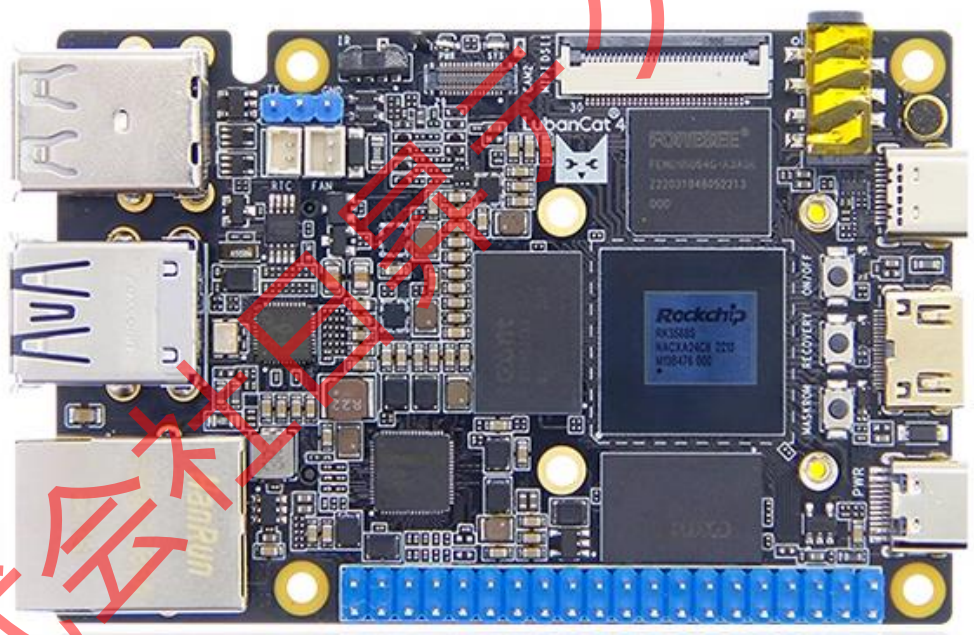
AI エッジ基板 CSAIEG358803 で AI 開発実践ガイド

株式会社日昇テクノロジー
https://www.csun.co.jp
info@csun.co.jp

作成日 2024/06/15



- **修正履歴**

NO	バージョン	修正内容	修正日
1	Ver1.0	新規作成	2024/06/15

※ この文書の情報は、文書を改善するため、事前の通知なく変更されることがあります。最新版は弊社ホームページからご参照ください。「https://www.csun.co.jp」

目　　次

AI エッジマニュアルシリーズについて

本マニュアル説明

このチュートリアルは、開発者が RK3588 基板で Python を使用してアプリケーションの開発、画像処理、周辺ハードウェアの制御などを行うのをガイドすることを目的としています。
このチュートリアルで使用される開発環境は以下の通りです：
· Python バージョン：Python3 以上
· 開発ボードシステム：基板付属するシステムイメージ（extboot パーティションの Ubuntu、Debian など）に適合するボード。

基板でセットにその他のマニュアル

《組み込み Linux イメージの構築と展開》
《Linux 基礎とアプリケーション開発実践ガイド》
《組み込み Linux ドライバ開発実践ガイド 》

日昇について

不可能への挑戦！
15 年以上の IOT 製品開発・量産経験、10 年以上のカメラモジュール設計・量産・画像処理実績、6 年以上の AI 開発システムノウハウを生かして、皆様により安い・最適な画像処理・コンピュータービジョン・AI 認識等のソリューションを続けて提供しております。
プロフェッショナルなカメラモジュールと AI 画像処理ソリューションの専門プロバイダとなり、HW と合わせて AI 活用を安く早く実現できます。ビジョン世界を専念にしてから専門の画像処理×IOT システム×AI ディープラーニングの土台を安く早く構築できます、AIOT 製品特に画像処理関連製品を開発する皆様に、ぜひこの経験を生かしてください。
連絡先：

ホームページ：https://www.dragonwake.com
通販サイト　：https://www.csun.co.jp
相談・問い合わせメール：info@csun.co.jp
電話：044-328-9098　　FAX：044-328-9097

本マニュアルのサンプルソースコードは下記 URL からダウンロードしてください。

https://www.dragonwake.com/download/LubanCat4/7-srccode/tutorial/CSAIEG358803_rk_code_storage.zip

解凍後：サブフォルダー「lubancat_ai_manual_code」のなか、すべてソースがマニュアルに使われるものです。

第 1 章 AI エッジボード CSAIEG358803 の紹介

AI 技術が急速に発展する中、クラウドでの展開にはプライバシー保護、通信遅延、コストなどの問題があります。これらの課題を解決するため、AI 技術と組み込みシステムを組み合わせてエッジコンピューティングを構築することが技術のホットスポットとなっています。

しかし、多くの人気 AI アルゴリズムには多大な計算能力とストレージが必要であり、組み込みデバイスには高い計算能力と低消費電力が求められます。AI 処理チップの開発により、複雑な AI アルゴリズムを実現する多くの方法が可能となりました。現在、組み込みで AI を実現する方法として、既存の組み込みプロセッサを最適化する方法、GPU マルチプロセッサを使用する方法、専用の計算アクセラレーションユニットを使用する方法などがあり、これらの方法はそれぞれ長所と短所があります。実際の応用分野に応じて異なる解決策を選択します。

AI エッジボード CSAIEG358803（Lubancat4）は、Rockchip 社設計の低消費電力で高性能なシングルボードコンピュータであり、豊富なハードウェアリソースを保持し、ユーザーのニーズを十分に考慮しつつ、コストを最適化し、できるだけ多くの周辺機能を引き出します。このシリーズのボードは、さまざまなシナリオで使用できます。CSAIEG358803（Lubancat4）は Rockchip rk3588S プロセッサを使用しています。

rk3588S プロセッサは、ニューラルネットワーク処理ユニット（NPU）を搭載しており、強力な計算能力を持ち、主流の深層学習フレームワークをサポートしています。また、GPU、VPU、RGA などのハードウェア単位が統合されており、これらのハードウェア演算単位を紹介します。

1.1 CPU

CSAIEG358803（Lubancat4）は、Rockchip rk3588S プロセッサを使用しています。このプロセッサの特徴は以下の通りです：

- 8 コア 64 ビットプロセッサ、4 コア Cortex-A76 と 4 コア Cortex-A55 のビッグリトル構造、および独立した NEON コプロセッサを統合
- 大コアは最大 2.4GHz、小コアは最大 1.8GHz をサポート
- ARM アーキテクチャ v8-A 命令セットを完全実装、ARM Neon Advanced SIMD（シングル命令・マルチデータ）によりメディアおよび信号処理を加速
- TrustZone 技術をサポート

1.2 NPU（ニューラルネットワーク処理ユニット）

rk3588 は、6TOPS の独立した NPU を内蔵しており、INT8 および INT16 畳み込み演算をサポートし、ディープラーニングフレームワーク（TensorFlow、TF-lite、Pytorch、Caffe、ONNX など）をサポートしています。

NPU は神経ネットワークのアルゴリズムを加速するために特別に設計された処理ユニットであり、機械視覚や自然言語処理などの人工知能分野における重要なアルゴリズムの実行を迅速化します。人工知能の応用範囲が拡大するにつれて、顔追跡、ジェスチャーおよび身体追跡、画像分類、ビデオ監視、音声認識（ASR）、および高度運転支援システム（ADAS）など、さまざまな機能を提供しています。

RKNPU を使用すると、Rockchip 社は公式に RKNN コンポーネントを提供しています。これには RKNPU2、RKNN Toolkit2、および RKNPU ドライバが含まれます。

RKNPU2 開発キットには、C/C++プログラミングインターフェースを提供するランタイムライブラリ（librknnrt.so）が含まれており、Linux または Android システム上で RKNN モデルの推論を展開するために使用されます。

RKNN Toolkit2 開発キット（Python インターフェース）は、PC および Rockchip NPU プラットフォーム上でのモデル変換、量子化、モデル推論、性能およびメモリ評価、量子化精度分析、モデル暗号化などの機能を提供します。このキットには、ボード上で RKNN モデルを展開するための Python プログラミングインターフェースを提供する RKNN Toolkit Lite2 も含まれています。

RKNN モデルは、Rockchip NPU 上でのモデル推論を加速するために、Rockchip の NPU ハードウェアアーキテクチャに基づいて定義されたモデルフォーマットであり、このフォーマットを使用して定義されたモデルは Rockchip NPU 上でより高い性能を発揮できます。

RKNPU ドライバは NPU ハードウェアインターフェースプログラムを提供しており、ボードのシステムファームウェアはすでに適応されています。

第 2 章 開発環境の紹介

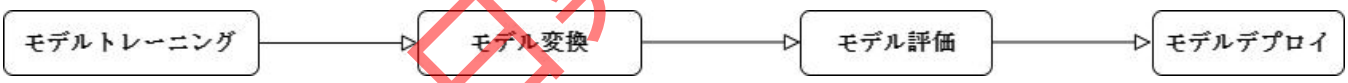
2.1 RKNN 開発環境

PC 端では主にモデルのトレーニングとモデルの変換などを行います。Windows システム、Windows 上の仮想マシン Ubuntu、Docker の Linux システム、クラウドサーバーなどを選択できます。PC 端には、Pycharm、Python、クロスコンパイラなどの一般的なソフトウェアとライブラリをインストールし、Pytorch、TensorFlow、PaddlePaddle などのディープラーニングフレームワークもインストールします。各ソフトウェアの使用には、仮想環境を作成して分離することをお勧めします。一般的な Python 仮想環境として Anaconda または Miniconda を使用します。

AI エッジボード CSAIEG358803（Lubancat4）上では、システムは Ubuntu または Debian を使用し、カーネルはデフォルトで rknn ドライバに適応しています。その他の rknn 関連コンポーネントも含まれており、必要に応じて後続の具体的なチュートリアルを参照してください。さらに、Python、cmake、make、gcc、opencv などの一般的な関連ソフトウェアとライブラリもインストールします。

後続のチュートリアルテストでは、PC 端は WSL2 あるいは Ubuntu20.04 を使用し（Pycharm と組み合わせて使用など）、モデルのトレーニングと変換を行い、ルバンキャットボードシステムは Debian10/11 を例にとり、デプロイと推論テストを行います。

2.2 RKNN 開発フロー



- **モデルトレーニング**：モデルのトレーニング前に具体的なプロジェクト問題に基づいてモデルを選択し、データを収集し、適切なディープラーニングフレームワークを使用してモデルのトレーニングを行います。RKNN モデルのオペレーターサポートについては、RKNN_Compiler_Support_Operator_List を参照してください。
- **モデル変換**：トレーニングされたディープラーニングモデルは RKNN 形式のモデルに変換されます。
- **モデル評価**：RKNN-Toolkit2 ツールを使用してモデルの量子化と性能評価を行い、精度、ボード推論性能、メモリ使用量などの重要な指標を評価します。評価に基づいてモデルを修正および最適化します。モデルの最適化については、RKNPU_User_Guide_RKNN_SDK を参照してください。
- **モデルデプロイ**：変換された RKNN モデルをボードにデプロイします。具体的な方法については、rknpu ランタイムライブラリと RKNN-Toolkit-lite2 の使用を参照してください。

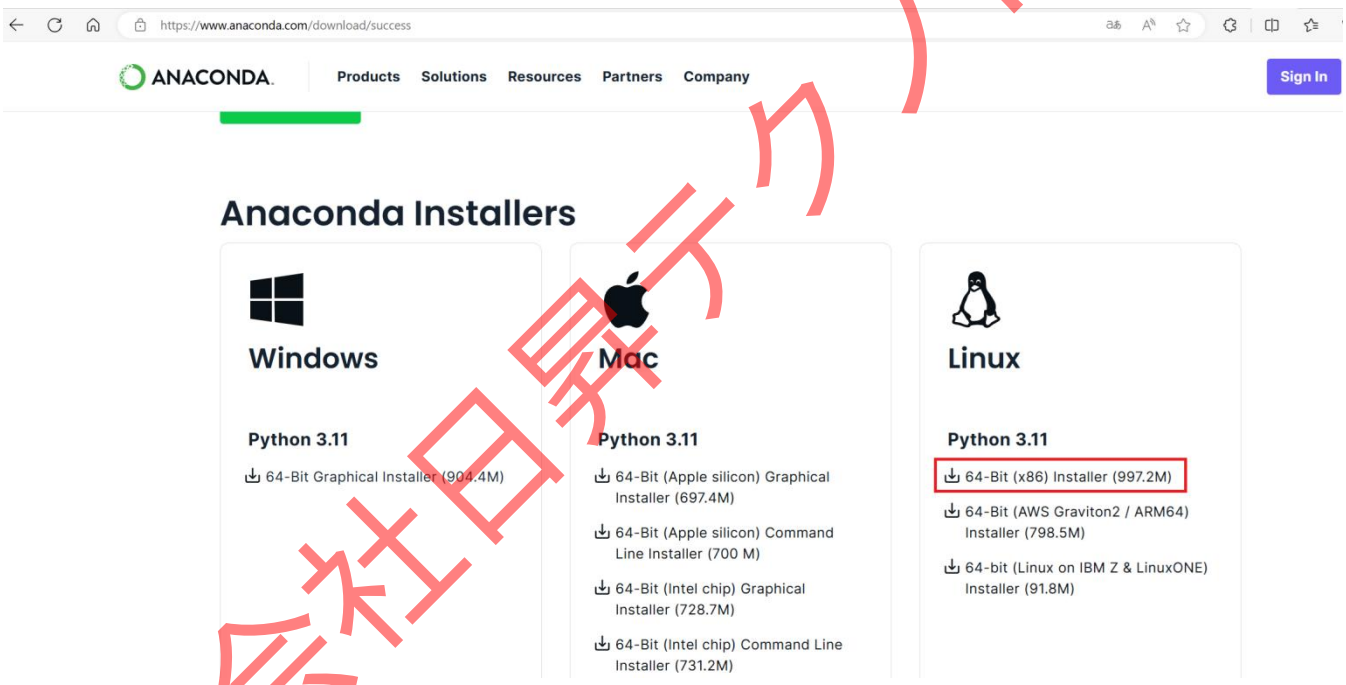
2.3 関連ソフトウェアのインストール

以下のソフトウェアとツールはPC側のUbuntu20.04にインストールします。

2.3.1 Anacondaのインストール

Anacondaは、パッケージを簡単に取得でき、パッケージ管理と環境管理を統一的に行えるディストリビューションです。Anacondaには、conda、Python、その他多くの科学計算、データ分析パッケージが含まれています。

Anacondaのインストールは、公式ウェブサイトから直接ダウンロードできます。
https://www.anaconda.com/download#downloadsを開き、システムに応じたバージョンを選択してインストールします。チュートリアルテストではUbuntu20.04を使用し、Linuxのインストールパッケージを選択します：



または、以下のコマンドを使用してインストールパッケージを直接取得します：

```
wget https://repo.anaconda.com/archive/Anaconda3-2024.02-1-Linux-x86_64.sh
```

インストールパッケージを取得したら、Linuxシステムでデフォルトのシェルを使用して以下のコマンドを実行してインストールします：

```
./Anaconda3-2024.02-1-Linux-x86_64.sh
```

次に Enter キーを押して、ライセンスを読み終えるまで下にスクロールし、最後に「yes」と入力して Enter キーを押します。その後、インストールが開始され、インストールが完了したら「yes」と入力して anaconda3 を初期化します：

```
done
installation finished.
Do you wish the installer to initialize Anaconda3
by running conda init? [yes|no]
[no] >>> yes
no change /home/csun/anaconda3/condabin/conda
no change /home/csun/anaconda3/bin/conda
no change /home/csun/anaconda3/bin/conda-env
no change /home/csun/anaconda3/bin/activate
no change /home/csun/anaconda3/bin/deactivate
no change /home/csun/anaconda3/etc/profile.d/conda.sh
no change /home/csun/anaconda3/etc/fish/conf.d/conda.fish
no change /home/csun/anaconda3/shell/condabin/Conda.psm1
no change /home/csun/anaconda3/shell/condabin/conda-hook.ps1
no change /home/csun/anaconda3/lib/python3.11/site-packages/xontrib/
,
→conda.xsh
no change /home/csun/anaconda3/etc/profile.d/conda.csh
modified /home/csun/.bashrc
==> For changes to take effect, close and re-open your current shell. <==
If you'd prefer that conda's base environment not be activated on startup,
set the auto_activate_base parameter to false:
conda config --set auto_activate_base false
Thank you for installing Anaconda3!
```

システムを再起動するか、以下のコマンドを使用して Anaconda 環境に入ります（コマンドラインのプロンプトの前に「base」が表示されます）：

```
source ~/.bashrc
#以下は一般的な conda コマンドの一部です。詳細なコマンドは「conda --help」を参照してください。
#環境を作成します。env_name は環境名で、後で Python のバージョンなどを指定できます：
conda create -n nc
#作成した仮想環境のリストを表示します：
conda info --envs
#環境を有効化します。env_name を指定しない場合は、デフォルトで base 環境に入ります：
conda activate nc
#環境を削除します。nc は環境名です：
conda remove -n nc --all
#現在の conda 環境から退出します：
conda deactivate
#ターミナルを開くたびに自動的に Anaconda 環境に入るのを防ぐには、以下のコマンドを使用して再起動します：
conda config --set auto_activate_base false
```

デフォルトの Anaconda インストールには多くのデフォルトライブラリが含まれています。軽量版を使用したい場合は、Miniconda を使用できます。これは Anaconda の軽量版で、科学計算やデータ分析パッケージが含まれていません。リンクをクリックしてダウンロードし、インストールと使用手順は同様です。

注：マニュアル≪AI エッジ基板 CSAIEG358803 で Python 開発及び実践≫で環境を作ったら、本節を参考のみです、筆者が python 自身のコマンド「python3 -m venv .toolkit2_env」を作成しました。

2.3.2 rknn-toolkit2 のインストール

RKNN Toolkit2 開発キットは PC x86_64 プラットフォーム上で実行され、モデルの変換、量子化、モデル推論、性能評価、メモリ評価、量子化精度分析、モデル暗号化などの機能を提供します。

rknn-toolkit2 のインストールについては、後続の RKNN Toolkit2 紹介セクションを参照するか、Anaconda を使用してインストールします（チュートリアルテスト時の toolkit2 バージョンは 2.0.0 です）：

#toolkit2_2.0 という名前の環境を作成し、Python バージョンを指定します：

```
conda create -n toolkit2_2.0 python=3.8
conda activate toolkit2_2.0
```

#toolkit2 のソースコードをクローンします：

```
git clone https://github.com/airockchip/rknn-toolkit2
```

#pip ソースを設定します：

```
pip3 config set global.index-url https://pypi.org/simple
```

- pip を使用して指定バージョンのライブラリをインストールします（チュートリアルテスト時の toolkit2 バージョンは 2.0.0beta です。Python バージョンに応じてファイルを選択してインストールします）：

```
cd rknn-toolkit2
pip3 install -r packages/requirements_cp38-2.0.0b0.txt
```

- Python バージョンと rknn_toolkit2 バージョンに応じて whl ファイルを選択します。例として、python3.8 環境を作成した場合、「cp38」を含む whl ファイルを使用します：

```
pip3 install packages/rknn_toolkit2-2.0.0b0+9bab5682-cp38-cp38-linux_x86_64.whl
```

インストールが成功すると「Successfully」が表示されます。簡単なテストも行えます。

```
(.toolkit2_env) (base) csun@CSUN-PC-0013:~/project-Toolkit2/rknn-toolkit2-2.0.0/rknn-toolkit2$ python
Python 3.8.10 (default, Nov 22 2023, 10:22:35)
[GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from rknn.api import RKNN
>>> rknn = RKNN()
I rknn-toolkit2 version: 2.0.0b0+9bab5682
```

2.3.3 Jupyter Notebook のインストール

Jupyter Notebook は、ウェブベースのインタラクティブコンピューティングアプリケーションです。Jupyter Notebook はウェブ形式で開き、各コマンドが記述されると、その後すぐにコード実行後のデータ

変化を確認できます。プログラミング中に説明文書を記述する必要がある場合、同じページ内で直接記述でき、タイムリーな説明と解釈が可能です。

チュートリアルの一部の章には Jupyter Notebook ファイルが含まれており、conda コマンドを使用して Jupyter Notebook をインストールして、表示または操作することができます：

- 環境に Jupyter をインストールします：

```
conda install jupyter notebook
```

または、pip3 を使用してインストールすることもできます：

- Jupyter をインストールします：

```
pip3 install jupyter -i https://pypi.org/simple
```

詳細な設定変更については、Jupyter 公式ウェブサイトまたは 図解！Jupyter Notebook を徹底解説！（インストール・使い方・起動・終了方法） - ビジ Py（ai-inter1.com）を参照してください。

2.3.4 いくつかの学習フレームワークのインストール

以下にいくつかのディープラーニングフレームワークのインストールを示します。詳細は対応するディープラーニングフレームワークの公式ウェブサイトを参照してください。

1. PaddlePaddle のインストール

ここでは conda を使用して PaddlePaddle をインストールします。具体的なコマンドは公式ウェブサイトを参照し、対応するバージョン、オペレーティングシステム、conda インストール方法、計算プラットフォームを選択して、対応するインストールコマンドを取得します：

- paddle という名前の環境を作成し、Python バージョンを指定します：

```
conda create -n paddle python=3.8
```

- 環境に入ります：

```
conda activate paddle
```

- 最新バージョンの PaddlePaddle をインストールします（テスト時の最新バージョンは 2.6.1 です）：

```
conda install paddlepaddle==2.6.1 --channel
https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/Paddle/
```

- インストールの成功を検証し、バージョンを確認します：

```
python3 -c "import paddle;paddle.utils.run_check()"
```

```
(.toolkit2_env) (base) csun@CSUN-PC-0013:~/project-Toolkit2/rknn-toolkit2-2.0.0/rknn-toolkit2$ python3 -c "import paddle;paddle.utils.run_check()"

Running verify PaddlePaddle program ...
I0615 19:03:40.161260 122822 program_interpreter.cc:212] New Executor is Running.
W0615 19:03:40.166601 122822 gpu_resources.cc:119] Please NOTE: device: 0, GPU Compute Capability: 6.1, Driver API Version: 12.2, Runtime API Version: 11.8
W0615 19:03:40.222450 122822 gpu_resources.cc:164] device: 0, cuDNN Version: 8.9.
I0615 19:03:41.538120 122822 interpreter_util.cc:624] Standalone Executor is Used.
PaddlePaddle works well on 1 GPU.
PaddlePaddle is installed successfully! Let's start deep learning with PaddlePaddle now.
```

GPU バージョンをインストールする場合は、最初に nvidia-smi コマンドを使用してサポートされている CUDA の最高バージョンを確認します。このコマンドがない場合は、グラフィックドライバを更新またはインストールする必要があります。

ヒント: WSL2 を使用している場合、GPU ドライバは Windows システムのものを使用し、WSL2 内ではサブシステムをサポートするグラフィックドライバをインストールするだけで、nvidia-smi コマンドを使用して上記と同様の表示が得られます。サポートされている CUDA の最高バージョンを確認し、そのバージョン以下の CUDA バージョンをサポートする PaddlePaddle のインストールコマンドを見つけてインストールします：

- paddle_gpu という名前の環境を作成し、Python バージョンを指定します：

```
conda create -n paddle_gpu python=3.8
```

- 環境に入ります：

```
conda activate paddle_gpu
```

- CUDA バージョン 12.1 の GPU バージョンの PaddlePaddle をインストールします：

```
conda install paddlepaddle-gpu==2.6.1 cudatoolkit=12.1 -c
https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/Paddle/ -c conda-forge
```

- インストールの成功を検証し、バージョンを確認します：

```
python3 -c "import paddle;paddle.utils.run_check()"
```

CUDA と cudnn を自分でインストールすることもできます（この場合、Conda にインストールされた CUDA と自分でインストールした CUDA の使用に注意してください）。詳細は次の URL を参照してください：https://developer.nvidia.com/cuda-toolkit-archive、https://developer.nvidia.com/rdp/cudnn-archive。

2. TensorFlow のインストール

python3-venv または Anaconda を使用して仮想環境を作成し、その後 pip を使用してインストールします。

Python 仮想環境を使用する場合：

- Python をインストールし、pip を更新します。Python 3.6-3.9 および pip 19.0 以上のバージョンを使用します：

```
sudo apt update
sudo apt install python3-dev python3-pip python3-venv
sudo python3 -m pip install pip --upgrade
```

- 通常は仮想環境を作成して使用します：

```
sudo python3 -m venv .tensorflow_venv
```

- 環境を有効化します：

```
source .tensorflow_venv/bin/activate
```

- 最新の安定版 TensorFlow をインストールします。以前のバージョンを指定することもできます（GPU と CPU バージョンはすでに統合されています）。GPU バージョンを使用するには、CUDA と cudnn をインストールする必要があります。詳細は前述のリンクを参照してください：

```
pip3 install tensorflow
```

- インストールの成功を検証し、バージョンを確認します：

```
python3 -c "import tensorflow as tf;print(tf.__version__)"
```

または、conda を使用して仮想環境を作成します：

- TensorFlow という名前の環境を作成し、Python バージョンを指定します：

conda create -n TensorFlow python=3.8

- 環境に入ります：

conda activate TensorFlow

- 最新の安定版 TensorFlow をインストールします。以前のバージョンを指定することもできます（GPU と CPU バージョンはすでに統合されています）。GPU バージョンを使用するには、CUDA と cudnn をインストールする必要があります。詳細は前述のリンクを参照してください：

pip3 install tensorflow==2.8.0

- インストールの成功を検証し、バージョンを確認します：

python3 -c "import tensorflow as tf;print(tf.__version__)"

詳細なインストール手順は TensorFlow を参照してください。

TensorFlow バージョンが CUDA、Python バージョンの対応関係：

https://www.tensorflow.org/install/source?hl=ja#gpu

注：rknn_toolkit2 2.0.0b0 バージョンは TensorFlow 2.8.0 までにサーポートしますので、Python が 3.7～3.10、CUDA が 11.2 までサーポートします。

3. PyTorch のインストール

PyTorch のインストールについては、公式ウェブサイトを参照し、対応するバージョン、システム、言語、CUDA を選択し、インストールコマンドを確認します：

要求に合わない場合は、以前のバージョンのインストールを選択できます。以下に Conda を使用して仮想環境を作成し、pip を使用して PyTorch をインストールします：

- pytorch という名前の環境を作成し、Python バージョンを指定します：

conda create -n pytorch python=3.8

- 環境に入ります：

conda activate pytorch

- pip3 を使用して最新バージョンの PyTorch をインストールします（計算プラットフォームは GPU です）。PyTorch 公式ウェブサイトを参照し、自分のシステム環境に応じたインストールコマンドを確認します：

pip3 install torch torchvision torchaudio

- インストールの成功を検証し、バージョンを確認します。また、conda list を使用しても確認できます

python3 -c "import torch;print(torch.__version__)"

GPU のバージョンをインストールしている場合、下記のコマンドにより True を出力されれば使用可能
python3 -c "import torch;print(torch.cuda.is_available())"

2.4 参考リンク

https://developer.nvidia.com/cuda-toolkit-archive
https://developer.nvidia.com/rdp/cudnn-archive
https://www.anaconda.com/download#downloads
https://www.paddlepaddle.org.cn/documentation/docs/en/install/conda/linux-conda_en.html
https://www.tensorflow.org/install/source?hl=ja#gpu

第 3 章 RKNN Toolkit2 紹介

RKNN Toolkit2 開発キット（Python インターフェース）は PC プラットフォーム上で動作し、モデル変換、量子化機能、モデル推論、性能とメモリ評価、量子化精度分析、モデル暗号化などの機能を提供します。詳細な機能説明は、RKNN-Toolkit2 プロジェクトファイルの「RKNN-Toolkit2 ユーザーガイド」を参照してください。

本章では、PC（Ubuntu システム）上で RKNN-Toolkit2 を使用してモデル変換、モデル推論、性能評価などを行う方法を簡単に紹介します。

重要：テスト環境：CSAIEG358803（Lubancat4）ボード、イメージシステムは Debian11、PC 環境は Ubuntu20.04、チュートリアル作成時の RKNN-Toolkit2 バージョンは 2.0.0b0、NPU ドライバは 0.9.2。

3.1 Toolkit2 のインストール

RKNN-Toolkit2 の現在のバージョンは Ubuntu18.04/Ubuntu20.04/Ubuntu22.04 に対応しており、詳細な依存関係と使用情報は RKNN Toolkit2 マニュアルを参照してください。

Toolkit2 をインストールするには、Python のパッケージマネージャ pip3 を使用するか、Docker を使用して Toolkit2 環境を構築します。関連する依存ライブラリと Docker ファイルは、Rockchip 公式の RKNN-Toolkit2 プロジェクト、ダウンロードした RKNN-Toolkit2 ファイルには、RKNN Toolkit Lite2 も含まれています。

以下のテストでは、Python の venv 仮想環境を使用して Toolkit2 をインストールしますが、Anaconda や Miniconda を使用して環境を作成することもできます。Miniconda は Anaconda の軽量版です。

```
# python ツールのインストール、ubuntu20.04 ではデフォルトで python3.8.10 がインストールされています。
# 仮想環境として python3.8-venv を使用するか、Anaconda/Miniconda をインストールし、conda で環境を管理します。
sudo apt update
sudo apt-get install python3-dev python3-pip python3.8-venv gcc

# 関連ライブラリとソフトウェアパッケージのインストール
sudo apt-get install libxslt1-dev zlib1g-dev libglib2.0 libsm6 libgl1-mesa-glx libprotobuf-dev gcc
```

RKNN-Toolkit2 のインストール：

```
# ディレクトリを作成します。テストに使用する ubuntu20.04 にすでにインストールされているパッケージと
# RKNN-Toolkit2 の動作に必要なパッケージのバージョンが異なる可能性があるため、他の問題を避けるために
# python venv で環境を分離します。
```

```
mkdir project-Toolkit2 && cd project-Toolkit2

# .toolkit2_env は環境の名前で、自由に変更できます
python3 -m venv .toolkit2_env

# 環境を有効化して入りましょう
source .toolkit2_env/bin/activate

# 公式の RKNN-Toolkit2 リポジトリから最新バージョンをクローンするか、関連するサンプルコードを取得します（toolkit2 バージョンは 2.0.0b0）。
git clone https://github.com/airockchip/rknn-toolkit2.git

# pip ソースの設定
pip3 config set global.index-url https://pypi.org/simple

# 依存ライブラリのインストール。rknn-toolkit2/packages/requirements_cp38-2.0.0b0.txt に従ってインストールします。
pip3 install numpy
pip3 install -r doc/requirements_cp38-2.0.0b0.txt

# rknn_toolkit2 のインストール。システムの Python バージョンに応じて異なる whl ファイルを選択してインストールします。
pip3 install packages/rknn_toolkit2-2.0.0b0+9bab5682-cp38-cp38-linux_x86_64.whl
```

インストールが成功したかどうかを確認する：

```
(.toolkit2_env) (base) csun@CSUN-PC-0013:~$ python3
Python 3.8.10 (default, Nov 22 2023, 10:22:35)
[GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from rknn.api import RKNN
>>>
```

`quit()`と入力するか、Ctrl+D を使用して終了します。

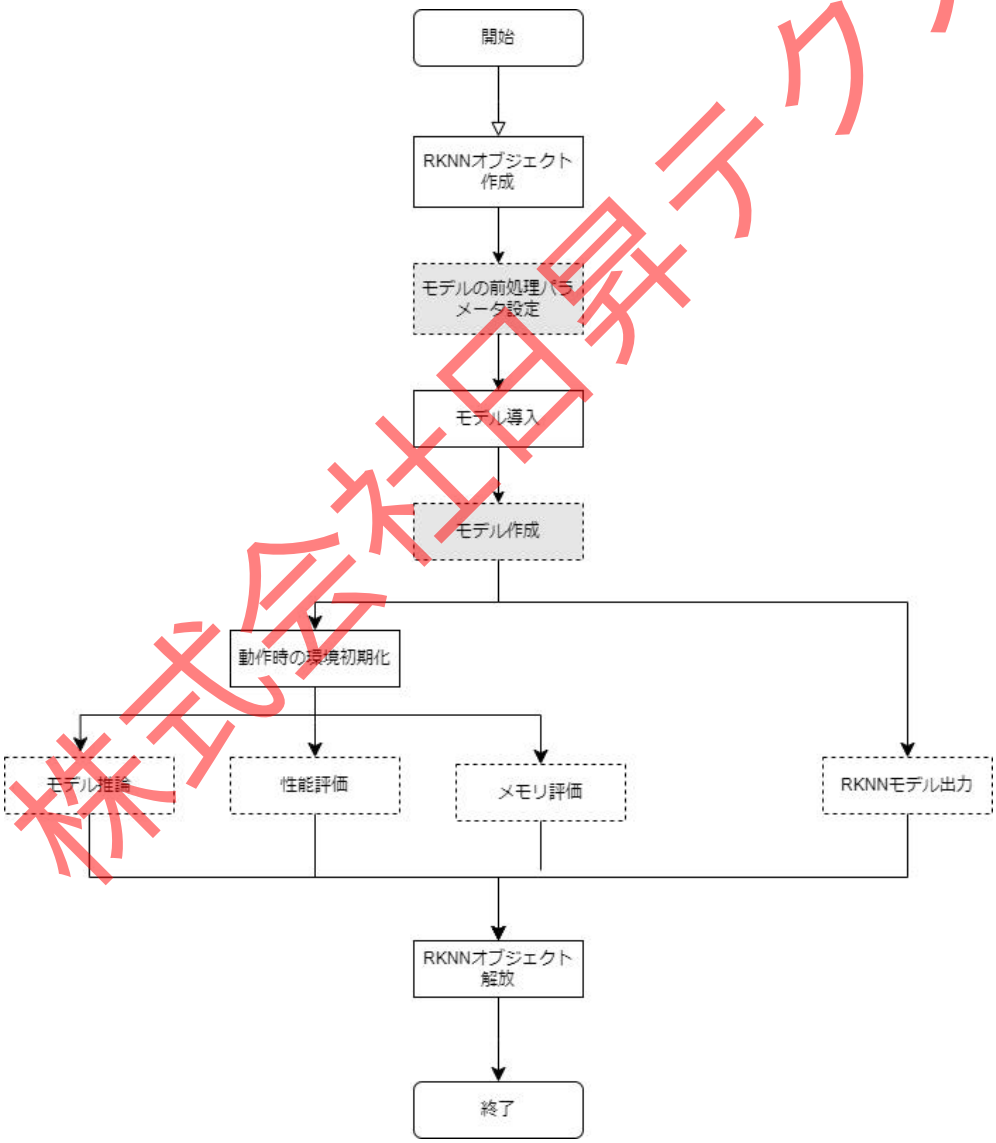
3.2 RKNN Toolkit2 インターフェースの使用

この節では、Toolkit-lite2 ツールを紹介します。このツールは PC プラットフォーム上で使用され、Python インターフェースを提供してモデルのデプロイと実行を簡素化します。ユーザーはこのツールを使用して以下の機能を簡単に実行できます：

- **モデル変換**：Toolkit-lite2 ツールは、Caffe、TensorFlow、TensorFlow Lite、ONNX、Pytorch、MXNet などの元のモデルを RKNN モデルに変換できます。また、RKNN モデルをインポートして NPU プラットフォームで推論を実行することもサポートしています。
- **量子化機能**：浮動小数点モデルを固定小数点モデルに量子化することをサポートしています。現在サポ

ートされている量子化方法は非対称量子化（asymmetric_quantized-8）であり、混合量子化機能もサポートしています。
- **モデル推論**：PC 上で NPU の動作をシミュレートして RKNN モデルを実行し、推論結果を取得できます。または、RKNN モデルを指定された NPU デバイスにデプロイして推論を実行し、推論結果を取得できます。
- **性能とメモリ評価**：ボードに接続して、RKNN モデルを指定された NPU デバイスにデプロイして実行し、実際のデバイス上でのモデルの性能とメモリ使用量を評価します。
- **量子化精度分析**：量子化されたモデルの各層の推論結果と浮動小数点モデルの推論結果のコサイン距離を提供し、量子化誤差を分析して量子化モデルの精度を向上させるための手がかりを提供します。
- **モデル暗号化機能**：指定された暗号化レベルで RKNN モデル全体を暗号化します。RKNN モデルの復号化は NPU ドライバ内で行われるため、暗号化モデルを使用する際には通常の RKNN モデルと同様に読み込み、NPU ドライバが自動的に復号化します。

Toolkit-lite2 を使用すると、PC 上で動作し、シミュレータを通じてモデルを実行し、推論やモデル変換などの操作を行うことができます。また、接続されたボードの NPU 上で実行し、RKNN モデルを NPU デバイスに転送して実行し、推論結果、性能情報などを取得することもできます。
Toolkit-lite2 でモデルを実行する際の簡単なフロー：



- RKNN オブジェクトを作成し、RKNN 環境を初期化します。
- モデルの前処理パラメータを設定します。PC 上でシミュレータを使用してモデルを実行する場合、config インターフェースを呼び出してモデルの前処理パラメータを設定します。接続されたボードの NPU で RKNN モデルをインポートして実行する場合は、設定は不要です。
- モデルをインポートします。PC 上でシミュレータを使用してモデルを実行する場合、load_caffe、load_tensorflow などのインターフェースを使用して対応する非 RKNN モデルをインポートします。接続されたボードの NPU で実行する場合は、load_rknn インターフェースを使用して RKNN モデルをインポートします。
- RKNN モデルを構築します。PC 上でシミュレータを使用してモデルを実行する場合、build インターフェースを呼び出して RKNN モデルを構築し、RKNN モデルをエクスポートしたり、実行環境を初期化して推論などの操作を行ったりします。接続されたボードの NPU で実行する場合は、これらの操作は不要です。
- 実行環境を初期化します。モデルの推論や性能評価を行うには、まず init_runtime インターフェースを呼び出して実行環境を初期化し、モデルの実行プラットフォーム（シミュレータまたは接続されたボードのハードウェア NPU）を指定する必要があります。
- 実行環境の初期化後、inference インターフェースを呼び出して推論を行い、eval_perf インターフェースを使用してモデル性能を評価したり、eval_memory インターフェースを使用してハードウェアプラットフォーム上でのモデルのメモリ使用量を取得したりします（モデルはハードウェアプラットフォーム上で実行する必要があります）。
- 最後に release インターフェースを呼び出して RKNN オブジェクトを解放します。

load_rknn インターフェースを使用して RKNN モデルをインポートする場合、accuracy_analysis 精度分析を呼び出すことはできません。非 RKNN モデルをインポートし、モデル構築時に量子化を設定する必要があります。

詳細なインターフェース説明は、RKNN-Toolkit2 プロジェクトの doc/ディレクトリ内のユーザーマニュアルを参照してください。具体的な使用例については、RKNN-Toolkit2 プロジェクトの examples/functions ディレクトリ内のサンプルを参照してください。

3.2.1 モデル変換とモデル推論

この節では、PC 上でシミュレータを使用してモデルを実行し、モデル変換と推論を行う方法を紹介します。

サンプルコードは、チュートリアルのサンプルコードまたは RKNN-Toolkit2 プロジェクトファイルの example ディレクトリ内の onnx/yolov5 から取得できます。

すでに RKNN-Toolkit2 の実行環境が整っている場合は、以下のコマンドを実行します：

```
# サンプルコードの examples/onnx/yolov5 ディレクトリに切り替えます（チュートリアルのサンプルコードを使用）
(.toolkit2_env) (base) csun@CSUN-PC-0013:~$cd ~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/dev_env/rknn-toolkit2/examples/onnx/yolov5
```

test.py を実行し、モデル変換とモデル推論を行います
python3 test.py

```
  I rknn-toolkit2 version: 2.0.0b0+9bab5682
  --> Config model
  done
  --> Loading model
  I It is recommended onnx opset 19, but your onnx model opset is 12!
  I Model converted from pytorch, 'opset_version' should be set 19 in torch.onnx.export for
successful convert!
  I Loading : 100%|████████████████████████████████████████████| 125/125 [00:00<00:00,
22431.35it/s]
  done
  --> Building model
  ......(省略)
  D RKNN: [02:45:16.753] -----------------------------------------
  D RKNN: [02:45:16.753] <<<<<<<<< end: rknn::RKNNMemStatisticsPass
  I rknn buiding done.
  done
  --> Export rknn model
  done
  --> Init runtime environment
  I Target is None, use simulator!
  done
  --> Running model
  W inference: The 'data_format' is not set, and its default value is 'nhwc'!
  I GraphPreparing : 100%|████████████████████████████████████| 153/153 [00:00<00:00,
3604.71it/s]
  I SessionPreparing : 100%|██████████████████████████████████| 153/153 [00:00<00:00,
1672.65it/s]
  W inference: The dims of input(ndarray) shape (640, 640, 3) is wrong, expect dims is 4! Try
expand dims to (1, 640, 640, 3)!
  done
     class        score        xmin, ymin, xmax, ymax
  -------------------------------------------------
     person       0.884      [ 208,   244,   286,   506]
     person       0.868      [ 478,   236,   559,   528]
     person       0.825      [ 110,   238,   230,   533]
     person       0.334      [  79,   353,   122,   517]
     bus          0.705      [  92,   128,   554,   467]
  Save results to result.jpg!
```

test.py メインプログラムでは、onnx モデル（yolov5s.onnx）を設定してインポートし、モデルを量子化して RKNN モデル（yolov5s.rknn）をエクスポートします。その後、実行環境を初期化し、シミュレータを使用して推論を実行し、出力データを処理して最終結果の画像を生成します。

# mean_values は入力の平均値を設定し、std_values は入力の正規化値を設定します。
print('--> Config model')
rknn.config(mean_values=[[0, 0, 0]], std_values=[[255, 255, 255]], target_platform='rk3588')

```
print('done')
```

エクスポートが成功すると、現在のディレクトリに yolov5s.rknn ファイルが生成されます。ボードにデプロイするには、前述の RKNN API または後述の Toolkit Lite2 セクションを参照してください。PC 上で RKNN モデルをシミュレートして実行し、推論結果を取得することもできます。この場合、結果の画像（result.jpg）が現在のディレクトリに出力されます。

3.2.2 性能とメモリ評価

この節では、RKNN-Toolkit2 を使用して、接続されたボードの NPU 上で実行し、性能とメモリ評価、推論などを行う方法を紹介します。

RKNN Toolkit2 は PC 上で動作し、PC から USB で NPU デバイスに接続します。RKNN Toolkit2 は RKNN モデルを NPU デバイスに転送して実行し、推論結果や性能情報を取得します。

サンプルを開始する前に、以下の操作が必要です：

1. ボードに接続します。ここではネットワーク adb を使用してボードに接続します。ボード上で adbd を起動します（サンプルコードから取得可能）：

```
# ボード上で以下のコマンドを実行します：
cat@lubancat:~$ ./adbd &
[1] 41260
cat@lubancat:~$ install_listener('tcp:5037','*smartsocket*')
using port=5555
```

PC の Ubuntu20.04 で adb をインストールし、以下のコマンドを実行して adb server を起動します。ボード上で adbd を起動します。その後、ボードと PC を同じ LAN に接続し（ping で相互接続を確認）、以下のコマンドを使用します：

```
# PC 側で adb をインストールします
(.toolkit2_env) (base) csun@CSUN-PC-0013:~$ sudo apt install -y adb

# adb server を開始します
(.toolkit2_env) (base) csun@CSUN-PC-0013:~$ adb start-server

# ボードに接続します。IP は具体的なボードに応じて設定します。デフォルトポートは 5555 です。
(.toolkit2_env) (base) csun@CSUN-PC-0013:~$ adb connect 192.168.11.241
already connected to 192.168.11.241:5555

# 接続されたデバイスを確認します。接続が成功すると、これが RKNN の実行時の device_id となります。
```

1.2 有線接続 adb も可能です。まず、USB ガジェットを使用して OTG ポートを adb デバッグ用に設定します。Type-C ケーブルでボードと PC を接続します（仮想マシンを使用する場合は仮想マシンに接続することに注意してください）。その後、以下のコマンドを使用して id を確認します。

2. ボード上で rknn_server サービスを開始します。このサービスはボード上で実行されるバックグラウンドエージェントで、PC から USB 経由で送信されるプロトコルを受信し、ボード側のランタイム対応インターフェースを実行して結果を PC に返します。この rknn_server ファイルは、LubanCat ボードシステムのファームウェアにデフォルトで含まれています（RKNPU2 プロジェクトファイルから取得するか、サンプルコードから取得可能です）。

Linux プラットフォームでは、librknnrt.so ファイルを追加する必要があります（LubanCat ボードシステムにはデフォルトで含まれていますが、更新が必要な場合があります）。このファイルは、RKNPU2 の runtime/XXXX/Linux/rknn_server ディレクトリから取得するか、チュートリアルのサンプルコードから取得可能です。

次に、サンプルコードを取得してテストします（前の節でエクスポートした rknn モデルを使用）。

test.py メインプログラム：

上記のプログラムは実行環境を初期化してメモリと性能の評価を行い、NPU デバイス上で推論を行うこともできます。詳細な実装は前の節の PC シミュレータ上での推論を参照してください。

その他の rknn-toolkit2 の機能テストの例は、https://github.com/rockchip-linux/rknn-toolkit2/tree/master/examples/functions を参照してください。

3.3 ボード情報の確認と設定

3.3.1 RK3588（RK3588s/RK3588）の場合

1. NPU 周波数の確認と設定：

```
# ドライババージョンを確認
cat /sys/kernel/debug/rknpu/version

# 電源状態を確認
cat /sys/kernel/debug/rknpu/power

# NPU の使用率を確認（root 権限が必要）
cat /sys/kernel/debug/rknpu/load

# NPU の利用可能な周波数を確認し、周波数を設定
cat /sys/class/devfreq/fdab0000.npu/available_frequencies
echo userspace > /sys/class/devfreq/fdab0000.npu/governor
echo 1000000000 > /sys/class/devfreq/fdab0000.npu/userspace/set_freq

# 現在の NPU 動作周波数を確認
cat /sys/kernel/debug/rknpu/freq
```

3.3.2 NPU その他関連情報

```
# librknnrt ライブラリバージョンを確認
strings /usr/lib/librknnrt.so | grep "librknnrt version"

librknnrt version: 2.0.0b0 (35a6907d79@2024-03-24T10:31:14)

# rknn_server バージョンを確認
strings /usr/bin/rknn_server | grep build

2.0.0b0 (18eacd0 build@2024-03-22T14:07:19)
rknn_server version: 2.0.0b0 (18eacd0 build@2024-03-22T14:07:19)
.note.gnu.build-id

# NPU ドライババージョンを確認
dmesg | grep -i rknpu
```

NPU ドライバの更新については、カーネルをコンパイルしてボードのカーネルを更新する必要があります。LubanCat ボードは、コマンドを使用して直接更新するか、クラウドストレージの最新イメージをフラッシュすることができます。

```
sudo apt update
# lubancat4 ボードの場合
sudo apt install linux-image-5.10.160
```

3.4 参考リンク

- https://github.com/airockchip/rknn-toolkit2
- https://github.com/rockchip-linux/rknn-toolkit2
- https://github.com/rockchip-linux/rknpu2

第 4 章 RKNN Toolkit Lite2 介绍

RKNN Toolkit Lite2 は、Rockchip NPU プラットフォームのプログラミングインターフェース（Python）であり、ボード上で RKNN モデルをデプロイするために使用されます。本章では、Toolkit Lite2 の使用フローとボード上でのデプロイの例を簡単に紹介します。

重要：テスト環境：LubanCat RK シリーズボードのイメージシステムは Debian11、チュートリアル作成時の RKNN-Toolkit2 バージョンは 2.0.0b0、ボードの NPU ドライバは 0.9.2。

4.1 Toolkit Lite2 のインストール

Toolkit Lite2 は現在、Debian10/11（aarch64）システムに対応しています。詳細な依存関係と使用情報は「RKNN Toolkit Lite2 ユーザー使用ガイド」を参照してください。

ボード上で RKNN Toolkit Lite2 を入手するには、公式の GitHub から直接取得するか、サンプルコードから取得します：

#サンプルソースコードは下記 URL からダウンロードしてください。
https://www.dragonwake.com/download/LubanCat4/7-srccode/tutorial/CSAIEG358803_rk_code_storage.zip
解凍後：サブフォルダー「lubancat_ai_manual_code」であります。
#toolkit2 ツール（2.0.0b2）ダウンロード URL：
https://www.dragonwake.com/download/LubanCat4/7-srccode/tutorial/rknn_toolkit2-dev_env.zip
cd lubancat_ai_manual_code/dev_env/rknn_toolkit_lite2/
または
git clone https://github.com/airockchip/rknn-toolkit2.git
もし取得したのがチュートリアルの toolkit2 ツールであれば、ディレクトリ構造は以下のようになります：

```
.
├── docs
│   ├── change_log.txt
│   ├── 02_Rockchip_RKNPU_User_Guide_RKNN_SDK_V2.0.0beta0_EN.pdf
│   └── 02_Rockchip_RKNPU_User_Guide_RKNN_SDK_V2.0.0beta0_CN.pdf
├── examples
│   ├── inference_with_lite
│   │   ├── resnet18_for_rk3562.rknn
│   │   ├── resnet18_for_rk3566_rk3568.rknn
│   │   ├── resnet18_for_rk3588.rknn
│   │   ├── space_shuttle_224.jpg
│   │   └── test.py
│   └── yolov5_inference
│       ├── bus.jpg
│       ├── test.py
│       ├── yolov5s_for_rk3562.rknn
│       ├── yolov5s_for_rk3566_rk3568.rknn
│       └── yolov5s_for_rk3588.rknn
└── packages
    ├── rknn_toolkit_lite2-2.0.0b0-cp311-cp311-linux_aarch64.whl
    ├── rknn_toolkit_lite2-2.0.0b0-cp310-cp310-linux_aarch64.whl
    ├── rknn_toolkit_lite2-2.0.0b0-cp37-cp37m-linux_aarch64.whl
    ├── rknn_toolkit_lite2-2.0.0b0-cp38-cp38-linux_aarch64.whl
    ├── rknn_toolkit_lite2-2.0.0b0-cp39-cp39-linux_aarch64.whl
    └── rknn_toolkit_lite2_2.0.0b0_packages.md5sum
5 directories, 19 file
```

環境インストール（LubanCat4 ボード、Debian11）：

```
sudo apt update
# Python ツールなどをインストール
sudo apt-get install python3-dev python3-pip gcc
# 依存関係とソフトウェアパッケージをインストール
pip3 install wheel
sudo apt-get install -y python3-opencv
```

Toolkit Lite2 ツールのインストール：

```
# rknn_toolkit_lite2 ディレクトリに移動
cd rknn_toolkit_lite2/
# ボードシステムの Python バージョンに応じて Debian11 ARM64 with Python3.9.2 の whl ファイルをインストール
# Debian11 Python3.9
pip3 install packages/rknn_toolkit2-2.0.0b0+9bab5682-cp38-cp38-linux_x86_64.whl
```

インストールが成功しました：
下記のような実行してエラーが出なければ、成功ということです。

```
cat@lubancat:~$ python
Python 3.9.2 (default, Feb 28 2021, 17:03:44)
[GCC 10.2.1 20210110] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from rknnlite.api import RKNNLite
>>>
```

4.2 Toolkit Lite2 インターフェースの使用

Toolkit Lite2 は主に RKNN モデルをボードにデプロイするために使用されます。他のモデルを RKNN モデルに変換するには RKNN Toolkit2 ツールを使用します。Toolkit Lite2 を使用して RKNN モデルをデプロイするフローは次のとおりです：

1. RKNNLite オブジェクトを作成します。
2. load_rknn インターフェースを呼び出して RKNN モデルをインポートします。対応するプラットフォーム（rk356x/rk3588）のモデルが必要です。
3. init_runtime インターフェースを呼び出して実行時環境を初期化します。
4. inference インターフェースを呼び出して入力データを推論し、推論結果を返し、結果を処理します。
5. 推論が完了したら、release インターフェースを呼び出して RKNNLite オブジェクトを解放します。

RKNN Toolkit Lite2 の詳細なインターフェース説明は、RKNN-Toolkit2 プロジェクトの rknn_toolkit_lite2/docs ディレクトリ内のユーザーマニュアルを参照してください。

4.3 ボード上での推論テスト

ボード上での推論テストでは、librknnrt.so ライブラリを呼び出します。このライブラリはボード上のランタイムライブラリです。デフォルトでボードのイメージの /usr/lib ディレクトリに librknnrt.so ライブラリがあり、デフォルトのライブラリバージョンは 1.5.0 です。必要に応じて、チュートリアルのサンプルコードまたは rknpu2 プロジェクトファイルから更新してください。

rknpu2 プロジェクトは https://github.com/rockchip-linux/rknpu2 から入手できます、或いは、toolkit2 ツール（2.0.0b2）ダウンロード URL：https://www.dragonwake.com/download/LubanCat4/7-srccode/tutorial/rknn_toolkit2-dev_env.zip からダウンロードできます。

librknnrt.so は、rknpu2 プロジェクトの runtime/ ディレクトリ内にあり、異なるボード、プラットフォーム、システムに応じてディレクトリを選択します。例えば、LubanCat4 ボードの場合、runtime/RK3588/Linux/librknn_api/aarch64/ ディレクトリ内の librknnrt.so を選択し、そのライブラリをボードシステムの /usr/lib/ ディレクトリにコピーします。

重要：RKNN-Toolkit2、RKNPU2 ランタイムライブラリの異なるバージョンは互換性がない場合があり、Invalid RKNN model version 6 エラーが発生する可能性があります。librknnrt.so ライブラリを更新するか、対応するバージョンの RKNN-Toolkit2 を使用して再度 rknn モデルを変換してください。

4.3.1 resnet18 推論テスト

RKNN Toolkit Lite2 のデモを実行し、取得した Toolkit Lite2 ディレクトリに移動し、../examples/inference_with_lite ディレクトリ内で以下のコマンドを実行します：

```
# チュートリアルのサンプルコード取得
toolkit2 ツール（2.0.0b2）ダウンロード URL:
https://www.dragonwake.com/download/LubanCat4/7-srccode/tutorial/rknn_toolkit2-dev_env.zip
# 解凍後 rknn_toolkit_lite2/examples/inference_with_lite ディレクトリに移動
cd rknn_toolkit2-dev_env/dev_env/rknn_toolkit_lite2/examples/inference_with_lite
# 推論コマンドを実行（以下は LubanCat-4 ボードでのテスト）
```

```
  cat@lubancat:~/lubancat_ai_manual_code/dev_env/rknn_toolkit_lite2/examples/yolov5_inference$
python3 test.py
  --> Load RKNN model
  done
  --> Init runtime environment
  I RKNN: [12:36:27.361] RKNN Runtime Information, librknnrt version: 2.0.0b0 (35a6907d79@2024-
03-24T10:31:14)
  I RKNN: [12:36:27.361] RKNN Driver Information, version: 0.9.2
  I RKNN: [12:36:27.362] RKNN Model Information, version: 6, toolkit version:
```

```
2.0.0b0+9bab5682(compiler version: 2.0.0b0 (35a6907d79@2024-03-24T02:34:11)), target: RKNPU v2,
target platform: rk3588, framework name: ONNX, framework layout: NCHW, model inference type:
static_shape
  done
  --> Running model
  Save results to result.jpg!
```

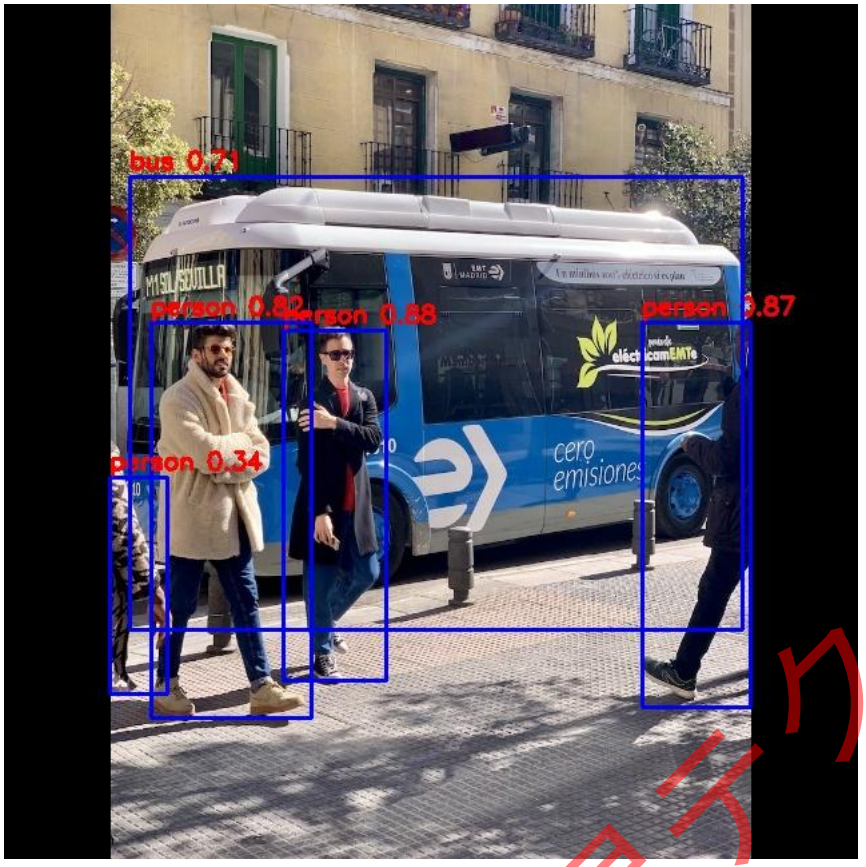
テスト例は、ボード（上記のテストは LubanCat-4 ボード）に応じて対応するプラットフォームの RKNN モデルを使用し、最終的に推論結果を出力して表示します。

4.3.2 yolov5 推論テスト

以下では、前の Toolkit Lite2 節で変換した yolov5s.rknn モデルをデプロイする簡単なテストを行い、test.py を作成し、ボードに応じてモデルをロードし、推論し、結果を後処理するなどの操作を行います（test.py ソースファイルはサンプルコードを参照してください）。

```
  cat@lubancat:~/lubancat_ai_manual_code/dev_env/rknn_toolkit_lite2/examples/yolov5_inference$
python3 test.py
  --> Load RKNN model
  done
  --> Init runtime environment
  I RKNN: [12:36:27.361] RKNN Runtime Information, librknnrt version: 2.0.0b0 (35a6907d79@2024-
03-24T10:31:14)
  I RKNN: [12:36:27.361] RKNN Driver Information, version: 0.9.2
  I RKNN: [12:36:27.362] RKNN Model Information, version: 6, toolkit version:
2.0.0b0+9bab5682(compiler version: 2.0.0b0 (35a6907d79@2024-03-24T02:34:11)), target: RKNPU v2,
target platform: rk3588, framework name: ONNX, framework layout: NCHW, model inference type:
static_shape
  done
  --> Running model
  Save results to result.jpg!
```

正常に動作すると、出力画像 out.jpg を表示できます。デフォルトのサンプルコード



yolov5_inference/test.py は cv2.imshow() を使用して画像を表示しないため、result.jpg を手動で確認できます。

4.4 参考リンク

- https://github.com/airockchip/rknn-toolkit2
- https://github.com/rockchip-linux/rknn-toolkit2
- https://github.com/rockchip-linux/rknpu2

第 5 章 RKNPU2

RKNPU2 は、RKNPU を搭載したチッププラットフォーム向けのクロスプラットフォームプログラミングインターフェース（C/C++）です。RKNN Toolkit2 でエクスポートされた RKNN モデルをデプロイし、AI アプリケーションの実現を加速させることができます。

RKNN API を使用する前に、まず RKNN Toolkit2 ツールを使用してユーザーのモデルを RKNN モデルに変換する必要があります（注意：RKNN Toolkit2 と RKNPU2 のバージョンが異なると互換性がない場合がありますので、同じバージョンを選択してください）。RKNN モデルが得られたら、RKNN API インターフェースを使用してプラットフォーム上でアプリケーションを開発することができます。

本章では、RKNN API ライブラリの開発フローおよび使用例を簡単に紹介します。

重要：テスト環境：LubanCat4 ボード、イメージシステムは Debian11、PC 環境は ubuntu20.04、チュートリアル作成時の RKNN Toolkit2 は 2.0.0b0 バージョン、RKNPU2 は 2.0.0b0 バージョン、NPU ドライバは 0.9.2 です。

5.1 RKNN API の使用

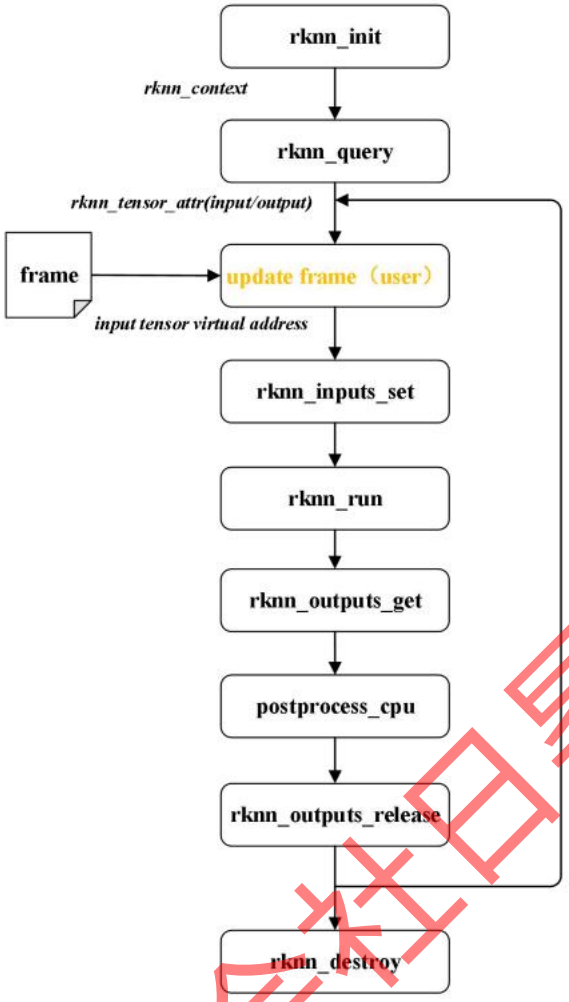
RKNN API ライブラリファイルは librknnrt.so（RK356X/RK3588 用）で、ヘッダーファイルには rknn_api.h および rknn_matmul_api.h があります。Linux システムでこのライブラリを使用するには、ライブラリファイルをシステムライブラリ検索パスに配置する必要があります。通常は/usr/lib に配置し、ヘッダーファイルは/usr/include に配置します。

LubanCat ボードの提供するイメージ（Debian11、ubuntu20.04）にはこのライブラリがインストールされています。更新が必要な場合は、rknpu2 プロジェクトの rknpu2/runtime/ディレクトリから最新のランタイムライブラリを取得してください。

RKNN API の呼び出しには、一般 API インターフェースとゼロコピー API インターフェースの 2 つの方法があります。RK356X/RK3588 では、これらのインターフェースの両方を使用できますが、ユーザー入力データが仮想アドレスのみの場合は一般 API インターフェースのみを使用できます。ユーザー入力データに物理アドレスや fd がある場合は、両方のインターフェースを使用できますが、一般 API とゼロコピー API を混在させることはできません。

5.1.1 一般 API インターフェース

RKNN 一般 API インターフェースでは、データの量子化、正規化、変換などの操作はすべて CPU で行われ、モデル推論は NPU で実行されます。フローは以下の通りです（画像は rknpu api ユーザーガイドを参照）：



関連関数の紹介：

- rknn_init：この関数は、渡された rknn_context オブジェクトを初期化し、RKNN モデルをロードし、flag および rknn_init_extend 構造体に基づいて特定の初期化動作を実行します。
- rknn_query：この関数は、モデルの入力出力情報、レイヤごとの実行時間、モデル推論の総時間、SDK バージョン、メモリ使用情報、ユーザー定義の文字列などの情報を取得できます。
- rknn_inputs_set：この関数は、モデルの入力データを設定できます。複数の入力をサポートし、各入力は rknn_input 構造体オブジェクトです。渡される前に入力データを設定する必要があります。
- rknn_run：この関数は、モデル推論呼び出しを 1 回実行します。呼び出す前に rknn_inputs_set 関数が必要です。
- rknn_outputs_get：この関数は、モデル推論の出力データを取得できます。複数の出力データを取得できます。
- rknn_outputs_release：この関数は、rknn_outputs_get 関数によって取得された出力に関連するリソー

スを解放します。
- rknn_destroy：この関数は、rknn_context およびその関連リソースを解放します。

一般 API インターフェースの簡単な呼び出しフロー：
1. **rknn_init**関数で rknn_context オブジェクトを初期化し、RKNN モデルをロードします。
2. **rknn_query**関数でモデルの入力出力情報を取得し、後続の関数で使用します。
3. **rknn_inputs_set**関数でモデルの入力を設定し、**rknn_run**関数で推論を実行します。
4. 推論が終了したら、**rknn_outputs_get**関数で出力データを取得し、ユーザーは関連後処理を行います。
5. 処理が完了したら、**rknn_outputs_release**関数で出力に関連するリソースを解放します。
6. フレームデータを更新し、**rknn_run**関数で推論を繰り返します。すべての推論が完了したら、**rknn_destroy**関数で rknn_context およびその関連リソースを解放します。

後続の rknn_yolov5_demo テスト例を実践し、ソースコードの呼び出しフローを確認すると、一般 API インターフェースの呼び出しフローがより明確になります。

5.1.2 ゼロコピー API インターフェース

一般 API インターフェースに加えて、ゼロコピー API インターフェースがあります。これは一般 API のデータ処理フローを最適化し、データの量子化、正規化、変換などの操作を NPU で行い、出力データの配置と逆量子化を CPU または NPU で行います。

2 つの API の主な違いは、一般インターフェースではフレームデータを更新するたびに外部モジュールで割り当てられたデータを NPU 実行時の入力メモリにコピーする必要があるのに対し、ゼロコピーインターフェースでは事前に割り当てられたメモリを直接使用するため、メモリコピーのオーバーヘッドが削減される点です。ゼロコピー API インターフェースの呼び出しフローは一般 API インターフェースと似ていますが、モデルの入力出力データメモリを設定する際に rknn_create_mem/_from_phys/_from_fd、rknn_set_io_mem、rknn_destroy_mem などのインターフェースを呼び出す必要があります。

2 つの API インターフェースのデータ処理フローは以下の通りです
（02_Rockchip_RKNPU_User_Guide_RKNN_SDK_V2.0.0beta0_EN.pdf を参照）：

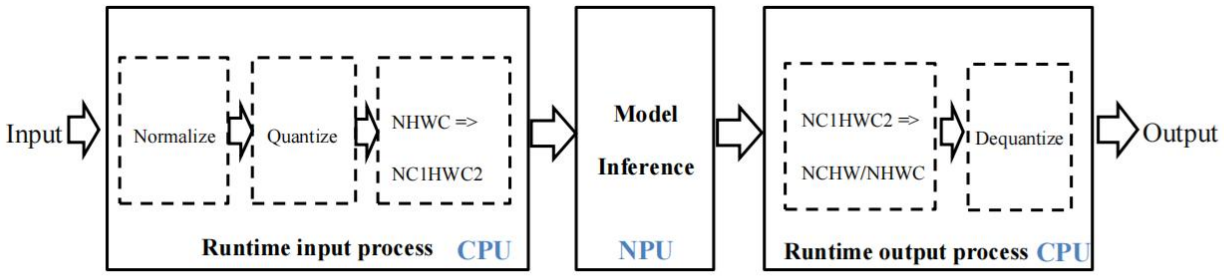


Figure 5-5 Data processing flow of general API

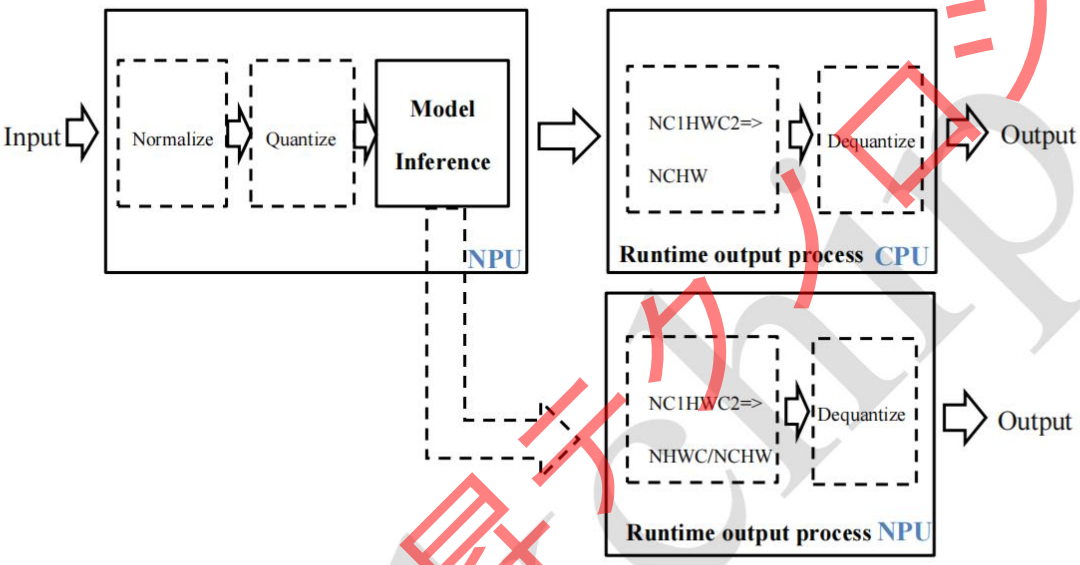


Figure 5-6 Data processing flow of zero-copy API

実際の呼び出しシナリオは比較的複雑で、モデルのモジュールメモリ（入力/出力/重み/中間結果）の自己割り当ての必要性や、メモリ表現方法（ファイルディスクリプタ/物理アドレスなど）によって、3 つの典型的なゼロコピー呼び出しフローに分類できます：

1. 入力/出力メモリがランタイムによって割り当てられる場合
2. 入力/出力メモリが外部によって割り当てられる場合
3. 入力/出力/重み/中間結果メモリが外部によって割り当てられる場合

詳細な呼び出しフローについては、RKNPU のユーザーマニュアルを参照してください。

詳細な RKNN API プラットフォームサポート状況とインターフェースの説明については、rknpu2 プロジェクトドキュメント「Rockchip_RKNPU_User_Guide_RKNN_API.pdf」、およびライブラリヘッダーファイルを参照してください。

5.2 Linux プラットフォームでのテスト

5.2.1 rknn_yolov5_demo 例のテスト

この例は一般 API インターフェースを使用しており、具体的なテスト手順は以下の通りです：

1. テスト例を取得：rknpu2 の rknn_yolov5_demo 例を使用します：
   # チュートリアルのサンプルコードを取得
toolkit2 ツール（2.0.0b2）ダウンロード URL:
https://www.dragonwake.com/download/LubanCat4/7-srccode/tutorial/rknn_toolkit2-dev_env.zip
解凍後 lubancat_ai_manual_code/dev_env/rknpu2/rknn_yolov5_demo ディレクトリに移動

   cd lubancat_ai_manual_code/dev_env/rknpu2/rknn_yolov5_demo
   または rknpu2 の GitHub から取得
   # git clone https://github.com/airockchip/rknpu2.git
   cd rknpu2-master/examples/rknn_yolov5_demo

2. テストモデルの更新：前述の RKNN Toolkit2 で変換された yolov5 RKNN モデルを使用するか、例のモデルを直接使用します。

3. シェルスクリプトを実行：例をコンパイルしてインストールします。PC 環境でクロスコンパイルするか、ボード上でコンパイルします。
   - PC 仮想マシンでクロスコンパイルする場合は、まずクロスコンパイラを取得し、環境変数を設定してからコンパイルしてインストールし、最後に scp や rsync などの方法でインストールディレクトリのファイルをボードにコピーします。
   # デフォルトのクロスコンパイラをインストール
   sudo apt update
   sudo apt install gcc-aarch64-linux-gnu g++-aarch64-linux-gnu
   # または以下のクロスコンパイラを使用
   # git clone https://github.com/LubanCat/gcc-buildroot-9.3.0-2020.03-x86_64_aarch64-rockchip-linux-gnu.git
   # 例のディレクトリに移動し、GitHub から取得したクロスコンパイラを使用する場合は、スクリプトファイルの GCC_COMPILER パスを変更
   cd lubancat_ai_manual_code/dev_env/rknpu2/rknn_yolov5_demo
   # ボードプロセッサに応じてスクリプトを実行します。ここでは LubanCat-4（プロセッサは rk3588s）をテストし、build-linux_RK3588.sh を実行します：

```
  .....省略......
  Install the project...
  -- Install configuration: ""
  -- Installing:
/home/cat/lubancat_ai_manual_code/dev_env/rknpu2/rknn_yolov5_demo/install/rknn_yolov5_demo_Linux/./rknn_yolov5_demo
```

```
  -- Installing:
/home/cat/lubancat_ai_manual_code/dev_env/rknpu2/rknn_yolov5_demo/install/rknn_yolov5_demo_Linu
x/lib/librknnrt.so
  -- Installing:
/home/cat/lubancat_ai_manual_code/dev_env/rknpu2/rknn_yolov5_demo/install/rknn_yolov5_demo_Linu
x/lib/librga.so
  -- Installing:
/home/cat/lubancat_ai_manual_code/dev_env/rknpu2/rknn_yolov5_demo/install/rknn_yolov5_demo_Linu
x/.//model
  -- Installing:
/home/cat/lubancat_ai_manual_code/dev_env/rknpu2/rknn_yolov5_demo/install/rknn_yolov5_demo_Linu
x/.//model/coco_80_labels_list.txt
  -- Installing:
/home/cat/lubancat_ai_manual_code/dev_env/rknpu2/rknn_yolov5_demo/install/rknn_yolov5_demo_Linu
x/.//model/RK3562
  -- Installing:
/home/cat/lubancat_ai_manual_code/dev_env/rknpu2/rknn_yolov5_demo/install/rknn_yolov5_demo_Linu
x/.//model/RK3562/yolov5s-640-640.rknn
  -- Installing:
/home/cat/lubancat_ai_manual_code/dev_env/rknpu2/rknn_yolov5_demo/install/rknn_yolov5_demo_Linu
x/.//model/test.mp4
  -- Installing:
/home/cat/lubancat_ai_manual_code/dev_env/rknpu2/rknn_yolov5_demo/install/rknn_yolov5_demo_Linu
x/.//model/RV110X
  -- Installing:
/home/cat/lubancat_ai_manual_code/dev_env/rknpu2/rknn_yolov5_demo/install/rknn_yolov5_demo_Linu
x/.//model/RV110X/yolov5s-640-640.rknn
  -- Installing:
/home/cat/lubancat_ai_manual_code/dev_env/rknpu2/rknn_yolov5_demo/install/rknn_yolov5_demo_Linu
x/.//model/test.h264
  -- Installing:
/home/cat/lubancat_ai_manual_code/dev_env/rknpu2/rknn_yolov5_demo/install/rknn_yolov5_demo_Linu
x/.//model/RK3566_RK3568
  -- Installing:
/home/cat/lubancat_ai_manual_code/dev_env/rknpu2/rknn_yolov5_demo/install/rknn_yolov5_demo_Linu
x/.//model/RK3566_RK3568/yolov5s-640-640.rknn
  -- Installing:
/home/cat/lubancat_ai_manual_code/dev_env/rknpu2/rknn_yolov5_demo/install/rknn_yolov5_demo_Linu
x/.//model/bus.jpg
  -- Installing:
/home/cat/lubancat_ai_manual_code/dev_env/rknpu2/rknn_yolov5_demo/install/rknn_yolov5_demo_Linu
x/.//model/RK3588
  -- Installing:
/home/cat/lubancat_ai_manual_code/dev_env/rknpu2/rknn_yolov5_demo/install/rknn_yolov5_demo_Linu
x/.//model/RK3588/yolov5s-640-640.rknn
  -- Installing:
/home/cat/lubancat_ai_manual_code/dev_env/rknpu2/rknn_yolov5_demo/install/rknn_yolov5_demo_Linu
x/./rknn_yolov5_video_demo
  -- Installing:
/home/cat/lubancat_ai_manual_code/dev_env/rknpu2/rknn_yolov5_demo/install/rknn_yolov5_demo_Linu
x/lib/librockchip_mpp.so
  -- Installing:
```

```
/home/cat/lubancat_ai_manual_code/dev_env/rknpu2/rknn_yolov5_demo/install/rknn_yolov5_demo_Linux/lib/libmk_api.so
  /home/cat/lubancat_ai_manual_code/dev_env/rknpu2/rknn_yolov5_demo
```

```
./build-linux_RK3588.sh
```

コンパイルが完了すると、プログラムは install ディレクトリにインストールされます。このディレクトリの内容をボードにコピーします。ここでは rsync コマンド或いは scp コマンドを使用してボードに同期します：

```
rsync -avz ./install cat@192.168.11.241:~/
```

或いは

```
scp -r ./install cat@192.168.11.241:~/
```

4. テスト例を実行：

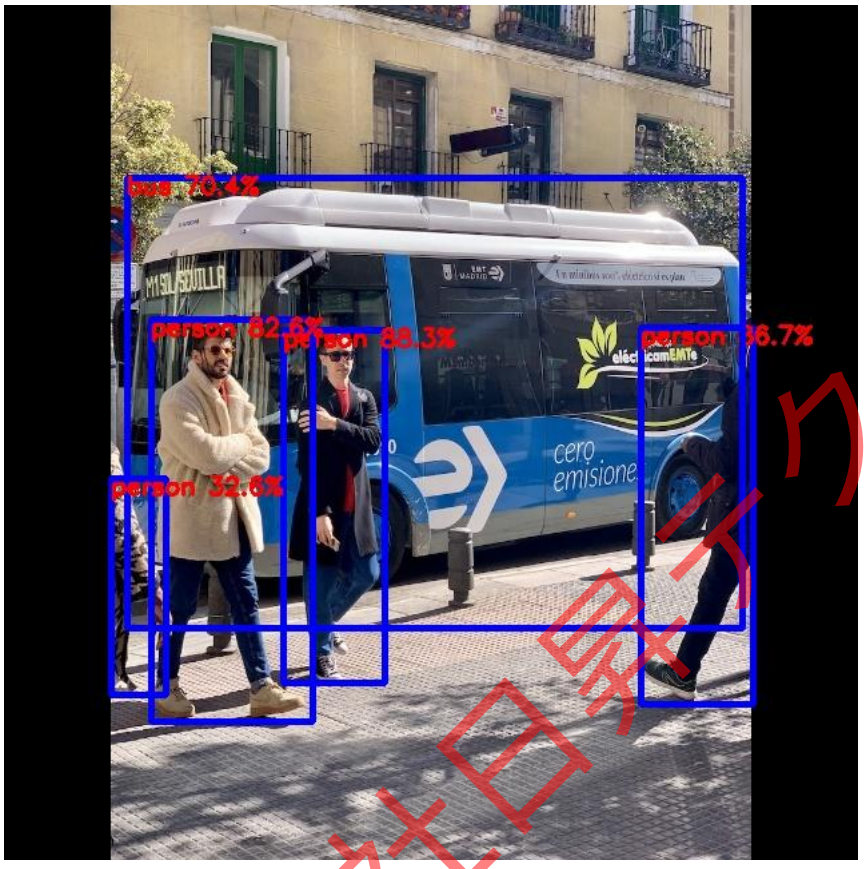
```
  # インストールディレクトリに切り替え、例のインストールライブラリを使用してプログラムを実行
  cat@lubancat:~/install/rknn_yolov5_demo_Linux$ export LD_LIBRARY_PATH=./lib
  # コマンドの使用法に従って、ボードに対応するモデルを使用
./rknn_yolov5_demo <rknn model> <jpg>
```

デフォルトのモデルと画像を使用して、コマンドを実行します（LubanCat-4 ボードをテストする場合は、build-linux_RK3588.sh でコンパイルされたプログラムを使用します）。

```
  cat@lubancat:~/install/rknn_yolov5_demo_Linux$  ./rknn_yolov5_demo ./model/RK3588/yolov5s-640-640.rknn ./model/bus.jpg
  post process config: box_conf_threshold = 0.25, nms_threshold = 0.45
  Read ./model/bus.jpg ...
  img width = 640, img height = 640
  Loading mode...
  sdk version: 2.0.0b0 (35a6907d79@2024-03-24T10:31:14) driver version: 0.9.2
  model input num: 1, output num: 3
    index=0, name=images, n_dims=4, dims=[1, 640, 640, 3], n_elems=1228800, size=1228800, w_stride = 640, size_with_stride=1228800, fmt=NHWC, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003922
    index=0, name=269, n_dims=4, dims=[1, 255, 80, 80], n_elems=1632000, size=1632000, w_stride = 0, size_with_stride=1638400, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=83, scale=0.093136
    index=1, name=271, n_dims=4, dims=[1, 255, 40, 40], n_elems=408000, size=408000, w_stride = 0, size_with_stride=491520, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=48, scale=0.089854
    index=2, name=273, n_dims=4, dims=[1, 255, 20, 20], n_elems=102000, size=102000, w_stride = 0, size_with_stride=163840, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=46, scale=0.078630
  model is NHWC input fmt
  model input height=640, width=640, channel=3
  once run use 31.821000 ms
  loadLabelName ./model/coco_80_labels_list.txt
  person @ (209 243 285 507) 0.883131
```

```
person @ (477 241 561 523) 0.866942
person @ (110 235 231 536) 0.825886
bus @ (92 129 553 466) 0.703667
person @ (80 354 121 516) 0.326333
loop count = 10 , average run  26.382100 ms
```

最後に `out.jpg` 画像を出力し、ボードのシステム上で表示するか、PC に転送して表示することができます。



ローカルビデオを認識するために、`rknn_yolov5_video_demo`プログラムを実行します：

```
# コマンド使用方法
./rknn_yolov5_video_demo <rknn_model> <video_path> <video_type 264/265>
# ビデオを取得して、h264 および h265 ストリームビデオに変換するために以下のコマンドを使用します：
# h264 ストリームビデオに変換
ffmpeg -i xxx.mp4 -vcodec h264 out.h264
# h265 ストリームビデオに変換
ffmpeg -i xxx.mp4 -vcodec hevc out.hevc
# デフォルトのモデルとビデオを使用してコマンドを実行します（テストは build-linux_RK3588.sh でコンパイルされたプログラムを使用し、デフォルトの test.h264 ビデオを使用しています）：
cat@lubancat:~/install/rknn_yolov5_demo_Linux$ ./rknn_yolov5_video_demo ./model/RK3588/yolov5
```

```
s-640-640.rknn ./model/test.h264 264
  cat@lubancat:~/install/rknn_yolov5_demo_Linux$ ./rknn_yolov5_video_demo ./model/RK3588/yolov5
s-640-640.rknn ./model/test.h264 264
  Loading mode...
  sdk version: 2.0.0b0 (35a6907d79@2024-03-24T10:31:14) driver version: 0.9.2
  model input num: 1, output num: 3
    index=0, name=images, n_dims=4, dims=[1, 640, 640, 3], n_elems=1228800, size=1228800,
fmt=NHWC, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003922
    index=0, name=269, n_dims=4, dims=[1, 255, 80, 80], n_elems=1632000, size=1632000,
fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=83, scale=0.093136
    index=1, name=271, n_dims=4, dims=[1, 255, 40, 40], n_elems=408000, size=408000, fmt=NCHW,
type=INT8, qnt_type=AFFINE, zp=48, scale=0.089854
    index=2, name=273, n_dims=4, dims=[1, 255, 20, 20], n_elems=102000, size=102000, fmt=NCHW,
type=INT8, qnt_type=AFFINE, zp=46, scale=0.078630
  model is NHWC input fmt
  model input height=640, width=640, channel=3
  app_ctx=0x7fca0855e8 decoder=0x14b80ea0
  read video size=4473796
  0x150d43e0 encoder test start w 1920 h 1080 type 7
  input image 1920x1080 stride 1920x1088 format=2560
  resize with RGA!
  rga_api version 1.9.1_[4]
  once run use 35.674000 ms
  ...省略...
  reset decoder
  waiting finish
  # 最終的に、現在のディレクトリに out.h264 ビデオが生成され、ボードのシステム上でビデオを再生して結果を確認できます。
```

rtsp ビデオストリーミングのテストも rknn_yolov5_video_demo を使われます。

```
  # コマンド使用方法
  ./rknn_yolov5_video_demo model/<TARGET_PLATFORM>/yolov5s-640-640.rknn <RTSP_URL>
265
  # ローカルで ffmpeg を使用して RTSP ストリーミングをテストするには、まず RTSP サーバーを起動します。このサービスはバックグラウンドで実行されます。
  # MediaServer を使用します（サンプルコードの dev_env/rknpu2/examples/3rdparty/zlmediakit ディレクトリから取得します）。
   sudo ./MediaServer &
  # ffmpeg ストリーミングテスト、出力先は rtsp://127.0.0.1/live/stream、テストビデオ test.mp4 を使用します。
  ffmpeg -re -i "./model/test.mp4" -vcodec h264 -f rtsp -rtsp_transport tcp
rtsp://127.0.0.1/live/stream
  # コマンドを実行して、前述のローカルストリーミングビデオ rtsp://127.0.0.1:8554/stream をテストします。これはテスト専用です。
```

```
cat@lubancat:~/install/rknn_yolov5_demo_Linux$ ./rknn_yolov5_video_demo ./model/RK3588/yolov5s-640-640.rknn rtsp://127.0.0.1/live/stream 264

# 任意のキーを押して終了します。最後に現在のディレクトリに out.h264 ビデオが生成され、ボードのシステム上でビデオを再生して結果を確認できます。
```

5.2.2 ゼロコピー API インターフェースの例

テスト例は RKNPU プロジェクトファイルの `examples/rknn_api_demo_Linux` ディレクトリから取得するか、チュートリアルのコード例から取得できます。この例では、MobileNet-v1 を使用して画像認識を行います。Linux プラットフォーム上でテストする場合、RGA を使用する場合と使用しない場合の 2 つの例があります。

1. サンプルソースコードの取得：

```
# チュートリアルのサンプルコードを取得
toolkit2 ツール（2.0.0b2）ダウンロード URL:
https://www.dragonwake.com/download/LubanCat4/7-srccode/tutorial/rknn_toolkit2-dev_env.zip
# 解凍後 lubancat_ai_manual_code/dev_env/rknpu2/rknn_yolov5_demo ディレクトリに移動
# または rknpu2 の GitHub から取得
# git clone https://github.com/rockchip-linux/rknpu2.git
# cd rknpu2/examples/rknn_api_demo/src/rknn_create_mem_demo.cpp
# サンプルソースコードのディレクトリに切り替え
cd rknpu2/rknn_api_demo
```

2. サンプルのコンパイル：

LubanCat-4 上で直接コンパイルする例を示します：

コンパイルが完了すると、プログラムは現在のディレクトリの `install/` にインストールされます。

```
cat@lubancat:~/lubancat_ai_manual_code/dev_env/rknpu2/rknn_api_demo_Linux$ ./build-linux_RK3588.sh
-- Configuring done
-- Generating done
-- Build files have been written to: /home/cat/lubancat_ai_manual_code/dev_env/rknpu2/rknn_api_demo_Linux/build/build_linux_aarch64
[ 25%] Linking CXX executable rknn_create_mem_with_rga_demo
[ 50%] Linking CXX executable rknn_create_mem_demo
[ 75%] Built target rknn_create_mem_demo
[100%] Built target rknn_create_mem_with_rga_demo
[ 50%] Built target rknn_create_mem_with_rga_demo
[100%] Built target rknn_create_mem_demo
```

```
  Install the project...
  -- Install configuration: ""
  -- Installing:
/home/cat/lubancat_ai_manual_code/dev_env/rknpu2/rknn_api_demo_Linux/install/rknn_api_demo_Linu
x/./rknn_create_mem_demo
  -- Set runtime path of
"/home/cat/lubancat_ai_manual_code/dev_env/rknpu2/rknn_api_demo_Linux/install/rknn_api_demo_Lin
ux/./rknn_create_mem_demo" to "lib"
  -- Installing:
/home/cat/lubancat_ai_manual_code/dev_env/rknpu2/rknn_api_demo_Linux/install/rknn_api_demo_Linu
x/./rknn_create_mem_with_rga_demo
  -- Set runtime path of
"/home/cat/lubancat_ai_manual_code/dev_env/rknpu2/rknn_api_demo_Linux/install/rknn_api_demo_Lin
ux/./rknn_create_mem_with_rga_demo" to "lib"
  -- Installing:
/home/cat/lubancat_ai_manual_code/dev_env/rknpu2/rknn_api_demo_Linux/install/rknn_api_demo_Linu
x/.//model
  -- Installing:
/home/cat/lubancat_ai_manual_code/dev_env/rknpu2/rknn_api_demo_Linux/install/rknn_api_demo_Linu
x/.//model/dog_224x224.jpg
  -- Installing:
/home/cat/lubancat_ai_manual_code/dev_env/rknpu2/rknn_api_demo_Linux/install/rknn_api_demo_Linu
x/.//model/RK3562
  -- Installing:
/home/cat/lubancat_ai_manual_code/dev_env/rknpu2/rknn_api_demo_Linux/install/rknn_api_demo_Linu
x/.//model/RK3562/mobilenet_v1.rknn
  -- Installing:
/home/cat/lubancat_ai_manual_code/dev_env/rknpu2/rknn_api_demo_Linux/install/rknn_api_demo_Linu
x/.//model/RK3566_RK3568
  -- Installing:
/home/cat/lubancat_ai_manual_code/dev_env/rknpu2/rknn_api_demo_Linux/install/rknn_api_demo_Linu
x/.//model/RK3566_RK3568/mobilenet_v1.rknn
  -- Installing:
/home/cat/lubancat_ai_manual_code/dev_env/rknpu2/rknn_api_demo_Linux/install/rknn_api_demo_Linu
x/.//model/RK3588
  -- Installing:
/home/cat/lubancat_ai_manual_code/dev_env/rknpu2/rknn_api_demo_Linux/install/rknn_api_demo_Linu
x/.//model/RK3588/mobilenet_v1.rknn
  -- Installing:
/home/cat/lubancat_ai_manual_code/dev_env/rknpu2/rknn_api_demo_Linux/install/rknn_api_demo_Linu
x/lib/librknnrt.so
  -- Installing:
/home/cat/lubancat_ai_manual_code/dev_env/rknpu2/rknn_api_demo_Linux/install/rknn_api_demo_Linu
x/lib/librga.so
  /home/cat/lubancat_ai_manual_code/dev_env/rknpu2/rknn_api_demo_Linux
```

3. サンプルのテスト：

```
  # 現在のディレクトリ `install/rknn_api_demo_Linux` で `rknn_create_mem_demo` を使用してテスト
  cat@lubancat:~/lubancat_ai_manual_code/dev_env/rknpu2/rknn_api_demo_Linux/install/rknn_api_demo_Linux$  ./rknn_create_mem_demo model/RK3588/mobilenet_v1.rknn model/dog_224x224.jpg
  rknn_api/rknnrt version: 2.0.0b0 (35a6907d79@2024-03-24T10:31:14), driver version: 0.9.2
  model input num: 1, output num: 1
  input tensors:
    index=0, name=input, n_dims=4, dims=[1, 224, 224, 3], n_elems=150528, size=150528, fmt=NHWC, type=INT8, qnt_type=AFFINE, zp=0, scale=0.007812
  output tensors:
    index=0, name=MobilenetV1/Predictions/Reshape_1, n_dims=2, dims=[1, 1001, 0, 0], n_elems=1001, size=1001, fmt=UNDEFINED, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003906
  custom string:
  Begin perf ...
     0: Elapse Time = 2.84ms, FPS = 352.11
  ---- Top5 ----
  0.984375 - 156
  0.007812 - 155
  0.003906 - 205
  0.000000 - 0
  0.000000 - 1

  # 現在のディレクトリ `install/rknn_api_demo_Linux` で `rknn_create_mem_with_rga_demo` を使用してテスト
  cat@lubancat:~/lubancat_ai_manual_code/dev_env/rknpu2/rknn_api_demo_Linux/install/rknn_api_demo_Linux$ ./rknn_create_mem_with_rga_demo model/RK3588/mobilenet_v1.rknn model/dog_224x224.jpg
  rknn_api/rknnrt version: 2.0.0b0 (35a6907d79@2024-03-24T10:31:14), driver version: 0.9.2
  model input num: 1, output num: 1
  input tensors:
    index=0, name=input, n_dims=4, dims=[1, 224, 224, 3], n_elems=150528, size=150528, fmt=NHWC, type=INT8, qnt_type=AFFINE, zp=0, scale=0.007812
  output tensors:
    index=0, name=MobilenetV1/Predictions/Reshape_1, n_dims=2, dims=[1, 1001, 0, 0], n_elems=1001, size=1001, fmt=UNDEFINED, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003906
  custom string:
  rga_api version 1.9.1_[4]
  Begin perf ...
     0: Elapse Time = 2.84ms, FPS = 352.73
  ---- Top5 ----
  0.984375 - 156
  0.007812 - 155
  0.003906 - 205
  0.000000 - 0
  0.000000 - 1
```

最終的に、出力される 5 つのカテゴリとその確率が表示されます。最大の確率を持つカテゴリは 156 です。

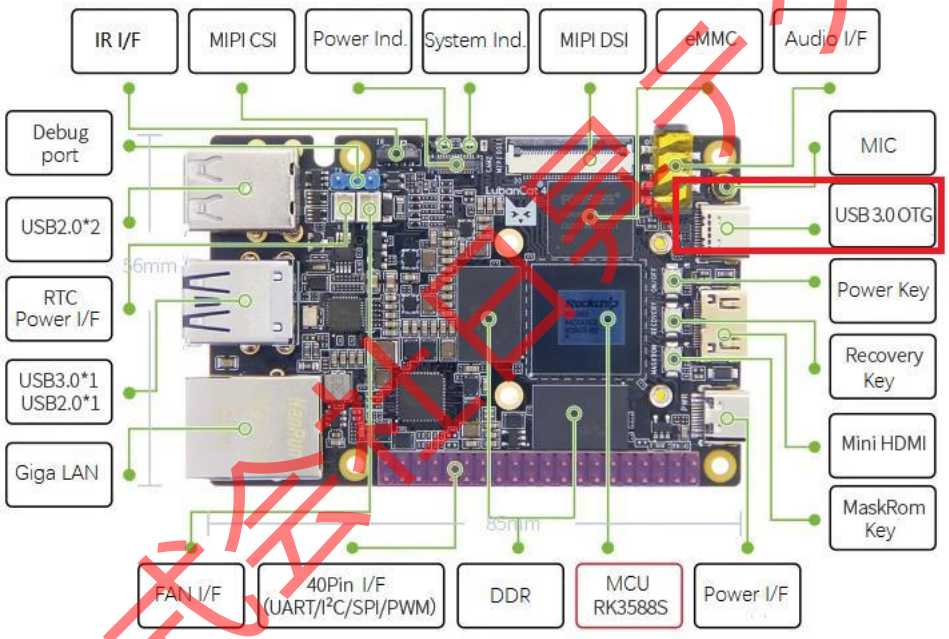
4. ゼロコピー API インターフェースの例のソースコード：
   RGA を使用しない場合の例（`rknn_create_mem_demo.cpp`）ソースをご参考にしてください。

5.3 Android プラットフォームでのテスト

Android プラットフォームで RKNN モデルを実行する場合も同様です。以下では RKNPU2 プロジェクトを参考に簡単なテストを行います。

重要：LubanCat の Android イメージを焼く手順については、Android チュートリアルを参照してください。チュートリアルのテストには LubanCat-4 および Android 12 イメージを使用します。

システムを起動後、adb を使用してボードに接続する必要があります。具体的な接続の USB ポートについては、クイックスタートガイドのハードウェア説明を参照してください。以下では LubanCat-4 を例に、USB-Type C ケーブルを使用してボードの USB3.0 OTG ポートに接続し、もう一方を PC に接続します。



PC（ubuntu システム）で adb を使用してデバイスに接続します：

```
root@YH-LONG:~# adb devices
* daemon not running; starting now at tcp:5037
* daemon started successfully
List of devices attached
7fbff6f219991594
device
```

5.3.1 rknn_yolov5_demo 例のテスト

1. 例を取得およびコンパイラの取得：

```
# rknpu2 github から取得
git clone https://github.com/airockchip/rknn-toolkit2.git
cd rknn-toolkit2/rknpu2/examples/rknn_yolov5_demo
# 本マニュアル用の toolkit2 ツール（2.0.0b2）ダウンロード URL から取得
https://www.dragonwake.com/download/LubanCat4/7-srccode/tutorial/rknn_toolkit2-dev_env.zip
# 解凍後 rknpu2/examples/rknn_yolov5_demo ディレクトリに移動
#cd dev_env/rknpu2/rknn_yolov5_demo
```

NDK ツールを https://developer.android.com/ndk/downloads?hl=ja から取得し、チュートリアルのテストには推奨される NDK バージョン（android-ndk-r17c、Linux x86 64 ビット）を使用します。取得した zip ファイルを関連ディレクトリに解凍します。

r17c

r17c Changelog

```
android {
    ndkVersion "17.2.4988734"
}
```

Platform	Package	Size (bytes)	SHA1 Checksum
Linux	android-ndk-r17c-linux-x86_64.zip	709387703	12cacc70c3fd2f40574015631c00f41fb8a39048
Mac	android-ndk-r17c-darwin-x86_64.zip	675091485	f97e3d7711497e3b4faf9e7b3fa0f0da90bb649c
Windows 32-bit	android-ndk-r17c-windows-x86.zip	608358310	5bb25bf13fa494ee6c3433474c7aa90009f9f6a9
Windows 64-bit	android-ndk-r17c-windows-x86_64.zip	650626501	3e3b8d1650f9d297d130be2b342db956003f5992

2. rknn_yolov5_demo 例のコンパイル：

```
# rknn-toolkit2/rknpu2/examples/rknn_yolov5_demo に切り替えます
cd rknn-toolkit2/rknpu2/examples/rknn_yolov5_demo
# または、チュートリアルのサンプルコードから
# cd dev_env/rknpu2/rknn_yolov5_demo
# 次に、コンパイルスクリプト build-android_RK3588.sh（テスト用の Android システム、LubanCat-4）の ANDROID_NDK_PATH パスを変更します
# 具体的には前述の android-ndk-r26b の保存パスに従ってください
ANDROID_NDK_PATH=/mnt/d/rk/android-ndk-r17c
# スクリプトを実行し、例をコンパイルします
./build-android_RK3588.sh
-- Check for working C compiler: /mnt/d/rk/android-ndk-r17c/toolchains/llvm/prebuilt/linux-x86_64/bin/clang
```

```
  -- Check for working C compiler: /mnt/d/rk/android-ndk-r17c/toolchains/llvm/prebuilt/linux-x86_64/bin/clang -- works
  -- Detecting C compiler ABI info
  -- Detecting C compiler ABI info - done
  -- Detecting C compile features
  -- Detecting C compile features - done
  -- Check for working CXX compiler: /mnt/d/rk/android-ndk-r17c/toolchains/llvm/prebuilt/linux-x86_64/bin/clang++
  -- Check for working CXX compiler: /mnt/d/rk/android-ndk-r17c/toolchains/llvm/prebuilt/linux-x86_64/bin/clang++ -- works
  -- Detecting CXX compiler ABI info
  -- Detecting CXX compiler ABI info - done
  -- Detecting CXX compile features
  -- Detecting CXX compile features - done
  -- Found OpenCV: /mnt/d/rk/rknpu2-1.5.2/examples/3rdparty/opencv/OpenCV-android-sdk (found version "3.4.5")
  -- Configuring done
  -- Generating done
  -- Build files have been written to: /mnt/d/rk/rknpu2-1.5.2/examples/rknn_yolov5_demo/build/build_android_v8a
  Scanning dependencies of target rknn_yolov5_video_demo
  Scanning dependencies of target rknn_yolov5_demo
  [ 11%] Building CXX object CMakeFiles/rknn_yolov5_video_demo.dir/src/main_video.cc.o
  [ 22%] Building CXX object CMakeFiles/rknn_yolov5_video_demo.dir/src/postprocess.cc.o
  [ 33%] Building CXX object CMakeFiles/rknn_yolov5_video_demo.dir/utils/mpp_decoder.cpp.o
  # 省略...
  [ 88%] Built target rknn_yolov5_video_demo
  [100%] Linking CXX executable rknn_yolov5_demo
  [100%] Built target rknn_yolov5_demo
  [ 66%] Built target rknn_yolov5_video_demo
  [100%] Built target rknn_yolov5_demo
  Install the project...
  # 省略...
  # コンパイルされたプログラムは、現在のディレクトリ内の install フォルダにインストールされます。
```

3. プログラムをボードに転送：

```
  # rknpu2 github から取得
  git clone https://github.com/airockchip/rknn-toolkit2.git
  cd rknn-toolkit2/rknpu2/examples/rknn_yolov5_demo
  # 本マニュアル用の toolkit2 ツール（2.0.0b2）ダウンロード URL から取得
https://www.dragonwake.com/download/LubanCat4/7-srccode/tutorial/rknn_toolkit2-dev_env.zip
  # 解凍後 rknpu2/examples/rknn_yolov5_demo ディレクトリに移動
  #cd dev_env/rknpu2/rknn_yolov5_demo
  # 前のコンパイル済みの例を /data ディレクトリに転送します
  adb root
  adb push install/rknn_yolov5_demo_Android /data
  # ボードのシステムターミナルに入り、例のディレクトリに切り替えます
  adb shell
  rk3588s_lubancat_4_hdmi:/# cd /data/rknn_yolov5_demo_Android
```

```
  rk3588s_lubancat_4_hdmi:/data/rknn_yolov5_demo_Android
  # 次に、ライブラリの検索パスを設定し、転送したライブラリを使用します
  rk3588s_lubancat_4_hdmi:/data/rknn_yolov5_demo_Android# export LD_LIBRARY_PATH=./lib
  # `rknn_yolov5_demo` に実行権限を追加し、コマンドを実行します。コマンドの使用法
は：./rknn_yolov5_demo <rknn model> <jpg>

  rk3588s_lubancat_4_hdmi:/data/rknn_yolov5_demo_Android# chmod +x rknn_yolov5_demo
  rk3588s_lubancat_4_hdmi:/data/rknn_yolov5_demo_Android# ./rknn_yolov5_demo ./model/RK3588/yol
ov5s-640-640.rknn ./model/bus.jpg
  model input num: 1, output num: 3
  index=0, name=images, n_dims=4, dims=[1, 640, 640, 3], n_elems=1228800, size=1228800,
w_stride=640, size_with_stride=1228800, fmt=NHWC, type=INT8, qnt_type=AFFINE,
  zp=-128, scale=0.003922
  index=0, name=output, n_dims=4, dims=[1, 255, 80, 80], n_elems=1632000, size=1632000,
w_stride=0, size_with_stride=1638400, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp
  =-128, scale=0.003860
  index=1, name=283, n_dims=4, dims=[1, 255, 40, 40], n_elems=408000, size=408000, w_stride=0,
size_with_stride=491520, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128,
  scale=0.003922
  index=2, name=285, n_dims=4, dims=[1, 255, 20, 20], n_elems=102000, size=102000, w_stride=0,
size_with_stride=163840, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128,
  scale=0.003915
  model is NHWC input fmt
  model input height=640, width=640, channel=3
  once run use 31.285000 ms
  loadLabelName ./model/coco_80_labels_list.txt
  person @ (209 244 286 506) 0.884139
  person @ (478 238 559 526) 0.867678
  person @ (110 238 230 534) 0.824685
  bus @ (94 129 553 468) 0.705055
  person @ (79 354 122 516) 0.339254
  loop count = 10, average run 29.712800 ms
  # 結果は out.jpg に保存されます。exit コマンドでターミナルを終了し、adb コマンドを使用して PC
に転送して結果を確認します
  adb pull /data/rknn_yolov5_demo_Android/out.jpg ./
  /data/rknn_yolov5_demo_Android/out.jpg: 1 file pulled, 0 skipped. 25.3 MB/s (190629 bytes in
0.007s)
```

実行時にライブラリ検索パスを指定します。システムにデフォルトでインストールされているライブラリを使用する場合、ボードのランタイムライブラリを更新する必要があるかもしれません（Android システムの/vendor/lib64/librknnrt.so を置き換える）、および rknn_server。RKNN Toolkit2 バージョンとランタイムライブラリバージョンを一致させます。

5.4 参考リンク

-https://github.com/rockchip-linux/rknpu2
-https://github.com/airockchip/rknn-toolkit2

第 6 章 人工知能

本章では、人工知能に関連する基本的な知識を簡単に紹介します。人工知能は非常に多くの研究分野に関わるため、すべての知識を詳細に説明することはできません。さらに詳しい内容については、推奨書籍や参考リンクをご覧ください。

6.1 人工知能の概要

人工知能（Artificial Intelligence、略称 AI）は、その名が示す通り、コンピュータに人間と同じように考え、学び、問題を解決する能力を持たせる技術です。人工知能は、人間の知能を模倣し、機械が自動でタスクを実行できるようにするものです。例えば、画像中の物体の認識や、自動運転車の運転などがあります。人工知能は、機械学習、深層学習、自然言語処理、コンピュータビジョン、エキスパートシステム、推論エンジンなど、さまざまな技術と手法を採用しています。

人工知能の具体的な実現方法としては、機械学習とディープラーニングが現在最も効果的な手法です。機械学習は、大量のデータからパターンや法則を学び取ることができる技術です。モデルとアルゴリズムの訓練を通じて、機械学習はコンピュータが自主的に予測、分類、クラスタリング、意思決定を行うことを可能にします。

ディープラーニングは、機械学習の一分野であり、人間の脳神経ネットワークの構造と機能を模倣します。深層神経ネットワークを構築することで、大規模なデータの高度な抽象化と特徴抽出を行い、複雑な問題の精密なモデリングと処理を実現します。

人工知能は、科学、工学、社会の発展を推進する前沿技術として、私たちに多くの機会と可能性を提供しています。生活を改善し、イノベーションを促進し、世界的な課題を解決するための手段として、人工知能は研究と発展が続けられています。

6.2 機械学習

機械学習（Machine Learning、略称 ML）は、人工知能の重要な分野であり、データや経験を通じてコンピュータの性能を自動的に向上させる方法を研究しています。サンプルデータから学習して経験（モデル）を獲得し、その後予測を行います。

機械学習の実現には、大量のデータサンプルが訓練データとして必要であり、それに対応するラベルや目標値が参考として使用されます。これらのデータを用いてモデルの訓練とパラメータの調整を行うことで、機械学習アルゴリズムは自身を最適化し、より正確な結果と予測を提供することができます。

機械学習アルゴリズムは、問題の性質や要求に応じて選択および適用されます。一般的な機械学習アルゴリズムには、線形回帰、ロジスティック回帰、決定木、サポートベクターマシン、ナイーブベイズ、ニューラルネットワーク、深層学習などがあります。

機械学習は、医療、金融、推薦システムなど、さまざまな分野で広く応用されています。例えば、医療分

野では、機械学習を利用して病気の予測と診断、薬物開発、個別化治療を支援することができます。金融分野では、リスク評価、信用スコアリング、投資意思決定に使用されます。推薦システムでは、個別化された製品推薦やコンテンツフィルタリングを提供します。

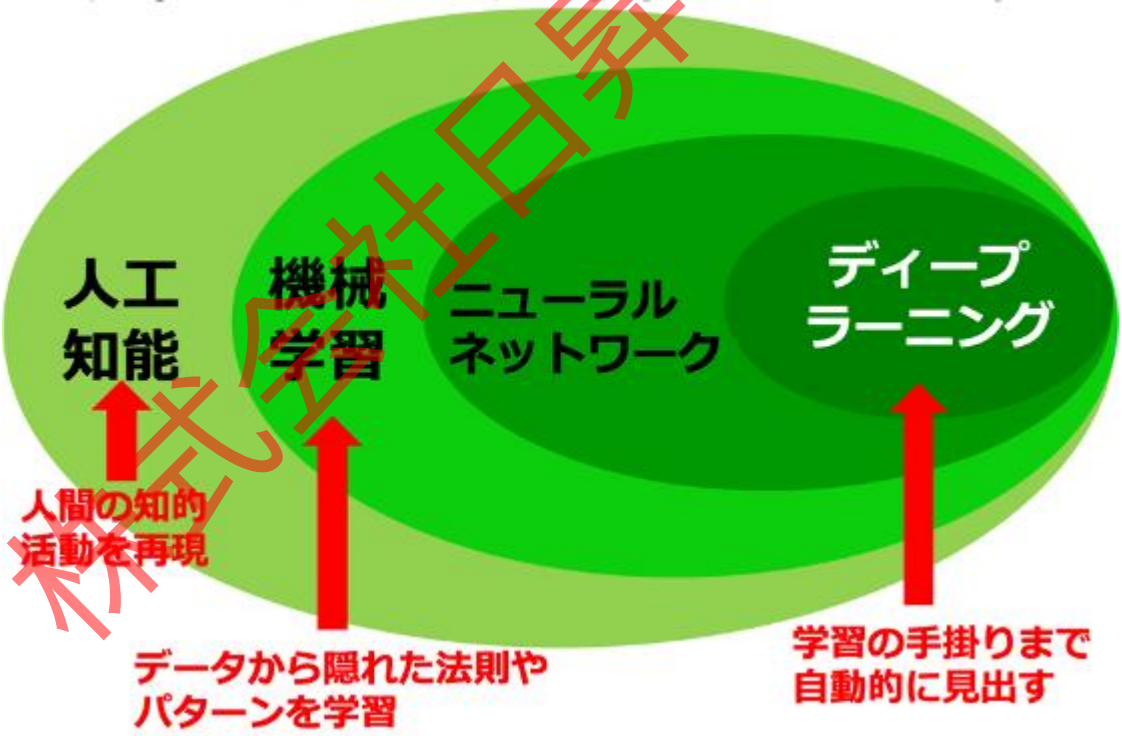
6.3 ディープラーニング

ディープラーニング（Deep Learning、略称 DL）は、機械学習アルゴリズムの中で最も注目されている分野の一つであり、近年大きな進展を遂げ、従来の多くの機械学習アルゴリズムに取って代わりました。深層学習は、人工神経ネットワークに基づく機械学習アルゴリズムであり、人間の脳の働きを模倣して、複雑なデータの高度な抽象化と解析を行います。

ディープラーニングの中心となる考え方は、深層神経ネットワークの構築です。これは、複数のレベルのニューロンで構成されており、各レベルが入力データを処理および変換し、その結果を次のレベルに伝達します。これらのネットワーク層間の接続重みとバイアスは、大量の訓練データを用いて自動的に調整され、ネットワークがデータから特徴を学習および抽出し、分類または予測の精度を向上させることができます。

ディープラーニングの成功は、主に 2 つの要因に起因します。第一に、大量のデータからパターンを自動的に学習し発見する能力です。第二に、ハードウェアの計算能力の向上と大規模データセットの利用可能性により、深層学習モデルの訓練がより実現可能かつ効果的になったことです。

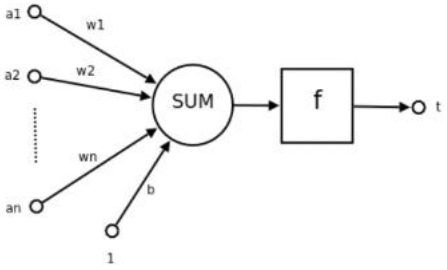
人工知能、機械学習、ディープラーニングの関係を簡単に示すと以下のようになります。



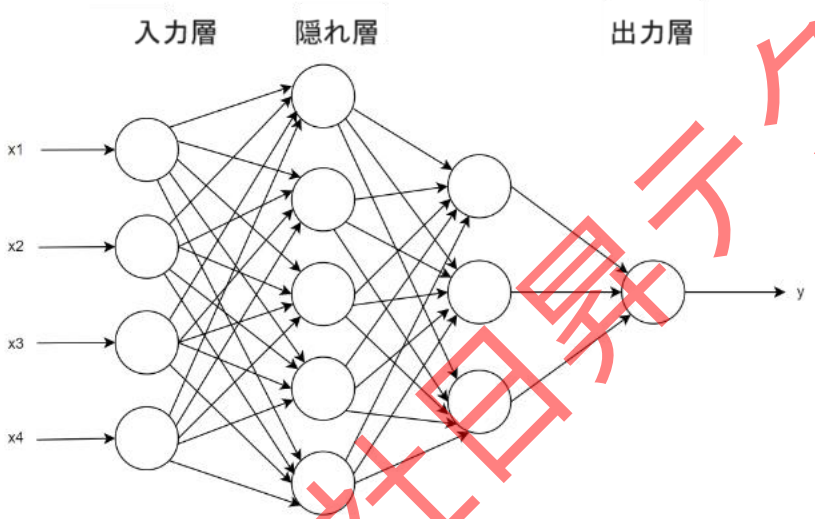
6.3.1 人工神経ネットワーク

人工神経ネットワーク（Artificial Neural Network、略称 ANN）は、神経ネットワークまたは神経モデルとも呼ばれ、生物の神経ネットワークの構造と機能を模倣した数学モデルです。

以下は、単一のニューロンの構造図です：



図の a1~an はニューロンの入力、w1~wn は各入力に対応する重み、b はバイアス、f は非線形関数（活性化関数）、t はニューロンの出力を示しています。



単一の人工ニューロンの機能は単純ですが、複雑な機能を実現するためには、複数のニューロンが協力して働く必要があります。このように、特定の接続方法や情報伝達方法で協力するニューロンのネットワークが、神経ネットワークです。

神経ネットワークの階層構造は通常、入力層、隠れ層、出力層で構成されます。一般的に、隠れ層の数が 2 層以上の神経ネットワークを深層神経ネットワークと呼び、深層学習はこのような深層構造を持つ機械学習手法です。

神経ネットワークの接続重みとバイアスは、訓練中に自動的に調整され、データから学習して正確な予測や分類を行うことができるようになります。訓練プロセスでは、逆伝播アルゴリズムを使用して損失関数の勾配を計算し、各層の接続重みとバイアスを調整して、予測結果と実際の結果の誤差を最小化します。

6.3.2 ディープラーニングフレームワーク

ディープラーニングフレームワークは、深層学習モデルの構築、訓練、デプロイを支援するソフトウェアライブラリです。これらのフレームワークは、高度な抽象化、事前定義された層とオプティマイザ、自動微分などの機能を提供し、ディープラーニングコードの作成を簡単かつ効率的にします。以下は、一般的な深層学習フレームワークの例です：

- TensorFlow：Google がオープンソース化した、最も人気のあるディープラーニングフレームワークの一つ。データフローグラフに基づいた計算方式を採用し、複数のプラットフォームと言語をサポートします。
- PyTorch：Facebook がオープンソース化したディープラーニングフレームワーク。Torch ライブラリに基づき、Python で記述され、動的グラフと自動微分をサポートします。
- PaddlePaddle：Baidu がオープンソース化した産業用ディープラーニングプラットフォームで、深層学習のコアトレーニングと推論フレームワーク、基礎モデルライブラリ、エンドツーエンドの開発キット、豊富なツールコンポーネントを統合しています。
- Caffe：コンピュータビジョンタスクに適したディープラーニングフレームワークで、高効率の前方計算と訓練速度を特徴とします。Caffe は、画像分類、物体検出、画像セグメンテーションなどのタスクに使用される事前訓練済みモデルを提供します。
- MXNet：Amazon、ワシントン大学、カーネギーメロン大学などが開発・保守しているディープラーニングフレームワーク。効率と生産性を最大化するために、シンボリックプログラミングと命令型プログラミングの混合使用をサポートし、複数の GPU および複数のマシンに効率的にスケールアップできます。

上記のフレームワークの他にも、Theano、Keras、Chainer など、多くのディープラーニングフレームワークがあります。

6.4 学習書籍の推薦

- Dive into Deep Learning (D2L)(https://d21.ai)
- Deep Learning Book(https://www.deeplearningbook.org/)
- Neural Networks and Deep Learning(http://neuralnetworksanddeeplearning.com/index.html)

6.5 参考リンク

- [機械学習 (Wikipedia)]
- [ディープラーニング(Wikipedia)]
- [ニューラルネットワーク (Wikipedia)]
- [PaddlePaddle チュートリアル]
- [TensorFlow 学習リソース]
- [PyTorch チュートリアル]

第 7 章 手書き数字認識 - PaddlePaddle

7.1 PaddlePaddle についての紹介

PaddlePaddle は、Baidu の多年にわたるディープラーニング技術の研究とビジネス応用を基盤に、ディープラーニングのコアトレーニングと推論フレームワーク、基礎モデルライブラリ、エンドツーエンドの開発キット、豊富なツールコンポーネントを一体化した、機能豊富、オープンソースの産業級ディープラーニングプラットフォームです。

PaddlePaddle はすでに 477 万人の開発者を集め、PaddlePaddle を基に 56 万のモデルを作成し、18 万社の企業や団体にサービスを提供しています。PaddlePaddle は開発者が迅速に AI のアイデアを実現し、AI 応用を革新することを支援し、基盤プラットフォームとしてますます多くの産業のインテリジェント化アップグレードを支えています。2022 年 12 月時点で、通信通院の最新報告によると、PaddlePaddle は中国のディープラーニング市場で最も広く応用されているディープラーニングフレームワークと支援プラットフォームとなり、535 万人の開発者を集め、20 万社の企業や団体にサービスを提供し、PaddlePaddle を基に 67 万のモデルを構築しています。

本章では、PaddlePaddle を基に、簡単な手書き数字認識タスクを完成し、Lubancat ボードにデプロイします。この章を通して PaddlePaddle 及びディープラーニングモデルについて簡単に理解します。

注意: テスト環境は Lubancat ボードで Debian11 を使用し、PC は ubuntu20.04 です。PaddlePaddle は CPU バージョンであり、rknn-Toolkit2 バージョンは 2.0.0 です。

7.2 PaddlePaddle のインストール

pip のバージョンが 20.2.2 以上に必要となり、インストール前に、
pip のバージョンを確認しましょう。

```
source ~/project-Toolkit2/.toolkit2_env/bin/activate
(.toolkit2_env) (base) csun@CSUN-PC-0013:~$ pip --version
pip 24.0 from /home/csun/project-Toolkit2/.toolkit2_env/lib/python3.8/site-packages/pip (python 3.8)
```

バージョンが低い場合、アップグレードします。

```
pip install --upgrade pip
# PC で pip3 ツールを使用して PaddlePaddle CPU バージョンをインストール
pip3 install paddlepaddle
GPU バージョンをインストール
pip install paddlepaddle-gpu
*現時点の 2.6.1 バージョンをインストールすることになります。
#NVIVIA 社の pypi index をインストール
pip install nvidia-pyindex
#
#cuDNN をインストール
```

```
pip install nvidia-cudnn-cu12
インストールした場合、下記メッセージが出られます。
(.toolkit2_env) csun@CSUN-PC-0013:~/project-Toolkit2/rknn-toolkit2-2.0.0/rknn-toolkit2/examples/onnx/yolov5$ pip install nvidia-cudnn-cu12
Looking in indexes: https://pypi.org/simple, https://pypi.ngc.nvidia.com
Requirement already satisfied: nvidia-cudnn-cu12 in /home/csun/project-Toolkit2/.toolkit2_env/lib/python3.8/site-packages (8.9.2.26)
Requirement already satisfied: nvidia-cublas-cu12 in /home/csun/project-Toolkit2/.toolkit2_env/lib/python3.8/site-packages (from nvidia-cudnn-cu12) (12.1.3.1)
#ライブラリのリンクを作成
cd /home/csun/project-Toolkit2/.toolkit2_env/lib/python3.8/site-packages/nvidia/cudnn/lib
ln -s libcudnn.so.8 libcudnn.so
cd /home/csun/project-Toolkit2/.toolkit2_env/lib/python3.8/site-packages/nvidia/cublas/lib
ln -s libcublas.so.12 libcublas.so
#仮想環境設定
nano ~/project-Toolkit2/.toolkit2_env/bin/activate
#下記パスを追加
export LD_LIBRARY_PATH=/home/csun/project-Toolkit2/.toolkit2_env/lib/python3.8/site-packages/nvidia/cublas/lib:$LD_LIBRARY_PATH
export LD_LIBRARY_PATH=/home/csun/project-Toolkit2/.toolkit2_env/lib/python3.8/site-packages/nvidia/cudnn/lib:$LD_LIBRARY_PATH
#仮想環境再起動
deactivate
source ~/project-Toolkit2/.toolkit2_env/bin/activate
# インストールを確認するため、Python インタープリタに入ります。
python
import paddle
paddle.utils.run_check()
或いは
python -c "import paddle; paddle.utils.run_check()"
# インストールを確認し、バージョン情報を表示します。
print(paddle.__version__)

#CPU バージョンの場合：
Running verify PaddlePaddle program ...
I0608 06:58:04.996460 626764 program_interpreter.cc:212] New Executor is Running.
I0608 06:58:05.024521 626764 interpreter_util.cc:624] Standalone Executor is Used.
PaddlePaddle works well on 1 CPU.
PaddlePaddle is installed successfully! Let's start deep learning with PaddlePaddle now.
#或いは GPU バージョンの場合：
Running verify PaddlePaddle program ...
I0608 07:02:20.359321 628464 program_interpreter.cc:212] New Executor is Running.
W0608 07:02:20.359508 628464 gpu_resources.cc:119] Please NOTE: device: 0, GPU Compute Capability: 6.1, Driver API Version: 12.2, Runtime API Version: 11.8
```

```
W0608 07:02:20.359935 628464 gpu_resources.cc:164] device: 0, cuDNN Version: 8.9.
I0608 07:02:20.965610 628464 interpreter_util.cc:624] Standalone Executor is Used.
PaddlePaddle works well on 1 GPU.
PaddlePaddle is installed successfully! Let's start deep learning with PaddlePaddle now.

# 最後に「PaddlePaddle is installed successfully! Let's start deep learning with PaddlePaddle now.」
# と表示されれば、インストールは成功です。
```

詳細なインストール方法は［インストールガイド］
（Run the following command to install (paddlepaddle.org.cn)）を参照してください。

7.3 手書き数字認識タスク

手書き数字認識タスクは、機械学習分野で最も広く使用されている例の一つであり、比較的シンプルなモデルであるため、初心者に非常に適しています。

タスクの全体的な流れは以下の図（図は Paddle のチュートリアルから）を参考にしてください：
PaddlePaddle を基にディープラーニング入門チュートリアル
https://github.com/PaddlePaddle/book?tab=readme-ov-file

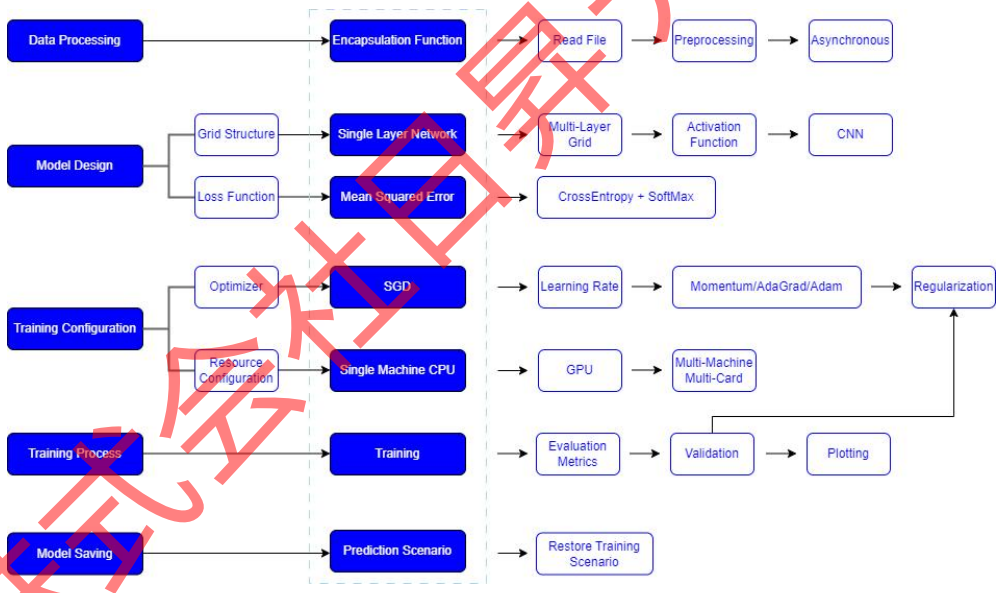


図 1 ：The vertical minimalist solution

7.3.1 畳み込みニューラルネットワーク

畳み込みニューラルネットワーク（Convolutional Neural Network，CNN）は、現在コンピュータビジョンで最も広く使用されているモデル構造です。CNN アーキテクチャに基づくモデルは、コンピュータビジョンの分野で支配的な地位を占めており、今日のほとんどの画像認識、物体検出、またはセマンティックセグメンテーション関連の問題はこの方法に基づいています。

7.3.2 手書き数字認識モデルの訓練

手書き数字認識は、典型的な画像分類問題の一つです。画像分類は、CNN の重要な応用の一つであり、画像のセマンティック情報に基づいて異なるカテゴリの画像を区別することです。これは、物体検出、画像セグメンテーション、物体追跡、行動分析、顔認識などの他の高次視覚タスクの基礎です。

画像分類は多くの分野で広く応用されています。例えば、セキュリティ分野の顔認識やインテリジェントビデオ分析、交通分野の交通シーン認識、インターネット分野のコンテンツベースの画像検索やアルバム自動分類、医療分野の画像認識などです。

一般的な画像分類の CNN には、LeNet、AlexNet、VGG、GoogLeNet、ResNet などがあります。

以下は、手書き数字認識モデルを訓練し、そのモデルをデバイス上でデプロイするまでの完全なプロセスです：

- データセットの準備
- ニューラルネットワークモデルの構築
- モデルの訓練
- モデルの評価
- モデルのエクスポートとモデル変換、シミュレーション推論
- デバイス上でのデプロイと推論

これらのステップに従って、ボード上で手書き数字認識モデルを訓練します。具体的な操作は `handwritten.ipynb` を参照してください。

7.3.3 訓練データセットとテストデータセットの準備

MNIST データセットは機械学習でよく使用されるデータセットで、次の URL からダウンロードできます：http://yann.lecun.com/exdb/mnist/。このデータセットには 4 つのファイルが含まれています：

- `t10k-images-idx3-ubyte.gz`：テストセットの画像、合計 10,000 枚。
- `t10k-labels-idx1-ubyte.gz`：テストセット画像のラベル（0〜9 の数字）、合計 10,000 件。
- `train-images-idx3-ubyte.gz`：トレーニングセットの画像、合計 60,000 枚。
- `train-labels-idx1-ubyte.gz`：トレーニングセット画像のラベル（0〜9 の数字）、合計 60,000 件。

PaddlePaddle ディープラーニングフレームワークを使用すると、内蔵の API インターフェースを利用して MNIST データセットを直接ダウンロードして読み取ることができます（他のディープラーニングフレームワークでも同様です）。デフォルトでは、ユーザーディレクトリのキャッシュデータセットパス（`~/.cache/paddle/dataset/mnist/`）にダウンロードされます。

通常、サンプルセットはトレーニングセット、検証セット、およびテストセットに分けられます。

- トレーニングセット：モデルのパラメータをトレーニングするために使用されます。これはトレーニングプロセスで主に行われる作業です。
- 検証セット：モデルのハイパーパラメータを選択するために使用されます。たとえば、ネットワーク構造の調整や正則化項の重みの選択などです。
- テストセット：モデルの実際の適用後の効果をシミュレートするために使用されます。テストセットはモデルの最適化やパラメータトレーニングには一切使用されないため、モデルにとっては完全に未知のサンプルです。検証データを使用してネットワーク構造やモデルのハイパーパラメータを最適化しない場合、検証データとテストデータの効果は類似しており、モデルの効果をより正確に反映します。

ここでは、トレーニングデータとテストデータをインポートし、トレーニングセットとテストセットのみに分割します。

手書き数字認識は、異なる人が書いた数字をグレースケール画像で表現します。各画像のサイズは 28x28 です。我々は MNIST データセットを使用し、公式サイトからダウンロードすることができます。または Paddle を使用して MNIST を直接ロードすることもできます。

ここでは、MNIST 訓練データセット（mode='train'）とテストデータセット（mode='test'）をロードします。訓練データセットはモデルの訓練に使用され、テストデータセットはモデルの評価に使用されます。

```
# データセットをダウンロードして初期化します
train_dataset = paddle.vision.datasets.MNIST(mode='train', transform=transform)
test_dataset = paddle.vision.datasets.MNIST(mode='test', transform=transform)
```

7.3.4 モデルネットワークの構築

```
# モデル構造の定義
# 多層畳み込みニューラルネットワークの実装
class MNIST(paddle.nn.Layer):
    def __init__(self):
        super(MNIST, self).__init__()

        # 畳み込み層の定義、出力特徴チャネル out_channels は 20、畳み込みカーネルのサイズ kernel_size は 5、畳み込みストライド stride=1、
padding=2
        self.conv1 = Conv2D(in_channels=1, out_channels=20, kernel_size=5, stride=1, padding=2)
        # プーリング層の定義、プーリングカーネルのサイズ kernel_size は 2、プーリングストライドは 2
        self.max_pool1 = MaxPool2D(kernel_size=2, stride=2)
        # 畳み込み層の定義、出力特徴チャネル out_channels は 20、畳み込みカーネルのサイズ kernel_size は 5、畳み込みストライド stride=1、
padding=2
        self.conv2 = Conv2D(in_channels=20, out_channels=20, kernel_size=5, stride=1, padding=2)
        # プーリング層の定義、プーリングカーネルのサイズ kernel_size は 2、プーリングストライドは 2
        self.max_pool2 = MaxPool2D(kernel_size=2, stride=2)
        # 全結合層の定義、出力次元は 10
        self.fc = Linear(in_features=980, out_features=10)

    # ネットワークの前方計算プロセスの定義、畳み込み後にプーリング層を使用し、最後に全結合層を使用して最終出力を計算
    # 畳み込み層の活性化関数に Relu を使用し、全結合層の活性化関数に softmax を使用
    def forward(self, inputs, label=None):
        x = self.conv1(inputs)
        x = F.relu(x)
        x = self.max_pool1(x)
        x = self.conv2(x)
        x = F.relu(x)
        x = self.max_pool2(x)
        x = paddle.reshape(x, [x.shape[0], -1])
        x = self.fc(x)
        if label is not None:
            acc = paddle.metric.accuracy(input=x, label=label)
            return x, acc
        else:
            return x
```

モデル定義の前の部分は、paddle.nn が提供する Conv2D と MaxPool2D 関数を使用し、ネットワークの第一層は畳み込み層です。

7.3.5 モデルの訓練

まず、paddle.io.DataLoader API を使用してデータを読み込みます。このインターフェースは非同期データ読み込みを実現し、データは Python スレッドによって事前に読み込まれ、非同期的にキューに送信されます。読み込まれたデータはキャッシュに格納され、モデルの訓練を待たずに次のデータ読み込みを開始でき、データ読み込みとモデル訓練が並行して行われます。

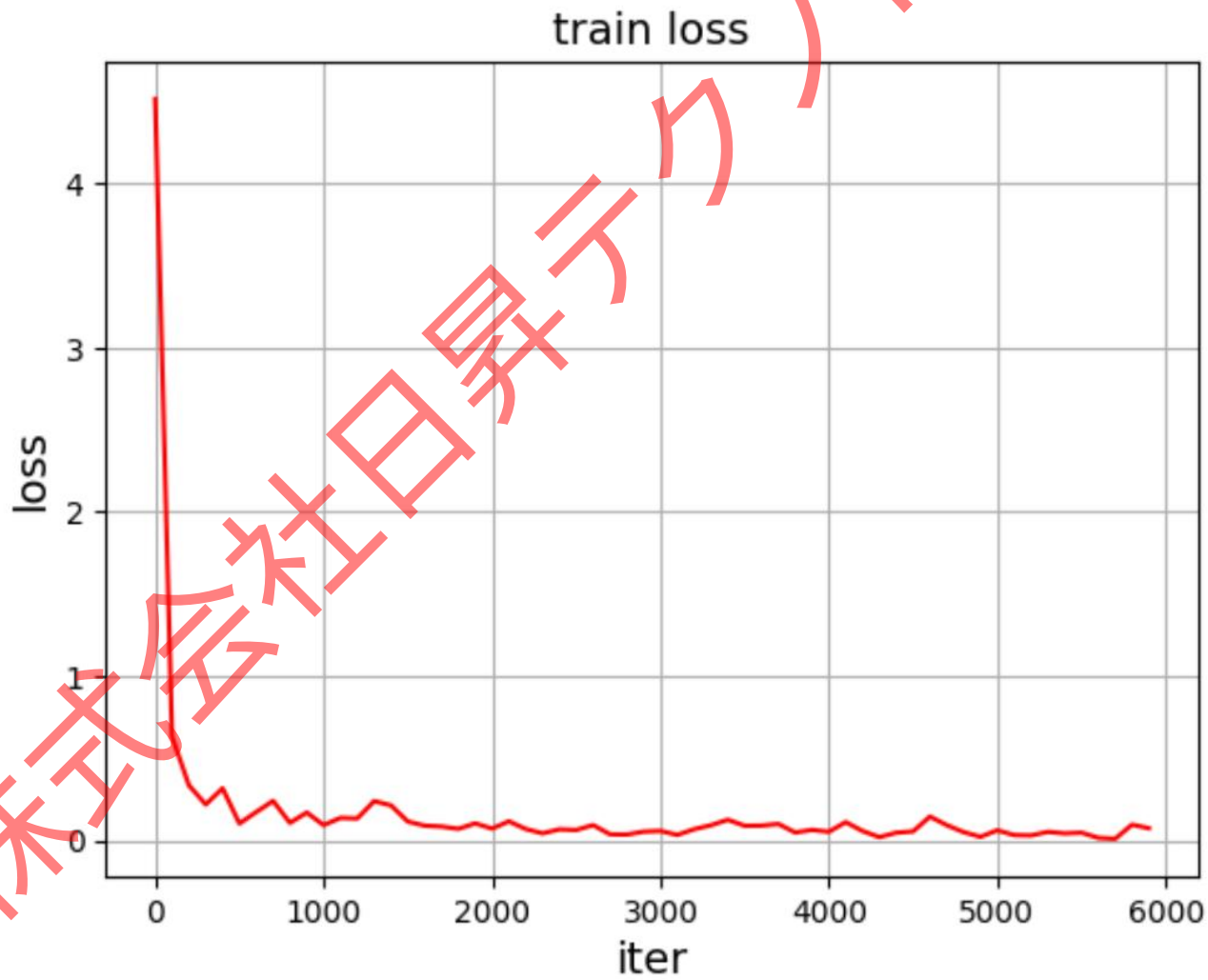
```
def train(model):
    model.train()

    # データのロード関数を呼び出し、ミニバッチのサンプル数を 100 に設定し、shuffle=True で順序をシャッフル
    train_loader = paddle.io.DataLoader(train_dataset, batch_size=100, shuffle=True)

    # 4 種類の最適化アルゴリズムの設定方法、効果を逐次試すことができます
    opt = paddle.optimizer.SGD(learning_rate=0.01, parameters=model.parameters())
    # opt = paddle.optimizer.Momentum(learning_rate=0.01, momentum=0.9, parameters=model.parameters())
    # opt = paddle.optimizer.Adagrad(learning_rate=0.01, parameters=model.parameters())
    # opt = paddle.optimizer.Adam(learning_rate=0.01, parameters=model.parameters())
```

```
    EPOCH_NUM = 10
    iter=0
    iters=[]
    losses=[]

    for epoch_id in range(EPOCH_NUM):
        for batch_id, data in enumerate(train_loader()):
            # データの準備
            images, labels = data
            images = paddle.to_tensor(images)
            labels = paddle.to_tensor(labels)

            # 前向き計算のプロセスと同時に分類精度を計算
            predicts, acc = model(images, labels)
            #predicts = model(images)

            # 損失を計算し、バッチサンプルの平均損失を取得
            loss = F.cross_entropy(predicts, labels)
            avg_loss = paddle.mean(loss)

            # 100 バッチごとに現在の損失状況を表示
            if batch_id % 100 == 0:
                print("epoch: {}, batch: {}, loss is: {}, acc is {}".format(epoch_id, batch_id, avg_loss.item(), acc.item()))
                iters.append(iter)
                losses.append(avg_loss.item())
                iter = iter + 100

            # 逆伝播、パラメータ更新のプロセス
            avg_loss.backward()
            # 損失を最小化し、パラメータを更新
            opt.step()
            # 勾配をクリア
            opt.clear_grad()

    # モデルパラメータを保存
    paddle.save(model.state_dict(), './paddle/mnist.pdparams')

    return iters, losses

# モデルを作成
model = MNIST()

#paddle.summary(model, (1, 1, 28, 28))

# トレーニングプロセスを開始
# GPU を有効にする
#use_gpu = True
#paddle.device.set_device('gpu:0') if use_gpu else paddle.device.set_device('cpu')
iters, losses = train(model)

# ネットワーク構造を表示
paddle.summary(model,(1,1,28,28))


import matplotlib.pyplot as plt

# トレーニング中の損失の変化曲線を描画
plt.figure()
plt.title("train loss", fontsize=14)
plt.xlabel("iter", fontsize=14)
plt.ylabel("loss", fontsize=14)
plt.plot(iters, losses, color='red')
plt.grid()
plt.show()
```

上記のプログラムで paddle.summary(model, (1, 1, 28, 28))のコメントを解除すると、ネットワーク構造が表示されます。

```
---------------------------------------------------------------------------
 Layer (type)       Input Shape          Output Shape         Param #
===========================================================================
   Conv2D-1       [[1, 1, 28, 28]]     [1, 20, 28, 28]          520
  MaxPool2D-1     [[1, 20, 28, 28]]    [1, 20, 14, 14]           0
   Conv2D-2       [[1, 20, 14, 14]]    [1, 20, 14, 14]       10,020
  MaxPool2D-2     [[1, 20, 14, 14]]     [1, 20, 7, 7]            0
   Linear-1          [[1, 980]]            [1, 10]             9,810
===========================================================================
Total params: 20,350
Trainable params: 20,350
Non-trainable params: 0
---------------------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.19
Params size (MB): 0.08
Estimated Total Size (MB): 0.27
---------------------------------------------------------------------------
```

ネットワーク構造を表示せずに直接訓練を行うと、Loss の変化曲線が表示されます。



7.3.6 モデルのテスト

ここではテストセットを使用してモデルを簡単に評価します。

```
def evaluation(model):
    params_file_path = './paddle/mnist.pdparams'
    # モデルパラメータをロード
    print('loading mnist model from ', params_file_path)
    param_dict = paddle.load(params_file_path)
    model.load_dict(param_dict)

    model.eval()
    eval_loader = paddle.io.DataLoader(test_dataset)


    acc_set = []
    avg_loss_set = []
    for batch_id, data in enumerate(eval_loader()):
        images, labels = data
        images = paddle.to_tensor(images)
        labels = paddle.to_tensor(labels)
        predicts, acc = model(images, labels)
        loss = F.cross_entropy(input=predicts, label=labels)
        avg_loss = paddle.mean(loss)
        acc_set.append(float(acc.item()))
        avg_loss_set.append(float(avg_loss.item()))

    # 複数バッチの平均損失と精度を計算
    acc_val_mean = np.array(acc_set).mean()
    avg_loss_val_mean = np.array(avg_loss_set).mean()


    print('loss={}, acc={}'.format(avg_loss_val_mean, acc_val_mean))

# モデル評価
model = MNIST()
evaluation(model)
```

テスト結果の出力：

```
loading mnist model from  ./paddle/mnist.pdparams
loss=0.05354610707261737, acc=0.9829
```

7.3.7 RKNN モデルのエクスポートとシミュレーション推論テスト

Paddle モデルは直接 RKNN モデルに変換できないため、まず ONNX モデルに変換し、その後 rknn-toolkit2 を使用して RKNN モデルに変換します。

1. トレーニングしたモデルを ONNX モデルとして保存します。

```
# 保存するパスを指定、現在の onnx ディレクトリ内
save_path = 'onnx/handwritten'
# モデルに入力の形状とデータ型を指定
x_spec = paddle.static.InputSpec([1, 1, 28, 28], 'float32', 'x')
# ONNX モデルを生成
paddle.onnx.export(model, save_path, input_spec=[x_spec], opset_version=11)
```

変換が成功すると、現在の onnx ディレクトリに handwritten.onnx モデルファイルが生成され、実

行結果が表示されます。

```
2024-06-20 22:51:51 [INFO]Static PaddlePaddle model saved in onnx/paddle_model_static_onnx_temp
_dir.
[Paddle2ONNX] Start to parse PaddlePaddle model...
[Paddle2ONNX] Model file path: onnx/paddle_model_static_onnx_temp_dir/model.pdmodel
[Paddle2ONNX] Parameters file path: onnx/paddle_model_static_onnx_temp_dir/model.pdiparams
[Paddle2ONNX] Start to parsing Paddle model...
2024-06-20 22:51:51 [INFO]ONNX model saved in onnx/handwritten.onnx.
[Paddle2ONNX] Use opset_version = 11 for ONNX export.
[Paddle2ONNX] PaddlePaddle model is exported as ONNX format now.
```

2. rknn-toolkit2 を使用してモデルを RKNN モデルに変換し、簡単な推論テストを行います。

PC 上で rknn-toolkit2 をインストールした環境で変換を行います。rknn-toolkit2 のインストールは前の章を参照してください。この例では、rknn-toolkit2 の変換モデルと推論の手順は以下の通りです：

●RKNN オブジェクトを作成し、RKNN 環境を初期化します。

●モデルの前処理パラメータを設定します。シミュレータでモデルを実行する場合は、config インターフェースを使用してモデルの前処理パラメータを設定します。

●モデルをインポートし、load_onnx インターフェースを使用してモデルをインポートします。

●RKNN モデルを構築し、build インターフェースを使用して RKNN モデルを構築します。

●export_rknn インターフェースを使用して RKNN モデルをエクスポートし、同時に init_runtime を使用してランタイム環境を初期化し、シミュレータでの推論をシミュレートします。

●ランタイム環境を初期化した後、inference インターフェースを使用してシミュレーション推論を行います。

●最後に release インターフェースを使用して RKNN オブジェクトを解放します。

テストファイルの主要ソースコード：

サンプルソース 1：onnx_to_rknn.py

```
  # 詳しくはマニュアルを参考：https://www.dragonwake.com/download/LubanCat4/1-manual/CSAIEG358803_Python-
application-guide.pdf

  import numpy as np
  import cv2
  from rknn.api import RKNN

  ONNX_MODEL = './handwritten.onnx'
  RKNN_MODEL ='./model/RK3588/handwritten.rknn'
  IMG_PATH = './0.jpg'
  IMG_SIZE = 28
  QUANTIZE_ON = False

  if __name__ == '__main__':

      # RKNN オブジェクトを作成
      rknn = RKNN()

      # ここでは lubancat-4 で指定プラットフォームは rk3588 です。lubancat0/1/2 の場合、指定プラットフォームは
rk3566、rk3568 です。
      print('--> モデルを設定')
      rknn.config(mean_values=[[127.5]], std_values=[[127.5]], target_platform='rk3588')
      print('完了')
```

```
    # ONNX モデルをロード
    print('--> モデルをロード中')
    ret = rknn.load_onnx(model=ONNX_MODEL)
    if ret != 0:
        print('モデルのロードに失敗しました!')
        exit(ret)
    print('完了')

    # モデルをビルド
    print('--> モデルをビルド中')
    #ret = rknn.build(do_quantization=QUANTIZE_ON, dataset=DATASET)
    ret = rknn.build(do_quantization=QUANTIZE_ON)
    if ret != 0:
        print('モデルのビルドに失敗しました!')
        exit(ret)
    print('完了')

    # RKNN モデルをエクスポート
    print('--> rknn モデルをエクスポート')
    ret = rknn.export_rknn(RKNN_MODEL)
    if ret != 0:
        print('rknn モデルのエクスポートに失敗しました!')
        exit(ret)
    print('完了')

    # 実行環境を初期化
    print('--> 実行環境を初期化')
    ret = rknn.init_runtime()
    if ret != 0:
        print('実行環境の初期化に失敗しました!')
        exit(ret)
    print('完了')

    # 入力を設定
    img = cv2.imread(IMG_PATH)
    img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
    img = cv2.resize(img, (IMG_SIZE, IMG_SIZE))
    img = np.expand_dims(img, 0)
    img = np.expand_dims(img, 0)

    # 推論
    print('--> モデルを実行中')
    outputs = rknn.inference(inputs=[img], data_format='nchw')
    print('完了')

    # 出力
    print("出力: ", outputs)
    print("今回の予測数字は:", np.argmax(outputs))

    rknn.release()
```


onnx_to_rknn.py を実行すると、以下が表示されます：

```
(.toolkit2_env) (base) csun@CSUN-PC-0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/base/handwritten$ python3 onnx_to_rknn.py
I rknn-toolkit2 version: 2.0.0b0+9bab5682
--> モデルを設定
完了
```

```
--> モデルをロード中
I It is recommended onnx opset 19, but your onnx model opset is 11!
I Loading : 100%|███████████████████████████████████████████████| 7/7 [00:00<00:00, 51690.37it/s]
完了
--> モデルをビルド中
I rknn building ...
I rknn buiding done.
完了
--> rknn モデルをエクスポート
完了
--> 実行環境を初期化
I Target is None, use simulator!
完了
--> モデルを実行中
I GraphPreparing : 100%|████████████████████████████████████| 8/8 [00:00<00:00, 1071.65it/s]
I SessionPreparing : 100%|██████████████████████████████████| 8/8 [00:00<00:00, 464.51it/s]
完了
出力: [array([[ 8.6328125 , -0.21862793,  3.5253906 ,  0.38549805, -1.8681641 ,
        -0.39990234,  1.3476562 , -0.90722656, -1.328125  ,  0.8574219 ]],
      dtype=float32)]
今回の予測数字は: 0
```

変換された handwritten.rknn ファイルは、変数 RKNN_MODEL で指定されたディレクトリに保存されます。

7.3.8 ボード側でのデプロイ推論

前のセクションでは、rknn-toolkit2 ツールを使用して handwritten.rknn ファイルを変換しましたが、ボード側でデプロイする方法は？RKNN API インターフェース（C/C++）または RKNN Toolkit Lite2 インターフェース（Python）を使用できます。これらのインターフェースの使用方法は、前の章を参照してください。ここでは、LubanCat ボード上でモデルをデプロイする手順を示します。

1. Toolkit Lite2 を使用したデプロイテスト

Toolkit Lite2 の使用手順は前の「RKNN Toolkit Lite2 紹介」を参照してください。以下はボード端での手書き数字認識モデルのデプロイ推論プログラムです：

サンプルソース 2: rknn_inference.py

```
# 詳しくはマニュアルを参考：https://www.dragonwake.com/download/LubanCat4/1-manual/CSAIEG358803_Python-application-guide.pdf
import numpy as np
import cv2
import sys
import platform
from rknnlite.api import RKNNLite

IMG_PATH = './9.jpg'
RK3588_RKNN_MODEL = '../model/RK3588/handwritten.rknn'
RK3566_RK3568_RKNN_MODEL = '../model/RK356X/handwritten.rknn'
DEVICE_COMPATIBLE_NODE = '/proc/device-tree/compatible'

def get_host():
```

```
    system = platform.system()
    machine = platform.machine()
    os_machine = system + '-' + machine
    if os_machine == 'Linux-aarch64':
        try:
            with open(DEVICE_COMPATIBLE_NODE) as f:
                device_compatible_str = f.read()
                if 'rk3588' in device_compatible_str:
                    host = 'RK3588'
                elif 'rk3562' in device_compatible_str:
                    host = 'RK3562'
                else:
                    host = 'RK3566_RK3568'
        except IOError:
            print('デバイスノード {} の読み取りに失敗しました。'.format(DEVICE_COMPATIBLE_NODE))
            exit(-1)
    else:
        host = os_machine
    return host

if __name__ == '__main__':

    # 対応するプラットフォームのモデルを取得
    host_name = get_host()
    if host_name == 'RK3566_RK3568':
      rknn_model = RK3566_RK3568_RKNN_MODEL
    elif host_name == 'RK3588':
      rknn_model = RK3588_RKNN_MODEL
    else:
      print("このデモは現在のプラットフォームでは実行できません: {}".format(host_name))
      exit(-1)

    # RKNNLite オブジェクトを作成
    # rknn_lite = RKNNLite(verbose=True)
    rknn_lite = RKNNLite()

    # load_rknn インターフェースを呼び出して RKNN モデルをインポートし、対応するプラットフォーム (rk356x/rk3588)
のモデルを使用
    print('--> RKNN モデルをロード')
    ret = rknn_lite.load_rknn(rknn_model)
    if ret != 0:
        print('RKNN モデルのロードに失敗しました')
        exit(ret)
    print('完了')

    # init_runtime インターフェースを呼び出して実行環境を初期化
    print('--> 実行環境を初期化')
    ret = rknn_lite.init_runtime()
    if ret != 0:
        print('実行環境の初期化に失敗しました!')
        exit(ret)
    print('完了')

    # 画像を読み込み、画像データを前処理
    img = cv2.imread(IMG_PATH)
    img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
    img = cv2.resize(img, (28, 28))

    # 推論
    print('--> モデルを実行中')
    outputs = rknn_lite.inference(inputs=[img])
```

```
    # 結果を出力
    print("出力: ", outputs)
    print("ボードのこの予測数字は:", np.argmax(outputs))

    # release インターフェースを呼び出して RKNNLite オブジェクトを解放
    rknn_lite.release()
```

ボード上での実行結果（lubancat-4）：

```
  cat@lubancat:~/lubancat_ai_manual_code/base/handwritten/python$ python rknn_inference.py
  --> RKNN モデルをロード
  完了
  --> 実行環境を初期化
  I RKNN: [23:48:04.811] RKNN Runtime Information, librknnrt version: 2.0.0b0 (35a6907d79@2024-03-24T10:31:14)
  I RKNN: [23:48:04.811] RKNN Driver Information, version: 0.9.2
  I RKNN: [23:48:04.811] RKNN Model Information, version: 6, toolkit version: 2.0.0b0+9bab5682(compiler
version: 2.0.0b0 (35a6907d79@2024-03-24T02:34:11)), target: RKNPU v2, target platform: rk3588, framework name:
ONNX, framework layout: NCHW, model inference type: static_shape
  完了
  --> モデルを実行中
  出力:  [array([[-1.0146484 ,  0.20922852, -0.66259766, -0.9946289 ,  1.0742188 ,
          -1.2353516 , -4.0273438 ,  0.79345703,  2.3144531 ,  3.296875  ]],
        dtype=float32)]
  ボードのこの予測数字は: 9
```

2. RKNN API を使用したデプロイテスト

ボード側で手書き数字認識モデルをデプロイするには、RKNN API インターフェースを使用します。具体的な呼び出し手順は「RKNN API」を参照してください。ここでは簡単なテストを行い、通用 API インターフェースを使用します。主要なコード部分は以下の通りです（詳細は付属の例を参照してください。付属の例は更新されていない場合があります）：

サンプルソース 3：main.cc

```
  //...省略...//
  /*-------------------------------------------
                  Main Functions
  -------------------------------------------*/
  int main(int argc, char** argv)
  {
    char*          model_name = NULL;
    rknn_context   ctx;
    int            img_width          = 0;
    int            img_height         = 0;
    int            ret;
    struct timeval start_time, stop_time;

    if (argc != 3) {
      printf("Usage: %s <rknn model> <jpg> ¥n", argv[0]);
      return -1;
    }

    model_name       = (char*)argv[1];
    char* image_name = argv[2];

    // Load RKNN Model
    printf("Loading mode %s ...¥n", model_name);
    int            model_data_size = 0;
```

```
unsigned char* model_data      = load_model(model_name, &model_data_size);
ret                            = rknn_init(&ctx, model_data, model_data_size, 0, NULL);
if (ret < 0) {
  printf("rknn_init error ret=%d¥n", ret);
  return -1;
}

// Get sdk version and driver version
rknn_sdk_version version;
ret = rknn_query(ctx, RKNN_QUERY_SDK_VERSION, &version, sizeof(rknn_sdk_version));
if (ret < 0) {
  printf("rknn_init error ret=%d¥n", ret);
  return -1;
}
printf("sdk version: %s driver version: %s¥n", version.api_version, version.drv_version);

// Get Model Input Output Info
rknn_input_output_num io_num;
ret = rknn_query(ctx, RKNN_QUERY_IN_OUT_NUM, &io_num, sizeof(io_num));
if (ret < 0) {
  printf("rknn_init error ret=%d¥n", ret);
  return -1;
}
printf("model input num: %d, output num: %d¥n", io_num.n_input, io_num.n_output);

rknn_tensor_attr input_attrs[io_num.n_input];
memset(input_attrs, 0, sizeof(input_attrs));
for (int i = 0; i < io_num.n_input; i++) {
  input_attrs[i].index = i;
  ret                  = rknn_query(ctx, RKNN_QUERY_INPUT_ATTR, &(input_attrs[i]), sizeof(rknn_tensor_attr));
  if (ret < 0) {
    printf("rknn_init error ret=%d¥n", ret);
    return -1;
  }
  dump_tensor_attr(&(input_attrs[i]));
}

rknn_tensor_attr output_attrs[io_num.n_output];
memset(output_attrs, 0, sizeof(output_attrs));
for (int i = 0; i < io_num.n_output; i++) {
  output_attrs[i].index = i;
  ret = rknn_query(ctx, RKNN_QUERY_OUTPUT_ATTR, &(output_attrs[i]), sizeof(rknn_tensor_attr));
  dump_tensor_attr(&(output_attrs[i]));
}

int channel = 0;
int width   = 0;
int height  = 0;
if (input_attrs[0].fmt == RKNN_TENSOR_NCHW) {
  printf("model is NCHW input fmt¥n");
  channel = input_attrs[0].dims[1];
  height  = input_attrs[0].dims[2];
  width   = input_attrs[0].dims[3];
} else {
  printf("model is NHWC input fmt¥n");
  height  = input_attrs[0].dims[1];
  width   = input_attrs[0].dims[2];
  channel = input_attrs[0].dims[3];
}

printf("model input height=%d, width=%d, channel=%d¥n", height, width, channel);

// Set Input Data
rknn_input inputs[1];
memset(inputs, 0, sizeof(inputs));
inputs[0].index        = 0;
inputs[0].type         = RKNN_TENSOR_UINT8;
inputs[0].size         = width * height * channel;
inputs[0].fmt          = RKNN_TENSOR_NHWC;
inputs[0].pass_through = 0;

// Load image
```

```
    printf("Read %s ...¥n", image_name);
    cv::Mat orig_img = cv::imread(image_name, 1);
    if (!orig_img.data) {
      printf("cv::imread %s fail!¥n", image_name);
      return -1;
    }

    cv::Mat img;
    cv::cvtColor(orig_img, img, cv::COLOR_RGB2GRAY);
    img_width  = img.cols;
    img_height = img.rows;
    printf("img width = %d, img height = %d¥n", img_width, img_height);

    // resize with Opencv
    if (img_width != width || img_height != height) {
      printf("resize with Opencv! ¥n");
      cv::Mat resize_image;
      cv::resize(img, resize_image, cv::Size(width, height), cv::INTER_LINEAR);
      inputs[0].buf = (void*)resize_image.data;
    } else {
      inputs[0].buf = (void*)img.data;
    }

    rknn_inputs_set(ctx, io_num.n_input, inputs);
    rknn_output outputs[1];
    outputs[0].want_float = 1;

    // Run
    gettimeofday(&start_time, NULL);
    ret = rknn_run(ctx, NULL);
    ret = rknn_outputs_get(ctx, io_num.n_output, outputs, NULL);
    gettimeofday(&stop_time, NULL);
    printf("once run use %f ms¥n", (__get_us(stop_time) - __get_us(start_time)) / 1000);

    // Get Output and Post Process
    float*   buffer = (float*)outputs[0].buf;
    printf("outputs: ");
    for (int i = 0; i < 10; i++) {
      printf("%.6f ", buffer[i]);
    }
    printf("¥n");

    findMaxAndPosition(buffer, 10);

    ret = rknn_destroy(ctx);

    if (model_data) {
      free(model_data);
    }

    return 0;
}
```

付属のファイルをボードにコピーし、ソースコードをコンパイルします（付属の例は PC でのクロスコンパイルに対応していません。必要に応じて CMakeFiles.txt を変更し、ライブラリファイルを追加してください）。コンパイルには、ボードにデフォルトでインストールされている RKNN ライブラリを使用します。ライブラリを更新するかヘッダーファイルを追加する必要がある場合は、前の「RKNN API」を参照してください。

プログラムをコンパイルするためのコマンドを実行します：

```
# 手書き数字の例 handwritten/c++ディレクトリに移動し、コンパイルします。ここでは lubancat-4 ボードをテストしています。
cat@lubancat:~/lubancat_ai_manual_code/base/handwritten/c++$ mkdir build && cd build
cat@lubancat:~/lubancat_ai_manual_code/base/handwritten/c++/build$ cmake ../
-- The C compiler identification is GNU 10.2.1
-- The CXX compiler identification is GNU 10.2.1
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /usr/bin/cc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/c++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Found OpenCV: /usr (found version "4.5.1")
-- Configuring done
-- Generating done
-- Build files have been written to: /home/cat/lubancat_ai_manual_code/base/handwritten/c++/build

# 次に make コマンドを実行してプログラムをコンパイルし、handwritten ファイルを生成します。
cat@lubancat:~/lubancat_ai_manual_code/base/handwritten/c++/build$ make
Scanning dependencies of target handwritten
[ 50%] Building CXX object CMakeFiles/handwritten.dir/main.cc.o
[100%] Linking CXX executable handwritten
[100%] Built target handwritten
```

ボード側の推論を実行：

```
# ボードに応じて異なるモデルを選択します。
cat@lubancat:~/lubancat_ai_manual_code/base/handwritten/c++/build$ ./handwritten ../../model/RK3588/handwritten.rknn ../0.jpg
Loading mode ../../model/RK3588/handwritten.rknn ...
sdk version: 2.0.0b0 (35a6907d79@2024-03-24T10:31:14) driver version: 0.9.2
model input num: 1, output num: 1
  index=0, name=x, n_dims=4, dims=[1, 28, 28, 1], n_elems=784, size=1568, w_stride = 32, size_with_stride=1792, fmt=NHWC, type=FP16, qnt_type=AFFINE, zp=0, scale=1.000000
  index=0, name=linear_2.tmp_1, n_dims=2, dims=[1, 10], n_elems=10, size=20, w_stride = 0, size_with_stride=20, fmt=UNDEFINED, type=FP16, qnt_type=AFFINE, zp=0, scale=1.000000
model is NHWC input fmt
model input height=28, width=28, channel=1
Read ../0.jpg ...
img width = 219, img height = 217
once run use 0.182000 ms
```

```
outputs: 8.632812 -0.218262 3.525391 0.385742 -1.869141 -0.399170 1.348633 -0.907715 -1.328125
0.857910
今回予測された数字は： 0
```

7.4 参考リンク

1. [Handwritten Digit Recognition]
2. [PaddlePaddle インストールガイド]
3. [PaddlePaddle クイックスタート]
4. [PaddlePaddle API リファレンス]
5. [PaddlePaddle github - PaddlePaddle

第 8 章 住宅価格予測

本章では、PaddlePaddle ディープラーニングフレームワークを使用して住宅価格予測を例に挙げて簡単に紹介します：

●神経ネットワークを使用して典型的な機械学習である回帰問題を解決する方法
●PaddlePaddle を使用して全結合神経ネットワークを作成する方法
●神経ネットワークのトレーニング方法（例えば、確率的勾配降下法など）

8.1 線形神経ネットワーク

機械学習において、線形回帰（Linear Regression）は最も基礎的で広く使用されているモデルです。これは、自変量と因変量の関係をモデル化する回帰分析の一種であり、自変量が 1 つの場合は単回帰、多変量が複数の場合は重回帰と呼ばれます。

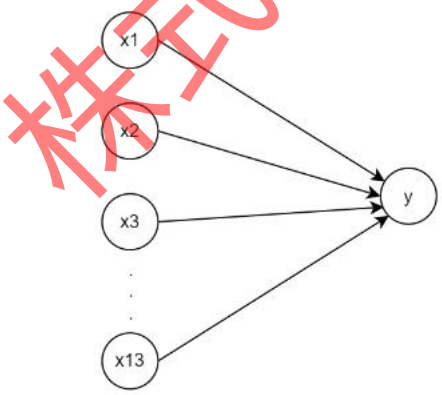
住宅価格予測タスクを単純な線形回帰問題として捉えることができます。予測出力のタイプが連続的な実数値か、または離散的なラベルかによって回帰タスクと分類タスクに分けられます。住宅価格は連続値であるため、住宅価格予測は明らかに回帰タスクです。

ここでは、ボストン住宅価格データセット（dataset）を例に取ります。このデータセットには、住宅価格に影響を与える可能性のある 13 の要因と住宅の平均価格が含まれています。以下は、住宅価格と各影響要因との関係が線形であると仮定します：

$$y = \sum_{j=1}^{M} x_j w_j + b$$

ここで、w は重みベクトル、b はバイアス、x は入力（住宅価格に影響を与える要因）、y は出力（計算された住宅価格）です。モデルの解法は、データを用いて w と b の値をフィッティングすることです。

線形回帰モデルを、単一の人工神経ニューロンから成る神経ネットワークと見なすことができます。この線形モデルを神経ネットワークの方法で説明すると、以下のように単純に表されます：



神経ネットワークの標準構造では、各ニューロンは加重和と非線形変換で構成されます。上図の神経ネッ

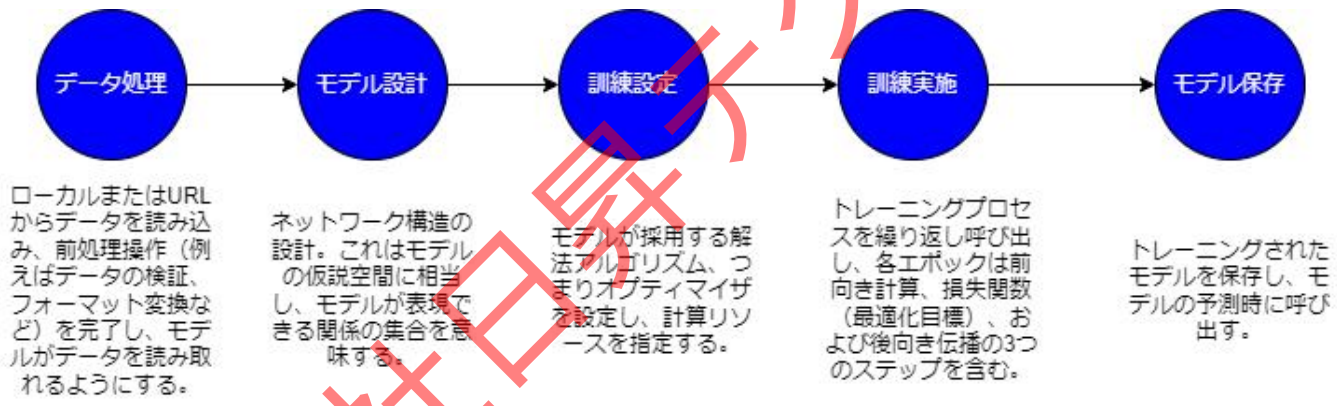
トワークユニットには、重み、バイアス、活性化関数が含まれています。線形回帰の場合、各入力は各出力（この場合は 1 つの出力）に接続され、この変換を全結合層（fully-connected layer）と呼びます。

ボストン住宅価格データセットに含まれるサンプル（x1～x13 の 13 の要因）とターゲット（価格）を使用して w と b の値を計算します。これにより、新しいサンプル x を与えると、このモデルを使用して価格 y を予測することができます。

では、最適な w と b をどのように確認し、最良の予測価格を得るのでしょうか？
それは、いくつかのアルゴリズムを使用して実現します。トレーニング中に、予測価格とデータセットの実際の価格との差異を測定する必要があります。これが損失関数（loss function）です。回帰問題では、平方誤差関数が最も一般的に使用されます。そして、誤差を最小化する解を得るために、最適化アルゴリズムを使用します。住宅価格予測では確率的勾配降下法（SGD）を使用します。

8.2 住宅価格予測モデルのトレーニング

PaddlePaddle ディープラーニングフレームワークを使用して、ボストン住宅価格データセットに基づいて、シンプルな住宅価格予測モデルを構築・トレーニングします。以下は、ディープラーニング神経ネットワークを構築するための基本ステップです：



PaddlePaddle のインストール：（7.2 の PaddlePaddle のインストールを参照してもよい）

```
# conda で環境を作成し、Python のバージョンを指定する
conda activate
conda create -n paddle python=3.8
# 環境に入る
conda activate paddle
# 最新バージョンの paddlepaddle をインストールする（CPU 版、チュートリアル時の最新バージョンは 2.6.1）
conda install paddlepaddle==2.6.1 --channel
https://mirrors.tuna.tsinghua.edu.cn/anaconda/cloud/Paddle/
```

8.2.1 データ処理

データ処理部分は、データのインポート、データ変換、データセットの分割、データの正規化処理などの操作に分かれます。

●データのインポート：データはボストン住宅価格データセットを使用します。このデータセットには、アメリカ合衆国マサチューセッツ州ボストンの住宅価格に関する情報が含まれており、小規模なデータセットで 506 のケースがあります。[こちら]から取得できます。データセットには 14 の属性があり、最初の 13 は価格に影響を与える要因で、最後の 1 つは住宅の平均価格です。
●データ変換：インポートしたデータは 1 次元です。データの形状を変換して、506 行 14 列の 2 次元の行列にします。各行は 1 つのデータサンプル（14 の値）で、各データサンプルには 13 の要因と 1 つの平均価格が含まれます。
●データセットの分割：506 組のデータの 80%をトレーニングセットとして使用し、20%をテストセットとして使用します。つまり、404 のサンプルデータがトレーニングサンプルで、102 のデータがテストサンプルです。
●データの正規化処理：元のデータを[0, 1]の範囲に変換し、モデルのトレーニングを効率化します。

```
def load_data():
    # 現在のディレクトリの./data/housing.data からデータをインポート
    datafile = '/home/csun/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/base/housing/data/housing.data'
    data = np.fromfile(datafile, sep=' ', dtype=np.float32)

    # 各データは 14 項目を含み、最初の 13 項目は影響要因、14 項目目は対応する住宅価格の中央値
    feature_names = [ 'CRIM', 'ZN', 'INDUS', 'CHAS', 'NOX', 'RM', 'AGE', ¥
                      'DIS', 'RAD', 'TAX', 'PTRATIO', 'B', 'LSTAT', 'MEDV' ]
    feature_num = len(feature_names)

    # 元のデータを Reshape して[N, 14]の形状に変更
    data = data.reshape([data.shape[0] // feature_num, feature_num])

    # 元のデータセットをトレーニングセットとテストセットに分割
    # ここでは 80%のデータをトレーニング、20%をテストに使用
    # テストセットとトレーニングセットは重複しないようにする
    ratio = 0.8
    offset = int(data.shape[0] * ratio)
    training_data = data[:offset]

    # トレーニングデータセットの各列の最大値、最小値を計算
    maximums, minimums = training_data.max(axis=0), training_data.min(axis=0)

    # データの正規化パラメータを記録し、予測時にデータを正規化
    global max_values
    global min_values

    max_values = maximums
    min_values = minimums

    # データの正規化処理
    for i in range(feature_num):
        data[:, i] = (data[:, i] - min_values[i]) / (maximums[i] - minimums[i])

    # トレーニングセットとテストセットの分割比率
    training_data = data[:offset]
```

```
    test_data = data[offset:]
    return training_data, test_data
```

データ処理の具体的な実装関数は以下のとおりです。

8.2.2 神経ネットワークモデルの構築

前述の線形神経ネットワークセクションで簡単に分析したように、住宅価格予測タスクは単純な線形回帰問題です。ここでは、PaddlePaddle ディープラーニングフレームワークを使用して、ネットワークの構築を行います。まず、Paddle のいくつかのサブモジュールをインポートします。これには、線形変換層（Linear）、2 次元畳み込み層（Conv2D）、長短期記憶ネットワーク（LSTM）、損失関数、ReLU 活性化層などのクラスと関数が含まれます。Paddle の API インターフェースの詳細な使用方法については、[API ドキュメント]を参照してください。

次に、`Regressor`クラスを作成し、`paddle.nn.Layer`を継承してネットワークをカスタマイズします。`__init__`はクラスの初期化関数で、1 つの全結合層を定義します。入力次元は 13（価格に影響を与える要因）、出力次元は 1（予測される価格）です。続いて、`forward`関数（「前向き計算」を表す）を定義し、特徴量とパラメータから出力予測値の計算プロセスを完了し、このタスクでの住宅価格予測結果を返します。

```
class Regressor(paddle.nn.Layer):

    # self はクラスのインスタンス自身を表す
    def __init__(self):
        # 親クラスのパラメータを初期化
        super(Regressor, self).__init__()

        # 全結合層を定義し、入力次元は 13、出力次元は 1
        self.fc = Linear(in_features=13, out_features=1)

    # ネットワークの前向き計算
    def forward(self, inputs):
        x = self.fc(inputs)
        return x
```

この簡単なネットワークモデルについては、`nn.Sequential`を直接インスタンス化することもできます。これにより、コードを減らして迅速にモデルを構築できます。

```
model = paddle.nn.Sequential(paddle.nn.Linear(13, 1))
```

`Sequential`に追加された層は、それらが追加された順序で実行されます。Sequential モデルで前向き伝播を行うとき、入力データは Sequential モデルに追加された順に各層を通過し、出力が得られます。

8.2.3 モデルのトレーニング

ネットワークモデルを構築し、ハイパーパラメータを設定した後、モデルのトレーニングフェーズに入ります。トレーニングによって、損失関数を最小化するパラメータ値を見つけます。トレーニングが終了した後のパラメータを使用して評価やテストを行うことができます。また、モデルを保存して後で使用することもできます。

トレーニングには、確率的勾配降下法（Stochastic Gradient Descent, SGD）を使用します。全データセットからランダムに小部分のデータを抽出し、この部分データに基づいて勾配と損失を計算し、パラメータを更新します。一部のパラメータの説明：
●mini_batch：各イテレーションで抽出されるデータの小部分を 1 つのミニバッチと呼びます。
●batch_size：1 つのミニバッチに含まれるサンプル数をバッチサイズと呼びます。
●epoch：プログラムがイテレーションする際、ミニバッチでデータを順次抽出し、データセット全体を 1 回遍歴することを 1 エポックと呼びます。

モデルのトレーニングは、内側と外側の 2 つのループ（内側ループと外側ループ）と単一トレーニングの 4 つのステップ（前向き計算、損失関数計算、勾配計算、パラメータ更新）で構成されます。

外側ループ：データセット全体をトレーニングする回数を表し、各ループ前にサンプルデータをランダムにシャッフルします。
内側ループ：データセット全体を遍歴する際、サンプルセットが複数のバッチに分割され、それぞれのバッチでトレーニングを実行します。
前向き計算：1 バッチのサンプルデータをネットワークに投入し、出力結果を計算します。
損失関数計算：前向き計算結果と実際の住宅価格を入力として、損失関数（`square_error_cost`インターフェース）で損失関数値（Loss）を計算します。詳細は[PaddlePaddle の API マニュアル]を参照してください。
勾配計算：逆伝播アルゴリズム（`backward`関数）を実行し、各層の勾配を逐次計算し、設定された最適化アルゴリズムに基づいてパラメータを更新します（`opt.step`関数）。
パラメータ更新：次の計算のために勾配変数をクリアします。

```
# モデル
model = Regressor()

# モデルのトレーニングモードを開始
model.train()

#データの読み込み
training_data, test_data = load_data()

# 最適化アルゴリズムを定義し、確率的勾配降下法（SGD）を使用
# 学習率を 0.01 に設定
opt = paddle.optimizer.SGD(learning_rate=0.01, parameters=model.parameters())

EPOCH_NUM = 10   # 外部ループ回数を設定
BATCH_SIZE = 10  # バッチサイズを設定

# 外部ループを定義
for epoch_id in range(EPOCH_NUM):
    # 各イテレーションの開始前にトレーニングデータの順序をランダムにシャッフル
    np.random.shuffle(training_data)
```

```
    # トレーニングデータを分割し、各バッチに 10 件のデータを含む
    mini_batches = [training_data[k:k+BATCH_SIZE] for k in range(0, len(training_data), BATCH_SIZE)]
    # 内部ループを定義
    for iter_id, mini_batch in enumerate(mini_batches):
        x = np.array(mini_batch[:, :-1])
        y = np.array(mini_batch[:, -1:])
        # numpy データを tensor 形式に変換
        house_features = paddle.to_tensor(x)
        prices = paddle.to_tensor(y)

        # 前向き計算
        predicts = model(house_features)

        # 損失を計算
        loss = F.square_error_cost(predicts, label=prices)
        avg_loss = paddle.mean(loss)
        if iter_id % 40 == 0:
            print("epoch: {}, iter: {}, loss is: {}".format(epoch_id, iter_id, avg_loss.item()))

        # 逆伝播を行い、各層のパラメータの勾配値を計算
        avg_loss.backward()
        # パラメータを更新し、設定された学習率で一歩進む
        opt.step()
        # 次の計算のために勾配変数をクリア
        opt.clear_grad()
```

トレーニングが完了した後、モデルを保存します。

```
# モデルパラメータを保存し、ファイル名を LR_model.pdparams にする
paddle.save(model.state_dict(), './paddle/LR_model.pdparams')
print("モデルパラメータは./paddle/LR_model.pdparams に保存されました")
```

8.2.4 モデルのテスト

トレーニングが完了した後、保存されたモデルを使用してテストを行います。

```
def load_one_example():
    # 上記で読み込まれたテストセットからランダムに 1 つ選択してテストデータとする
    idx = np.random.randint(0, test_data.shape[0])
    one_data, label = test_data[idx, :-1], test_data[idx, -1]
    # このデータの shape を[1, 13]に変更
    one_data =  one_data.reshape([1, -1])

    return one_data, label
# モデルパラメータファイルをインポート
model_dict = paddle.load('./paddle/LR_model.pdparams')
model.load_dict(model_dict)
model.eval()

# データセットのファイルアドレスをパラメータとして設定
one_data, label = load_one_example()
# データを動的グラフの variable 形式に変換
one_data = paddle.to_tensor(one_data)
predict = model(one_data)

# 結果を逆正規化処理
```

```
predict = predict * (max_values[-1] - min_values[-1]) + min_values[-1]
# label データを逆正規化処理
label = label * (max_values[-1] - min_values[-1]) + min_values[-1]

print("推論結果は{}であり、対応するラベルは{}です".format(predict.numpy(), label))
```

実行結果：

```
推論結果は[[17.394253]]であり、対応するラベルは 16.799999237060547 です
```

8.3 参考リンク

1. [Handwritten Digit Recognition]
2. [PaddlePaddle インストールガイド]
3. [PaddlePaddle クイックスタート]
4. [PaddlePaddle API リファレンス]
5. [PaddlePaddle github - PaddlePaddle

第 9 章 MobileNetV2

MobileNet は、2017 年に Google の研究者によって開発された CNN アーキテクチャで、コンピュータビジョンを効率的に携帯電話やロボットなどの小型ポータブルデバイスに統合するためのものです。これは、精度を大幅に低下させることなく実現します。その後、実際のアプリケーションにおけるいくつかの問題を解決するために、v2、v3 バージョンが登場しました。

MobileNet は、標準的な畳み込みとは異なる「深度方向の分離可能な畳み込み（Depthwise Separable Convolutions）」を提案しました。この畳み込みは、モデルの規模を大幅に削減しながら、モデルの性能低下を最小限に抑えることができます。深度方向の分離可能な畳み込みは、深度畳み込み（DW）と逐点畳み込み（PW）の 2 つの操作に分かれます。

- 深度畳み込み（DW）と標準畳み込みの違いは、標準畳み込みでは、畳み込みカーネルがすべての入力チャネルに適用されるのに対し、DW 畳み込みでは、各入力チャネルに異なる畳み込みカーネルを使用する点です。つまり、1 つのカーネルが 1 つの入力チャネルに対応します。
- 逐点畳み込み（PW）は実際には普通の畳み込みですが、1x1 のカーネルを使用します。

MobileNet は、ネットワークのサイズを制御するための 2 つのグローバルハイパーパラメータ（幅の乗数と解像度の乗数）を設計しました。これらのハイパーパラメータを使用して、速度と精度のトレードオフを行い、ユーザーはデバイスの制限に応じてネットワークを調整できます。

MobileNetV2 は、MobileNet の基盤の上に「線形ボトルネック（Linear Bottlenecks）」と「反転残差構造（Inverted Residual）」を提案しました。以下にそのモデル構造を示します：

Input	Operator	t	c	n	s
$224^2 \times 3$	conv2d	-	32	1	2
$112^2 \times 32$	bottleneck	1	16	1	1
$112^2 \times 16$	bottleneck	6	24	2	2
$56^2 \times 24$	bottleneck	6	32	3	2
$28^2 \times 32$	bottleneck	6	64	4	2
$14^2 \times 64$	bottleneck	6	96	3	1
$14^2 \times 96$	bottleneck	6	160	3	2
$7^2 \times 160$	bottleneck	6	320	1	1
$7^2 \times 320$	conv2d 1x1	-	1280	1	1
$7^2 \times 1280$	avgpool 7x7	-	-	1	-
$1 \times 1 \times 1280$	conv2d 1x1	-	k	-	

Table 2: MobileNetV2 : Each line describes a sequence of 1 or more identical (modulo stride) layers, repeated n times. All layers in the same sequence have the same number c of output channels. The first layer of each sequence has a stride s and all others use stride 1. All spatial convolutions use 3×3 kernels. The expansion factor t is always applied to the input size as described in Table 1.

MobileNetV2 のネットワーク構築については、PyTorch の torchvision の実装、TensorFlow の実装、または paddlepaddle の実装を参照できます。

本章では、PyTorch の深層学習フレームワークを使用して、MobileNetV2 を簡単にテストし、花の分類タスクを完了し、Lubancat 上でこのモデルをデプロイしてテストします。

メモ：テスト環境：Lubancat4 は Debian11 を使用し、PC は ubuntu20.04 です。PyTorch バージョン 2.1.0、torchvision バージョン 0.16.0、rknn-Toolkit2 バージョン 2.0.0b0。

9.1 MobileNetV2

MobileNetV2 のテストには花のデータセットを使用します。以下の URL からダウンロードできます：

```
# データセット格納フォルダーを作成
mkdir data && cd data
# データセットをダウンロード
wget https://storage.googleapis.com/download.tensorflow.org/example_images/flower_photos.tgz
# 解凍、データセットのパスは./data/flower_photos
tar zxvf flower_photos.tgz
```

このデータセットには合計 3700 枚の画像が含まれており、5 つのカテゴリの花（デイジー、タンポポ、バラ、ヒマワリ、チューリップ）があります。

9.1.1 カスタムデータセット

ダウンロードした花のデータセットは、同じカテゴリの画像が同じディレクトリに配置され、そのディレクトリ名がカテゴリ名になっています。torchvision が提供する`datasets.ImageFolder`を使用してデータを直接ロードできます。

```
# データのロード
flower_dataset = torchvision.datasets.ImageFolder(root="path/flower_photos", transform=data_transform)
```

また、データセットのロードをカスタムすることもできます。以下に torchvision.datasets.ImageFolder の実装を参考にしたカスタムデータセットの実装例を示します：

サンプルソース 1：train.py

```
# FlowerData クラスを作成し、Dataset を継承します
class FlowerData(Dataset):
    def __init__(self, root_dir, transform=None):
        self.root_dir = root_dir
        self.transform = transform

        classes = sorted(entry.name for entry in os.scandir(self.root_dir) if entry.is_dir())
        class_to_idx = {cls_name: i for i, cls_name in enumerate(classes)}
        self.classes = classes
        self.class_to_idx = class_to_idx

        self.images = self.get_images(self.root_dir, self.class_to_idx)

    def __len__(self):
        return len(self.images)

    def __getitem__(self, index):
        path, target = self.images[index]
        with open(path, "rb") as f:
            img = Image.open(f)
            image = img.convert("RGB")

        if self.transform:
```

```
            image = self.transform(image)  # サンプルの変換

        return image, target

    def get_images(self, directory, class_to_idx):
        images = []
        for target_class in sorted(class_to_idx.keys()):
            class_index = class_to_idx[target_class]
            target_dir = os.path.join(directory, target_class)
            if not os.path.isdir(target_dir):
                continue
            for root, _, fnames in sorted(os.walk(target_dir, followlinks=True)):
                for fname in sorted(fnames):
                    path = os.path.join(root, fname)
                    item = path, class_index
                    images.append(item)

        return images
```

カスタムクラス FlowerData を Dataset から継承します。以下の 3 つの関数を実装する必要があります：

- `def __init__`：初期化
- `def __getitem__`：データの取得、サンプルインデックスに基づいてデータを返し、データ変換などを設定可能
- `def __len__`：データセットの長さを返し、最終的に訓練に使用されるデータセットのサンプル数を表す

その後、カスタムデータセットをインスタンス化し、torch.utils.data.DataLoader を使用してデータセットを読み込み、データ拡張や分割などの操作を行います。

サンプルソース 2：train.py

```
    #この前処理パイプラインは、訓練データをランダムに切り取り、水平反転し、テンソルに変換し、
    #さらに正規化する一連の操作を行います。これにより、モデルの汎化性能を向上させ、学習を安定させることができます。
    data_transform = transforms.Compose([transforms.RandomResizedCrop(224),
                                         transforms.RandomHorizontalFlip(),
                                         transforms.ToTensor(),
                                         transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])])

    # カスタムFlowerDataクラスを初期化、データセットのパスと変換を設定
    flower_data = FlowerData('../flower_photos', transform=data_transform)
    print("Dataset class: {}".format(flower_data.class_to_idx))

    # データセットをランダムに訓練セット（80%）と検証セット（20%）に分割
    train_size = int(len(flower_data) * 0.8)
    validate_size = len(flower_data) - train_size
    train_dataset, validate_dataset = torch.utils.data.random_split(flower_data, [train_size, validate_size])
    print("using {} images for training, {} images for validation.".format(len(train_dataset),
len(validate_dataset)))

    nw = min([os.cpu_count(), batch_size if batch_size > 1 else 0, 8])
    print('Using {} dataloader workers every process ¥n'.format(nw))

    train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True, num_workers=nw)
    validate_loader = DataLoader(validate_dataset, batch_size=1, shuffle=True, num_workers=nw)
```

データを torch.utils.data.random_split を使用して訓練セット（80%）と検証セット（20%）に分割し、その後 DataLoader を使用してデータセットを読み込みます。

9.1.2 MobileNetV2 ネットワーク

MobileNetV2 のネットワーク構造の詳細と説明については、論文および関連資料を参照してください。MobileNetV2 の実装は、torchvision が提供する`models.MobileNetV2`を直接使用することも、自分で構築することもできます。以下に torchvision.models.MobileNetV2 の実装を参考にしたネットワーク構築例を示します：

サンプルソース 3：train.py

```
# 畳み込み、バッチ正規化、活性化関数を定義するクラス。普通の畳み込みと DW 畳み込みの両方に適用可能
class ConvBNReLU(nn.Sequential):
    def __init__(self, in_channel, out_channel, kernel_size=3, stride=1, groups=1):
        padding = (kernel_size - 1) // 2
        super(ConvBNReLU, self).__init__(
            nn.Conv2d(in_channel, out_channel, kernel_size, stride, padding, groups=groups, bias=False),
            nn.BatchNorm2d(out_channel),
            nn.ReLU6(inplace=True)
        )
```

上記の畳み込みにおいて、`groups` パラメータが 1 の場合、これは通常の畳み込みです。`groups` が入力チャンネルと一致する場合、1 つの入力チャンネルが個別のカーネルとだけ畳み込みを行います。これがグループ化畳み込みです。

サンプルソース 4：train.py

```
# 反転残差構造
class InvertedResidual(nn.Module):
    def __init__(self, in_channel, out_channel, stride, expand_ratio):
        super(InvertedResidual, self).__init__()
        # 1×1 畳み込みを通じてチャンネルを拡張する
        hidden_channel = in_channel * expand_ratio
        # ストライドが 1 かつ入力チャンネルと出力チャンネルが等しい場合はショートカット分岐を使用
        self.use_shortcut = stride == 1 and in_channel == out_channel

        layers = []
        if expand_ratio != 1:
            # 1x1 逐点畳み込み
            layers.append(ConvBNReLU(in_channel, hidden_channel, kernel_size=1))
        layers.extend([
            # 3x3 深度方向の畳み込み
            ConvBNReLU(hidden_channel, hidden_channel, stride=stride, groups=hidden_channel),
            # 逐点畳み込み (pw-linear 1x1)
            nn.Conv2d(hidden_channel, out_channel, kernel_size=1, bias=False),
            nn.BatchNorm2d(out_channel),
            # 最後の 1×1 畳み込みには線形活性化関数を使用するため、省略する
        ])

        self.conv = nn.Sequential(*layers)

    def forward(self, x):
        if self.use_shortcut:
```

```
            return x + self.conv(x)
        else:
            return self.conv(x)
```

MobileNetV2 の実装例

サンプルソース 5：train.py

```
# MobileNet-v2 のネットワーク構造を定義
class MobileNetV2(nn.Module):
    def __init__(self, num_classes=1000, alpha=1.0, round_nearest=8):
        super(MobileNetV2, self).__init__()
        block = InvertedResidual
        input_channel = _make_divisible(32 * alpha, round_nearest)
        last_channel = _make_divisible(1280 * alpha, round_nearest)

        # 反転残差ブロックの設定
        inverted_residual_setting = [
            # t, c, n, s
            [1, 16, 1, 1],
            [6, 24, 2, 2],
            [6, 32, 3, 2],
            [6, 64, 4, 2],
            [6, 96, 3, 1],
            [6, 160, 3, 2],
            [6, 320, 1, 1],
        ]

        features = []
        # 通常の conv1 レイヤーを追加
        features.append(ConvBNReLU(3, input_channel, stride=2))
        # 反転残差ブロック
        for t, c, n, s in inverted_residual_setting:
            output_channel = _make_divisible(c * alpha, round_nearest)
            for i in range(n):
                # 最初のみストライドを s に設定し、それ以降は 1 に設定
                stride = s if i == 0 else 1
                features.append(block(input_channel, output_channel, stride, expand_ratio=t))
                input_channel = output_channel

        features.append(ConvBNReLU(input_channel, last_channel, 1))
        self.features = nn.Sequential(*features)

        # 分類層
        self.avgpool = nn.AdaptiveAvgPool2d((1, 1))
        self.classifier = nn.Sequential(
            nn.Dropout(0.2),
            nn.Linear(last_channel, num_classes)
        )

        # 重みの初期化
        for m in self.modules():
            if isinstance(m, nn.Conv2d):
                nn.init.kaiming_normal_(m.weight, mode='fan_out')
                if m.bias is not None:
                    nn.init.zeros_(m.bias)
            elif isinstance(m, nn.BatchNorm2d):
                nn.init.ones_(m.weight)
```

```
                nn.init.zeros_(m.bias)
            elif isinstance(m, nn.Linear):
                nn.init.normal_(m.weight, 0, 0.01)
                nn.init.zeros_(m.bias)

    def forward(self, x):
        x = self.features(x)
        x = self.avgpool(x)
        x = torch.flatten(x, 1)
        x = self.classifier(x)
        return x
```

9.1.3 モデルの訓練と検証

モデルの初期化には、PyTorch が提供する MobileNetV2 の事前学習済みの重み（ImageNet-1K データセットに基づく）を使用し、特徴抽出のパラメータを凍結し、ネットワークの一部の層だけを訓練する、いわゆる転移学習を行います。これにより、他人からの訓練済みパラメータを利用してモデルの訓練時間を大幅に節約できます。

以下に訓練スクリプトの例を示します：

サンプルソース 5：train.py

```
    # GPU があれば、GPU で訓練
    device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
    print("using {} device.".format(device))

    # ハイパーパラメータ
    batch_size = 32
    epochs = 10
    learning_rate = 0.0001

    #この前処理パイプラインは、訓練データをランダムに切り取り、水平反転し、テンソルに変換し、
    #さらに正規化する一連の操作を行います。これにより、モデルの汎化性能を向上させ、学習を安定させることができます。
    data_transform = transforms.Compose([transforms.RandomResizedCrop(224),
                                         transforms.RandomHorizontalFlip(),
                                         transforms.ToTensor(),
                                         transforms.Normalize([0.485, 0.456, 0.406], [0.229, 0.224, 0.225])])

    # カスタム FlowerData クラスを初期化、データセットのパスと変換を設定
    flower_data = FlowerData('../flower_photos', transform=data_transform)
    print("Dataset class: {}".format(flower_data.class_to_idx))

    # データセットをランダムに訓練セット（80%）と検証セット（20%）に分割
    train_size = int(len(flower_data) * 0.8)
    validate_size = len(flower_data) - train_size
    train_dataset, validate_dataset = torch.utils.data.random_split(flower_data, [train_size, validate_size])
    print("using {} images for training, {} images for validation.".format(len(train_dataset),
len(validate_dataset)))

    nw = min([os.cpu_count(), batch_size if batch_size > 1 else 0, 8])
    print('Using {} dataloader workers every process ¥n'.format(nw))
```

```
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True, num_workers=nw)
validate_loader = DataLoader(validate_dataset, batch_size=1, shuffle=True, num_workers=nw)

# モデルをインスタンス化、クラス数 num_classes を設定
net = MobileNetV2(num_classes=5).to(device)

# 事前学習済み重みを使用 https://download.pytorch.org/models/mobilenet_v2-b0353104.pth
model_weight_path = "./mobilenet_v2-b0353104.pth"
assert os.path.exists(model_weight_path), "file {} dose not exist.".format(model_weight_path)

pre_weights = torch.load(model_weight_path, map_location=device)
# print("The type is:".format(type(pre_weights)))

pre_dict = {k: v for k, v in pre_weights.items() if net.state_dict()[k].numel() == v.numel()}
missing_keys, unexpected_keys = net.load_state_dict(pre_dict, strict=False)

# requires_grad == False にして特徴抽出層の重みを凍結し、後ろのプーリング層と classifier 層のみを訓練
for param in net.features.parameters():
    param.requires_grad = False

# クロスエントロピー損失関数を使用
loss_function = nn.CrossEntropyLoss()

# adam オプティマイザを使用、最後のプーリング層と classifier 層のみを最適化
params = [p for p in net.parameters() if p.requires_grad]
optimizer = optim.Adam(params, lr=learning_rate)

# ネットワーク構造を出力
#print(summary(net, (3, 224, 224)))

# モデルを訓練および検証
fit(epochs, net, loss_function, optimizer, train_loader, validate_loader, device)
```

train.py を実行しモデルの訓練と評価を行い、10 epochs をテスト：

```
(.toolkit2_env) (base) csun@CSUN-PC-0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/base/mobilenetv2$ python train.py
using cuda:0 device.
Dataset class: {'daisy': 0, 'dandelion': 1, 'roses': 2, 'sunflowers': 3, 'tulips': 4}
using 2936 images for training, 734 images for validation.
Using 8 dataloader workers every process

train epoch[1/10] loss:1.115: 100%|████████████████████████████████████████████████████████████| 92/92 [00:06<00:00, 13.63it/s]
valid epoch[1/10]: 100%|████████████████████████████████████████████████████████████| 734/734 [00:04<00:00, 171.95it/s]
[epoch 1] train_loss: 1.375 - train_accuracy: 0.520 - val_accuracy: 0.747
train epoch[2/10] loss:0.942: 100%|████████████████████████████████████████████████████████████| 92/92 [00:04<00:00, 22.23it/s]
valid epoch[2/10]: 100%|████████████████████████████████████████████████████████████| 734/734 [00:03<00:00, 197.49it/s]
[epoch 2] train_loss: 1.014 - train_accuracy: 0.767 - val_accuracy: 0.804
train epoch[3/10] loss:0.711: 100%|████████████████████████████████████████████████████████████| 92/92 [00:04<00:00, 22.32it/s]
valid epoch[3/10]: 100%|████████████████████████████████████████████████████████████| 734/734 [00:03<00:00, 189.73it/s]
[epoch 3] train_loss: 0.825 - train_accuracy: 0.801 - val_accuracy: 0.819
train epoch[4/10] loss:0.647: 100%|████████████████████████████████████████████████████████████| 92/92 [00:04<00:00, 22.28it/s]
```

```
valid epoch[4/10]: 100%|████████████████████████████████████████████████████████████| 734/734 [00:03<00:00, 204.88it/s]
[epoch 4] train_loss: 0.715 - train_accuracy: 0.817 - val_accuracy: 0.835
train epoch[5/10] loss:0.546: 100%|████████████████████████████████████████████████| 92/92 [00:04<00:00, 22.41it/s]
valid epoch[5/10]: 100%|████████████████████████████████████████████████████████████| 734/734 [00:03<00:00, 191.47it/s]
[epoch 5] train_loss: 0.652 - train_accuracy: 0.826 - val_accuracy: 0.839
train epoch[6/10] loss:0.687: 100%|████████████████████████████████████████████████| 92/92 [00:04<00:00, 22.37it/s]
valid epoch[6/10]: 100%|████████████████████████████████████████████████████████████| 734/734 [00:03<00:00, 193.22it/s]
[epoch 6] train_loss: 0.600 - train_accuracy: 0.843 - val_accuracy: 0.847
train epoch[7/10] loss:0.527: 100%|████████████████████████████████████████████████| 92/92 [00:04<00:00, 21.61it/s]
valid epoch[7/10]: 100%|████████████████████████████████████████████████████████████| 734/734 [00:03<00:00, 196.67it/s]
[epoch 7] train_loss: 0.558 - train_accuracy: 0.842 - val_accuracy: 0.843
train epoch[8/10] loss:0.554: 100%|████████████████████████████████████████████████| 92/92 [00:04<00:00, 21.81it/s]
valid epoch[8/10]: 100%|████████████████████████████████████████████████████████████| 734/734 [00:03<00:00, 197.53it/s]
[epoch 8] train_loss: 0.543 - train_accuracy: 0.845 - val_accuracy: 0.851
train epoch[9/10] loss:0.537: 100%|████████████████████████████████████████████████| 92/92 [00:04<00:00, 22.16it/s]
valid epoch[9/10]: 100%|████████████████████████████████████████████████████████████| 734/734 [00:03<00:00, 188.93it/s]
[epoch 9] train_loss: 0.501 - train_accuracy: 0.857 - val_accuracy: 0.875
train epoch[10/10] loss:0.495: 100%|███████████████████████████████████████████████| 92/92 [00:04<00:00, 22.20it/s]
valid epoch[10/10]: 100%|███████████████████████████████████████████████████████████| 734/734 [00:03<00:00, 194.45it/s]
[epoch 10] train_loss: 0.500 - train_accuracy: 0.845 - val_accuracy: 0.876

10 epochs completed in 1m 24s.
```

torchvision の MobileNetV2 を使用する場合、モデル訓練ファイルに MobileNetV2 をインポートするだけで、自分で定義した MobileNetV2 を使用する必要はありません。

```
#Pytorch のモデル MobileNetV2
from torchvision.models import MobileNetV2 #<---コメントアウト
#カスタマイズのモデル MobileNetV2
#from model import MobileNetV2  #<---コメント
```

上記内容のみを修正してもう一回実施してみて結果を確認できます。

訓練は、精度が最も高い MobileNetV2.pth モデルの重みファイルを保存します。最後にモデル予測の推論は行わず、後で Toolkit2 を使用してボード上で推論テストを行います。

9.1.4 torchscript モデル保存

訓練時に保存した MobileNetV2.pth のモデル重みを使用して、ボードにデプロイするために torchscript 形式のモデルをエクスポートします。以下にその例を示します：

サンプルソース 6：export.py

```
import torch
import os
from model import MobileNetV2


if __name__ == '__main__':

    # モデル
    model = MobileNetV2(num_classes=5)

    # 重みを読み込む
    model.load_state_dict(torch.load("./MobileNetV2.pth"))

    model.eval()
    # モデルを保存
    trace_model = torch.jit.trace(model, torch.Tensor(1, 3, 224, 224))
    trace_model.save('./MobileNetV2.pt')
```

```
# モデルをエクスポート
    (.toolkit2_env) (base) csun@CSUN-PC-
0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/base/mobilenetv2$ python3 export.py
```

上記プログラムを実行して変換後の「MobileNetV2.pt」ファイルが生成されます。

9.2 ボードへのデプロイとテスト

9.2.1 rknn モデルへの変換

RKNN Toolkit2 を使用して、エクスポートされたモデルを rknn モデルに変換し、簡単なモデルテストを行います。以下にその例を示します：

サンプルソース 7：pt2rknn.py

```
if __name__ == '__main__':

    model = './MobileNetV2.pt'

    input_size_list = [[1, 3, 224, 224]]

    # Create RKNN object
    rknn = RKNN()

    # Pre-process config, デフォルトで rk3588 を設定される
    print('--> Config model')
    rknn.config(mean_values=[[128, 128, 128]], std_values=[[128, 128, 128]], target_platform='rk3588')
    print('done')
```

```
    # Load model
    print('--> Loading model')
    ret = rknn.load_pytorch(model=model, input_size_list=input_size_list)
    if ret != 0:
        print('Load model failed!')
        exit(ret)
    print('done')

    # Build model
    print('--> Building model')
    # ret = rknn.build(do_quantization=True, dataset='./dataset.txt')
    ret = rknn.build(do_quantization=False)
    if ret != 0:
        print('Build model failed!')
        exit(ret)
    print('done')

    # Export rknn model
    print('--> Export rknn model')
    ret = rknn.export_rknn('./MobileNetV2.rknn')
    if ret != 0:
        print('Export rknn model failed!')
        exit(ret)
    print('done')

    #Set inputs
    img = cv2.imread('./sun.jpg')
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = cv2.resize(img, (224,224))
    img = np.expand_dims(img, 0)

    # Init runtime environment
    print('--> Init runtime environment')
    ret = rknn.init_runtime()
    if ret != 0:
        print('Init runtime environment failed!')
        exit(ret)
    print('done')

    # Inference
    print('--> Running model')
    outputs = rknn.inference(inputs=[img])
    # np.save('./MobileNetV2.npy', outputs[0])
    print(outputs[0][0])
    show_outputs(softmax(np.array(outputs[0][0])))
    print('done')

    rknn.release()
```

プログラムを PC 側で実行し、rknn モデルをエクスポートします。

```
(.toolkit2_env) (base) csun@CSUN-PC-
0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/base/mobilenetv2$ python3 pt2rknn.py
I rknn-toolkit2 version: 2.0.0b0+9bab5682
--> Config model
done
--> Loading model
I PtParse: 100%|████████████████████████████████████████| 937/937 [00:00<00:00, 2092.01it/s]
I Loading : 100%|████████████████████████████████████████| 332/332 [00:00<00:00, 1995.60it/s]
```

```
done
--> Building model
I rknn building ...
I rknn buiding done.
done
--> Export rknn model
done
--> Init runtime environment
I Target is None, use simulator!
done
--> Running model
W inference: The 'data_format' is not set, and its default value is 'nhwc'!
I GraphPreparing : 100%|████████████████████████████████| 100/100 [00:00<00:00, 5261.29it/s]
I SessionPreparing : 100%|████████████████████████████████| 100/100 [00:00<00:00, 2201.39it/s]
[-1.9589844  -0.43530273 -1.3818359   0.22485352 -3.578125  ]

 class    prob
sunflowers:    54.0%
dandelion:    27.9%
roses:    10.8%
daisy:    6.08%
tulips:    1.2%

done
```

9.2.2 Toolkit Lite2 を使用したデプロイとテスト

Toolkit Lite2 を使用して簡単なデプロイテストを行います。Toolkit Lite2 の使用方法については「第 4 章 RKNN Toolkit Lite2 介紹」を参照してください：

サンプルソース 8：test.py

```python
if __name__ == '__main__':
    rknn_lite = RKNNLite()

    # Load RKNN model
    print('--> Load RKNN model')
    ret = rknn_lite.load_rknn(RK3588_RKNN_MODEL)
    if ret != 0:
        print('Load RKNN model failed')
        exit(ret)
    print('done')

    # Load and preprocess image
    ori_img = cv2.imread('./tulips.jpg')
    if ori_img is None:
        print('Image not found.')
        exit(1)

    img = cv2.cvtColor(ori_img, cv2.COLOR_BGR2RGB)
    img = cv2.resize(img, (224, 224))
    img = np.expand_dims(img, 0)

    # Print the shape and first few pixels of the input to confirm it matches the expected input shape
    print(f'Input shape: {img.shape}')
    print(f'First few pixels: {img.flatten()[:10]}')
```

```
    # Init runtime environment
    print('--> Init runtime environment')
    ret = rknn_lite.init_runtime()
    if ret != 0:
        print('Init runtime environment failed')
        exit(ret)
    print('done')

    # Inference
    print('--> Running model')
    outputs = rknn_lite.inference(inputs=[img])
    if outputs is not None:
        # Print the raw output from the model
        print(f'Raw outputs: {outputs}')
        print(f'Outputs shape: {np.array(outputs).shape}')

        # Flatten the output if necessary
        flat_outputs = np.array(outputs).flatten()

        print(f'Flattened outputs: {flat_outputs}')

        show_outputs(softmax(flat_outputs))
    else:
        print("Inference failed.")
    print('done')

    rknn_lite.release()
```

基板上のテスト結果

```
cat@lubancat:~/lubancat_ai_manual_code/base/mobilenetv2$ python test.py
--> Load RKNN model
done
Input shape: (1, 224, 224, 3)
First few pixels: [60 81 48 37 51 32 37 38 15 26]
--> Init runtime environment
I RKNN: [15:59:03.739] RKNN Runtime Information, librknnrt version: 2.0.0b0 (35a6907d79@2024-03-24T10:31:14)
I RKNN: [15:59:03.739] RKNN Driver Information, version: 0.9.2
I RKNN: [15:59:03.739] RKNN Model Information, version: 6, toolkit version: 2.0.0b0+9bab5682(compiler
version: 2.0.0b0 (35a6907d79@2024-03-24T02:34:11)), target: RKNPU v2, target platform: rk3588, framework name:
PyTorch, framework layout: NCHW, model inference type: static_shape
done
--> Running model
Raw outputs: [array([[-2.9511719, -3.8125   , -1.3916016, -1.2236328,  2.0058594]],
      dtype=float32)]
Outputs shape: (1, 1, 5)
Flattened outputs: [-2.9511719 -3.8125    -1.3916016 -1.2236328  2.0058594]

 Class     Prob
tulips:    92.333%
sunflowers:    3.654%
roses:    3.089%
daisy:    0.649%
dandelion:    0.274%

done
```

9.3 参考リンク

- https://pytorch.org
- https://arxiv.org/abs/1801.04381
- https://arxiv.org/pdf/1704.04861
- https://github.com/rockchip-linux/rknn-toolkit2

第 10 章 YOLOv3

目標検出はコンピュータビジョンの基本的なタスクであり、その目標は画像やビデオの中で興味のあるすべての目標（物体）を見つけ、それらの物体のカテゴリと位置を特定することです。実際のアプリケーションでは、目標検出には広範な用途があります。例えば、安全監視、スマートドライビング、電子商取引などがあります。

目標検出タスクの主な課題は、画像やビデオの中の目標物体を正確かつ効率的に検出し、それらのカテゴリと位置を正確に予測することです。実際のアプリケーションでは、物体の外観、形状、姿勢の違い、撮影時の光照や遮蔽などの要素を考慮する必要があります。また、複数の目標物体をどのように確定するか、背景のノイズや異常値をどのように処理するかといった問題も解決しなければなりません。

目標検出は一般的にワンステップ目標検出と二段階目標検出に分けられます：
●二段階目標検出：最初の段階で画像やビデオの中の物体を分類し、異なるカテゴリに分けます。次の段階で各カテゴリの物体の位置を回帰器を用いて予測します。代表的なアルゴリズムには R-CNN、Fast R-CNN などがあります。
●ワンステップ：1 つのネットワークを使用して候補領域を生成し、物体のカテゴリと位置を同時に予測します。一般的なアルゴリズムには YOLO、SSD などがあります。

その他にも、Faster R-CNN、DenseNet などのアルゴリズムがあります。

YOLOv3（You Only Look Once version 3）は、目標検出アルゴリズムの一種で、YOLO 目標検出アルゴリズムの重要なバージョンです。YOLOv3 は YOLOv2 に基づいてネットワークの主幹を改良し、多尺度の特徴マップを利用して検出を行い、softmax に代わる複数の独立したロジスティック回帰分類器を用いてカテゴリ分類を予測する方法を改善しました。

本章では、Pytorch ディープラーニングフレームワークを使用して、yolov3 のモデル構造、推論テスト、モデルの訓練、および rk356x/rk3588 上での yolov3 モデルの展開などを簡単に紹介します。

注意：テスト環境：Lubancat4 ボードは Debian11 システムを使用、PC システムは Ubuntu20.04、rknn-Toolkit2 バージョン 2.0.0b0 を使用。

10.1 YOLOv3 の簡単な分析

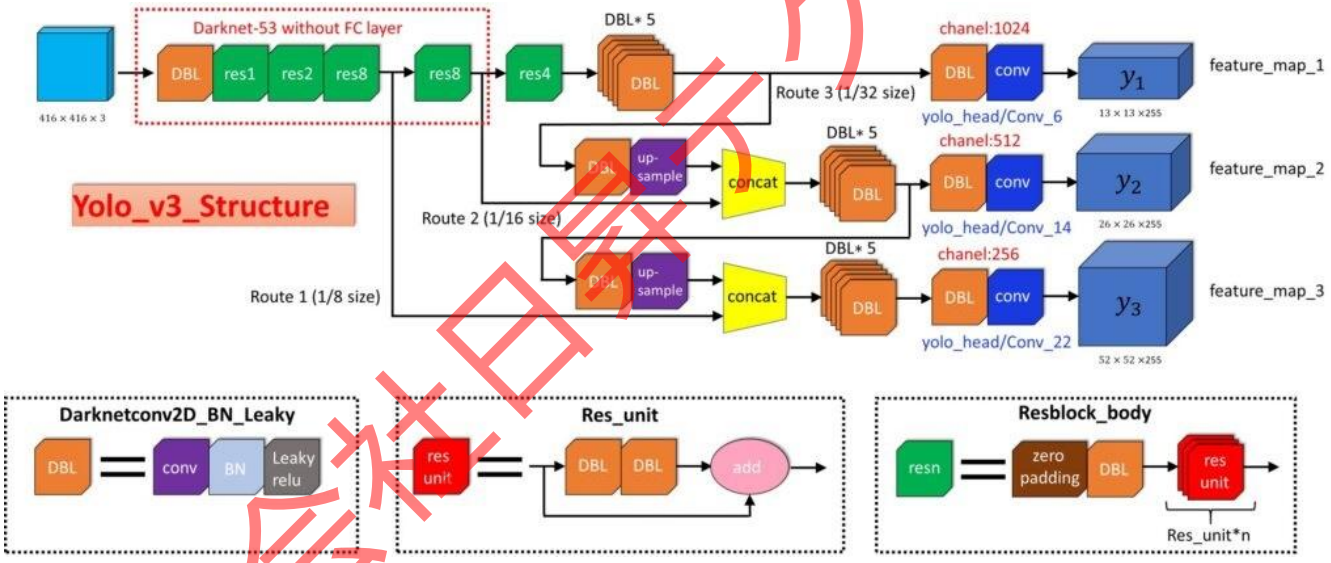
Joseph Redmon らは 2015 年に YOLO アルゴリズムを提案し、通常は YOLOv1 とも呼ばれます。2016 年に彼らはアルゴリズムを改良し、YOLOv2 バージョンを提案しました。2018 年には YOLOv3 バージョンが発展しました。YOLOv3 の原理についての詳細は論文を参照してください。

YOLOv3 は Darknet-53 を深層神経ネットワークの主幹特徴抽出ネットワークとして採用しています（論文から）。

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Table 1. **Darknet-53.**

全体のネットワーク構造は以下の通りです：



YOLOv3 には 3 つの出力特徴層（13x13x255、26x26x255、52x52x255）があり、それぞれが大中小の 3 つのスケールの目標を検出する役割を果たします。これにより小さな目標の検出能力が向上します。13x13x255 はネットワークが最終的に 13X13 のグリッドを出力し、各グリッドに 3 つのアンカーがあり、各アンカーに対して 1 つの予測枠が対応します。各予測枠には 4 つの位置座標と 1 つの信頼度パラメータがあり、さらに coco データセットの 80 のカテゴリが加わります。3 つのアンカーが合計で 255 のパラメータを持ちます。これは 13x13x3x(80+5) となり、この層の特徴受容野は大きく、大きな物体の予測を担当します。他の 2 つの特徴層のパラメータの解釈も同様です。

10.1.1 YOLOv3 の実装

YOLOv3 の論文は Darknet を使用して実装されており、Darknet は C と CUDA に基づいたオープンソースの深層学習フレームワークです。Darknet を使用して実装された YOLOv3 のコードリポジトリは、

https://github.com/pjreddie/darknet/tree/master にあります。

YOLOv3 は PyTorch でも実装されており、人気のあるリポジトリには
https://github.com/ultralytics/yolov3、
https://github.com/eriklindernoren/PyTorch-YOLOv3 などがあります。

YOLOv3 は TensorFlow でも実装されており、
https://github.com/zzh8829/yolov3-tf2 を参照できます。PaddlePaddle 実装については公式の PaddleDetection を参照してください。

本チュートリアルのテストは、PyTorch で実装された ultralytics yolov3 リポジトリを使用し、以前のバージョン（archive ブランチ）を使用します。

```
git clone https://github.com/ultralytics/yolov3 -b archive
```

10.2 ultralytics yolov3 のテスト

10.2.1 環境のインストール

関連する環境をインストールします。conda を使用して仮想環境を作成し、前の環境構築章を参照できます：

```
# conda を使用して yolov3 という名前の環境を作成し、Python バージョンを指定します
conda create -n yolov3 python=3.8
# 環境に入る
conda activate yolov3
```

ultralytics yolov3 のソースコードを取得し、関連する環境をインストールします：

```
# プログラムを取得する
git clone https://github.com/ultralytics/yolov3 -b archive
# yolov3 ディレクトリに移動
cd yolov3
# 関連ソフトウェアライブラリをインストールする
pip install -r requirements.txt
# デフォルトでは pytorch、gpu サポートなどがインストールされます。PyTorch 公式サイト
https://pytorch.org/get-started/previous-versions でコマンドを確認してインストールできます。以下のコマンドは cuda11.6 のバージョンをインストールする例です：
pip install torch==1.13.0+cu116 torchvision==0.14.0+cu116 torchaudio==0.13.0 --extra-index-url
https://download.pytorch.org/whl/cu116
```

PyTorch と NumPy は、requirements.txt の要件に従ってインストールすることをお勧めします。高バージョンをインストールすると多くの小さな問題が発生する可能性があります（インターネットで解決策を見つけることができます）。

10.2.2 目標検出

事前に訓練されたモデルを使用して目標検出を行います。まず darknet の重みを取得します：

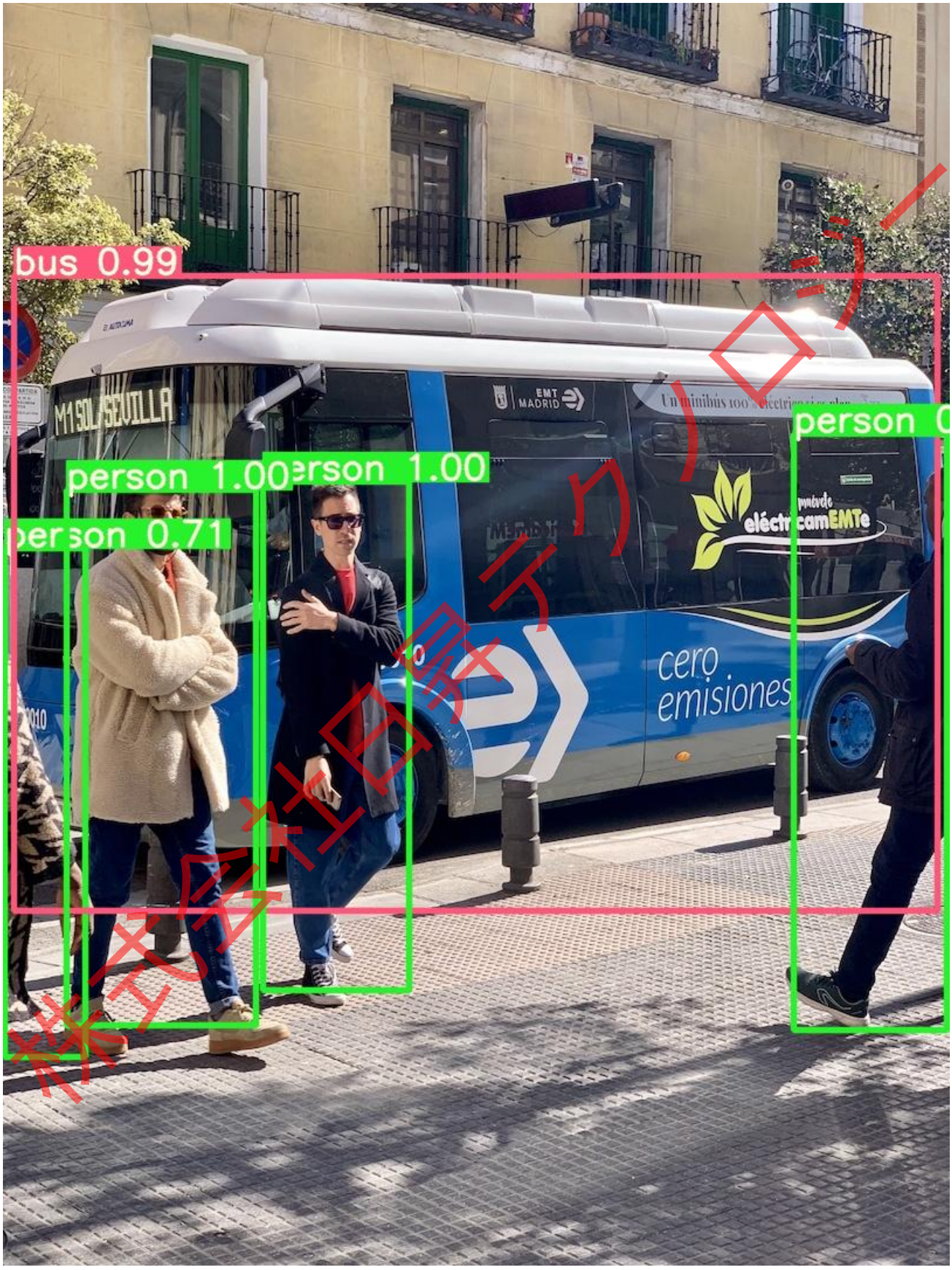
```
# darknet の重みファイルを取得します。yolov3-tiny は軽量版の yolov3、yolov3-spp は spp ネットワークが追加された重みファイルです、「weights」というフォルダーにダウンロードしてください。
cd weights
wget -c https://pjreddie.com/media/files/yolov3.weights
wget -c https://pjreddie.com/media/files/yolov3-tiny.weights
wget -c https://pjreddie.com/media/files/yolov3-spp.weights
# または pytorch の重みファイルを取得します
# darknet53 の重み（最初の 75 レイヤーのみ）
# wget -c https://pjreddie.com/media/files/darknet53.conv.74
# またはプロジェクトの download_yolov3_weights.sh スクリプトを使用して重みを取得します
```

目標検出プログラムを実行します。-weights で重みファイルを指定し、ここでは yolov3.weights をテストします。-cfg で設定ファイルを指定し、指定しない場合はデフォルトで cfg/yolov3-spp.cfg が使用されます。ここでは cfg/yolov3.cfg をテストします。-source で検出する画像のパス、または画像フォルダのパスを指定します。

```
# 単一の画像推論をテストします(yolov3 フォルダーのパスで実行)
python3 detect.py --source data/samples/bus.jpg --weights weights/yolov3.weights --cfg cfg/yolov3.cfg
# 複数の画像推論をテストします。プロジェクトディレクトリ data/samples 内のすべての画像を推論します
python3 detect.py --source data/samples --weights weights/yolov3.weights --cfg cfg/yolov3.cfg
```

単一の画像推論の出力：

```
(.toolkit2_env) (base) csun@CSUN-PC-0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/base/yolov3$ python3 detect.py --source data/samples/bus.jpg --weights weights/yolov3.weights --cfg cfg/yolov3.cfg
Namespace(agnostic_nms=False, augment=False, cfg='cfg/yolov3.cfg', classes=None, conf_thres=0.3, device='', fourcc='mp4v', half=False, img_size=512, iou_thres=0.6, names='data/coco.names', output='output', save_txt=False, source='data/samples/bus.jpg', view_img=False, weights='weights/yolov3.weights')
Using CUDA device0 _CudaDeviceProperties(name='NVIDIA GeForce GTX 1070 Ti', total_memory=8105MB)

/home/csun/project-Toolkit2/.toolkit2_env/lib/python3.8/site-packages/torch/functional.py:504: UserWarning: torch.meshgrid: in an upcoming release, it will be required to pass the indexing argument. (Triggered internally at ../aten/src/ATen/native/TensorShape.cpp:3526.)
  return _VF.meshgrid(tensors, **kwargs)  # type: ignore[attr-defined]
Model Summary: 222 layers, 6.19491e+07 parameters, 6.19491e+07 gradients, 117.5 GFLOPS
image 1/1 data/samples/bus.jpg: 512x384 4 persons, 1 buss, Done. (0.029s)
Results saved to /home/csun/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/base/yolov3/output
Done. (1.898s)
```



複数の画像推論の出力：

```
(.toolkit2_env) (base) csun@CSUN-PC-
0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/base/yolov3$ python3 detect.py --
source data/samples --weights weights/yolov3.weights --cfg cfg/yolov3.cfg
Namespace(agnostic_nms=False, augment=False, cfg='cfg/yolov3.cfg', classes=None,
conf_thres=0.3, device='', fourcc='mp4v', half=False, img_size=512, iou_thres=0.6,
names='data/coco.names', output='output', save_txt=False, source='data/samples',
view_img=False, weights='weights/yolov3.weights')
Using CUDA device0 _CudaDeviceProperties(name='NVIDIA GeForce GTX 1070 Ti',
total_memory=8105MB)

/home/csun/project-Toolkit2/.toolkit2_env/lib/python3.8/site-packages/torch/functional.py:504:
UserWarning: torch.meshgrid: in an upcoming release, it will be required to pass the indexing
argument. (Triggered internally at ../aten/src/ATen/native/TensorShape.cpp:3526.)
  return _VF.meshgrid(tensors, **kwargs)  # type: ignore[attr-defined]
Model Summary: 222 layers, 6.19491e+07 parameters, 6.19491e+07 gradients, 117.5 GFLOPS
image 1/2 data/samples/bus.jpg: 512x384 4 persons, 1 buss, Done. (0.028s)
image 2/2 data/samples/zidane.jpg: 288x512 2 persons, 1 ties, Done. (0.028s)
Results saved to
/home/csun/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/base/yolov3/output
Done. (0.616s)
```

「output」ディレクトリに./data/samples と同じ写真名の複数予測写真が生成されます。

10.2.3 モデルの訓練

自分の yolov3 モデルを再訓練できます。独自のデータセット、ハイパーパラメータ、系列戦略などに基づいて行います。以下では、coco データセットに基づいて yolo モデルを再訓練します。独自のデータセットを作成して訓練することもできます。

プロジェクト内には、coco2017 データセットを取得するスクリプト（get_coco2017.sh）が用意されています。このスクリプトを実行してデータセットを取得します：

```
# データセットを取得します
bash get_coco2017.sh
```

スクリプトが正常に動作しない場合は、データセットをインターネットから手動(下に書いてある)でダウンロードしてプロジェクトのルートディレクトリに配置することもできます。データセットのディレクトリ構造は以下の通りです：

```
.
├── LICENSE
├── README.txt
├── annotations
│   └── instances_val2017.json
├── images
│   ├── train2017 # 訓練データセット画像（118k images, 19G）
│   └── val2017 # 評価データセット画像（5k images, 1G）
├── labels
│   ├── train2017
│   └── val2017
├── test-dev2017.txt
├── train2017.txt
└── val2017.txt
7 directories, 6 files
```

上記構成にならない場合、下記通り手動ダウンロードできます。

```
# https://cocodataset.org/#download
# データセット構成を取得します
wget https://github.com/ultralytics/yolov5/releases/download/v1.0/coco2017labels.zip
# get_coco2017.sh により取得した train2017、val2017 二つフォルダーを上記解凍後の images にコピー
# get_coco2017.sh から train2017、val2017 を取得できない場合、下記通り直接ダウンロードできます。
# wget http://images.cocodataset.org/zips/train2017.zip
# wget http://images.cocodataset.org/zips/val2017.zip
# 最終的に yolov3/data/coco に上記構成になれば、訓練可能
```

実際のテストでは、coco2017 データセットから 64 枚の画像を抽出した coco64 データセットを使用します。これはテスト専用です。

準備した coco64 データセットを使用し、data/coco64.data ファイルを設定します：

```
classes=80
train=data/coco64/train2017.txt
valid=data/coco64/val2017.txt
names=data/coco.names
```

ここで、classes はカテゴリ数、coco2017 データセットには 80 のカテゴリがあります。train は訓練に使用する画像リスト、valid は検証に使用する画像リストです。train と valid は同じに設定されていますが、これは単なるテストの例です。最後の names はカテゴリ名を対応する実際のカテゴリ ID と一致させます。独自のデータセットを使用する場合、リストファイルを生成し、classes を修正し、ネットワークの設定ファイルも修正する必要があります。

train.py を実行してモデルを訓練します。実行コマンド中の-data はデータ設定ファイルを指定し、-weights は重みファイルを指定し、-cfg はモデルのネットワーク設定ファイルを指定します。-epochs は訓練ループのエポック数を指定し、-batch-size は一度に訓練する画像のグループを設定します。

```
# ここでテストのため、50epochs のみを訓練
python3 train.py --data data/coco64.data --weights weights/yolov3.weights --cfg cfg/yolov3.cfg
```

```
--epochs 50 --batch-size 16
```

訓練エラー１：

AttributeError: module 'numpy' has no attribute 'int'

修正：np.int--->np.int64 に修正

訓練エラー２：

indices.append((b, a, gj.clamp_(0, gain[3] - 1), gi.clamp_(0, gain[2] - 1))) # image, anchor, grid indices

RuntimeError: result type Float can't be cast to the desired output type long int

gain は float タイプですので、修正：

```
indices.append((b, a, gj.clamp_(0, gain[3].long() - 1), gi.clamp_(0, gain[2].long() - 1))) # image, anchor, grid indices
```

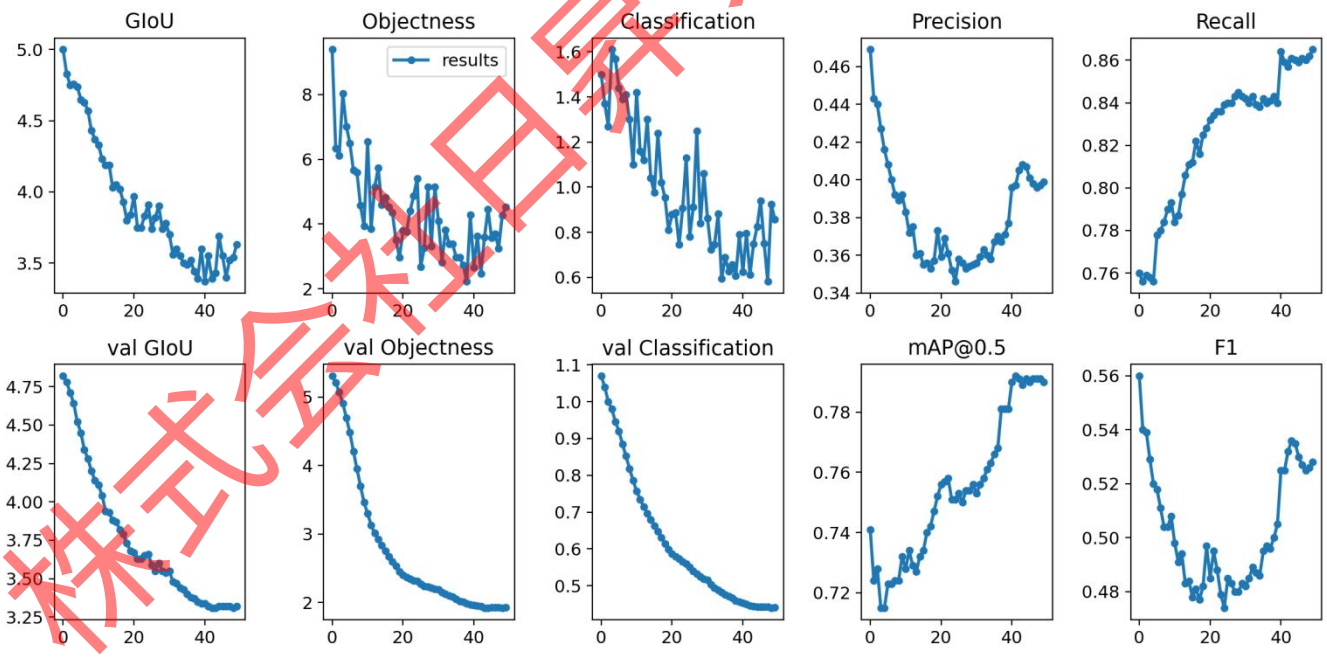
訓練エラー３：

torch.cuda.OutOfMemoryError: CUDA out of memory. Tried to allocate 32.00 MiB. GPU 0 has a total capacty of 7.92 GiB of which 26.31 MiB is free.

実行パラメータ修正：（遅くなりますが、CPU で訓練）

```
# ここでテストのため、50epochs のみを訓練
python3 train.py --data data/coco64.data --weights weights/yolov3.weights --cfg cfg/yolov3.cfg --epochs 50 --batch-size 16 --device cpu
```

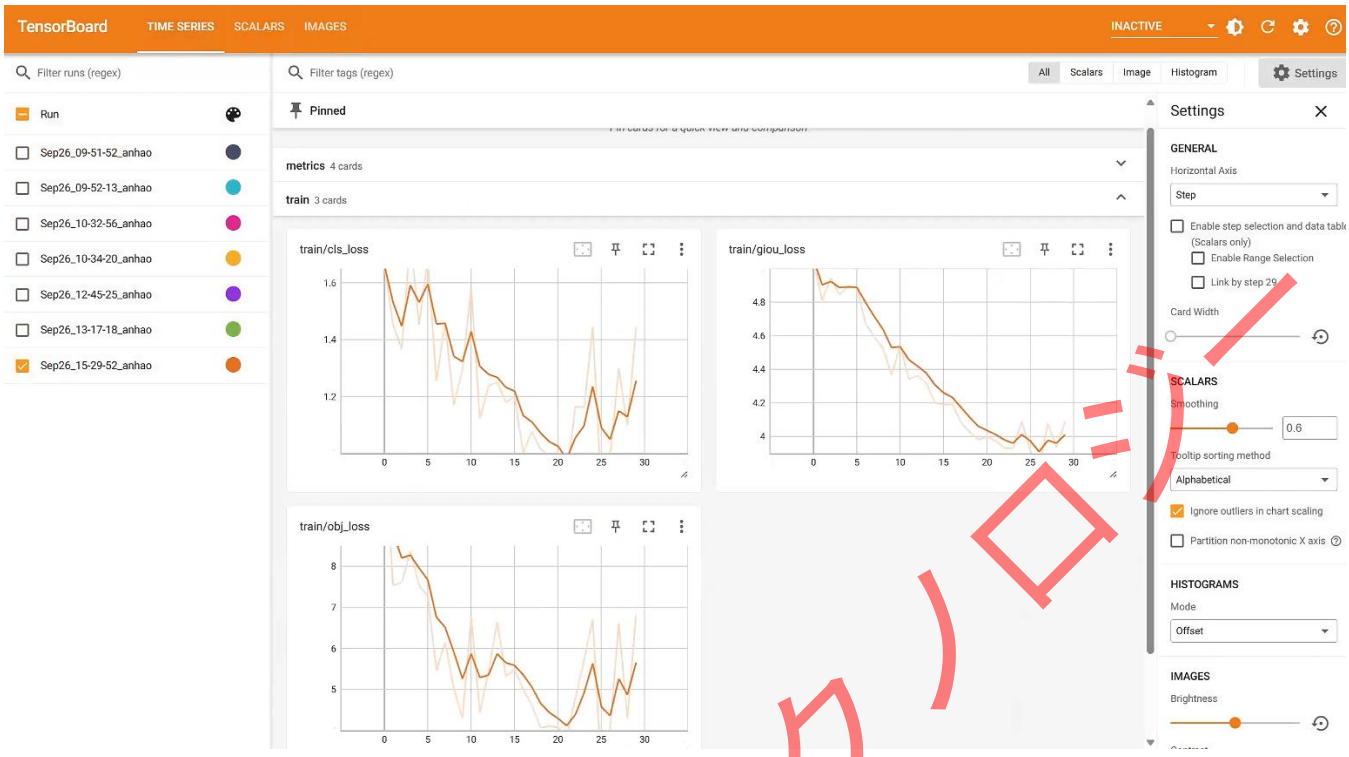
訓練されたモデルの重みは weights/last.pt に保存され、いくつかの訓練パラメータ結果は results.png ファイルに保存されます。以下は例です：



訓練の可視化：

```
# コマンド実行：
tensorboard --logdir=runs --bind_all
# 出力
TensorFlow installation not found - running with reduced feature set.
TensorBoard 2.14.0 at http://anhao.:6006/ (Press CTRL+C to quit)
```

ブラウザで IP:6006 にアクセスすると、以下のように表示されます：



訓練のパラメータ、結果画像などを確認できます。

10.2.4 モデルの保存

前述の訓練が正常に完了すると、weights ディレクトリにモデルの重み last.pt が保存されます。ボード上でテストするために、これを darknet の重みに変換します。主に models.py で定義された convert 関数を使用します。

```
# コマンドラインで python3 コマンドを使用し、models をロードして、convert 関数を使用して変換し、
weights/last.weights に保存します。
(yolov3) csun@CSUN-PC-0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/base/yolov3$ python3
Python 3.8.19 (default, Mar 20 2024, 19:58:24)
[GCC 11.2.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from models import *
>>> convert('cfg/yolov3.cfg', 'weights/last.pt')
WARNING: smart bias initialization failure.
WARNING: smart bias initialization failure.
WARNING: smart bias initialization failure.
Model Summary: 222 layers, 6.19491e+07 parameters, 6.19491e+07 gradients
Success: converted 'weights/last.pt' to 'weights/last.weights'
```

10.3 RKNN モデルへの変換

rknn-toolkit2 ツールは darknet の重みと設定ファイルのインポートをサポートしており、モデルを RKNN モデルに変換できます。前述の darknet の重みと設定ファイルを使用し、rknn.load_darknet を使用してモデルをインポートし、RKNN モデルに変換します。以下のプログラムを参照してください：

サンプルソース 1：yolov3_to_rknn.py

```
import numpy as np
import cv2
from rknn.api import RKNN

from yolov3_utils import yolov3_post_process, draw

GRID0 = 13
GRID1 = 26
GRID2 = 52
LISTSIZE = 85
SPAN = 3

if __name__ == '__main__':

    MODEL_PATH = './cfg/yolov3.cfg'
    WEIGHT_PATH = './weights/last.weights'
    RKNN_MODEL_PATH = './model/RK3588/yolov3_quantization.rknn'
    image = './dog_bike_car_416x416.jpg'
    DATASET = './dataset.txt'

    # Create RKNN object
    rknn = RKNN(verbose=True)

    # Pre-process config
    print('--> Config model')
    rknn.config(mean_values=[0, 0, 0], std_values=[255, 255, 255], target_platform='rk3588')
    print('done')

    # Load model
    print('--> Loading model')
    ret = rknn.load_darknet(model=MODEL_PATH, weight=WEIGHT_PATH)
    if ret != 0:
        print('Load model failed!')
        exit(ret)
    print('done')


    # Build model
    print('--> Building model')
    ret = rknn.build(do_quantization=True, dataset=DATASET)
    if ret != 0:
        print('Build model failed!')
        exit(ret)
    print('done')

    # Export rknn model
    print('--> Export rknn model')
    ret = rknn.export_rknn(RKNN_MODEL_PATH)
    if ret != 0:
        print('Export rknn model failed!')
        exit(ret)
    print('done')

    # Set inputs
    img = cv2.imread(image)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    # Init runtime environment
    print('--> Init runtime environment')
```

```
    ret = rknn.init_runtime()
    if ret != 0:
        print('Init runtime environment failed!')
        exit(ret)
    print('done')

    # Inference
    print('--> Running model')
    outputs = rknn.inference(inputs=[img])
    print('done')

    input0_data = outputs[0]
    np.save('./darknet_yolov3_416x416_0.npy', input0_data) # 1*255*13*13
    input1_data = outputs[1]
    np.save('./darknet_yolov3_416x416_1.npy', input1_data)
    input2_data = outputs[2]
    np.save('./darknet_yolov3_416x416_2.npy', input1_data)

    input0_data = input0_data.reshape(SPAN, LISTSIZE, GRID0, GRID0) # 3*85*13*13
    input1_data = input1_data.reshape(SPAN, LISTSIZE, GRID1, GRID1)
    input2_data = input2_data.reshape(SPAN, LISTSIZE, GRID2, GRID2)

    input_data = []
    input_data.append(np.transpose(input0_data, (2, 3, 0, 1)))  # 13*13*3*85
    input_data.append(np.transpose(input1_data, (2, 3, 0, 1)))
    input_data.append(np.transpose(input2_data, (2, 3, 0, 1)))

    boxes, classes, scores = yolov3_post_process(input_data)

    image = cv2.imread(image)
    if boxes is not None:
        draw(image, boxes, scores, classes)

    print('Save results to results.jpg!')
    cv2.imwrite('result.jpg', image)

    rknn.release()
```

ここでは後処理部分を簡単に説明します。モデルの出力は 3 つの特徴マップ（1*255*13*13、1*255*26*26、1*255*52*52）で、reshape と transpose 操作を経て（13*13*3*85、26*26*3*85、52*52*3*85）に変換されます。その後、yolov3_post_process 関数に渡されます。

10.4 RKNN モデルのデプロイと実行

10.4.1 Toolkit Lite2 を使用したデプロイテスト

Toolkit Lite2 の使用フローについては、前の《RKNN Toolkit Lite2 介绍》を参照してください。以下は前述の変換済み yolov3 モデルを使用し、具体的なボードでデプロイ推論を行うプログラムです：

サンプルソース 2: yolov3_inference.py

```
if __name__ == '__main__':

    host_name = get_host()
    if host_name == 'RK3566_RK3568':
```

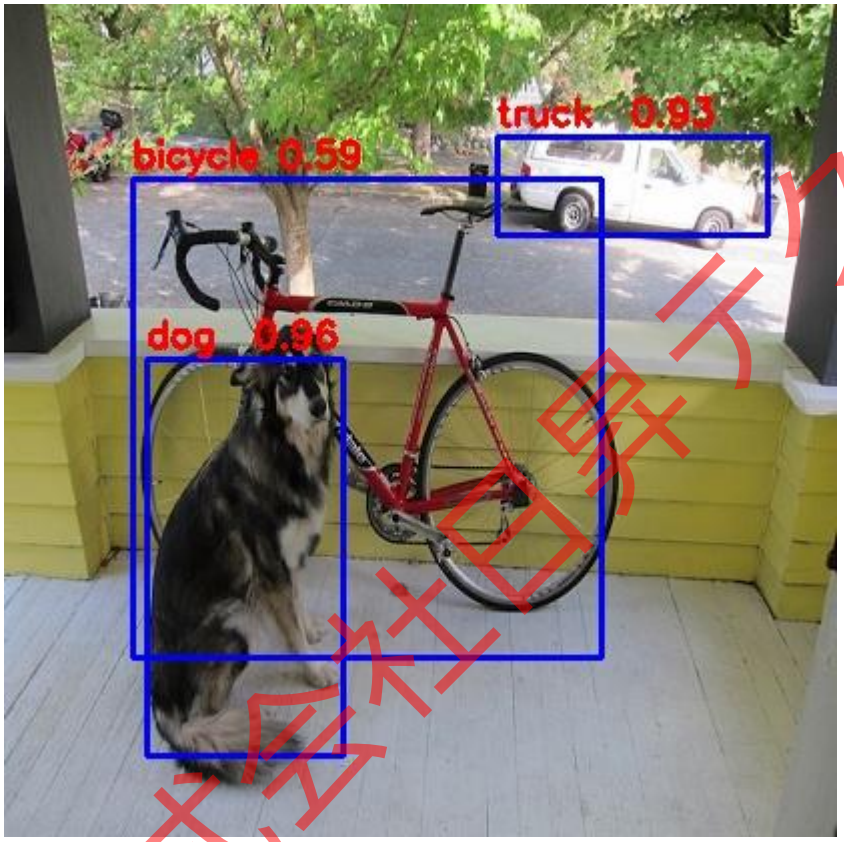
```
        rknn_model = RK3566_RK3568_RKNN_MODEL
    elif host_name == 'RK3588':
        rknn_model = RK3588_RKNN_MODEL
    else:
        print("This demo cannot run on the current platform: {}".format(host_name))
        exit(-1)

    # Create RKNN object
    rknn_lite = RKNNLite()

      # load RKNN model
    print('--> Load RKNN model')
    ret = rknn_lite.load_rknn(rknn_model)
    if ret != 0:
        print('Load RKNN model failed')
        exit(ret)
    print('done')

    # Init runtime environment
    print('--> Init runtime environment')
    # run on RK356x/RK3588 with Debian OS, do not need specify target.
    if host_name == 'RK3588':
        ret = rknn_lite.init_runtime(core_mask=RKNNLite.NPU_CORE_0)
    else:
        ret = rknn_lite.init_runtime()
    if ret != 0:
        print('Init runtime environment failed!')
        exit(ret)
    print('done')

    # Set inputs
    img = cv2.imread(IMG_PATH)
    #img, ratio, (dw, dh) = letterbox(img, new_shape=(IMG_SIZE, IMG_SIZE))
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = cv2.resize(img, (IMG_SIZE, IMG_SIZE))

    # Inference
    print('--> Running model')
    outputs = rknn_lite.inference(inputs=[img])
    np.save('./onnx_yolov3_0.npy', outputs[0])
    np.save('./onnx_yolov3_1.npy', outputs[1])
    np.save('./onnx_yolov3_2.npy', outputs[2])
    print('done')

    # post process
    input0_data = outputs[0]
    input1_data = outputs[1]
    input2_data = outputs[2]

    input0_data = input0_data.reshape(SPAN, LISTSIZE, GRID0, GRID0) # 3*85*13*13
    input1_data = input1_data.reshape(SPAN, LISTSIZE, GRID1, GRID1)
    input2_data = input2_data.reshape(SPAN, LISTSIZE, GRID2, GRID2)

    input_data = []
    input_data.append(np.transpose(input0_data, (2, 3, 0, 1)))  # 13*13*3*85
    input_data.append(np.transpose(input1_data, (2, 3, 0, 1)))
    input_data.append(np.transpose(input2_data, (2, 3, 0, 1)))

    boxes, classes, scores = yolov3_post_process(input_data)

    img = cv2.imread(IMG_PATH)
```

```
    if boxes is not None:
        draw(img, boxes, scores, classes)

    # show output
    print('Save results to results.jpg!')
    cv2.imwrite("result.jpg", img)

    rknn_lite.release()
```

Toolkit Lite2 を使用したボードでのデプロイ推論は、toolkit2 を使用して仮想マシンでシミュレーション推論を行う手順と類似しており、出力後処理のプログラムも同じです。以下は Lubancat-4 ボードでの実行出力です：

```
```

結果画像：



10.5 参考リンク

- https://github.com/ultralytics/yolov3
- https://pjreddie.com/darknet/yolo/
- https://github.com/pjreddie/darknet

第 11 章 花卉画像分類 - TensorFlow

TensorFlow はデータフロープログラミング（dataflow programming）に基づくシンボリック数学システムであり、様々な機械学習（machine learning）アルゴリズムのプログラミング実装に広く使用されています。前身は Google の神経ネットワークアルゴリズムライブラリ DistBelief です。

本章では、TensorFlow の花卉画像分類タスクを簡単に紹介します。tf.keras.Sequential モデルを使用し、モデルを簡単に構築し、最終的に RKNN モデルに変換して LubanCat RK シリーズのボードにデプロイします。

ヒント: テスト環境：LubanCat ボードは Debian11、PC 端は Ubuntu20.04。TensorFlow バージョン 2.8.0、rknn-Toolkit2 バージョン 2.0.0beta。

11.1 tensorflow 環境のインストール

PC で Ubuntu または Windows システムを使用して tensorflow をインストールします。以下の例は Ubuntu20.04 システムです：

Python をインストールし、pip を更新します。Python 3.6-3.9 と pip 19.0 以降を使用する必要があります。

```sh
sudo apt update
sudo apt install python3-dev python3-pip python3-venv
sudo python3 -m pip install pip --upgrade
```

一般的には仮想環境を作成して使用します。

```sh
sudo python3 -m venv .tensorflow_venv
```

最新の安定版 tensorflow を直接インストールするか、バージョンを指定してインストールします（GPU と CPU が統合されています）。

```sh
pip3 install tensorflow
```

詳細なインストール手順と要件は、TensorFlow のチュートリアルを参照してください。

ヒント: Windows システム（Windows 64 ビット OS）では、Python、VC++、Anaconda などをインストールする必要があります。NVIDIA GPU をサポートするには、GPU ドライバー、CUDA、cuDNN をインストールする必要があります。それらのバージョンの対応関係については[こちら]を参照してください。さらに、グラフィックスドライバーと CUDA バージョンの対応については[こちら]を参照してください。オンラインには多数のインストールガイドがありますので、自分のデバイスに合った最新のドキュメントを検索してください。

11.2 画像分類

ここでは、tf.keras.Sequential モデルを使用して花弁画像を分類する簡単な例を紹介します。詳細については[こちら]を参照してください。

11.2.1 データセットの準備

データセットをダウンロードします。3700 枚の画像を含み、tf.keras.utils.image_dataset_from_directory を使用して 80%をトレーニングセット、残り 20%を検証セットとして分割します。

サンプルソース 1:ai/tensorflow/tensorflow_classification.py

```
# 詳しくはマニュアルを参考：https://www.dragonwake.com/download/LubanCat4/1-manual/CSAIEG358803_Python-application-guide.pdf
import numpy as np
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential

# データを取得
import pathlib
dataset_url = "https://storage.googleapis.com/download.tensorflow.org/example_images/flower_photos.tgz"
data_dir = tf.keras.utils.get_file('flower_photos', origin=dataset_url, untar=True)
data_dir = pathlib.Path(data_dir)

# パラメータを設定
batch_size = 32
img_height = 180
img_width = 180

# tf.keras.utils を使用してデータセットを分割し、トレーニングセットと検証セットに分ける
train_ds = tf.keras.utils.image_dataset_from_directory(
  data_dir,
  validation_split=0.2,
  subset="training",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)

val_ds = tf.keras.utils.image_dataset_from_directory(
  data_dir,
  validation_split=0.2,
  subset="validation",
  seed=123,
  image_size=(img_height, img_width),
  batch_size=batch_size)

class_names = train_ds.class_names
#print(class_names)

# データを処理
normalization_layer = layers.Rescaling(1./255)
train_ds = train_ds.map(lambda x, y: (normalization_layer(x), y))
val_ds = val_ds.map(lambda x, y: (normalization_layer(x), y))
num_classes = len(class_names)

......
```

ヒント：TensorFlow の一部の API インターフェースの説明と使用例は[こちら](tensorflow2.8)を参照してください。

11.2.2 モデルの構築と訓練

Keras Sequential モデルの構築：主に 3 つの畳み込みブロック（tf.Keras.layers.Conv2D）で構成され、それぞれの畳み込みブロックには 1 つの最大プーリング層（tf.karas.layers.MaxPooling2D）が含まれています。最後に、1 つの全結合層（tf.keras.layers.Dense）があり、すべて relu が活性化関数として使用されています。

サンプルソース 2:ai/tensorflow/tensorflow_classification.py

```
 ......
# ネットワーク構造を構築し、data_augmentation 前処理レイヤーを追加
data_augmentation = keras.Sequential(
  [
    layers.RandomFlip("horizontal",
                      input_shape=(img_height,
                                   img_width,
                                   3)),
    layers.RandomRotation(0.1),
    layers.RandomZoom(0.1),
  ]
)

# ネットワーク構造を構築
model = Sequential([
  data_augmentation,
  layers.Conv2D(16, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(32, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Conv2D(64, 3, padding='same', activation='relu'),
  layers.MaxPooling2D(),
  layers.Dropout(0.2),# Dropout レイヤーを追加
  layers.Flatten(),
  layers.Dense(128, activation='relu'),
  layers.Dense(num_classes, name="outputs")
])

# オプティマイザー'adam'、損失関数、評価指標を設定
model.compile(optimizer='adam',
              loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'])

# 各層のネットワーク構造を表示
model.summary()

# モデル訓練
epochs=15
history = model.fit(
  train_ds,
  validation_data=val_ds,
  epochs=epochs,
)

......
```

各レイヤーのネットワーク構造を確認

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 sequential (Sequential)     (None, 180, 180, 3)       0

 conv2d (Conv2D)             (None, 180, 180, 16)      448

 max_pooling2d (MaxPooling2  (None, 90, 90, 16)        0
 D)

 conv2d_1 (Conv2D)           (None, 90, 90, 32)        4640

 max_pooling2d_1 (MaxPoolin  (None, 45, 45, 32)        0
 g2D)

 conv2d_2 (Conv2D)           (None, 45, 45, 64)        18496

 max_pooling2d_2 (MaxPoolin  (None, 22, 22, 64)        0
 g2D)

 dropout (Dropout)           (None, 22, 22, 64)        0

 flatten (Flatten)           (None, 30976)             0

 dense (Dense)               (None, 128)               3965056

 outputs (Dense)             (None, 5)                 645

=================================================================
Total params: 3989285 (15.22 MB)
Trainable params: 3989285 (15.22 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
Epoch 1/15
92/92 [==============================] - 23s 236ms/step - loss: 1.4662 - accuracy: 0.3764 -
val_loss: 1.1302 - val_accuracy: 0.5599
Epoch 2/15
92/92 [==============================] - 22s 236ms/step - loss: 1.1277 - accuracy: 0.5494 -
val_loss: 1.1087 - val_accuracy: 0.5708
Epoch 3/15
92/92 [==============================] - 22s 239ms/step - loss: 1.0519 - accuracy: 0.5862 -
val_loss: 0.9931 - val_accuracy: 0.6022
Epoch 4/15
92/92 [==============================] - 22s 233ms/step - loss: 0.9932 - accuracy: 0.6175 -
val_loss: 0.9435 - val_accuracy: 0.6444
Epoch 5/15
92/92 [==============================] - 22s 232ms/step - loss: 0.9220 - accuracy: 0.6478 -
val_loss: 0.9704 - val_accuracy: 0.6308
Epoch 6/15
92/92 [==============================] - 22s 234ms/step - loss: 0.8961 - accuracy: 0.6502 -
```

```
val_loss: 0.8781 - val_accuracy: 0.6526
  Epoch 7/15
  92/92 [==============================] - 21s 232ms/step - loss: 0.8210 - accuracy: 0.6839 -
val_loss: 0.8892 - val_accuracy: 0.6567
  Epoch 8/15
  92/92 [==============================] - 21s 232ms/step - loss: 0.7899 - accuracy: 0.6890 -
val_loss: 0.9213 - val_accuracy: 0.6376
  Epoch 9/15
  92/92 [==============================] - 22s 234ms/step - loss: 0.7597 - accuracy: 0.7108 -
val_loss: 0.8036 - val_accuracy: 0.6853
  Epoch 10/15
  92/92 [==============================] - 22s 233ms/step - loss: 0.7402 - accuracy: 0.7217 -
val_loss: 0.8363 - val_accuracy: 0.6621
  Epoch 11/15
  92/92 [==============================] - 22s 234ms/step - loss: 0.7076 - accuracy: 0.7364 -
val_loss: 0.7853 - val_accuracy: 0.6866
  Epoch 12/15
  92/92 [==============================] - 22s 236ms/step - loss: 0.6889 - accuracy: 0.7333 -
val_loss: 0.8018 - val_accuracy: 0.6798
  Epoch 13/15
  92/92 [==============================] - 22s 240ms/step - loss: 0.6566 - accuracy: 0.7480 -
val_loss: 0.8327 - val_accuracy: 0.6771
  Epoch 14/15
  92/92 [==============================] - 22s 236ms/step - loss: 0.6342 - accuracy: 0.7551 -
val_loss: 0.7567 - val_accuracy: 0.7057
  Epoch 15/15
  92/92 [==============================] - 22s 235ms/step - loss: 0.6194 - accuracy: 0.7650 -
val_loss: 0.7603 - val_accuracy: 0.7125
```

訓練結果から見ると、訓練精度（accuracy）は時間とともに線形に増加し、99%に達しましたが、検証精度（val_accuracy）は約60%にとどまっています。訓練精度と検証精度の差が大きいのは過学習の兆候です。簡単に言うと、このモデルは訓練データ上では他のデータよりも良好な結果を得ることができますが、訓練データ外のデータセットでは同じ効果を得ることができません。これはモデルが新しいデータセットに対して一般化するのが難しいことを意味します。

一般的に、過学習の原因は、データセットのサンプル数が少なすぎる、サンプル中のデータノイズが多すぎる、学習モデルが複雑すぎるなどです。ここでは、訓練データセットが少ないため、モデルがノイズを記憶し、実際の入力出力関係を無視して過度に適応した可能性があります。

データセットの規模を拡大し、Keras の前処理レイヤーでランダムフリップ、回転、ズームを行い、tf.keras.layers.RandomFlip、tf.keras.layers.RandomRotation、tf.keras.layers.RandomZoom を使用します。さらに、Dropout レイヤーを追加し、一定の確率でニューラルネットワークのユニットを一時的にネットワークから除外します。プログラム例は以下の通りです。

```python
# ネットワーク構造を構築し、data_augmentation 前処理レイヤーを追加
data_augmentation = tf.keras.Sequential(
    [
```

```
            tf.keras.layers.RandomFlip("horizontal", input_shape=(img_height, img_width, 3)),
            tf.keras.layers.RandomRotation(0.1),
            tf.keras.layers.RandomZoom(0.1),
        ]
    )

    model = tf.keras.Sequential([
        data_augmentation,  # data_augmentation を追加
        tf.keras.layers.Conv2D(16, 3, padding='same', activation='relu'),
        tf.keras.layers.MaxPooling2D(),
        tf.keras.layers.Conv2D(32, 3, padding='same', activation='relu'),
        tf.keras.layers.MaxPooling2D(),
        tf.keras.layers.Conv2D(64, 3, padding='same', activation='relu'),
        tf.keras.layers.MaxPooling2D(),
        tf.keras.layers.Dropout(0.2),  # Dropout レイヤーを追加
        tf.keras.layers.Flatten(),


        tf.keras.layers.Dense(128, activation='relu'),
        tf.keras.layers.Dense(num_classes, name="outputs")
    ])
```

簡単なテスト結果：

Keras の前処理レイヤーと Dropout レイヤーを追加後、訓練結果が改善されました。

11.2.3 モデルのテスト

サンプルソース 5:ai/tensorflow/tensorflow_classification.py

```
 ......
# モデル評価 画像を取得
sunflower_url = "https://storage.googleapis.com/download.tensorflow.org/example_images/592px-Red_sunflower.jpg"
sunflower_path = tf.keras.utils.get_file('Red_sunflower', origin=sunflower_url)

img = tf.keras.utils.load_img(
    sunflower_path, target_size=(img_height, img_width)
)
img_array = tf.keras.utils.img_to_array(img)
img_array = tf.expand_dims(img_array, 0) # Create a batch

predictions = model.predict(img_array)
score = tf.nn.softmax(predictions[0])

print(
    "This image most likely belongs to {} with a {:.2f} percent confidence."
    .format(class_names[np.argmax(score)], 100 * np.max(score))
)

# Convert the model.
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()
```

```
# Save the model.
with open('model.tflite', 'wb') as f:
  f.write(tflite_model)
```

11.2.4 TensorFlow Lite モデル

訓練された Keras Sequential モデルを使用し、tf.lite.TFLiteConverter.from_keras_model を使用して TensorFlow Lite モデルを生成します。

```python
# モデルを変換
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

# モデルを保存
with open('model.tflite', 'wb') as f:
    f.write(tflite_model)
```

TensorFlow Lite コンバータの変換ワークフローの詳細は[チュートリアル]を参照してください。

11.2.5 モデルの変換とシミュレーションテスト

rknn-Toolkit2 を使用して RKNN モデルをエクスポートします。

サンプルソース 7:ai/tensorflow/rknn_transfer.py

```
# 詳しくはマニュアルを参考：https://www.dragonwake.com/download/LubanCat4/1-manual/CSAIEG358803_Python-application-guide.pdf
import numpy as np
import cv2

# モデルを変換
from rknn.api import RKNN
import tensorflow as tf

img_height = 180
img_width = 180
IMG_PATH = 'test.jpg'
class_names = ['daisy', 'dandelion', 'roses', 'sunflowers', 'tulips']

if __name__ == '__main__':

    # RKNN オブジェクトを作成
    #rknn = RKNN(verbose='Debug')
    rknn = RKNN()

    # 前処理の設定
    print('--> Config model')
    rknn.config(mean_values=[0, 0, 0], std_values=[255, 255, 255], target_platform='rk3588')
```

```
    print('done')

    # モデルをロード
    print('--> Loading model')
    ret = rknn.load_tflite(model='model.tflite')
    if ret != 0:
        print('Load model failed!')
        exit(ret)
    print('done')

    # モデルをビルド
    print('--> Building model')
    ret = rknn.build(do_quantization=False)
    #ret = rknn.build(do_quantization=True,dataset='./dataset.txt')
    if ret != 0:
        print('Build model failed!')
        exit(ret)
    print('done')

    # Export rknn model
    print('--> Export rknn model')
    ret = rknn.export_rknn('./model.rknn')
    if ret != 0:
        print('Export rknn model failed!')
        exit(ret)
    print('done')


#ランタイム環境を初期化
print('--> Init runtime environment')
ret = rknn.init_runtime()
#    if ret != 0:
#        print('Init runtime environment failed!')
#        exit(ret)
print('done')

# 画像
img = cv2.imread(IMG_PATH)
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
img = cv2.resize(img,(180,180))
img = np.expand_dims(img, 0)

#print('--> Accuracy analysis')
#rknn.accuracy_analysis(inputs=['./test.jpg'])
#print('done')

# 推論
print('--> Running model')
outputs = rknn.inference(inputs=[img])
print("結果: ", outputs)
outputs = tf.nn.softmax(outputs)
print(outputs)

print(
    "This image most likely belongs to {} with a {:.2f} percent confidence."
    .format(class_names[np.argmax(outputs)], 100 * np.max(outputs))
)
print("画像予測は:", class_names[np.argmax(outputs)])
print('--> done')

rknn.release()
```

```
CPU を使いますので、CUDA に関連する警告メッセージを無視してください。ここ削除して表示
I rknn-toolkit2 version: 2.0.0b0+9bab5682
--> Config model
done
--> Loading model
INFO: Created TensorFlow Lite XNNPACK delegate for CPU.
I Loading : 100%|██████████████████████████████████████████████| 11/11 [00:00<00:00, 2518.96it/s]
done
--> Building model
I rknn building ...
I rknn buiding done.
done
--> Export rknn model
done
--> Init runtime environment
I Target is None, use simulator!
done
--> Running model
W inference: The 'data_format' is not set, and its default value is 'nhwc'!
I GraphPreparing : 100%|██████████████████████████████████████| 20/20 [00:00<00:00, 5900.41it/s]
I SessionPreparing : 100%|████████████████████████████████████| 20/20 [00:00<00:00, 898.47it/s]
結果:  [array([[-2.0957031, -2.546875 ,  1.5195312,  6.2109375,  2.3046875]],
      dtype=float32)]
tf.Tensor([[[2.3975283e-04 1.5269412e-04 8.9092273e-03 9.7116256e-01 1.9535707e-02]]], shape=(1, 1, 5), dtype=float32)
This image most likely belongs to sunflowers with a 97.12 percent confidence.
画像予測は: sunflowers
--> done
```

11.3 デプロイと推論テスト

LubanCat ボードにデプロイして推論を実行し、rknn-toolkit-lite2 を使用します。以下は RK3588 の例です。

```python
# 詳しくはマニュアルを参考：https://www.dragonwake.com/download/LubanCat4/1-manual/CSAIEG358803_Python-application-guide.pdf
import numpy as np
import cv2
from rknnlite.api import RKNNLite

IMG_PATH = 'test.jpg'
RKNN_MODEL = 'model.rknn'
img_height = 180
img_width = 180
class_names = ['daisy', 'dandelion', 'roses', 'sunflowers', 'tulips']


# Create RKNN object
```

```
rknn_lite = RKNNLite()

# load RKNN model
print('--> Load RKNN model')
ret = rknn_lite.load_rknn(RKNN_MODEL)
if ret != 0:
    print('Load RKNN model failed')
    exit(ret)
print('done')

# Init runtime environment
print('--> Init runtime environment')
ret = rknn_lite.init_runtime()
if ret != 0:
    print('Init runtime environment failed!')
    exit(ret)
print('done')

# load image
img = cv2.imread(IMG_PATH)
img = cv2.resize(img,(180,180))
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
img = np.expand_dims(img, 0)

# runing model
print('--> Running model')
outputs = rknn_lite.inference(inputs=[img])
print("result: ", outputs)
print(
    "This image most likely belongs to {}."
    .format(class_names[np.argmax(outputs)])
)

rknn_lite.release()
```

サンプルソース 7:ai/tensorflow/rknnlite_inference.py

基板上に実行した結果：（PC 側とのテスト結果と同じ）

```
cat@lubancat:~/python/code/ai/tensorflow$ python3 rknnlite_inference.py
--> Load RKNN model
done
--> Init runtime environment
I RKNN: [23:21:04.675] RKNN Runtime Information, librknnrt version: 2.0.0b0
(35a6907d79@2024-03-24T10:31:14)
I RKNN: [23:21:04.675] RKNN Driver Information, version: 0.9.2
I RKNN: [23:21:04.675] RKNN Model Information, version: 6, toolkit version:
2.0.0b0+9bab5682(compiler version: 2.0.0b0 (35a6907d79@2024-03-24T02:34:11)), target: RKNPU v2,
target platform: rk3588, framework name: TFLite, framework layout: NHWC, model inference type:
static_shape
done
--> Running model
result:  [array([[-2.0957031, -2.5449219,  1.5205078,  6.2109375,  2.3046875]],
      dtype=float32)]
This image most likely belongs to sunflowers.
```

11.4 まとめ

TensorFlow を使用して簡単な花卉画像分類を行い、モデルを RKNN モデルに変換し、LubanCat ボードにデプロイしました。モデルの検証精度は約 70%であり、シンプルな学習に適しています。このモデルのテストは TensorFlow 公式の画像分類チュートリアルからのものです。

11.5 参考リンク

- [TensorFlow 学習](https://www.tensorflow.org/learn?hl=ja)
- [画像分類チュートリアル](https://www.tensorflow.org/tutorials/images/classification)
- [TensorFlow チュートリアル](https://www.tensorflow.org/tutorials?hl=ja)

第 12 章 ResNet18 ネットワーク-PyTorch

ResNet18 は 18 層の深さを持つ畳み込みニューラルネットワークで、重みを持つ畳み込み層と全結合層が含まれます。これは、残差接続（residual connection）を使用して深いネットワークの劣化問題を解決する ResNet シリーズの変種です。

この章では、PyTorch とそのインストール環境について簡単に紹介し、ResNet ニューラルネットワークの簡単な分析と PyTorch のソースコード実装について説明します。最後に、PyTorch を使用して簡単に ResNet18 ネットワークを構築し、Cifar-10 データセットを分類して、Lubancat ボードにデプロイします。

ヒント: テスト環境は、Lubancat ボードで Debian11、PC ubuntu20.04 です。PyTorch は 2.1.0 GPU バージョンで、rknn-Toolkit2 のバージョンは 2.0.0beta です。

12.1 PyTorch と ResNet18

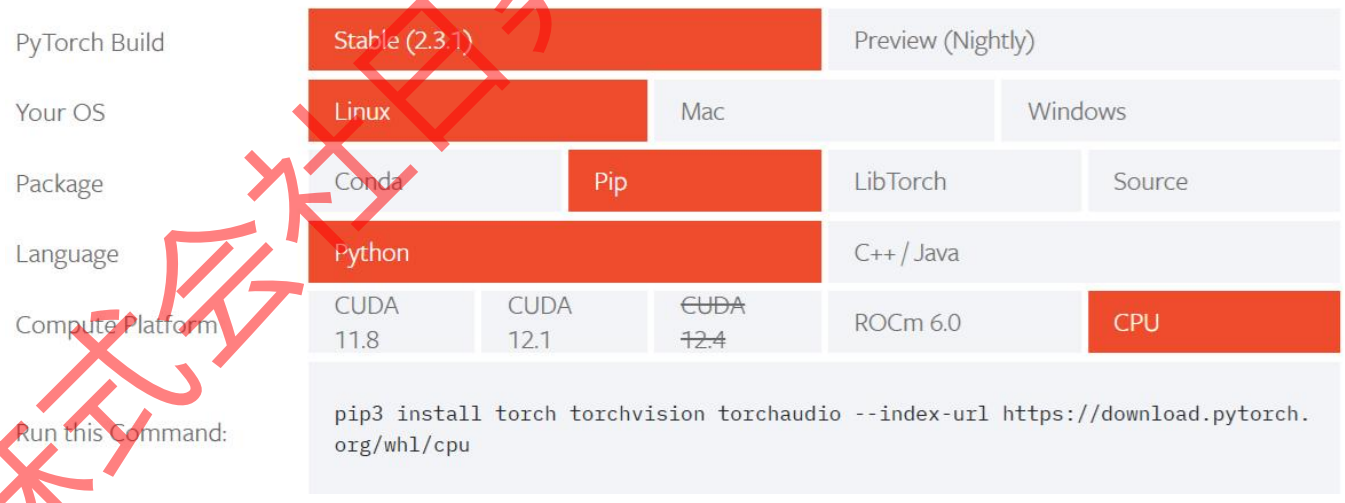
PyTorch は、Facebook 人工知能研究所の Torch7 チームによって開発されたオープンソースのディープラーニングフレームワークです。このフレームワークの基礎は Torch に基づいていますが、実装と使用はすべて Python で行われます。このフレームワークは主に人工知能分野の科学研究と応用開発に使用されます。

12.1.1 PyTorch のインストール

PyTorch は自分の環境に応じてインストールする必要があります。PyTorch 公式サイトにアクセスして詳細なインストール手順を確認してください。以下の例では、ubuntu20.04 で簡単にインストールします：

まず、PyTorch 公式サイトにアクセスし、対応する環境（例：Linux システム、Python 言語、CPU バージョンの PyTorch）を選択します。

Start Locally | PyTorch から選択しましょう。



このページの上部には最新の安定版の PyTorch がデフォルトで表示されています。以前のバージョンをインストールする場合は、このページの下部にある Previous version of PyTorch または直接[こちら](https://pytorch.org/get-started/previous-versions/)をクリックします。

PyTorch を使用するには、まず python3 と pip の基本環境をインストールする必要があります。これらについては検索して調べてください。

CPU バージョンのインストール：

```bash
```

```
pip3 install torch torchvision torchaudio --index-url
https://download.pytorch.org/whl/cpu
```

GPU バージョン（CUDA 12.1）のインストール：

```bash
pip3 install torch torchvision torchaudio --index-url
https://download.pytorch.org/whl/cu121
```

GPU バージョン（NVIDIA GPU）のインストールには、まず自分の GPU に対応した[ドライバー](https://www.nvidia.co.jp/Download/index.aspx?lang=jp)をインストールまたは更新し、[CUDA ツールキット](https://developer.nvidia.com/cuda-toolkit)をインストールし、cuDNN も必要に応じてインストールしてください。次に、[こちら](https://docs.nvidia.com/deeplearning/cudnn/install-guide/index.html)をクリックして Pytorch と CUDA のバージョン対応を確認します。

インストールが成功したかどうかを確認する：

インストールテスト、ターミナルに python と入力し、

```python
>>> import torch
>>> torch.__version__
'2.1.0+cu121'
```

CUDA バージョンの PyTorch をインストールした場合、PyTorch のバージョンと CUDA のバージョンを確認するためのコマンド

```python
>>> torch.version.cuda
'12.1'
>>> torch.cuda.is_available()
True
```

PyTorch のインストールは、自分の実際の環境とニーズに応じて、docker などの環境を使用することもできます。

プログラム編集ツールについては、Sublime Text、PyCharm、Vim などを使用できます。ここでは ubuntu20.04 を使用してテスト環境を設定し、編集ツールとして Jupyter Notebook を使用します。インストール手順は[こちら](https://jupyter.org/install)を参照してください。Linux システムでも同様に使用できます。

13.1.2 ResNet18 の構造概要

ResNet（Residual Neural Network）は、2015 年に Microsoft Research の Kaiming He らによって提案されたもので、ResNet の構造はニューラルネットワークのトレーニングを非常に高速化し、モデルの精度も大幅に向上させます。

ResNet は残差ネットワークであり、それをサブネットワークとして理解することができます。このサブネットワークを積み重ねることで非常に深いネットワークを構築できます。ResNet シリーズには ResNet18、ResNet34、ResNet50、ResNet101、および ResNet152 などのさまざまな変種があります。そのネットワーク構造は以下の通りです（参考文献：[論文](https://arxiv.org/abs/1512.03385)）。

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2 [3×3, 64 3×3, 64] ×2	[3×3, 64 3×3, 64] ×3	[1×1, 64 3×3, 64 1×1, 256] ×3	[1×1, 64 3×3, 64 1×1, 256] ×3	[1×1, 64 3×3, 64 1×1, 256] ×3
conv3_x	28×28	[3×3, 128 3×3, 128] ×2	[3×3, 128 3×3, 128] ×4	[1×1, 128 3×3, 128 1×1, 512] ×4	[1×1, 128 3×3, 128 1×1, 512] ×4	[1×1, 128 3×3, 128 1×1, 512] ×8
conv4_x	14×14	[3×3, 256 3×3, 256] ×2	[3×3, 256 3×3, 256] ×6	[1×1, 256 3×3, 256 1×1, 1024] ×6	[1×1, 256 3×3, 256 1×1, 1024] ×23	[1×1, 256 3×3, 256 1×1, 1024] ×36
conv5_x	7×7	[3×3, 512 3×3, 512] ×2	[3×3, 512 3×3, 512] ×3	[1×1, 512 3×3, 512 1×1, 2048] ×3	[1×1, 512 3×3, 512 1×1, 2048] ×3	[1×1, 512 3×3, 512 1×1, 2048] ×3
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

ここでは ResNet18 に注目します。ResNet18 の基本的な意味は、ネットワークの基本構造が ResNet であり、ネットワークの深さが 18 層であることです。これは BN 層やプーリング層を含まない、重みを持つ 18 層を指します。ResNet18 は基本的な残差ユニットを使用しており、各ユニットは 2 つの 3x3 畳み込み層で構成され、その間に BN 層と ReLU 活性化関数があります。

13.1.3 PyTorch での ResNet18 の実装

PyTorch での ResNet18 のソースコード実装：
https://github.com/pytorch/vision/blob/main/torchvision/models/resnet.py

12.2 ResNet18 の実装

12.2.1 データセットの準備とデータ前処理

次に、ResNet18 ネットワーク構造をカスタマイズし、CIFAR-10 データセットを使用して簡単にテストします。CIFAR-10 データセットは、10 カテゴリの 60000 枚の 32x32 カラーフォトで構成されており、各カテゴリには 6000 枚の画像があります。合計で 50000 枚のトレーニング画像と 10000 枚のテスト画像があります。

サンプルソースコード 1：resnet18.py（部分的な抜粋、参考用）

```
num_classes = 10
batch_size = 128
learning_rate = 0.001
num_epoches = 100
classes = ('plane', 'car', 'bird', 'cat', 'deer', 'dog', 'frog', 'horse', 'ship', 'truck')

# download=True パラメータは、インターネットからデータをダウンロードし、./data ディレクトリに保存することを示します。自分でダウンロードして指定のディレクトリに保存することもできます。
train_dataset = torchvision.datasets.CIFAR10('./data', download=True, train=True, transform=transform_train)
test_dataset = torchvision.datasets.CIFAR10('./data', download=True, train=False, transform=transform_test)



# ダウンロードしたデータセットをインポートし、torchvision を使用してトレーニングセットとテストセットをロードします。
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=batch_size, shuffle=False)
```

データセットを分割して前処理を行います

サンプルソースコード 2：resnet18.py（部分的な抜粋、参考用）

```
# データ増強と変換設定
transform_train = torchvision.transforms.Compose([
        torchvision.transforms.Pad(4),
        torchvision.transforms.RandomHorizontalFlip(),  # 画像を 50%の確率で水平反転します
        torchvision.transforms.RandomCrop(32),  # 画像をランダムに 32x32 にトリミングします
        torchvision.transforms.ToTensor(),  # Tensor に変換して正規化します
        torchvision.transforms.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])  # 正規化に使用する平均と標準偏差
])

transform_test = torchvision.transforms.Compose([
        torchvision.transforms.ToTensor(),
        torchvision.transforms.Normalize([0.5, 0.5, 0.5], [0.5, 0.5, 0.5])  # 正規化に使用する平均と標準偏差
])
```

12.2.2 モデルの構築

サンプルソースコード 3：resnet18.py（部分的な抜粋、参考用）

```
# 残差ブロックの実装
class ResidualBlock(nn.Module):
    def __init__(self, in_channels, out_channels, stride=1, downsample=None):
        super(ResidualBlock, self).__init__()
        self.conv1 = conv3x3(in_channels, out_channels, stride)
        self.bn1 = nn.BatchNorm2d(out_channels)
        self.relu = nn.ReLU(inplace=True)
        self.conv2 = conv3x3(out_channels, out_channels)
        self.bn2 = nn.BatchNorm2d(out_channels)
        self.downsample = downsample

    def forward(self, x):
        residual = x
        out = self.conv1(x)
        out = self.bn1(out)
        out = self.relu(out)
        out = self.conv2(out)
        out = self.bn2(out)
        if self.downsample:
            residual = self.downsample(x)
        out += residual
        out = self.relu(out)
        return out

# カスタムニューラルネットワークを定義し、nn.Module を使用して各層を初期化します。
# forward を使用してデータを接続
class ResNet(nn.Module):
    def __init__(self, block, layers, num_classes):
        super(ResNet, self).__init__()
        self.in_channels = 16
        self.conv = conv3x3(3, 16)
        self.bn = nn.BatchNorm2d(16)
        self.relu = nn.ReLU(inplace=True)
        self.layer1 = self._make_layers(block, 16, layers[0])
        self.layer2 = self._make_layers(block, 32, layers[1], 2)
        self.layer3 = self._make_layers(block, 64, layers[2], 2)
        self.layer4 = self._make_layers(block, 128, layers[3], 2)
        self.avg_pool = nn.AdaptiveAvgPool2d((1, 1))
        self.fc = nn.Linear(128, num_classes)
```

```
    # _make_layers 関数は、残差ブロックとショートカット部分を繰り返します
    def _make_layers(self, block, out_channels, blocks, stride=1):
        downsample = None
        # 畳み込みカーネルは 1 を使用してチャンネル数を増減します
        if (stride != 1) or (self.in_channels != out_channels):
            downsample = nn.Sequential(
                conv3x3(self.in_channels, out_channels, stride=stride),
                nn.BatchNorm2d(out_channels)
            )
        layers = []
        layers.append(block(self.in_channels, out_channels, stride, downsample))
        self.in_channels = out_channels
        for _ in range(1, blocks):
            layers.append(block(out_channels, out_channels))
        return nn.Sequential(*layers)

    def forward(self, x):
        out = self.conv(x)
        out = self.bn(out)
        out = self.relu(out)
        out = self.layer1(out)
        out = self.layer2(out)
        out = self.layer3(out)
        out = self.layer4(out)
        out = self.avg_pool(out)
        out = out.view(out.size(0), -1)
        out = self.fc(out)
        return out

# モデルを作成し、指定されたデバイスに移動します
model=ResNet(ResidualBlock, [2, 2, 2, 2], num_classes)
model.to(device=device)

# モデル構造を表示します
print(f"Model structure: {model}¥n¥n")

# 損失関数と最適化関数
criterion = nn.CrossEntropyLoss()  # クロスエントロピー損失関数
optimizer = torch.optim.Adam(model.parameters(), lr=learning_rate)  # 最適化アルゴリズムとしてアダムを使用
```

モデル構造をテストすると以下のようになります：

```
Model structure: ResNet(
  (conv): Conv2d(3, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
  (bn): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
  (relu): ReLU(inplace=True)
  (layer1): Sequential(
    (0): ResidualBlock(
      (conv1): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
    (1): ResidualBlock(
      (conv1): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(16, 16, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(16, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer2): Sequential(
    (0): ResidualBlock(
```

```
      (conv1): Conv2d(16, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(16, 32, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
        (1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): ResidualBlock(
      (conv1): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(32, 32, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(32, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer3): Sequential(
    (0): ResidualBlock(
      (conv1): Conv2d(32, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(32, 64, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
        (1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): ResidualBlock(
      (conv1): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(64, 64, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(64, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (layer4): Sequential(
    (0): ResidualBlock(
      (conv1): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (downsample): Sequential(
        (0): Conv2d(64, 128, kernel_size=(3, 3), stride=(2, 2), padding=(1, 1), bias=False)
        (1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      )
    )
    (1): ResidualBlock(
      (conv1): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn1): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
      (relu): ReLU(inplace=True)
      (conv2): Conv2d(128, 128, kernel_size=(3, 3), stride=(1, 1), padding=(1, 1), bias=False)
      (bn2): BatchNorm2d(128, eps=1e-05, momentum=0.1, affine=True, track_running_stats=True)
    )
  )
  (avg_pool): AdaptiveAvgPool2d(output_size=(1, 1))
  (fc): Linear(in_features=128, out_features=10, bias=True)
)
```

12.2.3 モデルのトレーニングとテスト

サンプルソースコード 4：resnet18.py（部分的な抜粋、参考用）

```
if __name__ == "__main__":
    # モデルをトレーニングします
    total_step = len(train_loader)
    for epoch in range(num_epoches):
        for i, (images, labels) in enumerate(train_loader):
            images = images.to(device=device)
            labels = labels.to(device=device)

            # 順伝播
            outputs = model(images)
            loss = criterion(outputs, labels)

            # 勾配をゼロにリセット
            optimizer.zero_grad()

            # 逆伝播
            loss.backward()

            # パラメータを更新
            optimizer.step()

            if (i + 1) % total_step == 0:
                print(f'Epoch [{epoch+1}/{num_epoches}], Step [{i+1}/{total_step}], Loss: {loss.item():.4f}')

    print("トレーニング完了")

    # モデルを保存
    # torch.save(model.state_dict(), 'model_weights.pth')
    # torch.save(model, 'model.pt')

    print('¥n モデルをテスト')
    model.eval()
    with torch.no_grad():
        correct = 0
        total = 0
        for images, labels in test_loader:
            images = images.to(device=device)
            labels = labels.to(device=device)
            outputs = model(images)
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
        print(f'10000 枚のテスト画像における精度: {100 * correct / total:.4f} %')
```


トレーニング後、テストセットでのモデルの精度は 89.07%に達します：

```
Epoch [1/100], Step [391/391], Loss: 1.3068
Epoch [2/100], Step [391/391], Loss: 0.9546
Epoch [3/100], Step [391/391], Loss: 0.6552
Epoch [4/100], Step [391/391], Loss: 0.6923
Epoch [5/100], Step [391/391], Loss: 0.7940
Epoch [6/100], Step [391/391], Loss: 0.4908
Epoch [7/100], Step [391/391], Loss: 0.4688
Epoch [8/100], Step [391/391], Loss: 0.3866
Epoch [9/100], Step [391/391], Loss: 0.4445
Epoch [10/100], Step [391/391], Loss: 0.4034
Epoch [11/100], Step [391/391], Loss: 0.4703
Epoch [12/100], Step [391/391], Loss: 0.3719
Epoch [13/100], Step [391/391], Loss: 0.5816
Epoch [14/100], Step [391/391], Loss: 0.2484
Epoch [15/100], Step [391/391], Loss: 0.3022
Epoch [16/100], Step [391/391], Loss: 0.4372
Epoch [17/100], Step [391/391], Loss: 0.3407
Epoch [18/100], Step [391/391], Loss: 0.4066
Epoch [19/100], Step [391/391], Loss: 0.2191
Epoch [20/100], Step [391/391], Loss: 0.3055
Epoch [21/100], Step [391/391], Loss: 0.3021
Epoch [22/100], Step [391/391], Loss: 0.1852
Epoch [23/100], Step [391/391], Loss: 0.2088
Epoch [24/100], Step [391/391], Loss: 0.3134
Epoch [25/100], Step [391/391], Loss: 0.3368
Epoch [26/100], Step [391/391], Loss: 0.1642
Epoch [27/100], Step [391/391], Loss: 0.1389
Epoch [28/100], Step [391/391], Loss: 0.3040
Epoch [29/100], Step [391/391], Loss: 0.2121
Epoch [30/100], Step [391/391], Loss: 0.4610
Epoch [31/100], Step [391/391], Loss: 0.3345
Epoch [32/100], Step [391/391], Loss: 0.1944
Epoch [33/100], Step [391/391], Loss: 0.2196
Epoch [34/100], Step [391/391], Loss: 0.1026
Epoch [35/100], Step [391/391], Loss: 0.1807
Epoch [36/100], Step [391/391], Loss: 0.1494
Epoch [37/100], Step [391/391], Loss: 0.1087
Epoch [38/100], Step [391/391], Loss: 0.2177
Epoch [39/100], Step [391/391], Loss: 0.1868
Epoch [40/100], Step [391/391], Loss: 0.1669
Epoch [41/100], Step [391/391], Loss: 0.2563
Epoch [42/100], Step [391/391], Loss: 0.1976
Epoch [43/100], Step [391/391], Loss: 0.2419
Epoch [44/100], Step [391/391], Loss: 0.3852
Epoch [45/100], Step [391/391], Loss: 0.0860
Epoch [46/100], Step [391/391], Loss: 0.3003
Epoch [47/100], Step [391/391], Loss: 0.2969
Epoch [48/100], Step [391/391], Loss: 0.1625
Epoch [49/100], Step [391/391], Loss: 0.0740
Epoch [50/100], Step [391/391], Loss: 0.0629
Epoch [51/100], Step [391/391], Loss: 0.0580
Epoch [52/100], Step [391/391], Loss: 0.1061
Epoch [53/100], Step [391/391], Loss: 0.1466
Epoch [54/100], Step [391/391], Loss: 0.0626
Epoch [55/100], Step [391/391], Loss: 0.1082
Epoch [56/100], Step [391/391], Loss: 0.2024
Epoch [57/100], Step [391/391], Loss: 0.1341
Epoch [58/100], Step [391/391], Loss: 0.0409
Epoch [59/100], Step [391/391], Loss: 0.0538
Epoch [60/100], Step [391/391], Loss: 0.1115
Epoch [61/100], Step [391/391], Loss: 0.0313
Epoch [62/100], Step [391/391], Loss: 0.0882
Epoch [63/100], Step [391/391], Loss: 0.1396
Epoch [64/100], Step [391/391], Loss: 0.0596
Epoch [65/100], Step [391/391], Loss: 0.0694
Epoch [66/100], Step [391/391], Loss: 0.1808
Epoch [67/100], Step [391/391], Loss: 0.0448
Epoch [68/100], Step [391/391], Loss: 0.0492
Epoch [69/100], Step [391/391], Loss: 0.1246
Epoch [70/100], Step [391/391], Loss: 0.0527
Epoch [71/100], Step [391/391], Loss: 0.0927
```

```
Epoch [72/100], Step [391/391], Loss: 0.0823
Epoch [73/100], Step [391/391], Loss: 0.1437
Epoch [74/100], Step [391/391], Loss: 0.0485
Epoch [75/100], Step [391/391], Loss: 0.0466
Epoch [76/100], Step [391/391], Loss: 0.0260
Epoch [77/100], Step [391/391], Loss: 0.0380
Epoch [78/100], Step [391/391], Loss: 0.0531
Epoch [79/100], Step [391/391], Loss: 0.0462
Epoch [80/100], Step [391/391], Loss: 0.1365
Epoch [81/100], Step [391/391], Loss: 0.0698
Epoch [82/100], Step [391/391], Loss: 0.0752
Epoch [83/100], Step [391/391], Loss: 0.1638
Epoch [84/100], Step [391/391], Loss: 0.0812
Epoch [85/100], Step [391/391], Loss: 0.1274
Epoch [86/100], Step [391/391], Loss: 0.0750
Epoch [87/100], Step [391/391], Loss: 0.0153
Epoch [88/100], Step [391/391], Loss: 0.1163
Epoch [89/100], Step [391/391], Loss: 0.0661
Epoch [90/100], Step [391/391], Loss: 0.0459
Epoch [91/100], Step [391/391], Loss: 0.0519
Epoch [92/100], Step [391/391], Loss: 0.0275
Epoch [93/100], Step [391/391], Loss: 0.1451
Epoch [94/100], Step [391/391], Loss: 0.0110
Epoch [95/100], Step [391/391], Loss: 0.0541
Epoch [96/100], Step [391/391], Loss: 0.1009
Epoch [97/100], Step [391/391], Loss: 0.0511
Epoch [98/100], Step [391/391], Loss: 0.0700
Epoch [99/100], Step [391/391], Loss: 0.1173
Epoch [100/100], Step [391/391], Loss: 0.0289
トレーニング完了

モデルをテスト
10000 枚のテスト画像における精度: 89.0700 %
```

12.2.4 ONNX モデルとして保存

ここでは torch.onnx.export を使用してモデルを ONNX 形式で保存します（pt 形式でもエクスポート可能）。

サンプルソースコード 5：resnet18.py（部分的な抜粋、参考用）

```
    # ONNX モデルとしてエクスポート (rknn-toolkit2 は opset_version=12 までサポート)
    x = torch.randn((1, 3, 32, 32)).to(device=device)
    model.to(device=device)
    torch.onnx.export(model, x, './resnet18.onnx', opset_version=12, input_names=['input'], output_names=['output'])
```

12.2.5 RKNN モデルのエクスポートとシミュレーションテスト

サンプルソースコード 6：rknn_transfer.py

```
# 詳しくはマニュアルを参考：https://www.dragonwake.com/download/LubanCat4/1-manual/CSAIEG358803_Python-application-guide.pdf
import numpy as np
import cv2
from rknn.api import RKNN
import torchvision.models as models
import torch
import os

def softmax(x):
```

```
    return np.exp(x)/sum(np.exp(x))

def torch_version():
    import torch
    torch_ver = torch.__version__.split('.')
    torch_ver[2] = torch_ver[2].split('+')[0]
    return [int(v) for v in torch_ver]

if __name__ == '__main__':

    if torch_version() < [1, 9, 0]:
        import torch
        print("Your torch version is '{}', in order to better support the Quantization Aware Training (QAT) model,¥n"
              "Please update the torch version to '1.9.0' or higher!".format(torch.__version__))
        exit(0)

    MODEL = './resnet18.onnx'

    # Create RKNN object
    rknn = RKNN(verbose=True)

    # Pre-process config
    print('--> Config model')
    rknn.config(mean_values=[127.5, 127.5, 127.5], std_values=[255, 255, 255], target_platform='rk3568')
    #rknn.config(mean_values=[123.675, 116.28, 103.53], std_values=[58.395, 58.395, 58.395], target_platform='rk3568')
    #rknn.config(mean_values=[125.307, 122.961, 113.8575], std_values=[51.5865, 50.847, 51.255], target_platform='rk3568')
    print('done')

    # Load model
    print('--> Loading model')
    #ret = rknn.load_pytorch(model=model, input_size_list=input_size_list)
    ret = rknn.load_onnx(model=MODEL)
    if ret != 0:
        print('Load model failed!')
        exit(ret)
    print('done')

    # Build model
    print('--> Building model')
    ret = rknn.build(do_quantization=False)
    if ret != 0:
        print('Build model failed!')
        exit(ret)
    print('done')

    # Export rknn model
    print('--> Export rknn model')
    ret = rknn.export_rknn('./resnet_18_100.rknn')
    if ret != 0:
        print('Export rknn model failed!')
        exit(ret)
    print('done')

    # Set inputs
    img = cv2.imread('./0_125.jpg')
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = cv2.resize(img,(32,32))
    #img = np.expand_dims(img, 0)

    # Init runtime environment
    print('--> Init runtime environment')
    ret = rknn.init_runtime()
    if ret != 0:
        print('Init runtime environment failed!')
        exit(ret)
    print('done')

    # Inference
    print('--> Running model')
    outputs = rknn.inference(inputs=[img])
```

```
    np.save('./pytorch_resnet18_qat_0.npy', outputs[0])
    #show_outputs(softmax(np.array(outputs[0][0])))
    print(outputs)
    print('done')


    rknn.release()
```

12.2.6 基板上でのデプロイとテスト

12.2.6.1 簡単なテスト

サンプルソースコード 7: rknnlite_inference0.py

```
# 詳しくはマニュアルを参考：https://www.dragonwake.com/download/LubanCat4/1-manual/CSAIEG358803_Python-application-guide.pdf
import numpy as np
import cv2
import os
from rknnlite.api import RKNNLite

IMG_PATH = '2_67.jpg'
RKNN_MODEL = './resnet18.rknn'
img_height = 32
img_width = 32
class_names = ["plane","car","bird","cat","deer","dog","frog","horse","ship","truck"]


# Create RKNN object
rknn_lite = RKNNLite()


# load RKNN model
print('--> Load RKNN model')
ret = rknn_lite.load_rknn(RKNN_MODEL)
if ret != 0:
    print('Load RKNN model failed')
    exit(ret)
print('done')


# Init runtime environment
print('--> Init runtime environment')
ret = rknn_lite.init_runtime()
if ret != 0:
    print('Init runtime environment failed!')
    exit(ret)
print('done')


# load image
img = cv2.imread(IMG_PATH)
img = cv2.resize(img,(32,32))
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
img = np.expand_dims(img, 0)



# runing model
print('--> Running model')
outputs = rknn_lite.inference(inputs=[img])
print("result: ", outputs)
print(
    "This image most likely belongs to {}."
    .format(class_names[np.argmax(outputs)])
)
```

```
rknn_lite.release()
```

デバイス上でのテスト結果：

```
--> Load RKNN model
done
--> Init runtime environment
I RKNN: [02:04:53.190] RKNN Runtime Information, librknnrt version: 2.0.0b0 (35a6907d79@2024-03-24T10:31:14)
I RKNN: [02:04:53.190] RKNN Driver Information, version: 0.9.2
I RKNN: [02:04:53.191] RKNN Model Information, version: 6, toolkit version: 2.0.0b0+9bab5682(compiler version: 2.0.0b0 (35a6907d79@2024-03-24T02:34:11)), target: RKNPU v2, target platform: rk3588, framework name: ONNX, framework layout: NCHW, model inference type: static_shape
done
--> Running model
result:  [array([[ -5.2148438, -16.109375 ,   8.390625 ,  -7.9804688, -11.2734375,
        -21.734375 ,  -6.8242188, -13.296875 ,  -4.890625 , -17.359375 ]],
      dtype=float32)]
This image most likely belongs to bird.
```

12.2.6.2 テストセット画像を使用したテスト

テストセットを.jpg形式の画像に変換し、デバイスに転送してデプロイとテストを行います：

*pip install imageio

サンプルソースコード8：cifar10_to_jpg.py

```
"""
    cifar10 の test_batch を .jpg 形式の画像に変換し、各クラスを個別のフォルダに保存します。フォルダ名は 0-9 です。
"""
from imageio import imwrite
import numpy as np
import os
import pickle

# CIFAR-10 データセットの絶対パス。自分の実際のディレクトリに合わせて設定してください。
base_dir = "/home/csun/linuxshare/work/AI/jupyter/python_lubancat_RK_tutorial_code/ai/pytorch/resnet18_pytorch/"

data_dir = os.path.join(base_dir, "data", "cifar-10-batches-py")
test_o_dir = os.path.join(base_dir, "Data", "cifar-10-png", "raw_test")


# 解凍
def unpickle(file):
    with open(file, 'rb') as fo:
        dict_ = pickle.load(fo, encoding='bytes')
    return dict_


# テスト用画像の生成
```

```
if __name__ == '__main__':
    print("開始...")
    test_data_path = os.path.join(data_dir, "test_batch")
    test_data = unpickle(test_data_path)
    for i in range(0, 10000):
        img = np.reshape(test_data[b'data'][i], (3, 32, 32))
        img = img.transpose(1, 2, 0)

        label_num = str(test_data[b'labels'][i])
        o_dir = os.path.join(test_o_dir, label_num)
        if not os.path.isdir(o_dir):
            os.makedirs(o_dir)

        img_name = label_num + '_' + str(i) + '.jpg'
        img_path = os.path.join(o_dir, img_name)
        imwrite(img_path, img)
    print("完了。")
```

PC 側に上記プログラムでテストセットを.jpg 形式に変換
基板側実行のプログラムを修正
サンプルソースコード 9：rknnlite_inference1.py

```
# 詳しくはマニュアルを参考：https://www.dragonwake.com/download/LubanCat4/1-manual/CSAIEG358803_Python-application-guide.pdf
import numpy as np
import cv2
import os
from rknnlite.api import RKNNLite

#IMG_PATH = 'test_180.jpg'
#IMG_PATH = '0_125.jpg'
RKNN_MODEL = './resnet18.rknn'
img_height = 32
img_width = 32
class_names = ["plane","car","bird","cat","deer","dog","frog","horse","ship","truck"]


def rknn_inference(root):
  total=0
  correct=0
  for path in os.listdir(root):
    image_filenames = os.listdir(root + '/' + path)
    for image_filename in image_filenames:
      img = cv2.imread(root + '/' + path + '/' + image_filename)
      img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
      outputs = rknn_lite.inference(inputs=[img])
      total += 1
      if np.argmax(outputs) == int(path[:1]) :
        correct += 1
  print('テスト画像{}枚における精度:{:.2f} %'.format(total,100 * correct / total))

# RKNN オブジェクトを作成
rknn_lite = RKNNLite()

# RKNN モデルをロード
print('--> Load RKNN model')
ret = rknn_lite.load_rknn(RKNN_MODEL)
if ret != 0:
    print('Load RKNN model failed')
    exit(ret)
print('done')

# ランタイム環境を初期化
print('--> Init runtime environment')
ret = rknn_lite.init_runtime()
if ret != 0:
```

```
        print('Init runtime environment failed!')
        exit(ret)
    print('done')

    # 画像をロード
    #img = cv2.imread(IMG_PATH)
    #img = cv2.resize(img,(32,32))
    #img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    #img = np.expand_dims(img, 0)
    #dnormalize_img = np.zeros_like(img)

    #cv2.normalize(img, dnormalize_img, 0, 1, cv2.NORM_MINMAX)



    # モデルを実行
    print('--> Running model')
    #outputs = rknn_lite.inference(inputs=[img])
    # テスト画像の保存パス。前の cifar10_to_jpg.py で取得するか、サンプルコードを使用します。
    rknn_inference("/home/cat/python/code/ai/pytorch/resnet18_pytorch/data/cifar-10-png/raw_test")
    print('done')
    #print("result: ", outputs)
    #print(
    #    "This image most likely belongs to {}."
    #   .format(class_names[np.argmax(outputs)])
    #)

    rknn_lite.release()
```

基板側簡単なテスト結果：

```
  --> Load RKNN model
  done
  --> Init runtime environment
  I RKNN: [02:16:11.827] RKNN Runtime Information, librknnrt version: 2.0.0b0 (35a6907d79@2024-
03-24T10:31:14)
  I RKNN: [02:16:11.827] RKNN Driver Information, version: 0.9.2
  I RKNN: [02:16:11.827] RKNN Model Information, version: 6, toolkit version:
2.0.0b0+9bab5682(compiler version: 2.0.0b0 (35a6907d79@2024-03-24T02:34:11)), target: RKNPU v2,
target platform: rk3588, framework name: ONNX, framework layout: NCHW, model inference type:
static_shape
  done
  --> Running model
  テスト画像 10000 枚における精度:66.44 %
  done
```

12.3 参考文献

- [ResNet18 実装]
https://pytorch.org/vision/main/models/generated/torchvision.models.resnet18.html
- [PyTorch Hub の ResNet]
https://pytorch.org/hub/pytorch_vision_resnet
- [ResNet 論文]
https://arxiv.org/abs/1512.03385

第 13 章 PaddlePaddle FastDeploy

13.1 FastDeploy

FastDeploy は、産業応用のシナリオにおける重要な AI モデルのために、モデル API を標準化し、ダウンロードしてすぐに実行できるデモを提供します。FastDeploy は、オンライン（サービスとしてのデプロイ）およびオフラインデプロイ形式をサポートし、さまざまな開発者のデプロイ要件を満たします。FastDeploy は AI 開発者に最適なモデルデプロイソリューションを提供することを目的としており、全シナリオ対応、簡単で使いやすい、そして極めて効率的な 3 つの特徴を持っています。

- 全シナリオ対応：GPU、CPU、Jetson、ARM CPU、Rockchip NPU、Amlogic NPU、NXP NPU などの多くのハードウェアをサポートし、ローカルデプロイ、サービスデプロイ、Web エンドデプロイ、モバイルエンドデプロイなどをサポートします。CV、NLP、Speech の 3 つの分野をサポートし、画像分類、画像分割、セマンティックセグメンテーション、物体検出、文字認識（OCR）、顔検出認識、人物切り抜き、姿勢推定、テキスト分類、情報抽出、人物追跡、音声合成などの 16 の主要なアルゴリズムシナリオをサポートします。
- 簡単で柔軟：3 行のコードで AI モデルのデプロイを完了し、1 行のコードでバックエンド推論エンジンとデプロイハードウェアを迅速に切り替え、統一された API で異なるデプロイシナリオのゼロコスト移行を実現します。
- 極めて効率的：従来のディープラーニング推論エンジンがモデルの推論時間にのみ焦点を当てるのに対し、FastDeploy はモデルタスクのエンドツーエンドのデプロイパフォーマンスに注目しています。高性能の前処理と後処理、高性能推論エンジンの統合、一発自動圧縮などの技術を通じて、AI モデルの推論デプロイの極限パフォーマンス最適化を実現します。

GitHub 上の FastDeploy 詳細紹介：
（たくさん産業用のオープンソースモデルもあります、製品開発には素早くできます。）
FastDeploy/docs/docs_i18n/README_日本語.md at develop · PaddlePaddle/FastDeploy (github.com)

次に、環境を簡単にセットアップし、FastDeploy を使用して rk3588 に軽量検出ネットワーク PicoDet をデプロイします。

ヒント: テスト環境：LubanCat ボードは Debian11、PC 端は ubuntu20.04。FastDeploy バージョン 1.0.7、rknn-Toolkit2 バージョン 2.0.0beta。

13.2 環境設定

13.2.1 PC 端モデル変換推論環境の構築

rknn-Toolkit2 と FastDeploy ツールをインストールする必要があります。

1. **rknn-Toolkit2 のインストール**
   前の《第 3 章 RKNN Toolkit2 紹介》を参照してください。

2. **FastDeploy などのツールをインストール**

```bash
# 事前にコンパイルされたライブラリを使用してインストールするか、FastDeploy をコンパイルしてインストール
# 詳細は以下を参照
pip3 install fastdeploy-python -f https://www.paddlepaddle.org.cn/whl/fastdeploy.html

# paddle2onnx をインストール
pip3 install paddle2onnx

# paddlepaddle をインストール(≪7.2 PaddlePaddle のインストール≫を参考)
python -m pip3 install paddlepaddle
```

14.2.2 ボード側 FastDeploy RKNPU2 推論環境の構築

1. **RKNN ドライバ環境のインストール**

```bash
# 最新の RK ドライバを使用
git clone https://github.com/rockchip-linux/rknpu2
sudo cp ./rknpu2/runtime/RK3588/Linux/librknn_api/aarch64/* /usr/lib
sudo cp ./rknpu2/runtime/RK3588/Linux/rknn_server/aarch64/usr/bin/* /usr/bin/
```

前の《第 3 章 RKNN Toolkit2 紹介》を参照してください。

2. **FastDeploy のインストール**

ボード側で推論をデプロイするため、ボード側で Python SDK をコンパイルし、パッケージ化して FastDeploy をインストールします。

```bash
```

ヒント: rk3588 のコンパイル中にハングアップする場合、メモリ不足が原因かもしれません。swap 領域を追加することをお勧めします。少なくとも 4G の swap 領域が必要で、ボード端のコンパイルは遅いため、しばらく待つ必要があります。

```
#一時的に swap を追加、現在の Swap サイズを確認
free -h
cat@lubancat:~/FastDeploy/build$ free -h
               total        used        free      shared  buff/cache   available
Mem:           3.8Gi       3.7Gi        12Mi        14Mi        59Mi        18Mi
Swap:             0B          0B          0B
 #Swap 追加
 sudo fallocate -l 4G /swapfile
 #正しいパーミッション設定
 sudo chmod 600 /swapfile
 #ファイルをスワップとしてフォーマットする
```

```
sudo mkswap /swapfile
#ファイルのスワップを有効にする
sudo swapon /swapfile
#スワップが有効になっていることを確認
sudo swapon --show
NAME             TYPE SIZE USED PRIO
/swapfile        file  4G   0B   -2
```

```
# ソースコードを取得
git clone https://github.com/PaddlePaddle/FastDeploy.git
```

1）FastDeploy Python SDK コンパイル・インストール方法：

```
# 環境変数
export ENABLE_ORT_BACKEND=ON
export ENABLE_RKNPU2_BACKEND=ON  # rknpu2 をバックエンドエンジンとして使用
export ENABLE_VISION=ON
export RKNN2_TARGET_SOC=RK3588  # RK3588 プラットフォーム

cd FastDeploy/python

# コンパイル、パッケージ化
python3 setup.py build
python3 setup.py bdist_wheel

# インストール
cd dist
pip3 install fastdeploy_python-0.0.0-cp39-cp39-linux_aarch64.whl
```

```
コンパイルとパッケージ化が不要な場合、付属の例の`fastdeploy_python-0.0.0-cp39-cp39-linux_aarch64.whl`を直接使用してインストールすることもできます。
# 環境変数設定（しないとエラーが出ます。）
nano ~/.bashrc
export LD_LIBRARY_PATH=/home/cat/.local/lib/python3.9/site-packages/fastdeploy/libs/third_libs/onnxruntime/lib:$LD_LIBRARY_PATH
    export LD_LIBRARY_PATH=/home/cat/.local/lib/python3.9/site-packages/fastdeploy/libs/third_libs/opencv/lib:$LD_LIBRARY_PATH
#設定を有効にする
source ~/.bashrc
```

2）FastDeploy C++ SDK コンパイル・インストール方法：

```
#コンパイル用フォルダー作成
cd FastDeploy
mkdir build && cd build
#コンパイル前設定
cmake ..  -DENABLE_ORT_BACKEND=ON ¥
    -DENABLE_RKNPU2_BACKEND=ON ¥
```

```
        -DENABLE_VISION=ON ¥
        -DRKNN2_TARGET_SOC=RK3588 ¥
        -DCMAKE_INSTALL_PREFIX=${PWD}/fastdeploy-0.0.3

    #コンパイル
    make
    #インストール
    make install
```

13.3 デプロイ推論の例

13.3.1 軽量化検出ネットワーク PicoDet

参考 URL:
FastDeploy/examples/vision/detection/paddledetection/rknpu2/README.md at develop · PaddlePaddle/FastDeploy (github.com)

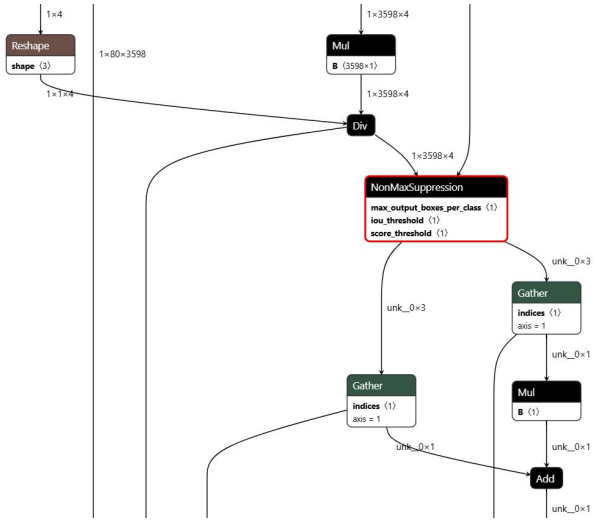
1. PC 側モデル変換

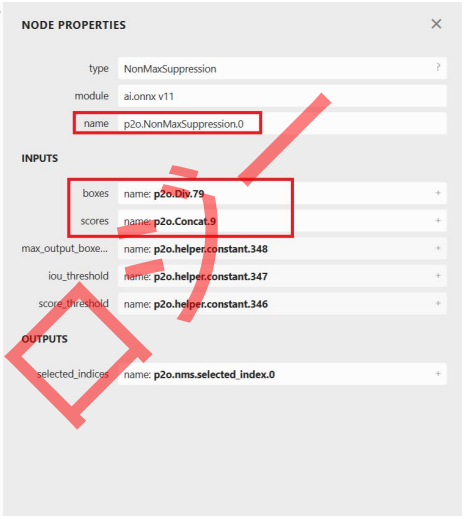
rknn-Toolkit2 は現在 Paddle モデルを直接 RKNN モデルとしてエクスポートすることをサポートしていません。Paddle2ONNX を使用して onnx モデルに変換し、次に rknn モデルをエクスポートする必要があります。

```bash
# Paddle の静的グラフモデルを取得し、解凍
wget https://paddledet.bj.bcebos.com/deploy/Inference/picodet_s_416_coco_lcnet.tar
＊直接サンプルソースコード python_lubancat_RK_tutorial_code¥ai¥fastdeplay を使ってもいい
# 解凍
tar xvf picodet_s_416_coco_lcnet.tar

# 静的グラフモデルを onnx モデルに変換
cd tools/rknpu2/
paddle2onnx --model_dir picodet_s_416_coco_lcnet ¥
            --model_filename model.pdmodel ¥
            --params_filename model.pdiparams ¥
            --save_file picodet_s_416_coco_lcnet/picodet_s_416_coco_lcnet.onnx

# shape の固定
python3 -m paddle2onnx.optimize --input_model
picodet_s_416_coco_lcnet/picodet_s_416_coco_lcnet.onnx ¥
    --output_model picodet_s_416_coco_lcnet/picodet_s_416_coco_lcnet.onnx ¥
    --input_shape_dict "{'image':[1,3,416,416],'scale_factor':[1,2]}"
```

2. RKNN モデルのエクスポート

Paddle2ONNX のバージョンの違いにより（テスト時には 1.0.7 を使用）、変換されたモデルの出力ノードの名前も異なります。Netron を使用してモデルを可視化し、NonMaxSuppression ノードを見つけ、出力ノードの名前を確認してから、`picodet_s_416_coco_lcnet.yaml`構成ファイルの`outputs_nodes`パラメータを変更する必要があります。





rknpu2/config/picodet_s_416_coco_lcnet_unquantized.yaml 構成ファイル

```
mean:
-
    - 123.675
    - 116.28
    - 103.53
std:
-
    - 58.395
    - 57.12
    - 57.375
model_path: ./picodet_s_416_coco_lcnet/picodet_s_416_coco_lcnet.onnx
outputs_nodes:
- 'p2o.Mul.179'
- 'p2o.Concat.9'
do_quantization: False
dataset:
output_folder: "./picodet_s_416_coco_lcnet"
```

```bash
# モデルをエクスポート
cd tools/rknpu2/
```

python export.py --config_path config/picodet_s_416_coco_lcnet_unquantized.yaml --target_platform rk3588
```

3. ボード側のデプロイと推論

```bash
# ボード端での推論デプロイ、前にエクスポートした rknn モデルまたは付属の例を使用可能
cd examples/vision/detection/paddledetection/rknpu2/python
python3 infer.py --model_file picodet_s_416_coco_lcnet_rk3588_unquantized.rknn --config_file infer_cfg.yml --image 00000014439.jpg
```

```
[INFO] fastdeploy/vision/common/processors/transform.cc(45)::FuseNormalizeCast  Normalize and Cast are fused to Normalize in preprocessing pipeline.
[INFO] fastdeploy/vision/common/processors/transform.cc(93)::FuseNormalizeHWC2CHW  Normalize and HWC2CHW are fused to NormalizeAndPermute  in preprocessing pipeline.
[INFO] fastdeploy/vision/common/processors/transform.cc(159)::FuseNormalizeColorConvert  BGR2RGB and NormalizeAndPermute are fused to NormalizeAndPermute with swap_rb=1
[INFO] fastdeploy/runtime/backends/rknpu2/rknpu2_backend.cc(81)::GetSDKAndDeviceVersion rknpu2 runtime version: 1.5.1b19 (32afb0e92@2023-07-14T12:46:17)
[INFO] fastdeploy/runtime/backends/rknpu2/rknpu2_backend.cc(82)::GetSDKAndDeviceVersion rknpu2 driver version: 0.9.2
index=0, name=image, n_dims=4, dims=[1, 416, 416, 3], n_elems=519168, size=1038336, fmt=NHWC, type=FP16, qnt_type=AFFINE, zp=0, scale=1.000000, pass_through=0
index=0, name=p2o.Mul.179, n_dims=3, dims=[1, 3598, 4, 0], n_elems=14392, size=28784, fmt=UNDEFINED, type=FP32, qnt_type=AFFINE, zp=0, scale=1.000000, pass_through=0
index=1, name=p2o.Concat.9, n_dims=3, dims=[1, 80, 3598, 0], n_elems=287840, size=575680, fmt=UNDEFINED, type=FP32, qnt_type=AFFINE, zp=0, scale=1.000000, pass_through=0
[INFO] fastdeploy/runtime/runtime.cc(367)::CreateRKNPU2Backend  Runtime initialized with Backend::RKNPU2 in Device::RKNPU.
[WARNING] fastdeploy/runtime/backends/rknpu2/rknpu2_backend.cc(420)::InitRKNNTensorMemory  The input tensor type != model's inputs type.The input_type need FP16,but inputs[0].type is UINT8
DetectionResult: [xmin, ymin, xmax, ymax, score, label_id]
413.461548,89.649635, 508.461548, 282.848541, 0.833984, 0
160.288467,81.880402, 202.307693, 166.795670, 0.812988, 0
264.615387,79.756004, 301.923096, 168.009613, 0.791992, 0
105.288467,46.251198, 126.730774, 93.594948, 0.770020, 0
583.846191,114.596146, 612.692322, 178.085327, 0.763672, 0
328.269257,40.211838, 344.423096, 80.545067, 0.637207, 0
379.038483,42.093449, 396.153870, 83.337135, 0.554199, 0
510.000031,116.113579, 599.230774, 278.235565, 0.540527, 0
24.350962,116.538452, 56.153847, 153.442307, 0.443359, 0
58.557693,136.325714, 107.211540, 173.593735, 0.426514, 0
352.307709,45.067604, 376.923096, 104.034851, 0.422119, 0
188.750000,45.795971, 200.192322, 61.668266, 0.415283, 0
352.692322,45.097954, 369.615387, 87.221748, 0.388672, 0
355.384644,60.970249, 387.884644, 114.353363, 0.364990, 0
505.000031,114.838936, 556.538452, 269.009613, 0.335449, 0
-1.550481,150.164658, 37.331734, 171.287247, 0.402100, 24
59.615387,143.609375, 104.038467, 171.530045, 0.311279, 24
163.173080,87.525238, 600.000000, 344.274017, 0.578125, 33
164.423080,84.975960, 320.000000, 344.516815, 0.390381, 33

Visualized result save in ./visualized_result.jpg
```

--model_file でモデルファイルを指定し、--config_file で構成ファイルを指定し、--image で推論する画像を指定
```

13.4 参考リンク

より詳細な環境設定、デプロイ最適化、デプロイの例などのドキュメントについては、以下を参照してください：

- https://github.com/PaddlePaddle/FastDeploy/tree/develop/docs
- https://github.com/PaddlePaddle/FastDeploy/tree/develop/docs/en/faq/rknpu2
- https://github.com/rockchip-linux/rknn-toolkit2

本章内容参考リンク：

FastDeploy/examples/vision/detection/paddledetection/rknpu2/README.md at develop · PaddlePaddle/FastDeploy (github.com)

第 14 章 PP-ORCv3

14.1 PP-OCRv3

PP-OCR は、Baidu が発表しオープンソース化した OCR 分野のアルゴリズムで、軽量な OCR システムです。最先端のアルゴリズムを実現する一方で、精度と速度のバランスを考慮し、モデルのスリム化と深度最適化を行い、産業の実用化ニーズをできるだけ満たすように設計されています。

PP-OCR は二段階の OCR システムであり、テキスト検出アルゴリズムに DB を、テキスト認識アルゴリズムに CRNN を採用し、検出と認識モジュールの間にテキスト方向分類器を追加して、異なる方向のテキスト認識に対応しています。

PP-OCRv2 は、PP-OCR を基にさらに 5 つの重要な点で最適化されています。検出モデルには CML 協調学習知識蒸留戦略と CopyPaste データ拡張戦略を採用し、認識モデルには LCNet 軽量バックボーンネットワーク、UDML 改良知識蒸留戦略、および Enhanced CTC loss 損失関数の改良（上図の赤枠参照）を取り入れ、推論速度と予測精度をさらに向上させました。

PP-OCRv3 は、PP-OCRv2 を基にさらにアップグレードされました。全体のフレームワークは PP-OCRv2 と同じパイプラインを維持し、検出モデルと認識モデルの両方を最適化しています。検出モジュールは DB アルゴリズムに基づいて最適化され、認識モジュールには CRNN の代わりに IJCAI 2022 で最新のテキスト認識アルゴリズム SVTR を採用し、産業への適用を図っています。

最新の PP-OCRv4 は、PP-OCRv3 を基にさらにアップグレードされました。全体のフレームワークは PP-OCRv3 と同じパイプラインを維持し、検出モデルと認識モデルに対してデータ、ネットワーク構造、訓練戦略などの複数のモジュールを最適化しています。

PaddleOCR は、FastDeploy を通じて RKNPU2 にモデルをデプロイすることをサポートしています。具体的なモデルリストについてはここを参照してください。次に、ボード上で FastDeploy を使用して PP-ORCv3 のデプロイテストを行います。

ヒント：チュートリアルのテスト環境:Lubancat-4 は Debian11 を使用、PC 端 ubuntu20.04 です。

14.2 環境インストール

FastDeploy の関連環境のインストールについては、前の FastDeploy 章を参照するか、FastDeploy ドキュメントを参照してください。PC またはクラウドサーバーなどに PaddlePaddle、FastDeploy、および paddle2onnx などのツールをインストールして、モデルの訓練および変換を行い、ボード上に FastDeploy 実行環境（Python または C++）をインストールして、モデルのデプロイ推論を行います。

14.3 モデル変換

PP-OCR シリーズモデルの訓練については、PaddleOCR チュートリアルドキュメントを参照してください。チュートリアルテストでは、PaddleOCR からモデルを直接取得し、RKNN モデルに変換します。

RKNPU2 に PP-OCR シリーズモデルをデプロイする際には、Paddle の推論モデルを RKNN モデルに変換する必要がありますが、rknn_toolkit2 ツールは直接 Paddle から RKNN モデルへの変換をサポートしていないため、まず Paddle 推論モデルを ONNX モデルに変換し、最後に RKNN モデルに変換する必要があります。

PC の ubuntu20.04 システム上で：

```
# PP-OCRv3 テキスト検出モデルをダウンロードし、解凍
wget https://paddleocr.bj.bcebos.com/PP-OCRv3/multilingual/Multilingual_PP-OCRv3_det_infer.tar
tar -xvf Multilingual_PP-OCRv3_det_infer.tar

# テキスト方向分類器モデルをダウンロードし、解凍
wget https://paddleocr.bj.bcebos.com/dygraph_v2.0/ch/ch_ppocr_mobile_v2.0_cls_infer.tar
tar -xvf ch_ppocr_mobile_v2.0_cls_infer.tar

# PP-OCRv3 テキスト認識モデルをダウンロードし、解凍
wget https://paddleocr.bj.bcebos.com/PP-OCRv3/multilingual/japan_PP-OCRv3_rec_infer.tar
tar -xvf japan_PP-OCRv3_rec_infer.tar

# paddle2onnx を使用して、モデルを ONNX モデルに変換する
paddle2onnx --model_dir Multilingual_PP-OCRv3_det_infer ¥
--model_filename inference.pdmodel ¥
--params_filename inference.pdiparams ¥
--save_file Multilingual_PP-OCRv3_det_infer/Multilingual_PP-OCRv3_det_infer.onnx ¥
# 新しいバージョンの paddle2onnx には enable_dev_version というパラメータがない
--enable_dev_version True

paddle2onnx --model_dir ch_ppocr_mobile_v2.0_cls_infer ¥
--model_filename inference.pdmodel ¥
--params_filename inference.pdiparams ¥
--save_file ch_ppocr_mobile_v2.0_cls_infer/ch_ppocr_mobile_v2.0_cls_infer.onnx ¥

paddle2onnx --model_dir japan_PP-OCRv3_rec_infer ¥
--model_filename inference.pdmodel ¥
--params_filename inference.pdiparams ¥
--save_file japan_PP-OCRv3_rec_infer/japan_PP-OCRv3_rec_infer.onnx

# モデルの入力 shape を固定する
python -m paddle2onnx.optimize --input_model Multilingual_PP-OCRv3_det_infer/Multilingual_PP-
OCRv3_det_infer.onnx ¥
```

```
  --output_model Multilingual_PP-OCRv3_det_infer/Multilingual_PP-OCRv3_det_infer.onnx ¥
  --input_shape_dict "{'x':[1,3,960,960]}"

  python -m paddle2onnx.optimize --input_model
ch_ppocr_mobile_v2.0_cls_infer/ch_ppocr_mobile_v2.0_cls_infer.onnx ¥
  --output_model ch_ppocr_mobile_v2.0_cls_infer/ch_ppocr_mobile_v2.0_cls_infer.onnx ¥
  --input_shape_dict "{'x':[1,3,48,192]}"

  python -m paddle2onnx.optimize --input_model japan_PP-OCRv3_rec_infer/japan_PP-OCRv3_rec_infer.onnx ¥
  --output_model japan_PP-OCRv3_rec_infer/japan_PP-OCRv3_rec_infer.onnx ¥
  --input_shape_dict "{'x':[1,3,48,320]}"
```

ツールソースコードを取得:

```
  # FastDeploy のソースコードを取得し、examples/vision/ocr/PP-OCR/rockchip ディレクトリに移動
(または付属の例から取得)
  git clone https://github.com/PaddlePaddle/FastDeploy.git
  cd FastDeploy/examples/vision/ocr/PP-OCR/rockchip
```

rknpu2_tools/config ディレクトリ内の設定ファイルを編集（通常はデフォルト）、設定ファイル内のモデルパス model_path とモデル出力パス output_folder を対応するモデルディレクトリに変更する必要があります。3 つのモデル設定ファイルすべてを変更し、プロジェクトでは ppocrv3_cls.yaml を例にします:

リスト 1:rknpu2_tools/config/ppocrv3_cls.yaml

```
  mean:
    -
      - 127.5
      - 127.5
      - 127.5
  std:
    -
      - 127.5
      - 127.5
      - 127.5
  model_path: ./ch_ppocr_mobile_v2.0_cls_infer/ch_ppocr_mobile_v2.0_cls_infer.onnx
  outputs_nodes:
  do_quantization: False
  dataset:
  output_folder: "./ch_ppocr_mobile_v2.0_cls_infer"
```

次に、export.py を使用して、前述の ONNX モデルを RKNN モデルに変換します:

```
# rknpu2_tools と同じディレクトリに移動し、現在のディレクトリ構造：
.
├─── Multilingual_PP-OCRv3_det_infer
├─── japan_PP-OCRv3_rec_infer
├─── ch_ppocr_mobile_v2.0_cls_infer
└─── rknpu2_tools


# FastDeploy が提供するツールを使用して、ONNX モデルを RKNN モデルに変換する（環境には rknn_toolkit2 ツールがインストールされている必要があります）
# --target_platform オプションでプラットフォームを指定します。lubancat-4 は rk3588、チュートリアルでは lubancat-4 ボードを使用しています。
python rknpu2_tools/export.py --config_path rknpu2_tools/config/ppocrv3_det.yaml ¥
--target_platform rk3588
python rknpu2_tools/export.py --config_path rknpu2_tools/config/ppocrv3_rec.yaml ¥
--target_platform rk3588
python rknpu2_tools/export.py --config_path rknpu2_tools/config/ppocrv3_cls.yaml ¥
--target_platform rk3588
```

onnx モデルをロードする際に発生する可能性のある問題:

```
E load_onnx: onnx.onnx_cpp2py_export.checker.ValidationError: Field 'shape' of type is required but missing.
```

モデルの一部のタイプに「shape」が欠けている場合は、onnxruntime のツールを使用して onnx の shape を再推論することができます。

```
python symbolic_shape_infer.py --input ch_ppocr_mobile_v2.0_cls_infer/ch_ppocr_mobile_v2.0_cls_infer_old.onnx ¥
--output ch_ppocr_mobile_v2.0_cls_infer/ch_ppocr_mobile_v2.0_cls_infer.onnx
```

symbolic_shape_infer.py プログラムの詳細については、onnxruntime を参照してください。
最後に、export.py を使用して変換された RKNN モデルは対応するディレクトリに保存されます。

14.4 Python デプロイテスト

デプロイ推論プログラムは、FastDeploy または PaddleOCR のソースコードから取得することができます（またはチュートリアルの付属例を参照）。次に、前述の変換モデルのフォルダをボードにコピーし、推論テストを行います（テストプラットフォームは lubancat-4、FastDeploy は develop ブランチです）：

```
# テスト画像（他のテスト画像でも可）と辞書ファイルをダウンロード
wget https://github.com/PaddlePaddle/PaddleOCR/raw/release/2.6/doc/imgs/japan_2.jpg
wget
https://github.com/PaddlePaddle/PaddleOCR/raw/release/2.6/ppocr/utils/dict/japan_dict.txt

# デプロイ例コードをダウンロードし、付属例から取得する場合は rockchip/python ディレクトリ
git clone https://github.com/PaddlePaddle/FastDeploy.git
```

```
cd FastDeploy/examples/vision/ocr/PP-OCR/rockchip/python

# 現在のディレクトリファイル
(.toolkit2_env) (base) csun@CSUN-PC-
0013:~/linuxshare/work/AI/jupyter/FastDeploy/examples/vision/ocr/PP-OCR/rockchip$ ls -ls
.
├──── Multilingual_PP-OCRv3_det_infer
├──── README.md
├──── ch_ppocr_mobile_v2.0_cls_infer
├──── cpp
├──── japan_2.jpg
├──── japan_PP-OCRv3_rec_infer
├──── japan_dict.txt
├──── python
└──── rknpu2_tools
```

cpu 推論を実行し、onnx モデルを使用：

```
# CPU 推論コード
cat@lubancat:~/lubancat_ai_manual_code/example/ppocrv3/rockchip$ python3 python/infer.py                --
det_model ./Multilingual_PP-OCRv3_det_infer/Multilingual_PP-OCRv3_det_infer.onnx                --
cls_model ./ch_ppocr_mobile_v2.0_cls_infer/ch_ppocr_mobile_v2.0_cls_infer.onnx                --
rec_model ./japan_PP-OCRv3_rec_infer/japan_PP-OCRv3_rec_infer.onnx                --
rec_label_file ./japan_dict.txt                --image ./japan_2.jpg                --device cpu
[INFO] fastdeploy/runtime/runtime.cc(326)::CreateOrtBackend     Runtime initialized with Backend::ORT in
Device::CPU.
[INFO] fastdeploy/runtime/runtime.cc(326)::CreateOrtBackend     Runtime initialized with Backend::ORT in
Device::CPU.
[INFO] fastdeploy/runtime/runtime.cc(326)::CreateOrtBackend     Runtime initialized with Backend::ORT in
Device::CPU.
det boxes: [[673,63],[848,63],[848,105],[673,105]]rec text: もちもち rec score:0.999996 cls label: 0 cls
score: 1.000000
det boxes: [[390,82],[537,79],[540,130],[393,132]]rec text: 天然の rec score:0.997521 cls label: 0 cls score:
1.000000
det boxes: [[873,90],[1019,90],[1019,129],[873,129]]rec text: とろっと rec score:0.999989 cls label: 0 cls
score: 0.961403
det boxes: [[1067,97],[1296,97],[1296,142],[1067,142]]rec text: 後味のよい rec score:0.988884 cls label: 0
cls score: 1.000000
det boxes: [[246,119],[324,119],[324,148],[246,148]]rec text: 濃厚な rec score:0.997802 cls label: 0 cls
score: 0.999997
det boxes: [[620,131],[720,131],[720,157],[620,157]]rec text: サクサク rec score:0.995820 cls label: 0 cls
score: 0.999996
det boxes: [[769,152],[892,152],[892,178],[769,178]]rec text: 味わい深い rec score:0.998777 cls label: 0 cls
score: 1.000000
det boxes: [[928,153],[1232,153],[1232,228],[928,228]]rec text: 焼きたて rec score:0.998969 cls label: 0 cls
score: 0.999998
det boxes: [[171,174],[292,174],[292,200],[171,200]]rec text: 深みのある rec score:0.999420 cls label: 0 cls
score: 0.999995
det boxes: [[416,158],[595,158],[595,189],[416,189]]rec text: なめらかな rec score:0.999909 cls label: 0 cls
score: 0.999991
det boxes: [[691,184],[843,186],[841,224],[689,222]]rec text: ふわふわ rec score:0.996401 cls label: 0 cls
score: 0.999944
det boxes: [[120,221],[558,221],[558,286],[120,286]]rec text: うま味のある rec score:0.999165 cls label: 0
cls score: 0.999845
det boxes: [[593,228],[720,228],[720,263],[593,263]]rec text: とろ一り rec score:0.978696 cls label: 0 cls
score: 0.999995
det boxes: [[1057,244],[1134,244],[1134,284],[1057,284]]rec text: 絶品 rec score:0.999971 cls label: 0 cls
```

score: 0.999925
  det boxes: [[1208,242],[1390,247],[1385,305],[1203,300]]rec text: 贅沢な rec score:0.919864 cls label: 0 cls score: 1.000000
  det boxes: [[752,254],[1028,250],[1030,298],[753,301]]rec text: 飽きのこない rec score:0.999389 cls label: 0 cls score: 0.999356
  det boxes: [[556,291],[654,291],[654,325],[556,325]]rec text: ピリ辛 rec score:0.822031 cls label: 0 cls score: 0.999186
  det boxes: [[347,305],[512,305],[512,326],[347,326]]rec text: やみつきになる rec score:0.999725 cls label: 0 cls score: 0.999317
  det boxes: [[1057,303],[1203,303],[1203,338],[1057,338]]rec text: ふわっと rec score:0.999295 cls label: 0 cls score: 0.999348
  det boxes: [[820,316],[960,316],[960,351],[820,351]]rec text: サクッと rec score:0.999706 cls label: 0 cls score: 0.999997
  det boxes: [[228,338],[350,338],[350,363],[228,363]]rec text: コクのある rec score:0.998522 cls label: 0 cls score: 0.999818
  det boxes: [[56,308],[211,308],[211,350],[56,350]]rec text: 香ばしい rec score:0.998818 cls label: 0 cls score: 0.999747
  det boxes: [[640,338],[804,336],[806,370],[641,372]]rec text: 脂の乗った rec score:0.998902 cls label: 0 cls score: 1.000000
  det boxes: [[1203,346],[1348,346],[1348,385],[1203,385]]rec text: カリカリ rec score:0.999788 cls label: 0 cls score: 1.000000
  det boxes: [[329,369],[603,363],[606,413],[332,419]]rec text: スパイシー rec score:0.995897 cls label: 0 cls score: 0.999993
  det boxes: [[939,361],[1166,356],[1171,415],[944,420]]rec text: 揚げたて rec score:0.999562 cls label: 0 cls score: 1.000000
  det boxes: [[153,386],[299,384],[300,421],[155,422]]rec text: とろとろ rec score:0.999991 cls label: 0 cls score: 0.826810
  det boxes: [[742,389],[923,383],[928,421],[747,427]]rec text: こだわりの rec score:0.991786 cls label: 0 cls score: 0.999864
  det boxes: [[560,406],[713,402],[718,442],[563,446]]rec text: 出来たて rec score:0.999884 cls label: 0 cls score: 1.000000
  det boxes: [[1224,415],[1300,415],[1300,443],[1224,443]]rec text: 新鮮な rec score:0.994382 cls label: 0 cls score: 1.000000
  det boxes: [[360,438],[464,438],[464,464],[360,464]]rec text: リッチな rec score:0.999920 cls label: 0 cls score: 1.000000
  det boxes: [[1041,436],[1206,436],[1206,475],[1041,475]]rec text: 炊きたて rec score:0.998127 cls label: 0 cls score: 1.000000
  det boxes: [[105,454],[233,452],[235,481],[107,483]]rec text: 風味豊かな rec score:0.999369 cls label: 0 cls score: 1.000000
  det boxes: [[710,452],[864,452],[864,494],[710,494]]rec text: 熟成した rec score:0.999973 cls label: 0 cls score: 1.000000
  det boxes: [[888,466],[1041,466],[1041,506],[888,506]]rec text: カリッと rec score:0.999974 cls label: 0 cls score: 1.000000
  det boxes: [[1302,464],[1400,464],[1400,510],[1302,510]]rec text: 美味 rec score:0.999916 cls label: 0 cls score: 1.000000
  det boxes: [[233,488],[382,488],[382,540],[233,540]]rec text: 芳醸な rec score:0.897710 cls label: 0 cls score: 1.000000
  det boxes: [[454,467],[595,464],[598,501],[457,505]]rec text: こんがり rec score:0.890585 cls label: 0 cls score: 0.999028
  det boxes: [[408,513],[772,515],[771,601],[406,599]]rec text: ふんわり rec score:0.994684 cls label: 0 cls score: 0.999797
  det boxes: [[1070,510],[1286,510],[1286,544],[1070,544]]rec text: 口どけのよい rec score:0.999709 cls label: 0 cls score: 0.996961
  det boxes: [[817,531],[993,531],[993,565],[817,565]]rec text: コシのある rec score:0.981016 cls label: 0 cls score: 0.985237
  det boxes: [[134,554],[256,554],[256,582],[134,582]]rec text: まろやかな rec score:0.998829 cls label: 0 cls score: 1.000000
  det boxes: [[905,570],[1097,584],[1084,637],[892,624]]rec text: パリッと rec score:0.999985 cls label: 0 cls score: 0.999667
  det boxes: [[1060,581],[1193,584],[1187,660],[1054,657]]rec text: 旬 rec score:0.619156 cls label: 0 cls score: 1.000000

```
  det boxes: [[1232,579],[1382,577],[1384,616],[1233,617]]rec text: 産地直送 rec score:0.999997 cls label: 0
cls score: 1.000000
  det boxes: [[251,612],[476,614],[475,658],[249,656]]rec text: クセになる rec score:0.999891 cls label: 0 cls
score: 1.000000
  det boxes: [[748,617],[854,617],[854,647],[748,647]]rec text: 食べごろ rec score:0.987618 cls label: 0 cls
score: 0.999981
  det boxes: [[1080,614],[1086,614],[1086,621],[1080,621]]rec text: - rec score:0.697572 cls label: 0 cls
score: 0.571474
  det boxes: [[564,629],[680,629],[680,666],[564,666]]rec text: 秘伝の rec score:0.999969 cls label: 0 cls
score: 1.000000
  det boxes: [[171,675],[316,675],[316,714],[171,714]]rec text: ほくほく rec score:0.876584 cls label: 0 cls
score: 0.997420
  det boxes: [[806,669],[985,669],[985,711],[806,711]]rec text: スイート rec score:0.996261 cls label: 0 cls
score: 1.000000
  det boxes: [[665,679],[782,678],[784,710],[667,712]]rec text: 流れたて rec score:0.900919 cls label: 0 cls
score: 0.999998
  det boxes: [[1027,675],[1164,677],[1163,714],[1025,712]]rec text: もっちり rec score:0.999722 cls label: 0
cls score: 0.999996
  det boxes: [[996,728],[1110,728],[1110,770],[996,770]]rec text: 木場の rec score:0.926558 cls label: 0 cls
score: 1.000000
  det boxes: [[396,684],[572,684],[572,718],[396,718]]rec text: ジューシー rec score:0.997624 cls label: 0 cls
score: 0.999645
  det boxes: [[283,740],[379,740],[379,776],[283,776]]rec text: 朝採り rec score:0.999268 cls label: 0 cls
score: 1.000000
  det boxes: [[505,741],[769,741],[769,787],[505,787]]rec text: 後味すっきり rec score:0.993246 cls label: 0
cls score: 1.000000
  det boxes: [[830,771],[924,771],[924,797],[830,797]]rec text: とろける rec score:0.999871 cls label: 0 cls
score: 1.000000

  可視化結果は./visualized_result.jpgに保存されました
```

NPU 推論を実行し、rknn モデルを使用：

```
  # NPU 推論コード
  cat@lubancat:~/lubancat_ai_manual_code/example/ppocrv3/rockchip$ python3 python/infer.py ¥
                --det_model ./Multilingual_PP-OCRv3_det_infer/Multilingual_PP-
OCRv3_det_infer_rk3588_unquantized.rknn ¥
                --
cls_model ./ch_ppocr_mobile_v2.0_cls_infer/ch_ppocr_mobile_v20_cls_infer_rk3588_unquantized.rkn
n ¥
                --rec_model ./japan_PP-OCRv3_rec_infer/japan_PP-
OCRv3_rec_infer_rk3588_unquantized.rknn ¥
                --rec_label_file ./japan_dict.txt ¥
                --image ./japan_2.jpg ¥
                --device npu
  [INFO] fastdeploy/runtime/backends/rknpu2/rknpu2_backend.cc(81)::GetSDKAndDeviceVersion
rknpu2 runtime version: 1.5.1b19 (32afb0e92@2023-07-14T12:46:17)
  [INFO] fastdeploy/runtime/backends/rknpu2/rknpu2_backend.cc(82)::GetSDKAndDeviceVersion
rknpu2 driver version: 0.9.2
  index=0, name=x, n_dims=4, dims=[1, 960, 960, 3], n_elems=2764800, size=5529600, fmt=NHWC,
type=FP16, qnt_type=AFFINE, zp=0, scale=1.000000, pass_through=0
  index=0, name=sigmoid_0.tmp_0, n_dims=4, dims=[1, 1, 960, 960], n_elems=921600, size=1843200,
fmt=NCHW, type=FP32, qnt_type=AFFINE, zp=0, scale=1.000000, pass_through=0
  [INFO] fastdeploy/runtime/runtime.cc(367)::CreateRKNPU2Backend  Runtime initialized with
Backend::RKNPU2 in Device::RKNPU.
  [INFO] fastdeploy/runtime/backends/rknpu2/rknpu2_backend.cc(81)::GetSDKAndDeviceVersion
```

```
rknpu2 runtime version: 1.5.1b19 (32afb0e92@2023-07-14T12:46:17)
  [INFO] fastdeploy/runtime/backends/rknpu2/rknpu2_backend.cc(82)::GetSDKAndDeviceVersion
rknpu2 driver version: 0.9.2
  index=0, name=x, n_dims=4, dims=[1, 48, 192, 3], n_elems=27648, size=55296, fmt=NHWC,
type=FP16, qnt_type=AFFINE, zp=0, scale=1.000000, pass_through=0
  index=0, name=softmax_0.tmp_0, n_dims=2, dims=[1, 2, 0, 0], n_elems=2, size=4, fmt=UNDEFINED,
type=FP32, qnt_type=AFFINE, zp=0, scale=1.000000, pass_through=0
  [INFO] fastdeploy/runtime/runtime.cc(367)::CreateRKNPU2Backend  Runtime initialized with
Backend::RKNPU2 in Device::RKNPU.
  [INFO] fastdeploy/runtime/backends/rknpu2/rknpu2_backend.cc(81)::GetSDKAndDeviceVersion
rknpu2 runtime version: 1.5.1b19 (32afb0e92@2023-07-14T12:46:17)
  [INFO] fastdeploy/runtime/backends/rknpu2/rknpu2_backend.cc(82)::GetSDKAndDeviceVersion
rknpu2 driver version: 0.9.2
  index=0, name=x, n_dims=4, dims=[1, 48, 320, 3], n_elems=46080, size=92160, fmt=NHWC,
type=FP16, qnt_type=AFFINE, zp=0, scale=1.000000, pass_through=0
  index=0, name=softmax_2.tmp_0, n_dims=3, dims=[1, 40, 4401, 0], n_elems=176040, size=352080,
fmt=UNDEFINED, type=FP32, qnt_type=AFFINE, zp=0, scale=1.000000, pass_through=0
  [INFO] fastdeploy/runtime/runtime.cc(367)::CreateRKNPU2Backend  Runtime initialized with
Backend::RKNPU2 in Device::RKNPU.
  [WARNING] fastdeploy/runtime/backends/rknpu2/rknpu2_backend.cc(420)::InitRKNNTensorMemory
The input tensor type != model's inputs type.The input_type need FP16,but inputs[0].type is
UINT8
  [WARNING] fastdeploy/runtime/backends/rknpu2/rknpu2_backend.cc(420)::InitRKNNTensorMemory
The input tensor type != model's inputs type.The input_type need FP16,but inputs[0].type is
UINT8
  [WARNING] fastdeploy/runtime/backends/rknpu2/rknpu2_backend.cc(420)::InitRKNNTensorMemory
The input tensor type != model's inputs type.The input_type need FP16,but inputs[0].type is
UINT8
  det boxes: [[673,63],[848,63],[848,105],[673,105]]rec text: もちもち rec score:0.788086 cls
label: 0 cls score: 1.000000
  det boxes: [[390,82],[537,79],[540,130],[393,132]]rec text: 天然の rec score:0.786621 cls
label: 0 cls score: 1.000000
  det boxes: [[873,90],[1019,90],[1019,129],[873,129]]rec text: とろっと rec score:0.788086 cls
label: 0 cls score: 0.970703
  det boxes: [[1067,97],[1296,97],[1296,142],[1067,142]]rec text: 後味のよい rec score:0.779297
cls label: 0 cls score: 1.000000
  det boxes: [[246,119],[324,119],[324,148],[246,148]]rec text: 濃厚な rec score:0.786784 cls
label: 0 cls score: 1.000000
  det boxes: [[620,131],[720,131],[720,157],[620,157]]rec text: サクサク rec score:0.785766 cls
label: 0 cls score: 1.000000
  det boxes: [[769,152],[892,152],[892,178],[769,178]]rec text: 味わい深い rec score:0.787695
cls label: 0 cls score: 1.000000
  det boxes: [[928,153],[1232,153],[1232,228],[928,228]]rec text: 焼きたて rec score:0.787720
cls label: 0 cls score: 1.000000
  det boxes: [[171,174],[292,174],[292,200],[171,200]]rec text: 深みのある rec score:0.787695
cls label: 0 cls score: 1.000000
  det boxes: [[416,158],[595,158],[595,189],[416,189]]rec text: なめらかな rec score:0.788086
cls label: 0 cls score: 1.000000
  det boxes: [[691,184],[843,186],[841,224],[689,222]]rec text: ふわふわ rec score:0.786011 cls
label: 0 cls score: 1.000000
  det boxes: [[116,221],[560,219],[560,286],[116,288]]rec text: うま味のある rec score:0.787842
cls label: 0 cls score: 1.000000
```

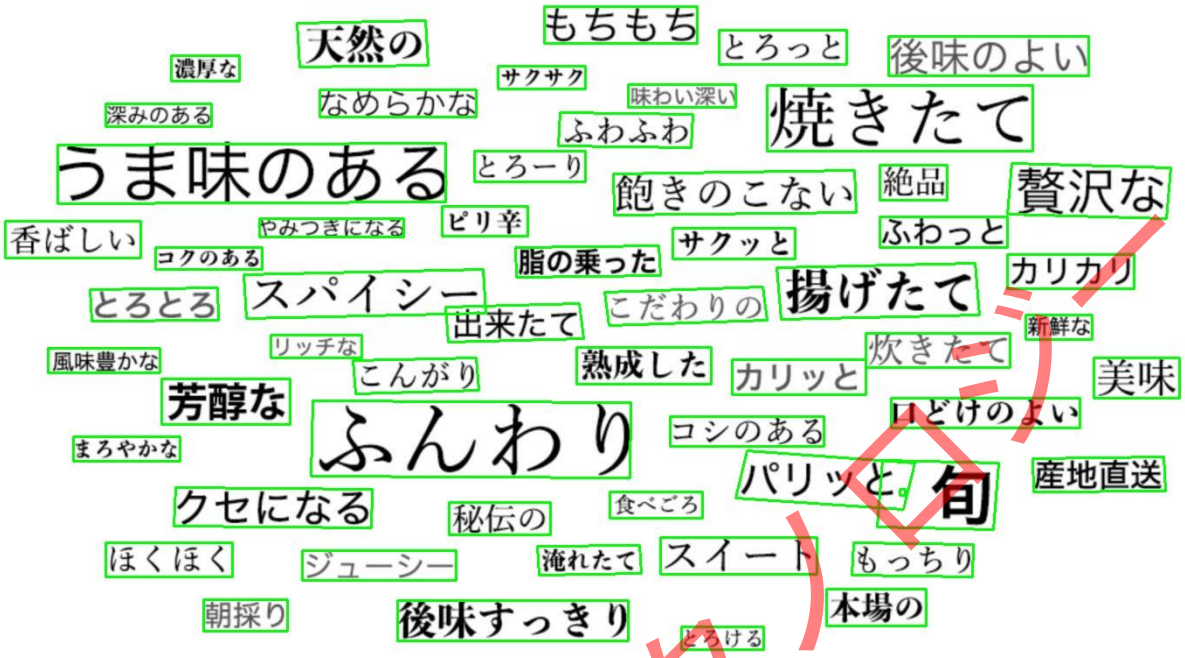
det boxes: [[593,228],[720,228],[720,263],[593,263]]rec text: とろ一り rec score:0.774780 cls label: 0 cls score: 1.000000
det boxes: [[1057,244],[1134,244],[1134,284],[1057,284]]rec text: 絶品 rec score:0.788086 cls label: 0 cls score: 1.000000
det boxes: [[1208,242],[1390,247],[1385,305],[1203,300]]rec text: 贅沢な rec score:0.733398 cls label: 0 cls score: 1.000000
det boxes: [[752,254],[1028,250],[1030,298],[753,301]]rec text: 飽きのこない rec score:0.787760 cls label: 0 cls score: 0.999023
det boxes: [[556,291],[654,291],[654,325],[556,325]]rec text: ピリ辛 rec score:0.663900 cls label: 0 cls score: 0.998047
det boxes: [[347,305],[512,305],[512,326],[347,326]]rec text: やみつきになる rec score:0.788016 cls label: 0 cls score: 0.999023
det boxes: [[1057,303],[1203,303],[1203,338],[1057,338]]rec text: ふわっと rec score:0.787720 cls label: 0 cls score: 0.999512
det boxes: [[820,316],[960,316],[960,351],[820,351]]rec text: サクッと rec score:0.787964 cls label: 0 cls score: 1.000000
det boxes: [[228,338],[350,338],[350,363],[228,363]]rec text: コクのある rec score:0.787305 cls label: 0 cls score: 1.000000
det boxes: [[56,308],[211,308],[211,350],[56,350]]rec text: 香ばしい rec score:0.787475 cls label: 0 cls score: 0.999512
det boxes: [[640,338],[804,336],[806,370],[641,372]]rec text: 脂の乗った rec score:0.787500 cls label: 0 cls score: 1.000000
det boxes: [[1203,346],[1348,346],[1348,385],[1203,385]]rec text: カリカリ rec score:0.788086 cls label: 0 cls score: 1.000000
det boxes: [[329,369],[603,363],[606,413],[332,419]]rec text: スパイシー rec score:0.785937 cls label: 0 cls score: 1.000000
det boxes: [[939,361],[1166,356],[1171,415],[944,420]]rec text: 揚げたて rec score:0.787842 cls label: 0 cls score: 1.000000
det boxes: [[153,386],[299,384],[300,421],[155,422]]rec text: とろとろ rec score:0.788086 cls label: 0 cls score: 0.826172
det boxes: [[742,389],[923,383],[928,421],[747,427]]rec text: こだわりの rec score:0.783203 cls label: 0 cls score: 1.000000
det boxes: [[560,406],[713,402],[718,442],[563,446]]rec text: 出来たて rec score:0.788086 cls label: 0 cls score: 1.000000
det boxes: [[1224,415],[1300,415],[1300,443],[1224,443]]rec text: 新鮮な rec score:0.784668 cls label: 0 cls score: 1.000000
det boxes: [[360,438],[464,438],[464,464],[360,464]]rec text: リッチな rec score:0.788086 cls label: 0 cls score: 1.000000
det boxes: [[1041,436],[1206,436],[1206,475],[1041,475]]rec text: 炊きたて rec score:0.786987 cls label: 0 cls score: 1.000000
det boxes: [[105,453],[233,453],[233,481],[105,481]]rec text: 風味豊かな rec score:0.785254 cls label: 0 cls score: 1.000000
det boxes: [[710,452],[864,452],[864,494],[710,494]]rec text: 熟成した rec score:0.788086 cls label: 0 cls score: 1.000000
det boxes: [[888,466],[1041,466],[1041,506],[888,506]]rec text: カリッと rec score:0.788086 cls label: 0 cls score: 1.000000
det boxes: [[1302,464],[1400,464],[1400,510],[1302,510]]rec text: 美味 rec score:0.788086 cls label: 0 cls score: 1.000000
det boxes: [[235,488],[382,488],[382,540],[235,540]]rec text: 芳醸な rec score:0.667480 cls label: 0 cls score: 1.000000
det boxes: [[454,467],[595,464],[598,501],[457,505]]rec text: こんがり rec score:0.715332 cls label: 0 cls score: 0.999023

```
  det boxes: [[408,513],[772,515],[771,601],[406,599]]rec text: ふんわり rec score:0.785156 cls label: 0 cls score: 1.000000
  det boxes: [[1070,510],[1286,510],[1286,544],[1070,544]]rec text: ロどけのよい rec score:0.788004 cls label: 0 cls score: 0.997070
  det boxes: [[817,531],[993,531],[993,565],[817,565]]rec text: コシのある rec score:0.776465 cls label: 0 cls score: 0.984375
  det boxes: [[134,554],[256,554],[256,582],[134,582]]rec text: まろやかな rec score:0.787500 cls label: 0 cls score: 1.000000
  det boxes: [[905,570],[1097,584],[1084,637],[892,624]]rec text: パリッと rec score:0.788086 cls label: 0 cls score: 0.999512
  det boxes: [[1060,581],[1193,584],[1187,660],[1054,657]]rec text: 旬 rec score:0.522460 cls label: 0 cls score: 1.000000
  det boxes: [[1232,579],[1382,577],[1384,616],[1233,617]]rec text: 産地直送 rec score:0.788086 cls label: 0 cls score: 1.000000
  det boxes: [[251,612],[476,614],[475,658],[249,656]]rec text: クセになる rec score:0.788086 cls label: 0 cls score: 1.000000
  det boxes: [[748,617],[854,617],[854,647],[748,647]]rec text: 食べごろ rec score:0.780639 cls label: 0 cls score: 1.000000
  det boxes: [[1080,614],[1086,614],[1086,621],[1080,621]]rec text: - rec score:0.600097 cls label: 0 cls score: 0.659668
  det boxes: [[564,629],[680,629],[680,666],[564,666]]rec text: 秘伝の rec score:0.788086 cls label: 0 cls score: 1.000000
  det boxes: [[171,675],[316,675],[316,714],[171,714]]rec text: ほくほく rec score:0.702759 cls label: 0 cls score: 0.998047
  det boxes: [[806,669],[985,669],[985,711],[806,711]]rec text: スイート rec score:0.786011 cls label: 0 cls score: 1.000000
  det boxes: [[665,679],[782,678],[784,710],[667,712]]rec text: 流れたて rec score:0.723144 cls label: 0 cls score: 1.000000
  det boxes: [[1027,675],[1164,677],[1163,714],[1025,712]]rec text: もっちり rec score:0.788086 cls label: 0 cls score: 1.000000
  det boxes: [[996,728],[1110,728],[1110,770],[996,770]]rec text: 木場の rec score:0.739746 cls label: 0 cls score: 1.000000
  det boxes: [[396,684],[572,684],[572,718],[396,718]]rec text: ジューシー rec score:0.786719 cls label: 0 cls score: 0.999512
  det boxes: [[283,740],[379,740],[379,776],[283,776]]rec text: 朝採り rec score:0.787760 cls label: 0 cls score: 1.000000
  det boxes: [[505,741],[769,741],[769,787],[505,787]]rec text: 後味すっきり rec score:0.784261 cls label: 0 cls score: 1.000000
  det boxes: [[830,771],[924,771],[924,797],[830,797]]rec text: とろける rec score:0.788086 cls label: 0 cls score: 1.000000

  可視化結果は./visualized_result.jpg に保存されました
```

上記ターミナルに出力された内容により、文字を認識されたことを確認できました。

推論結果画像の表示:



14.5 C++デプロイテスト

前述の python デプロイテストディレクトリファイルと同じ構成で、cpp ディレクトリに移動し、プログラムをコンパイルします:

```
  # デプロイ推論プログラムは FastDeploy または PaddleOCR のソースコードから取得できます（付属の例からも取得可能です）

  # モデルフォルダといくつかのテスト画像をコピーし、現在のディレクトリファイル
  cat@lubancat:~/lubancat_ai_manual_code/example/ppocrv3/rockchip$ ls
  Multilingual_PP-OCRv3_det_infer  ch_ppocr_mobile_v2.0_cls_infer  japan_2.jpg
japan_dict.txt  rknpu2_tools
  README.md                        cpp                             japan_PP-OCRv3_rec_infer
python          visualized_result.jpg

  cat@lubancat:~/pp-ocrv3$ cd cpp
  cat@lubancat:~/pp-ocrv3/cpp$ mkdir build && cd build
  cat@lubancat:~/pp-ocrv3/cpp/build$

  # cmake コマンドを実行し、FASTDEPLOY_INSTALL_DIR に前にコンパイルしてインストールした FastDeploy C++ SDK のディレクトリを指定します。チュートリアルのテストでは、~/FastDeploy/build/fastdeploy-0.0.3`です。
  cat@lubancat:~/lubancat_ai_manual_code/example/ppocrv3/rockchip/cpp/build$ cmake .. -
DFASTDEPLOY_INSTALL_DIR=~/FastDeploy/build/fastdeploy-0.0.3

  # make コマンドを実行してコンパイルします
  cat@lubancat:~/lubancat_ai_manual_code/example/ppocrv3/rockchip/cpp/build$ make -j
  Scanning dependencies of target infer_demo
  [ 50%] Building CXX object CMakeFiles/infer_demo.dir/infer.cc.o
```

```
[100%] Linking CXX executable infer_demo
[100%] Built target infer_demo
```

コンパイルの際に、下記エラーが出られた際：

```
    コンパイルエラー１：
    /usr/bin/ld: warning: libpaddle2onnx.so.1.0.8rc, needed by /home/cat/FastDeploy/build/fastdeploy-0.0.3/lib/libfastdeploy.so, not found (try using -rpath or -rpath-link)
    /usr/bin/ld: CMakeFiles/infer_demo.dir/infer.cc.o: undefined reference to symbol '_ZN2cv6imreadERKNS_6StringEi'
    /usr/bin/ld: /home/cat/FastDeploy/build/fastdeploy-0.0.3/third_libs/install/opencv/lib/libopencv_imgcodecs.so.3.4: error adding symbols: DSO missing from command line
    collect2: error: ld returned 1 exit status
    コンパイルエラー２：
    /usr/bin/ld: warning: libpaddle2onnx.so.1.0.8rc, needed by /home/cat/FastDeploy/build/fastdeploy-0.0.3/lib/libfastdeploy.so, not found (try using -rpath or -rpath-link)
    /usr/bin/ld: /home/cat/FastDeploy/build/fastdeploy-0.0.3/lib/libfastdeploy.so: undefined reference to `paddle2onnx::ConvertFP32ToFP16(char const*, int, char**, int*)'
    /usr/bin/ld: /home/cat/FastDeploy/build/fastdeploy-0.0.3/lib/libfastdeploy.so: undefined reference to `paddle2onnx::Export(void const*, long, void const*, long, char**, int*, int, bool, bool, bool, bool, bool, paddle2onnx::CustomOp*, int, char const*, char**, int*, char const*, bool*, bool, char**, int)'
    collect2: error: ld returned 1 exit status

  #ライブラリパスを追加するように build/CMakeLists.txt を修正

  PROJECT(infer_demo C CXX)
  CMAKE_MINIMUM_REQUIRED (VERSION 3.10)

  # 指定ダウンロード解凍後の fastdeploy ライブラリのパス
  option(FASTDEPLOY_INSTALL_DIR "Path of downloaded fastdeploy sdk.")

  include(${FASTDEPLOY_INSTALL_DIR}/FastDeploy.cmake)

  # FastDeploy 依存ヘッダファイルを追加
  include_directories(${FASTDEPLOY_INCS})

  # OpenCV のインクルードディレクトリを追加
  include_directories(${FASTDEPLOY_INSTALL_DIR}/third_libs/install/opencv/include)
  link_directories(${FASTDEPLOY_INSTALL_DIR}/third_libs/install/opencv/lib)

  # paddle2onnx のインクルードディレクトリを追加
  include_directories(${FASTDEPLOY_INSTALL_DIR}/third_libs/install/paddle2onnx/include)
  link_directories(${FASTDEPLOY_INSTALL_DIR}/third_libs/install/paddle2onnx/lib)

  # rpath を設定して libpaddle2onnx.so を見つける
  set(CMAKE_INSTALL_RPATH_USE_LINK_PATH TRUE)
  set(CMAKE_BUILD_WITH_INSTALL_RPATH TRUE)
  set(CMAKE_INSTALL_RPATH
${FASTDEPLOY_INSTALL_DIR}/lib:${FASTDEPLOY_INSTALL_DIR}/third_libs/install/paddle2onnx/lib)

  add_executable(infer_demo ${PROJECT_SOURCE_DIR}/infer.cc)

  # FastDeploy ライブラリ依存を追加
```

```
  target_link_libraries(infer_demo ${FASTDEPLOY_LIBS}
      paddle2onnx
      opencv_core
      opencv_imgproc
      opencv_highgui
      opencv_imgcodecs
      opencv_videoio)
```

build ディレクトリに infer_demo 推論プログラムが生成され、このプログラムをモデルファイルと同じディレクトリにコピーし、テストを実行:

```
  #現在のディレクトリ
  cat@lubancat:~/lubancat_ai_manual_code/example/ppocrv3/rockchip$ ls
  Multilingual_PP-OCRv3_det_infer  ch_ppocr_mobile_v2.0_cls_infer  infer_demo   japan_PP-
OCRv3_rec_infer  python        visualized_result.jpg
  README.md                        cpp                             japan_2.jpg  japan_dict.txt
rknpu2_tools
  # 推論テスト、infer_demo プログラムの最初の 3 つのパラメータはモデルのパス、4 番目は辞書ファイル、5 番目はテスト画像のパス、最後のパラメータ 1 は npu 推論を指定し、0 は cpu 推論を指定します。詳細は infer_demo.cc のソースコードを参照してください。
  export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:~/FastDeploy/build/fastdeploy-
0.0.3/third_libs/install/paddle2onnx/lib

  cat@lubancat:~/lubancat_ai_manual_code/example/ppocrv3/rockchip$ ./infer_demo
  Usage: infer_demo path/to/det_model path/to/cls_model path/to/rec_model
path/to/rec_label_file path/to/image run_option, e.g. ./infer_demo ./ch_PP-
OCRv3_det_infer ./ch_ppocr_mobile_v2.0_cls_infer ./ch_PP-
OCRv3_rec_infer ./ppocr_keys_v1.txt ./12.jpg 0
  The data type of run_option is int, 0: run with cpu; 1: run with ascend.
```

CPU 推論を実行

```
  #CPU 推論
  cat@lubancat:~/lubancat_ai_manual_code/example/ppocrv3/rockchip$ ./infer_demo ./Multilingual_
PP-OCRv3_det_infer/Multilingual_PP-OCRv3_det_infer.onnx ¥
                    ./ch_ppocr_mobile_v2.0_cls_infer/ch_ppocr_mobile_v2.0_cls_infer.onnx ¥
                    ./japan_PP-OCRv3_rec_infer/japan_PP-OCRv3_rec_infer.onnx ¥
                    ./japan_dict.txt ¥
                    ./japan_2.jpg ¥
                    0
  ONNX Model
  [INFO] fastdeploy/runtime/runtime.cc(326)::CreateOrtBackend     Runtime initialized with
Backend::ORT in Device::CPU.
  [INFO] fastdeploy/runtime/runtime.cc(326)::CreateOrtBackend     Runtime initialized with
Backend::ORT in Device::CPU.
  [INFO] fastdeploy/runtime/runtime.cc(326)::CreateOrtBackend     Runtime initialized with
Backend::ORT in Device::CPU.
  det boxes: [[673,63],[848,63],[848,105],[673,105]]rec text: もちもち rec score:0.999996 cls
label: 0 cls score: 1.000000
```

```
  det boxes: [[390,82],[537,79],[540,130],[393,132]]rec text: 天然の rec score:0.997521 cls
label: 0 cls score: 1.000000
  det boxes: [[873,90],[1019,90],[1019,129],[873,129]]rec text: とろっと rec score:0.999989 cls
label: 0 cls score: 0.961403
  det boxes: [[1067,97],[1296,97],[1296,142],[1067,142]]rec text: 後味のよい rec score:0.988884
cls label: 0 cls score: 1.000000
  det boxes: [[246,119],[324,119],[324,148],[246,148]]rec text: 濃厚な rec score:0.997802 cls
label: 0 cls score: 0.999997
  det boxes: [[620,131],[720,131],[720,157],[620,157]]rec text: サクサク rec score:0.995820 cls
label: 0 cls score: 0.999996
  det boxes: [[769,152],[892,152],[892,178],[769,178]]rec text: 味わい深い rec score:0.998777
cls label: 0 cls score: 1.000000
  det boxes: [[928,153],[1232,153],[1232,228],[928,228]]rec text: 焼きたて rec score:0.998969
cls label: 0 cls score: 0.999998
  det boxes: [[171,174],[292,174],[292,200],[171,200]]rec text: 深みのある rec score:0.999420
cls label: 0 cls score: 0.999995
  det boxes: [[416,158],[595,158],[595,189],[416,189]]rec text: なめらかな rec score:0.999909
cls label: 0 cls score: 0.999991
  det boxes: [[691,184],[843,186],[841,224],[689,222]]rec text: ふわふわ rec score:0.996401 cls
label: 0 cls score: 0.999944
  det boxes: [[120,221],[558,221],[558,286],[120,286]]rec text: うま味のある rec score:0.999165
cls label: 0 cls score: 0.999845
  det boxes: [[593,228],[720,228],[720,263],[593,263]]rec text: とろ一り rec score:0.978696 cls
label: 0 cls score: 0.999995
  det boxes: [[1057,244],[1134,244],[1134,284],[1057,284]]rec text: 絶品 rec score:0.999971 cls
label: 0 cls score: 0.999925
  det boxes: [[1208,242],[1390,247],[1385,305],[1203,300]]rec text: 贅沢な rec score:0.919864
cls label: 0 cls score: 1.000000
  det boxes: [[752,254],[1028,250],[1030,298],[753,301]]rec text: 飽きのこない rec
score:0.999389 cls label: 0 cls score: 0.999356
  det boxes: [[556,291],[654,291],[654,325],[556,325]]rec text: ピリ辛 rec score:0.822031 cls
label: 0 cls score: 0.999186
  det boxes: [[347,305],[512,305],[512,326],[347,326]]rec text: やみつきになる rec
score:0.999725 cls label: 0 cls score: 0.999317
  det boxes: [[1057,303],[1203,303],[1203,338],[1057,338]]rec text: ふわっと rec score:0.999295
cls label: 0 cls score: 0.999348
  det boxes: [[820,316],[960,316],[960,351],[820,351]]rec text: サクッと rec score:0.999706 cls
label: 0 cls score: 0.999997
  det boxes: [[228,338],[350,338],[350,363],[228,363]]rec text: コクのある rec score:0.998522
cls label: 0 cls score: 0.999818
  det boxes: [[56,308],[211,308],[211,350],[56,350]]rec text: 香ばしい rec score:0.998818 cls
label: 0 cls score: 0.999747
  det boxes: [[640,338],[804,336],[806,370],[641,372]]rec text: 脂の乗った rec score:0.998902
cls label: 0 cls score: 1.000000
  det boxes: [[1203,346],[1348,346],[1348,385],[1203,385]]rec text: カリカリ rec score:0.999788
cls label: 0 cls score: 1.000000
  det boxes: [[329,369],[603,363],[606,413],[332,419]]rec text: スパイシー rec score:0.995897
cls label: 0 cls score: 0.999993
  det boxes: [[939,361],[1166,356],[1171,415],[944,420]]rec text: 揚げたて rec score:0.999562
cls label: 0 cls score: 1.000000
  det boxes: [[153,386],[299,384],[300,421],[155,422]]rec text: とろとろ rec score:0.999991 cls
label: 0 cls score: 0.826810
```

```
  det boxes: [[742,389],[923,383],[928,421],[747,427]]rec text: こだわりの rec score:0.991786
cls label: 0 cls score: 0.999864
  det boxes: [[560,406],[713,402],[718,442],[563,446]]rec text: 出来たて rec score:0.999884 cls
label: 0 cls score: 1.000000
  det boxes: [[1224,415],[1300,415],[1300,443],[1224,443]]rec text: 新鮮な rec score:0.994382
cls label: 0 cls score: 1.000000
  det boxes: [[360,438],[464,438],[464,464],[360,464]]rec text: リッチな rec score:0.999920 cls
label: 0 cls score: 1.000000
  det boxes: [[1041,436],[1206,436],[1206,475],[1041,475]]rec text: 炊きたて rec score:0.998127
cls label: 0 cls score: 1.000000
  det boxes: [[105,454],[233,452],[235,481],[107,483]]rec text: 風味豊かな rec score:0.999369
cls label: 0 cls score: 1.000000
  det boxes: [[710,452],[864,452],[864,494],[710,494]]rec text: 熟成した rec score:0.999973 cls
label: 0 cls score: 1.000000
  det boxes: [[888,466],[1041,466],[1041,506],[888,506]]rec text: カリッと rec score:0.999974
cls label: 0 cls score: 1.000000
  det boxes: [[1302,464],[1400,464],[1400,510],[1302,510]]rec text: 美味 rec score:0.999916 cls
label: 0 cls score: 1.000000
  det boxes: [[233,488],[382,488],[382,540],[233,540]]rec text: 芳醸な rec score:0.897710 cls
label: 0 cls score: 1.000000
  det boxes: [[454,467],[595,464],[598,501],[457,505]]rec text: こんがり rec score:0.890585 cls
label: 0 cls score: 0.999028
  det boxes: [[408,513],[772,515],[771,601],[406,599]]rec text: ふんわり rec score:0.994684 cls
label: 0 cls score: 0.999797
  det boxes: [[1070,510],[1286,510],[1286,544],[1070,544]]rec text: ロどけのよい rec
score:0.999709 cls label: 0 cls score: 0.996961
  det boxes: [[817,531],[993,531],[993,565],[817,565]]rec text: コシのある rec score:0.981016
cls label: 0 cls score: 0.985237
  det boxes: [[134,554],[256,554],[256,582],[134,582]]rec text: まろやかな rec score:0.998829
cls label: 0 cls score: 1.000000
  det boxes: [[905,570],[1097,584],[1084,637],[892,624]]rec text: パリッと rec score:0.999985
cls label: 0 cls score: 0.999667
  det boxes: [[1060,581],[1193,584],[1187,660],[1054,657]]rec text: 旬 rec score:0.619156 cls
label: 0 cls score: 1.000000
  det boxes: [[1232,579],[1382,577],[1384,616],[1233,617]]rec text: 産地直送 rec score:0.999997
cls label: 0 cls score: 1.000000
  det boxes: [[251,612],[476,614],[475,658],[249,656]]rec text: クセになる rec score:0.999891
cls label: 0 cls score: 1.000000
  det boxes: [[748,617],[854,617],[854,647],[748,647]]rec text: 食べごろ rec score:0.987618 cls
label: 0 cls score: 0.999981
  det boxes: [[1080,614],[1086,614],[1086,621],[1080,621]]rec text: - rec score:0.697572 cls
label: 0 cls score: 0.571474
  det boxes: [[564,629],[680,629],[680,666],[564,666]]rec text: 秘伝の rec score:0.999969 cls
label: 0 cls score: 1.000000
  det boxes: [[171,675],[316,675],[316,714],[171,714]]rec text: ほくほく rec score:0.876584 cls
label: 0 cls score: 0.997420
  det boxes: [[806,669],[985,669],[985,711],[806,711]]rec text: スイート rec score:0.996261 cls
label: 0 cls score: 1.000000
  det boxes: [[665,679],[782,678],[784,710],[667,712]]rec text: 流れたて rec score:0.900919 cls
label: 0 cls score: 0.999998
  det boxes: [[1027,675],[1164,677],[1163,714],[1025,712]]rec text: もっちり rec score:0.999722
cls label: 0 cls score: 0.999996
```

```
  det boxes: [[996,728],[1110,728],[1110,770],[996,770]]rec text: 木場の rec score:0.926558 cls
label: 0 cls score: 1.000000
  det boxes: [[396,684],[572,684],[572,718],[396,718]]rec text: ジューシー rec score:0.997624
cls label: 0 cls score: 0.999645
  det boxes: [[283,740],[379,740],[379,776],[283,776]]rec text: 朝採り rec score:0.999268 cls
label: 0 cls score: 1.000000
  det boxes: [[505,741],[769,741],[769,787],[505,787]]rec text: 後味すっきり rec score:0.993246
cls label: 0 cls score: 1.000000
  det boxes: [[830,771],[924,771],[924,797],[830,797]]rec text: とろける rec score:0.999871 cls
label: 0 cls score: 1.000000

  Visualized result saved in ./vis_result.jpg
```

NPU 推論を実行:

```
  #NPU 推論
  cat@lubancat:~/lubancat_ai_manual_code/example/ppocrv3/rockchip$ ./infer_demo ./Multilingual_
PP-OCRv3_det_infer/Multilingual_PP-OCRv3_det_infer_rk3588_unquantized.rknn ¥
                        ./ch_ppocr_mobile_v2.0_cls_infer/ch_ppocr_mobile_v20_cls_infer_rk3588_u
nquantized.rknn ¥
                        ./japan_PP-OCRv3_rec_infer/japan_PP-
OCRv3_rec_infer_rk3588_unquantized.rknn ¥
                        ./japan_dict.txt ¥
                        ./japan_2.jpg ¥
                        1
  [INFO] fastdeploy/runtime/backends/rknpu2/rknpu2_backend.cc(81)::GetSDKAndDeviceVersion
rknpu2 runtime version: 2.0.0b0 (35a6907d79@2024-03-24T10:31:14)
  [INFO] fastdeploy/runtime/backends/rknpu2/rknpu2_backend.cc(82)::GetSDKAndDeviceVersion
rknpu2 driver version: 0.9.2
  index=0, name=x, n_dims=4, dims=[1, 960, 960, 3], n_elems=2764800, size=5529600, fmt=NHWC,
type=FP16, qnt_type=AFFINE, zp=0, scale=1.000000, pass_through=0
  index=0, name=sigmoid_0.tmp_0, n_dims=4, dims=[1, 1, 960, 960], n_elems=921600, size=1843200,
fmt=NCHW, type=FP32, qnt_type=AFFINE, zp=0, scale=1.000000, pass_through=0
  [INFO] fastdeploy/runtime/runtime.cc(367)::CreateRKNPU2Backend  Runtime initialized with
Backend::RKNPU2 in Device::RKNPU.
  [INFO] fastdeploy/runtime/backends/rknpu2/rknpu2_backend.cc(81)::GetSDKAndDeviceVersion
rknpu2 runtime version: 2.0.0b0 (35a6907d79@2024-03-24T10:31:14)
  [INFO] fastdeploy/runtime/backends/rknpu2/rknpu2_backend.cc(82)::GetSDKAndDeviceVersion
rknpu2 driver version: 0.9.2
  index=0, name=x, n_dims=4, dims=[1, 48, 192, 3], n_elems=27648, size=55296, fmt=NHWC,
type=FP16, qnt_type=AFFINE, zp=0, scale=1.000000, pass_through=0
  index=0, name=softmax_0.tmp_0, n_dims=2, dims=[1, 2, 0, 0], n_elems=2, size=4, fmt=UNDEFINED,
type=FP32, qnt_type=AFFINE, zp=0, scale=1.000000, pass_through=0
  [INFO] fastdeploy/runtime/runtime.cc(367)::CreateRKNPU2Backend  Runtime initialized with
Backend::RKNPU2 in Device::RKNPU.
  [INFO] fastdeploy/runtime/backends/rknpu2/rknpu2_backend.cc(81)::GetSDKAndDeviceVersion
rknpu2 runtime version: 2.0.0b0 (35a6907d79@2024-03-24T10:31:14)
```

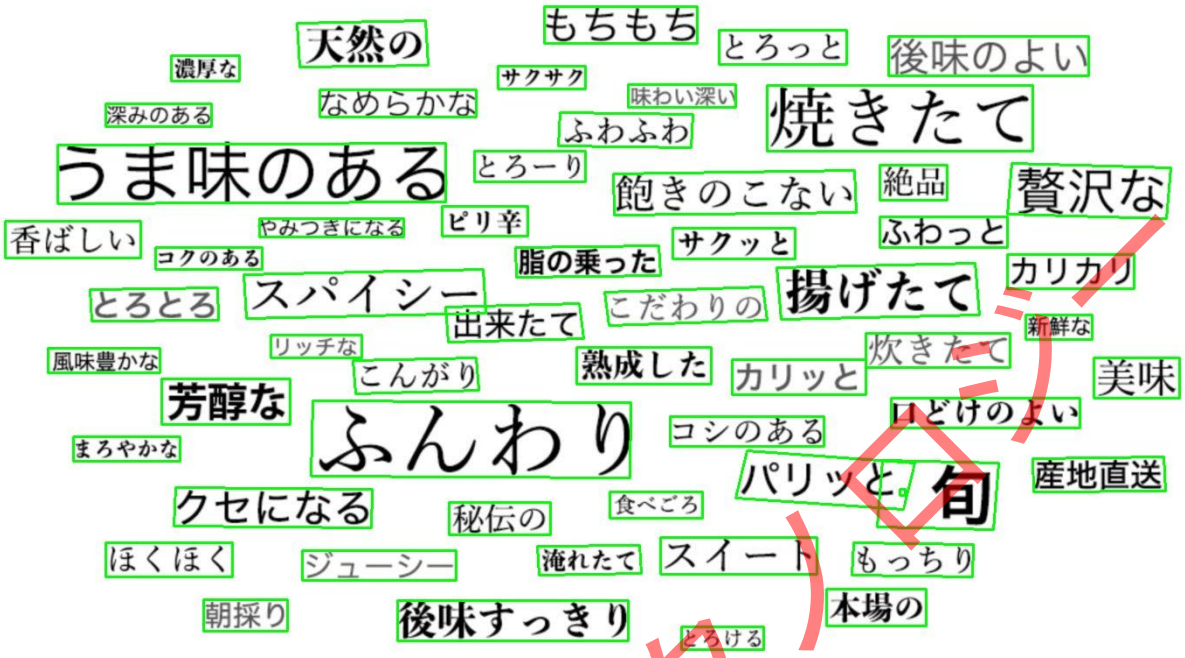
```
[INFO] fastdeploy/runtime/backends/rknpu2/rknpu2_backend.cc(82)::GetSDKAndDeviceVersion
rknpu2 driver version: 0.9.2
  index=0, name=x, n_dims=4, dims=[1, 48, 320, 3], n_elems=46080, size=92160, fmt=NHWC,
type=FP16, qnt_type=AFFINE, zp=0, scale=1.000000, pass_through=0
  index=0, name=softmax_2.tmp_0, n_dims=3, dims=[1, 40, 4401, 0], n_elems=176040, size=352080,
fmt=UNDEFINED, type=FP32, qnt_type=AFFINE, zp=0, scale=1.000000, pass_through=0
[INFO] fastdeploy/runtime/runtime.cc(367)::CreateRKNPU2Backend  Runtime initialized with
Backend::RKNPU2 in Device::RKNPU.
[WARNING] fastdeploy/runtime/backends/rknpu2/rknpu2_backend.cc(420)::InitRKNNTensorMemory
The input tensor type != model's inputs type.The input_type need FP16,but inputs[0].type is
UINT8
[WARNING] fastdeploy/runtime/backends/rknpu2/rknpu2_backend.cc(420)::InitRKNNTensorMemory
The input tensor type != model's inputs type.The input_type need FP16,but inputs[0].type is
UINT8
[WARNING] fastdeploy/runtime/backends/rknpu2/rknpu2_backend.cc(420)::InitRKNNTensorMemory
The input tensor type != model's inputs type.The input_type need FP16,but inputs[0].type is
UINT8
det boxes: [[673,63],[848,63],[848,105],[673,105]]rec text: もちもち rec score:0.788086 cls
label: 0 cls score: 1.000000
det boxes: [[390,82],[537,79],[540,130],[393,132]]rec text: 天然の rec score:0.786621 cls
label: 0 cls score: 1.000000
det boxes: [[873,90],[1019,90],[1019,129],[873,129]]rec text: とろっと rec score:0.788086 cls
label: 0 cls score: 0.970703
det boxes: [[1067,97],[1296,97],[1296,142],[1067,142]]rec text: 後味のよい rec score:0.779297
cls label: 0 cls score: 1.000000
det boxes: [[246,119],[324,119],[324,148],[246,148]]rec text: 濃厚な rec score:0.786784 cls
label: 0 cls score: 1.000000
det boxes: [[620,131],[720,131],[720,157],[620,157]]rec text: サクサク rec score:0.785766 cls
label: 0 cls score: 1.000000
det boxes: [[769,152],[892,152],[892,178],[769,178]]rec text: 味わい深い rec score:0.787695
cls label: 0 cls score: 1.000000
det boxes: [[928,153],[1232,153],[1232,228],[928,228]]rec text: 焼きたて rec score:0.787720
cls label: 0 cls score: 1.000000
det boxes: [[171,174],[292,174],[292,200],[171,200]]rec text: 深みのある rec score:0.787695
cls label: 0 cls score: 1.000000
det boxes: [[416,158],[595,158],[595,189],[416,189]]rec text: なめらかな rec score:0.788086
cls label: 0 cls score: 1.000000
det boxes: [[691,184],[843,186],[841,224],[689,222]]rec text: ふわふわ rec score:0.786011 cls
label: 0 cls score: 1.000000
det boxes: [[116,221],[560,219],[560,286],[116,288]]rec text: うま味のある rec score:0.787842
cls label: 0 cls score: 1.000000
det boxes: [[593,228],[720,228],[720,263],[593,263]]rec text: とろ一り rec score:0.774780 cls
label: 0 cls score: 1.000000
det boxes: [[1057,244],[1134,244],[1134,284],[1057,284]]rec text: 絶品 rec score:0.788086 cls
label: 0 cls score: 1.000000
det boxes: [[1208,242],[1390,247],[1385,305],[1203,300]]rec text: 贅沢な rec score:0.733398
cls label: 0 cls score: 1.000000
det boxes: [[752,254],[1028,250],[1030,298],[753,301]]rec text: 飽きのこない rec
score:0.787760 cls label: 0 cls score: 0.999023
det boxes: [[556,291],[654,291],[654,325],[556,325]]rec text: ピリ辛 rec score:0.663900 cls
label: 0 cls score: 0.998047
det boxes: [[347,305],[512,305],[512,326],[347,326]]rec text: やみつきになる rec
```

score:0.788016 cls label: 0 cls score: 0.999023
det boxes: [[1057,303],[1203,303],[1203,338],[1057,338]]rec text: ふわっと rec score:0.787720 cls label: 0 cls score: 0.999512
det boxes: [[820,316],[960,316],[960,351],[820,351]]rec text: サクッと rec score:0.787964 cls label: 0 cls score: 1.000000
det boxes: [[228,338],[350,338],[350,363],[228,363]]rec text: コクのある rec score:0.787305 cls label: 0 cls score: 1.000000
det boxes: [[56,308],[211,308],[211,350],[56,350]]rec text: 香ばしい rec score:0.787475 cls label: 0 cls score: 0.999512
det boxes: [[640,338],[804,336],[806,370],[641,372]]rec text: 脂の乗った rec score:0.787500 cls label: 0 cls score: 1.000000
det boxes: [[1203,346],[1348,346],[1348,385],[1203,385]]rec text: カリカリ rec score:0.788086 cls label: 0 cls score: 1.000000
det boxes: [[329,369],[603,363],[606,413],[332,419]]rec text: スパイシー rec score:0.785937 cls label: 0 cls score: 1.000000
det boxes: [[939,361],[1166,356],[1171,415],[944,420]]rec text: 揚げたて rec score:0.787842 cls label: 0 cls score: 1.000000
det boxes: [[153,386],[299,384],[300,421],[155,422]]rec text: とろとろ rec score:0.788086 cls label: 0 cls score: 0.826172
det boxes: [[742,389],[923,383],[928,421],[747,427]]rec text: こだわりの rec score:0.783203 cls label: 0 cls score: 1.000000
det boxes: [[560,406],[713,402],[718,442],[563,446]]rec text: 出来たて rec score:0.788086 cls label: 0 cls score: 1.000000
det boxes: [[1224,415],[1300,415],[1300,443],[1224,443]]rec text: 新鮮な rec score:0.784668 cls label: 0 cls score: 1.000000
det boxes: [[360,438],[464,438],[464,464],[360,464]]rec text: リッチな rec score:0.788086 cls label: 0 cls score: 1.000000
det boxes: [[1041,436],[1206,436],[1206,475],[1041,475]]rec text: 炊きたて rec score:0.786987 cls label: 0 cls score: 1.000000
det boxes: [[105,453],[233,453],[233,481],[105,481]]rec text: 風味豊かな rec score:0.785254 cls label: 0 cls score: 1.000000
det boxes: [[710,452],[864,452],[864,494],[710,494]]rec text: 熟成した rec score:0.788086 cls label: 0 cls score: 1.000000
det boxes: [[888,466],[1041,466],[1041,506],[888,506]]rec text: カリッと rec score:0.788086 cls label: 0 cls score: 1.000000
det boxes: [[1302,464],[1400,464],[1400,510],[1302,510]]rec text: 美味 rec score:0.788086 cls label: 0 cls score: 1.000000
det boxes: [[235,488],[382,488],[382,540],[235,540]]rec text: 芳醸な rec score:0.667480 cls label: 0 cls score: 1.000000
det boxes: [[454,467],[595,464],[598,501],[457,505]]rec text: こんがり rec score:0.715332 cls label: 0 cls score: 0.999023
det boxes: [[408,513],[772,515],[771,601],[406,599]]rec text: ふんわり rec score:0.785156 cls label: 0 cls score: 1.000000
det boxes: [[1070,510],[1286,510],[1286,544],[1070,544]]rec text: 口どけのよい rec score:0.788004 cls label: 0 cls score: 0.997070
det boxes: [[817,531],[993,531],[993,565],[817,565]]rec text: コシのある rec score:0.776465 cls label: 0 cls score: 0.984375
det boxes: [[134,554],[256,554],[256,582],[134,582]]rec text: まろやかな rec score:0.787500 cls label: 0 cls score: 1.000000
det boxes: [[905,570],[1097,584],[1084,637],[892,624]]rec text: パリッと rec score:0.788086 cls label: 0 cls score: 0.999512
det boxes: [[1060,581],[1193,584],[1187,660],[1054,657]]rec text: 旬 rec score:0.522460 cls

```
label: 0 cls score: 1.000000
  det boxes: [[1232,579],[1382,577],[1384,616],[1233,617]]rec text: 産地直送 rec score:0.788086
cls label: 0 cls score: 1.000000
  det boxes: [[251,612],[476,614],[475,658],[249,656]]rec text: クセになる rec score:0.788086
cls label: 0 cls score: 1.000000
  det boxes: [[748,617],[854,617],[854,647],[748,647]]rec text: 食べごろ rec score:0.780639 cls
label: 0 cls score: 1.000000
  det boxes: [[1080,614],[1086,614],[1086,621],[1080,621]]rec text: - rec score:0.600097 cls
label: 0 cls score: 0.659668
  det boxes: [[564,629],[680,629],[680,666],[564,666]]rec text: 秘伝の rec score:0.788086 cls
label: 0 cls score: 1.000000
  det boxes: [[171,675],[316,675],[316,714],[171,714]]rec text: ほくほく rec score:0.702759 cls
label: 0 cls score: 0.998047
  det boxes: [[806,669],[985,669],[985,711],[806,711]]rec text: スイート rec score:0.786011 cls
label: 0 cls score: 1.000000
  det boxes: [[665,679],[782,678],[784,710],[667,712]]rec text: 流れたて rec score:0.723144 cls
label: 0 cls score: 1.000000
  det boxes: [[1027,675],[1164,677],[1163,714],[1025,712]]rec text: もっちり rec score:0.788086
cls label: 0 cls score: 1.000000
  det boxes: [[996,728],[1110,728],[1110,770],[996,770]]rec text: 木場の rec score:0.739746 cls
label: 0 cls score: 1.000000
  det boxes: [[396,684],[572,684],[572,718],[396,718]]rec text: ジューシー rec score:0.786719
cls label: 0 cls score: 0.999512
  det boxes: [[283,740],[379,740],[379,776],[283,776]]rec text: 朝採り rec score:0.787760 cls
label: 0 cls score: 1.000000
  det boxes: [[505,741],[769,741],[769,787],[505,787]]rec text: 後味すっきり rec score:0.784261
cls label: 0 cls score: 1.000000
  det boxes: [[830,771],[924,771],[924,797],[830,797]]rec text: とろける rec score:0.788086 cls
label: 0 cls score: 1.000000

  Visualized result saved in ./vis_result.jpg
```

推論結果画像 vis_result.jpg:

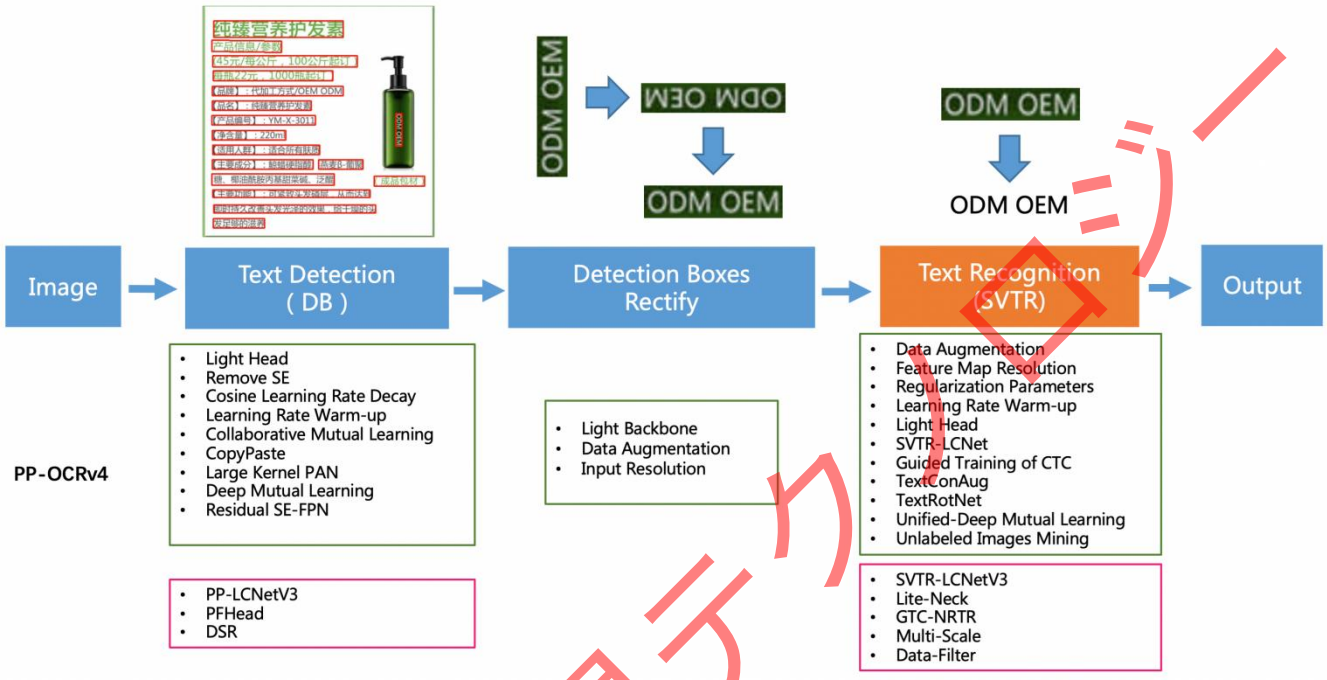


14.6 参考リンク

- [FastDeploy]
- [PaddleOCR]
- [rknn-toolkit2]

第 15 章 PP-ORCv4

PP-OCRv4 は PP-OCRv3 を基にさらにアップグレードされました。全体のフレームワークは PP-OCRv3 と同じパイプラインを維持し、検出モデルと認識モデルのデータ、ネットワーク構造、トレーニング戦略などの複数のモジュールを最適化しました。PP-OCRv4 システムのブロック図は以下の通りです。



効果として、速度が同等の場合、複数のシナリオで精度が大幅に向上しました：

- 中国語のシナリオでは、PP-OCRv3 中国語モデルと比較して 4%以上の向上。
- 英数字のシナリオでは、PP-OCRv3 英語モデルと比較して 6%の向上。
- 多言語のシナリオでは、80 の言語の認識効果を最適化し、平均精度が 8%以上向上しました。本章ではデプロイ環境を簡単に紹介し、Lubancat4 ボード上で PP-OCRv4 をデプロイして実行する方法をデモします。

注：本節は中国語認識しかありません。PP-ORCv3 を参照のうえ、日本語認識を行ってください。

15.1 環境インストール

PC Ubuntu システムに PaddleOCR 環境を作成し、関連ツールをインストールします。

```
# conda を使用して仮想環境を作成
conda create -n PaddleOCR python=3.8
conda activate PaddleOCR
```

PaddleOCR のソースコードを取得：

```
# conda を使用して仮想環境を作成
git clone https://github.com/PaddlePaddle/PaddleOCR.git
cd PaddleOCR
```

```
# 必要のパッケージをインストール
pip install -r requirements.txt

# toolkit2 のインストールは前の≪第 3 章 RKNN Toolkit2 紹介≫を参考
```

PP-OCRv4 のトレーニングと評価方法については、PaddleOCR のドキュメント（OCRv4 日本語マニュアルまだない）を参照してください。

15.2 モデルの準備

PP-OCRv4 モデルは PaddleOCR のメインページから直接ダウンロードできます：

または、チュートリアルの付属例から取得し、ターミナルで直接コマンドを使用してダウンロードできます：

```
# モデルダウンロード
wget -c https://paddleocr.bj.bcebos.com/PP-OCRv4/chinese/ch_PP-OCRv4_det_infer.tar
wget -c https://paddleocr.bj.bcebos.com/PP-OCRv4/chinese/ch_PP-OCRv4_rec_infer.tar
wget -c https://paddleocr.bj.bcebos.com/dygraph_v2.0/ch/ch_ppocr_mobile_v2.0_cls_infer.tar
# 解凍
tar -xvf ch_PP-OCRv4_rec_infer.tar
tar -xvf ch_PP-OCRv4_det_infer.tar
tar -xvf ch_ppocr_mobile_v2.0_cls_infer.tar
```

取得したモデルには、テキスト検出モデル、方向分類モデル、およびテキスト認識モデルの 3 つがあります：

- テキスト検出モデル：DB ベースの検出モデルで、画像内のテキスト領域を検出できます。
- 方向分類モデル：テキストの方向を判断し、補正を行い、後続のテキスト認識を容易にします。
- テキスト認識モデル：検出された枠内のテキストを認識し、各テキスト枠内の内容を取得します。

環境に paddle2onnx をインストールし、取得したテキスト検出モデルを onnx モデルに変換します：

```
# 検出モデルが Paddle 形式から ONNX に変換
(.toolkit2_env) (base) csun@CSUN-PC-
0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/ppocrv4$ paddle2onnx --
model_dir ./model/ch_PP-OCRv4_det_infer ¥
  --model_filename inference.pdmodel ¥
  --params_filename inference.pdiparams ¥
  --save_file ./model/ch_PP-OCRv4_det_infer/model.onnx ¥
  --opset_version 12
# Setting fix input shape
python -m paddle2onnx.optimize --input_model model/ch_PP-OCRv4_det_infer/model.onnx ¥
                               --output_model model/ch_PP-OCRv4_det_infer/ppocrv4_det.onnx ¥
                               --input_shape_dict "{'x':[1,3,480,480]}"
```

テキスト認識モデルを onnx モデルに変換し、入力の形状を固定

```
  # 認識モデルが Paddle 形式から ONNX に変換
  (.toolkit2_env) (base) csun@CSUN-PC-
0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/ppocrv4$ paddle2onnx --
model_dir ./model/ch_PP-OCRv4_rec_infer ¥
  --model_filename inference.pdmodel ¥
  --params_filename inference.pdiparams ¥
  --save_file ./model/ch_PP-OCRv4_rec_infer/model.onnx ¥
  --opset_version 12
# Seting fix input shape
python -m paddle2onnx.optimize --input_model model/ch_PP-OCRv4_rec_infer/model.onnx ¥
                          --output_model model/ch_PP-OCRv4_rec_infer/ppocrv4_rec.onnx ¥
                          --input_shape_dict "{'x':[1,3,48,320]}"
```

方向分類モデルを onnx モデルに変換

```
  # 方向分類モデルが Paddle 形式から ONNX に変換
  (.toolkit2_env) (base) csun@CSUN-PC-
0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/ppocrv4$ paddle2onnx --
model_dir ./model/ch_ppocr_mobile_v2.0_cls_infer ¥
              --model_filename inference.pdmodel ¥
              --params_filename inference.pdiparams ¥
              --
save_file ./model/ch_ppocr_mobile_v2.0_cls_infer/ch_ppocr_mobile_v2.0_cls_infer.onnx
# Seting fix input shape
python -m paddle2onnx.optimize --
input_model ./model/ch_ppocr_mobile_v2.0_cls_infer/ch_ppocr_mobile_v2.0_cls_infer.onnx ¥
                          --
output_model ./model/ch_ppocr_mobile_v2.0_cls_infer/ch_ppocr_mobile_v2.0_cls_infer.onnx ¥
                          --input_shape_dict "{'x':[1,3,48,192]}"
```

15.2.1 rknn モデル

チュートリアルでは、lubancat-4 の npu 上でデプロイをテストします。toolkit2 を使用してモデルを rknn モデルに変換する必要があります。モデル変換プログラムは、付属のチュートリアルコードリポジトリを参照してください。

```
  # プログラムの取得
wget https://www.dragonwake.com/download/LubanCat4/7-srccode/tutorial/CSAIEG358803_rk_code_storage.zip
  # 解凍後：サブフォルダー「lubancat_ai_manual_code」のなか、すべてソースがマニュアルに使われるものです。
  cd lubancat_ai_manual_code/example/ppocrv4

  # toolkit2 環境での文字認識モデルの変換
  (.toolkit2_env) (base) csun@CSUN-PC-0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/ppocrv4$ python convert_rec.py
  I rknn-toolkit2 version: 2.0.0b0+9bab5682
  --> Config model
  done
  --> Loading model
  I It is recommended onnx opset 19, but your onnx model opset is 12!
  I Loading : 100%|████████████████████████████████████████| 420/420 [00:00<00:00, 148558.58it/s]
  W load_onnx: The config.mean_values is None, zeros will be set for input 0!
  W load_onnx: The config.std_values is None, ones will be set for input 0!
  done
  --> Building model
  I rknn building ...
  I rknn buiding done.
  done
  --> Export rknn model
  done

  # 文字検出モデルの変換
  (.toolkit2_env) (base) csun@CSUN-PC-0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/ppocrv4$ python convert_rec.py
  I rknn-toolkit2 version: 2.0.0b0+9bab5682
  --> Config model
  done
  --> Loading model
  I It is recommended onnx opset 19, but your onnx model opset is 12!
  I Loading : 100%|████████████████████████████████████████| 420/420 [00:00<00:00, 148558.58it/s]
  W load_onnx: The config.mean_values is None, zeros will be set for input 0!
  W load_onnx: The config.std_values is None, ones will be set for input 0!
  done
  --> Building model
  I rknn building ...
  I rknn buiding done.
```

```
  done
  --> Export rknn model
  done
  (.toolkit2_env) (base) csun@CSUN-PC-
0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/ppocrv4$ python
convert_det.py
  I rknn-toolkit2 version: 2.0.0b0+9bab5682
  --> Config model
  done
  --> Loading model
  I It is recommended onnx opset 19, but your onnx model opset is 12!
  I Loading : 100%|████████████████████████████████████████████| 342/342 [00:00<00:00,
179328.91it/s]
  done
  --> Building model
  W build: found outlier value, this may affect quantization accuracy
                        const name          abs_mean    abs_std     outlier value
                        conv2d_398.w_0      6.53        8.93        58.173
                        conv2d_402.w_0      2.41        3.73        34.766
                        conv2d_403.w_0      0.14        0.16        9.510
                        conv2d_406.w_0      0.44        0.87        13.446
                        conv2d_412.w_0      0.29        0.86        -44.813
                        conv2d_416.w_0      0.37        0.94        29.860
                        conv2d_418.w_0      0.37        0.62        41.572
                        conv2d_420.w_0      0.33        0.69        -25.894
                        conv2d_421.w_0      0.08        0.09        11.623
  I GraphPreparing : 100%|████████████████████████████████████| 214/214 [00:00<00:00,
5953.31it/s]
  I Quantizating : 100%|██████████████████████████████████████| 214/214 [00:03<00:00,
60.30it/s]
  W build: The default input dtype of 'x' is changed from 'float32' to 'int8' in rknn model for
performance!
                        Please take care of this change when deploy rknn model with Runtime
API!
  W build: The default output dtype of 'sigmoid_0.tmp_0' is changed from 'float32' to 'int8' in
rknn model for performance!
                        Please take care of this change when deploy rknn model with Runtime
API!
  I rknn building ...
  I rknn building done.
  done
  --> Export rknn model
  done


  # 方向分類モデル
  (.toolkit2_env) (base) csun@CSUN-PC-
0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/ppocrv4$ python
convert_cls.py
  I rknn-toolkit2 version: 2.0.0b0+9bab5682
  --> Config model
  done
```

```
  --> Loading model
  I It is recommended onnx opset 19, but your onnx model opset is 9!
  I Loading : 100%|████████████████████████████████████████████| 250/250 [00:00<00:00,
295790.13it/s]
  W load_onnx: The config.mean_values is None, zeros will be set for input 0!
  W load_onnx: The config.std_values is None, ones will be set for input 0!
  done
  --> Building model
  I rknn building ...
  I rknn buiding done.
  done
  --> Export rknn model
  done
```

生成された rknn モデルはすべて model ディレクトリに保存されます。

15.3 ボード側のデプロイテスト

ボード上で例を取得し、前述の rknn モデルをボードに転送して、例の model ディレクトリに配置します。詳細な rknn 例の使用については、rknn_model_zoo を参照してください。

```
  # プログラムの取得
wget https://www.dragonwake.com/download/LubanCat4/7-
srccode/tutorial/CSAIEG358803_rk_code_storage.zip
  # 解凍後：サブフォルダー「lubancat_ai_manual_code」のなか、すべてソースがマニュアルに使われる
ものです。
  cd lubancat_ai_manual_code/example/ppocrv4/cpp

  # 直接コンパイル
  cat@lubancat:~/lubancat_ai_manual_code/example/ppocrv4/cpp$ ./build-linux.sh -t rk3588
  ./build-linux.sh -t rk3588
  ====================================
  TARGET_SOC=rk3588
  INSTALL_DIR=/home/cat/lubancat_ai_manual_code/example/ppocrv4/cpp/install/rk3588_linux
  BUILD_DIR=/home/cat/lubancat_ai_manual_code/example/ppocrv4/cpp/build/build_rk3588_linux
  CC=aarch64-linux-gnu-gcc
  CXX=aarch64-linux-gnu-g++
  ====================================
  -- Configuring done
  -- Generating done
  -- Build files have been written to:
/home/cat/lubancat_ai_manual_code/example/ppocrv4/cpp/build/build_rk3588_linux
  [100%] Built target rknn_ppocr_system_demo
  [100%] Built target rknn_ppocr_system_demo
  Install the project...
  -- Install configuration: ""
  -- Installing:
/home/cat/lubancat_ai_manual_code/example/ppocrv4/cpp/install/rk3588_linux/lib/librga.so
  -- Installing:
```

```
/home/cat/lubancat_ai_manual_code/example/ppocrv4/cpp/install/rk3588_linux/lib/librknnrt.so
  -- Installing:
/home/cat/lubancat_ai_manual_code/example/ppocrv4/cpp/install/rk3588_linux/./rknn_ppocr_system_
demo
  -- Set runtime path of
"/home/cat/lubancat_ai_manual_code/example/ppocrv4/cpp/install/rk3588_linux/./rknn_ppocr_system
_demo" to "$ORIGIN/lib"
  -- Installing:
/home/cat/lubancat_ai_manual_code/example/ppocrv4/cpp/install/rk3588_linux/model/test.jpg
  -- Installing:
/home/cat/lubancat_ai_manual_code/example/ppocrv4/cpp/install/rk3588_linux/model/11.jpg
  -- Installing:
/home/cat/lubancat_ai_manual_code/example/ppocrv4/cpp/install/rk3588_linux/model/ppocrv4_det_rk
3588.rknn
  -- Installing:
/home/cat/lubancat_ai_manual_code/example/ppocrv4/cpp/install/rk3588_linux/model/ch_ppocr_mobil
e_v2.0_cls_rk3588.rknn
  -- Installing:
/home/cat/lubancat_ai_manual_code/example/ppocrv4/cpp/install/rk3588_linux/model/ppocrv4_rec_rk
3588.rknn
```

install/rk3588_linux ディレクトリに切り替え、簡単なコマンドを実行してパラメータを確認します：

```
  # プログラムパラメータを確認
  cat@lubancat:~/lubancat_ai_manual_code/example/ppocrv4/cpp/install/rk3588_linux$ ./rknn_ppocr
_system_demo
  ./rknn_ppocr_system_demo <det_model_path> <rec_model_path> <image_path>
  or ./rknn_ppocr_system_demo <det_model_path> <cls_model_path> <rec_model_path> <image_path>
```

OCR プログラムをコンパイルし、2 つのコマンド形式をサポートしています。まずはテキスト検出とテキスト認識のみをテストします：

```
  cat@lubancat:~/lubancat_ai_manual_code/example/ppocrv4/cpp/install/rk3588_linux$ ./rknn_ppocr
_system_demo ./model/ppocrv4_det_rk3588.rknn ./model/ppocrv4_rec_rk3588.rknn ./model/test_align
ed.jpg
  model input num: 1, output num: 1
  input tensors:
    index=0, name=x, n_dims=4, dims=[1, 480, 480, 3], n_elems=691200, size=691200, fmt=NHWC,
type=INT8, qnt_type=AFFINE, zp=-14, scale=0.018658
  output tensors:
    index=0, name=sigmoid_0.tmp_0, n_dims=4, dims=[1, 1, 480, 480], n_elems=230400,
size=230400, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003922
  model is NHWC input fmt
  model input height=480, width=480, channel=3
  model input num: 1, output num: 1
  input tensors:
    index=0, name=x, n_dims=4, dims=[1, 48, 320, 3], n_elems=46080, size=92160, fmt=NHWC,
type=FP16, qnt_type=AFFINE, zp=0, scale=1.000000
```

```
  output tensors:
    index=0, name=softmax_11.tmp_0, n_dims=3, dims=[1, 40, 6625, 0], n_elems=265000,
size=530000, fmt=UNDEFINED, type=FP16, qnt_type=AFFINE, zp=0, scale=1.000000
  model is NHWC input fmt
  model input height=48, width=320, channel=3
  src width=512 height=500 fmt=0x1 virAddr=0x0x55acdf79c0 fd=0
  dst width=480 height=480 fmt=0x1 virAddr=0x0x55aceb31f0 fd=0
  color=0x0
  rga_api version 1.10.0_[2]
  [ WARN:0] global ../modules/core/src/matrix_expressions.cpp (1333) assign OpenCV/MatExpr:
processing of multi-channel arrays might be changed in the future:
https://github.com/opencv/opencv/issues/16739
  DRAWING OBJECT
  [0] @ [(28, 37), (309, 39), (308, 71), (27, 70)]
  regconize result: 纯, score=0.656249
  [1] @ [(26, 81), (177, 81), (177, 103), (26, 103)]
  regconize result: 产品信参数, score=0.654394
  [2] @ [(27, 111), (341, 113), (340, 134), (26, 132)]
  regconize result: 5 元 1 斤起订, score=0.532389
  [3] @ [(27, 141), (289, 142), (288, 164), (26, 162)]
  regconize result: 瓶 2 元, 起, score=0.641699
  [4] @ [(26, 180), (305, 178), (306, 192), (27, 194)]
  regconize result: 品牌加, score=0.571045
  [5] @ [(26, 209), (240, 209), (240, 227), (26, 227)]
  regconize result: 品营养护, score=0.600586
  [6] @ [(25, 240), (246, 240), (246, 259), (25, 259)]
  regconize result: 产品号】-301, score=0.529602
  [7] @ [(25, 270), (183, 270), (183, 289), (25, 289)]
  regconize result: 净含量 0m, score=0.544482
  [8] @ [(25, 335), (350, 333), (352, 352), (26, 353)]
  regconize result: 脂醇, score=0.624267
  [9] @ [(27, 365), (289, 365), (289, 384), (27, 384)]
  regconize result: 基, score=0.519042
  [10] @ [(376, 368), (486, 367), (487, 386), (377, 388)]
  regconize result: (成品), score=0.510010
  [11] @ [(26, 396), (371, 396), (371, 414), (26, 414)]
  regconize result: 能致层达, score=0.515930
  [12] @ [(27, 428), (379, 428), (379, 445), (27, 445)]
  regconize result: 头发光泽的给的头, score=0.518494
  [13] @ [(27, 459), (139, 459), (139, 477), (27, 477)]
  regconize result: 的, score=0.526855
```

テスト画像は現在のディレクトリに保存されています。以下のように：

もう 1 つのコマンド形式はテキスト方向検出を含み、認識時にテキストの方向に基づいて認識します。

```
  cat@lubancat:~/lubancat_ai_manual_code/example/ppocrv4/cpp/install/rk3588_linux$ ./rknn_ppocr
_system_demo ./model/ppocrv4_det_rk3588.rknn ./model/ch_ppocr_mobile_v2.0_cls_rk3588.rknn ./mod
el/ppocrv4_rec_rk3588.rknn ./model/11.jpg
```

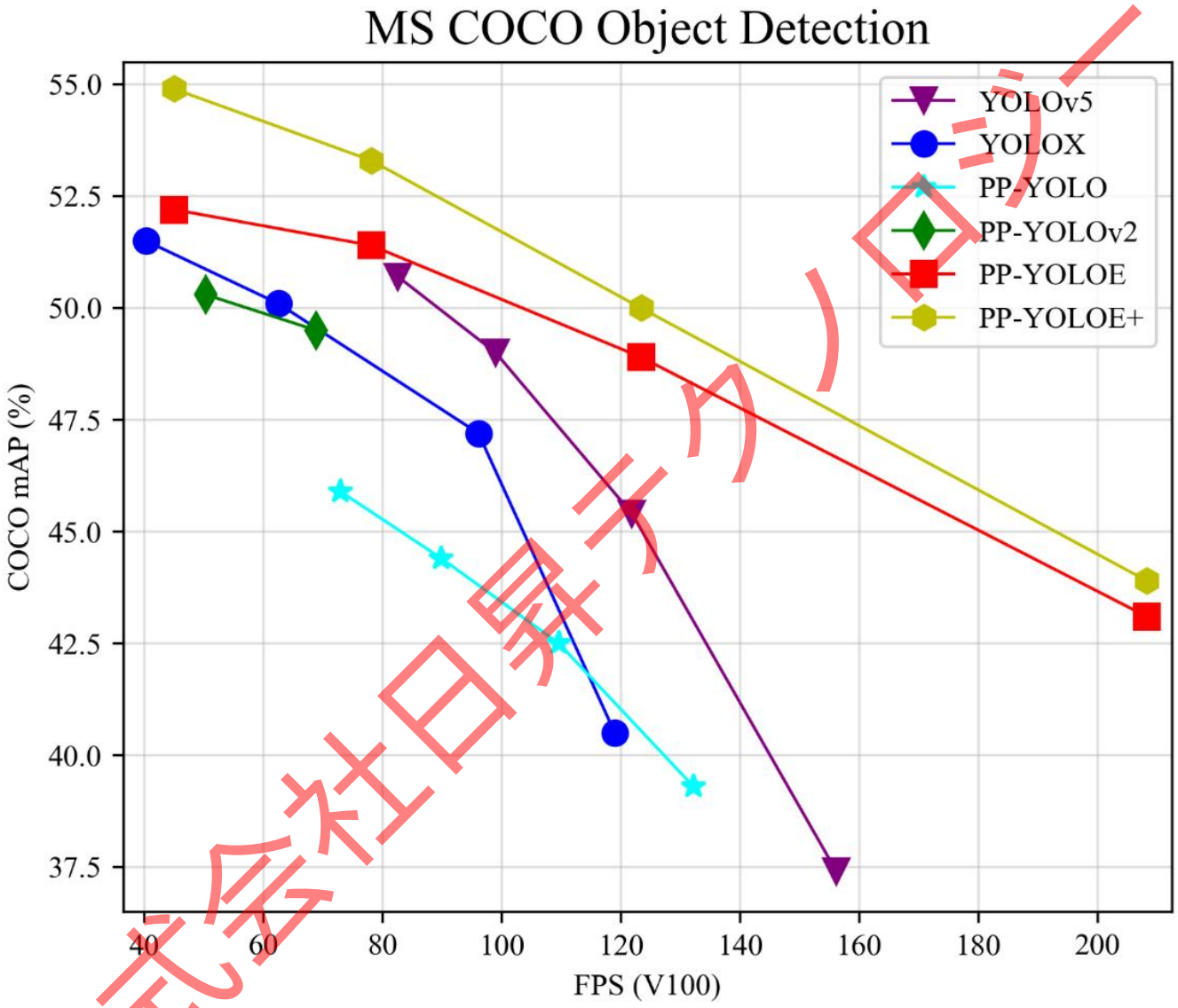
テスト画像は現在のディレクトリに保存されています。以下のように：

15.4 参考リンク

- https://github.com/PaddlePaddle/PaddleOCR
- https://github.com/PaddlePaddle/PaddleOCR/blob/release/2.7/doc/doc_ch/PP-OCRv4_introduction.md
- https://github.com/airockchip/rknn-toolkit2
- https://github.com/airockchip/rknn_model_zoo

第 16 章 PP-YOLOE

PP-YOLOE は、PP-YOLOv2 を基にした優れたシングルステージの Anchor-free であり、多くの人気のある YOLO モデルを超えています。PP-YOLOE には s/m/l/x というモデルシリーズがあり、幅と深さのマルチプライヤーで構成できます。PP-YOLOE は変形可能畳み込みや Matrix NMS などの特殊な演算子を避け、多様なハードウェアに簡単にデプロイできるよう設計されています。詳細は論文とここを参照してください。



本章では、CSAIEG Lubancat シリーズボードに PP-YOLOE モデルをデプロイする方法について簡単に説明します。

テスト環境： RK3588 ボードシステムは Debian 11 を採用し、PC は Ubuntu 20.04 を使用しています。rknn-Toolkit2 のバージョンは 2.0.0b0 であり、PaddleYOLO または PaddleDetection のバージョンは release/2.6 です。

16.1 PP-YOLOE 環境のインストール

ここでは、conda を使用して ppyoloe という名前の仮想環境を作成し、Paddle をインストールします。

```
# conda を使用して、python のバージョンを指定して PaddleYOLO 環境を作成します
conda create -n ppyoloe python=3.8
conda activate ppyoloe
# Paddle をインストールします。PaddleYOLO のコードライブラリでは、paddlepaddle-2.4.2 以上のバージョンを推奨しています。チュートリアルでは、GPU バージョンの paddlepaddle 2.6.1 を pip を使用してインストールします。
pip install paddlepaddle-gpu==2.6.1.post116 -f
https://www.paddlepaddle.org.cn/whl/linux/cudnnin/stable.html
```

Paddle を pip コマンドでインストールし、環境に応じた具体的なコマンドは公式サイトを参照してください。

16.2 PP-YOLOE+ モデルの簡単な使用

公式ウェイトファイルと coco データセットを使用して PP-YOLOE+ モデルを推論してテストし、モデルを保存し、モデル変換などの操作を行い、それを lubancat4 ボードにデプロイします。独自のデータセットでモデルをトレーニングする場合は、こちらを参照してください。さらに詳細な説明は、PaddleYOLO/docs ディレクトリのドキュメントを参照してください。

PaddleYOLO のソースコードを取得します：

```
# PaddleYOLO をクローンします。デフォルトでは、2.6 release を使用します。
git clone https://github.com/PaddlePaddle/PaddleYOLO.git
# PaddleYOLO ディレクトリに移動し、関連する依存パッケージをインストールします。
cd PaddleYOLO
# 全てブランチを確認
(.toolkit2_env) (base) csun@CSUN-PC-
0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/ppyoloe/PaddleYOLO$ git
branch -a
* develop
  remotes/origin/HEAD -> origin/develop
  remotes/origin/dev_npu
  remotes/origin/develop
  remotes/origin/fix_type
  remotes/origin/fix_v5_reader
  remotes/origin/ins_seg
  remotes/origin/release/2.5
  remotes/origin/release/2.5full
  remotes/origin/release/2.6
  remotes/origin/release/2.7
  remotes/origin/yolov5u
  remotes/origin/yolov6_v20
# 2.6 に切り替え
git checkout remotes/origin/release/2.6
pip install -r requirements.txt # インストール
```

16.2.1 モデルの推論

PP-YOLOE+ は最新の ppyoloe モデルで、s/m/l/x のシリーズがあり、ウェイトを取得します（ここからも取得できます）。

```
# PP-YOLOE+_s
wget https://bj.bcebos.com/v1/paddledet/models/ppyoloe_plus_crn_s_80e_coco.pdparams
# PP-YOLOE+_m
wget https://bj.bcebos.com/v1/paddledet/models/ppyoloe_plus_crn_m_80e_coco.pdparams
```

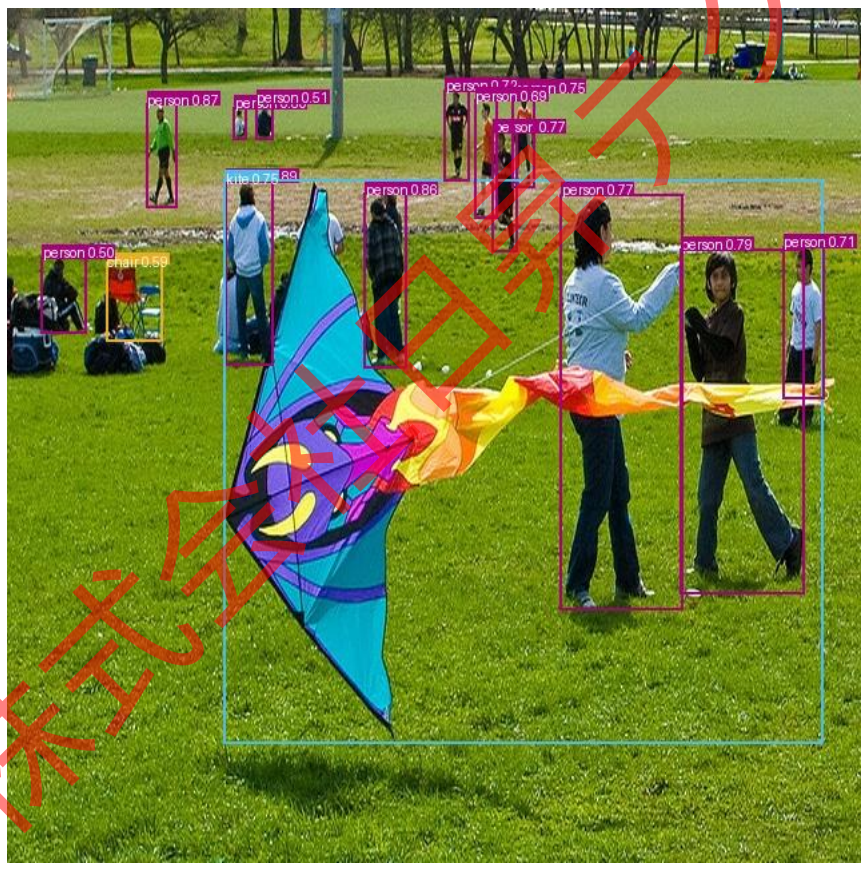
tools/infer.py を使用して推論します：

```
    #export CUDA_VISIBLE_DEVICES=0
    # 9.5.0 の Pillow をインストールする必要があるかもしれません
    pip install Pillow==9.5.0
    # tools/infer.py を使用して推論予測（単一の画像/画像フォルダ）
    # -c は設定ファイルを指定します。configs/ ディレクトリの設定ファイル（テストには ppyoloe_plus_crn_s_80e_coco.yml を使用）
    # -o または --opt は設定オプションを設定します。ここでは、手動でダウンロードしたウェイトを使用する weights を設定しましたが、直接 weights=https://bj.bcebos.com/v1/paddledet/models/ppyoloe_plus_crn_s_80e_coco.pdparams を設定することもできます。
    # --infer_dir は推論する画像のパスまたはフォルダを指定します。--draw_threshold はデフォルトで 0.5 で、画像サイズは 640*640 です。
(ppyoloe) csun@CSUN-PC-0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/ppyoloe/PaddleYOLO$ python tools/infer.py -c configs/ppyoloe/ppyoloe_plus_crn_s_80e_coco.yml  -o weights=ppyoloe_plus_crn_s_80e_coco.pdparams --infer_img=demo/000000014439_640x640.jpg --draw_threshold=0.5
Warning: import ppdet from source directory without installing, run 'python setup.py install' to install ppdet firstly
  #...省略...
  # ppdet を先にインストール
  python setup.py install
  #もう一回実行
  (ppyoloe) csun@CSUN-PC-0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/ppyoloe/PaddleYOLO$ python tools/infer.py -c configs/ppyoloe/ppyoloe_plus_crn_s_80e_coco.yml  -o weights=ppyoloe_plus_crn_s_80e_coco.pdparams --infer_img=demo/000000014439_640x640.jpg --draw_threshold=0.5
  W0626 00:01:46.539851 110352 gpu_resources.cc:119] Please NOTE: device: 0, GPU Compute Capability: 6.1, Driver API Version: 12.2, Runtime API Version: 11.6
  W0626 00:01:46.543087 110352 gpu_resources.cc:164] device: 0, cuDNN Version: 8.4.
  [06/26 00:01:47] ppdet.utils.checkpoint INFO: Finish loading model weights: ppyoloe_plus_crn_s_80e_coco.pdparams
```

```
  [06/26 00:01:47] ppdet.data.source.category WARNING: anno_file
'dataset/coco/annotations/instances_val2017.json' is None or not set or not exist, please
recheck TrainDataset/EvalDataset/TestDataset.anno_path, otherwise the default categories will
be used by metric_type.
  [06/26 00:01:47] ppdet.data.source.category WARNING: metric_type: COCO, load default
categories of COCO.
  [06/26 00:01:47] ppdet.engine INFO: Test loader length is 1, test batch_size is 1.
  [06/26 00:01:47] ppdet.engine INFO: Starting predicting ......

  100%|██████████████████████████████████████████████████████████████████████████████| 1/1 [00:00<00:00,  1.32it/s]
  [06/26 00:01:48] ppdet.engine INFO: Detection bbox results save in
output/000000014439_640x640.jpg
```

推論結果は output/000000014439_640x640.jpg に保存されます。



16.3 ボードへのモデルのデプロイ

16.3.1 RKNN 最適化

RKNPU でより良い推論性能を得るために、モデルの出力構造を調整しました。これらの調整は後処

理の論理に影響を与え、主に以下の内容を含みます。

- DFL 構造は後処理に移動されました
- 新しい追加の出力は、すべてのクラスのスコアの累積であり、後処理の候補ボックスのフィルタリングロジックを高速化するために使用されます。

詳細については、rknn_model_zoo を参照してください。

前述の PaddleDetection のソースコードまたは PaddleYOLO のソースコード（バージョンは release/2.6）を基に、ソースコード（黄色いコード）を簡単に変更します。

サンプルソース 1: ppdet/modeling/architectures/yolo.py

```
if self.training:
            yolo_losses = self.yolo_head(neck_feats, self.inputs)
            return yolo_losses
        else:
            yolo_head_outs = self.yolo_head(neck_feats)
    +       return yolo_head_outs
```

サンプルソース 2: ppdet/modeling/heads/ppyoloe_head.py

```
    +   rk_out_list = []
        for i, feat in enumerate(feats):
            _, _, h, w = feat.shape
            l = h * w
            avg_feat = F.adaptive_avg_pool2d(feat, (1, 1))
            cls_logit = self.pred_cls[i](self.stem_cls[i](feat, avg_feat) +
                                         feat)
            reg_dist = self.pred_reg[i](self.stem_reg[i](feat, avg_feat))
    +       rk_out_list.append(reg_dist)
    +       rk_out_list.append(F.sigmoid(cls_logit))
    +       rk_out_list.append(paddle.clip(rk_out_list[-1].sum(1, keepdim=True), 0, 1))
            reg_dist = reg_dist.reshape(
                [-1, 4, self.reg_channels, l]).transpose([0, 2, 3, 1])
            if self.use_shared_conv:
                reg_dist = self.proj_conv(F.softmax(
                    reg_dist, axis=1)).squeeze(1)
            else:
                reg_dist = F.softmax(reg_dist, axis=1)
            # cls and reg
            cls_score = F.sigmoid(cls_logit)
            cls_score_list.append(cls_score.reshape([-1, self.num_classes, l]))
            reg_dist_list.append(reg_dist)
    +   return rk_out_list
```

上記の簡単な変更はモデルのエクスポートにのみ使用され、モデルのトレーニング時はコメントアウトしてください。ソースコードの変更はパッチを直接適用することもできます（ソースコードのバージョンは release/2.5）。詳細については rknn_model_zoo を参照してください。

16.3.2 ONNX モデルのエクスポート

```
    # PaddleYOLO のソースコードディレクトリに移動し、tools/export_model.py を使用して paddle モデルをエクスポートします。
    cd PaddleYOLO
    # -c は設定ファイルを指定します。configs/ ディレクトリの設定ファイル
    # --output_dir はモデルの保存ディレクトリを指定します。デフォルトは output_inference です
    # -o または --opt は設定オプションを設定します。ここでは、手動でダウンロードしたウェイトなどを設定しました
python tools/export_model.py -c configs/ppyoloe/ppyoloe_plus_crn_s_80e_coco.yml ¥
                        -o weights=ppyoloe_plus_crn_s_80e_coco.pdparams ¥
                        exclude_nms=True exclude_post_process=True ¥
                        --output_dir inference_model
```

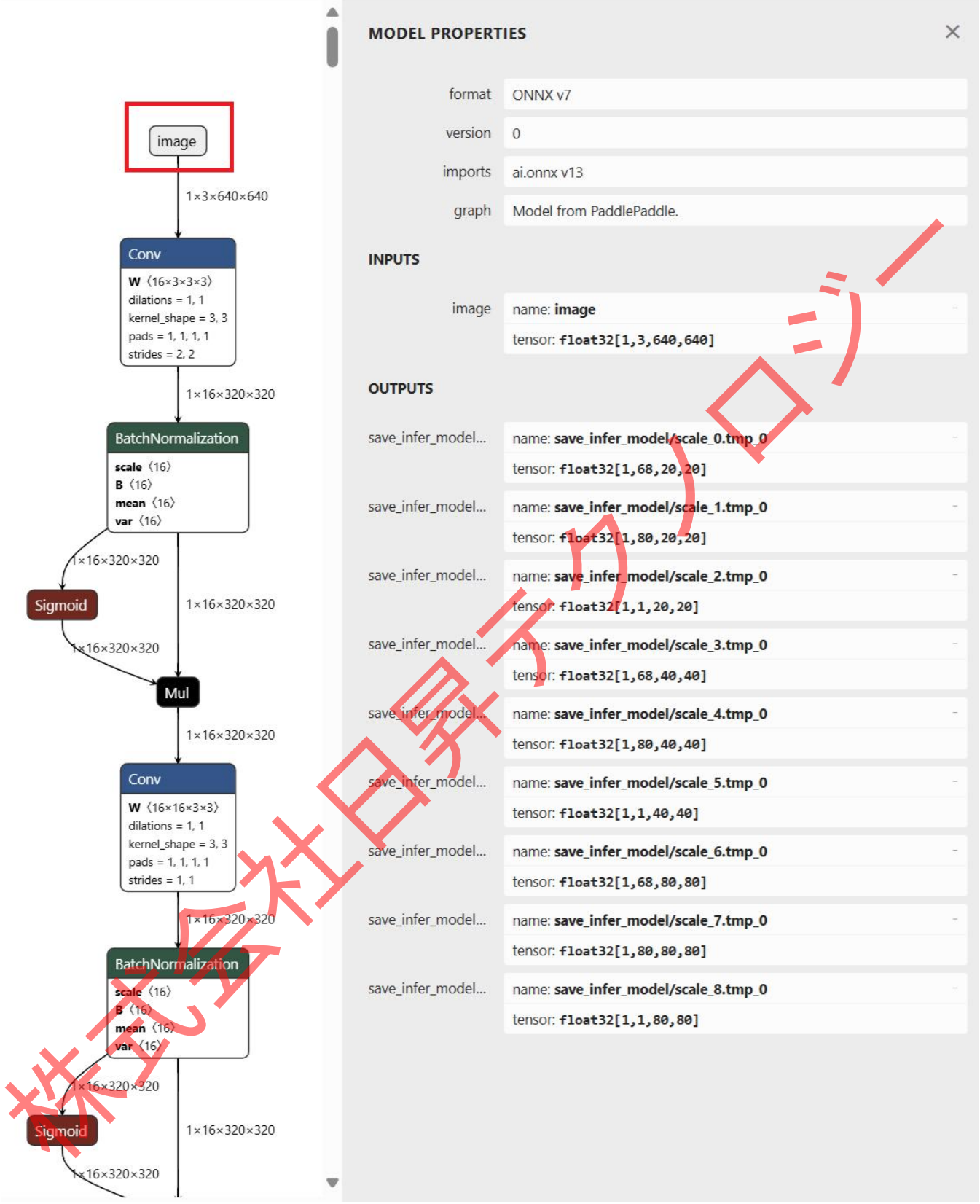
モデルは inference_model/ppyoloe_plus_crn_s_80e_coco に保存されます：

```
    ppyoloe_plus_crn_s_80e_coco
    ├─── infer_cfg.yml         # モデルの設定ファイル情報
    ├─── model.pdiparams       # 静的グラフモデルパラメータ
    ├─── model.pdiparams.info # パラメータの追加情報、通常は無視できます
    └─── model.pdmodel         # 静的グラフモデルファイル
```

次に、paddle モデルを ONNX モデルに変換します：

```
  # モデルの変換
  paddle2onnx --model_dir inference_model/ppyoloe_plus_crn_s_80e_coco ¥
      --model_filename model.pdmodel ¥
      --params_filename model.pdiparams ¥
      --opset_version 11 ¥
      --
save_file ./inference_model/ppyoloe_plus_crn_s_80e_coco/ppyoloe_plus_crn_s_80e_coco.onnx
  # モデルのシェープを固定する
  python -m paddle2onnx.optimize --input_model
inference_model/ppyoloe_plus_crn_s_80e_coco/ppyoloe_plus_crn_s_80e_coco.onnx ¥
      --output_model
inference_model/ppyoloe_plus_crn_s_80e_coco/ppyoloe_plus_crn_s_80e_coco.onnx ¥
      --input_shape_dict "{'image':[1,3,640,640]}"
```

Netron を使用してエクスポートされた onnx モデルの入力と出力を確認できます。



16.3.3 rknn-Toolkit2 で RKNN モデルをエクスポート

次に、onnx モデルを rknn モデルに変換します。rknn_model_zoo のプログラムを使用できます。ここでは rknn_model_zoo を参照してください。

rknn-Toolkit2 ツールを使用して、onnx モデルを rknn モデルに変換し、推論テストを行います（関連例も参照）：

ソースコード 3：ppyolo3/test.py（一部のプログラム）

```
if __name__ == '__main__':

    # Create RKNN object
    #rknn = RKNN(verbose=True)
    rknn = RKNN()

    # pre-process config,target_platform='rk3588'
    print('--> Config model')
    rknn.config(mean_values=[[0, 0, 0]], std_values=[[255, 255, 255]], target_platform='rk3588')
    print('done')

    # Load ONNX model
    print('--> Loading model')
    ret = rknn.load_onnx(model=ONNX_MODEL)
    if ret != 0:
        print('Load model failed!')
        exit(ret)
    print('done')

    # Build model
    print('--> Building model')
    ret = rknn.build(do_quantization=QUANTIZE_ON, dataset=DATASET)
    if ret != 0:
        print('Build model failed!')
        exit(ret)
    print('done')

    # Export RKNN model
    print('--> Export rknn model')
    ret = rknn.export_rknn(RKNN_MODEL)
    if ret != 0:
        print('Export rknn model failed!')
        exit(ret)
    print('done')

    # Init runtime environment
    print('--> Init runtime environment')
    ret = rknn.init_runtime()
    # ret = rknn.init_runtime('rk3566')
    if ret != 0:
        print('Init runtime environment failed!')
        exit(ret)
    print('done')

    # Set inputs
    img_src = cv2.imread(IMG_PATH)
    src_shape = img_src.shape[:2]
    img, ratio, (dw, dh) = letter_box(img_src, IMG_SIZE)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    #img = cv2.resize(img_src, IMG_SIZE)

    # Inference
    print('--> Running model')
    outputs = rknn.inference(inputs=[img])
```

```
    print('done')

    # post process
    boxes, classes, scores = post_process(outputs)

    img_p = img_src.copy()
    if boxes is not None:
        draw(img_p, get_real_box(src_shape, boxes, dw, dh, ratio), scores, classes)
        cv2.imwrite("result.jpg", img_p)
if __name__ == '__main__':

    # Create RKNN object
    #rknn = RKNN(verbose=True)
    rknn = RKNN()

    # pre-process config,target_platform='rk3588'
    print('--> Config model')
    rknn.config(mean_values=[[0, 0, 0]], std_values=[[255, 255, 255]], target_platform='rk3588')
    print('done')

    # Load ONNX model
    print('--> Loading model')
    ret = rknn.load_onnx(model=ONNX_MODEL)
    if ret != 0:
        print('Load model failed!')
        exit(ret)
    print('done')

    # Build model
    print('--> Building model')
    ret = rknn.build(do_quantization=QUANTIZE_ON, dataset=DATASET)
    if ret != 0:
        print('Build model failed!')
        exit(ret)
    print('done')

    # Export RKNN model
    print('--> Export rknn model')
    ret = rknn.export_rknn(RKNN_MODEL)
    if ret != 0:
        print('Export rknn model failed!')
        exit(ret)
    print('done')

    # Init runtime environment
    print('--> Init runtime environment')
    ret = rknn.init_runtime()
    # ret = rknn.init_runtime('rk3566')
    if ret != 0:
        print('Init runtime environment failed!')
        exit(ret)
    print('done')

    # Set inputs
    img_src = cv2.imread(IMG_PATH)
    src_shape = img_src.shape[:2]
    img, ratio, (dw, dh) = letter_box(img_src, IMG_SIZE)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    #img = cv2.resize(img_src, IMG_SIZE)

    # Inference
```

```
    print('--> Running model')
    outputs = rknn.inference(inputs=[img])
    print('done')

    # post process
    boxes, classes, scores = post_process(outputs)

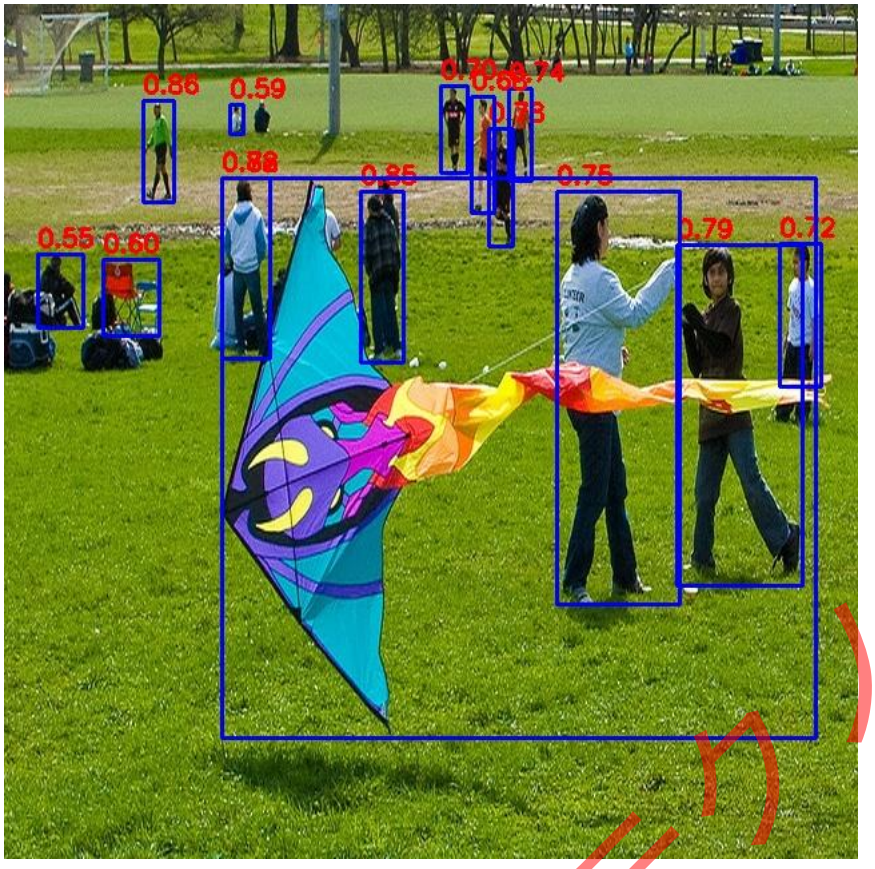
    img_p = img_src.copy()
    if boxes is not None:
        draw(img_p, get_real_box(src_shape, boxes, dw, dh, ratio), scores, classes)
        cv2.imwrite("result.jpg", img_p)
```

モデルを変換し、推論します：

```
(.toolkit2_env) (base) csun@CSUN-PC-0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/ppyoloe$ python test.py
I rknn-toolkit2 version: 2.0.0b0+9bab5682
--> Config model
done
--> Loading model
I It is recommended onnx opset 19, but your onnx model opset is 13!
I Loading : 100%|███████████████████████████████████████████| 347/347 [00:00<00:00, 54426.67it/s]
done
--> Building model
W build: found outlier value, this may affect quantization accuracy
                        const name         abs_mean    abs_std     outlier value
                        conv2d_97.w_0      0.11        0.19        23.804
I GraphPreparing : 100%|███████████████████████████████████| 230/230 [00:00<00:00, 6457.66it/s]
I Quantizating : 100%|████████████████████████████████████| 230/230 [00:00<00:00, 411.15it/s]
W build: The default input dtype of 'image' is changed from 'float32' to 'int8' in rknn model for performance!
                        Please take care of this change when deploy rknn model with Runtime API!
W build: The default output dtype of 'save_infer_model/scale_0.tmp_0' is changed from 'float32' to 'int8' in rknn model for performance!
                        Please take care of this change when deploy rknn model with Runtime API!
W build: The default output dtype of 'save_infer_model/scale_1.tmp_0' is changed from 'float32' to 'int8' in rknn model for performance!
                        Please take care of this change when deploy rknn model with Runtime API!
W build: The default output dtype of 'save_infer_model/scale_2.tmp_0' is changed from 'float32' to 'int8' in rknn model for performance!
                        Please take care of this change when deploy rknn model with Runtime API!
W build: The default output dtype of 'save_infer_model/scale_3.tmp_0' is changed from 'float32' to 'int8' in rknn model for performance!
                        Please take care of this change when deploy rknn model with Runtime API!
W build: The default output dtype of 'save_infer_model/scale_4.tmp_0' is changed from 'float32' to 'int8' in rknn model for performance!
                        Please take care of this change when deploy rknn model with Runtime API!
W build: The default output dtype of 'save_infer_model/scale_5.tmp_0' is changed from 'float32' to 'int8' in rknn model for performance!
                        Please take care of this change when deploy rknn model with Runtime API!
W build: The default output dtype of 'save_infer_model/scale_6.tmp_0' is changed from 'float32' to 'int8' in rknn model for performance!
                        Please take care of this change when deploy rknn model with Runtime API!
W build: The default output dtype of 'save_infer_model/scale_7.tmp_0' is changed from 'float32' to 'int8' in rknn model for performance!
                        Please take care of this change when deploy rknn model with Runtime API!
W build: The default output dtype of 'save_infer_model/scale_8.tmp_0' is changed from 'float32' to 'int8' in
```

```
rknn model for performance!
                        Please take care of this change when deploy rknn model with Runtime API!
  I rknn building ...
  I rknn buiding done.
  done
  --> Export rknn model
  done
  --> Init runtime environment
  I Target is None, use simulator!
  done
  --> Running model
  W inference: The 'data_format' is not set, and its default value is 'nhwc'!
  I GraphPreparing : 100%|████████████████████████████████| 240/240 [00:00<00:00, 6328.88it/s]
  I SessionPreparing : 100%|████████████████████████████████| 240/240 [00:00<00:00, 2177.78it/s]
  W inference: The dims of input(ndarray) shape (640, 640, 3) is wrong, expect dims is 4! Try expand dims to
(1, 640, 640, 3)!
  done
  input_data shapes: [(1, 68, 20, 20), (1, 80, 20, 20), (1, 1, 20, 20), (1, 68, 40, 40), (1, 80, 40, 40), (1,
1, 40, 40), (1, 68, 80, 80), (1, 80, 80, 80), (1, 1, 80, 80)]
  class: person, score: 0.8798694610595703
  box coordinate left,top,right,down: [163, 130, 199, 265]
  class: person, score: 0.8589202165603638
  box coordinate left,top,right,down: [104, 72, 127, 148]
  class: person, score: 0.8484455943107605
  box coordinate left,top,right,down: [267, 140, 299, 268]
  class: person, score: 0.7864068746566772
  box coordinate left,top,right,down: [504, 180, 598, 435]
  class: person, score: 0.7821062207221985
  box coordinate left,top,right,down: [363, 93, 381, 181]
  class: person, score: 0.7463155388832092
  box coordinate left,top,right,down: [414, 140, 506, 449]
  class: person, score: 0.7428423166275024
  box coordinate left,top,right,down: [378, 63, 395, 132]
  class: person, score: 0.7227499485015869
  box coordinate left,top,right,down: [581, 179, 612, 286]
  class: person, score: 0.7029375433921814
  box coordinate left,top,right,down: [327, 61, 347, 126]
  class: person, score: 0.681450366973877
  box coordinate left,top,right,down: [350, 69, 367, 156]
  class: person, score: 0.5924320816993713
  box coordinate left,top,right,down: [169, 75, 179, 97]
  class: person, score: 0.5516641736030579
  box coordinate left,top,right,down: [25, 187, 59, 242]
  class: kite, score: 0.7185599980392456
  box coordinate left,top,right,down: [163, 130, 608, 549]
  class: chair, score: 0.6005458235740662
  box coordinate left,top,right,down: [74, 191, 116, 248]
```

結果の画像は現在のディレクトリに result.jpg として保存されます。



16.3.4 ボードでの C++デプロイとテスト

1. 簡単なデプロイテストには rknn_model_zoo リポジトリの提供する RKNN_C_demo を使用し、lubancat-4 ボード、Debian11 システムでテストします。

```
# ボード側の操作
# rknn_model_zoo リポジトリのソースコードをクローンします。チュートリアルでテストされた
rknn_model_zoo リポジトリのバージョンは 2.0.0 です。
      cat@lubancat:~/lubancat_ai_manual_code/example/ppyoloe$ git clone
https://github.com/airockchip/rknn_model_zoo.git
# PC 側の操作
# 上記の onnx, rknn モデルをボード側に転送
(.toolkit2_env) (base) csun@CSUN-PC-
0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/ppyoloe$ scp
PaddleYOLO/inference_model/ppyoloe_plus_crn_s_80e_coco/ppyoloe_plus_crn_s_80e_coco.onnx
cat@192.168.11.241:/home/cat/lubancat_ai_manual_code/example/ppyoloe/rknn_model_zoo/examples/pp
yoloe/model/
(.toolkit2_env) (base) csun@CSUN-PC-
0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/ppyoloe$ scp
ppyoloe_plus_crn_s_80e_coco.rknn
cat@192.168.11.241:/home/cat/lubancat_ai_manual_code/example/ppyoloe/rknn_model_zoo/examples/pp
yoloe/model/
# ボード側に戻して操作
cat@lubancat:~/lubancat_ai_manual_code/example/ppyoloe$ ls
dataset.txt  rknn_model_zoo  test.jpg  test.py
```

```
cat@lubancat:~/lubancat_ai_manual_code/example/ppyoloe$ cd rknn_model_zoo
# コンパイルの際に、デフォルト名前が ppyoloe.rknn になりますので、モデル名前を変更
cat@lubancat:~/lubancat_ai_manual_code/example/ppyoloe/rknn_model_zoo$ mv
examples/ppyoloe/model/ppyoloe_plus_crn_s_80e_coco.rknn ppyoloe.rknn

cat@lubancat:~/lubancat_ai_manual_code/example/ppyoloe/rknn_model_zoo$ mv
examples/ppyoloe/model/ppyoloe_plus_crn_s_80e_coco.onnx ppyoloe.onnx

# コンパイル
cat@lubancat:~/lubancat_ai_manual_code/example/ppyoloe/rknn_model_zoo$ bash build-linux.sh -t
rk3588 -a aarch64 -d ppyoloe
build-linux.sh -t rk3588 -a aarch64 -d ppyoloe
aarch64-linux-gnu
==================================
BUILD_DEMO_NAME=ppyoloe
BUILD_DEMO_PATH=examples/ppyoloe/cpp
TARGET_SOC=rk3588
TARGET_ARCH=aarch64
BUILD_TYPE=Release
ENABLE_ASAN=OFF
INSTALL_DIR=/home/cat/lubancat_ai_manual_code/example/ppyoloe/rknn_model_zoo/install/rk3588_lin
ux_aarch64/rknn_ppyoloe_demo
BUILD_DIR=/home/cat/lubancat_ai_manual_code/example/ppyoloe/rknn_model_zoo/build/build_rknn_ppy
oloe_demo_rk3588_linux_aarch64_Release
CC=aarch64-linux-gnu-gcc
CXX=aarch64-linux-gnu-g++
==================================
-- !!!!!!!!!!!CMAKE_SYSTEM_NAME: Linux
-- Configuring done
-- Generating done
-- Build files have been written to:
/home/cat/lubancat_ai_manual_code/example/ppyoloe/rknn_model_zoo/build/build_rknn_ppyoloe_demo_
rk3588_linux_aarch64_Release
[ 20%] Built target imagedrawing
[ 40%] Built target fileutils
[ 60%] Built target imageutils
[100%] Built target rknn_ppyoloe_demo
[ 20%] Built target fileutils
[ 40%] Built target imageutils
[ 60%] Built target imagedrawing
[100%] Built target rknn_ppyoloe_demo
Install the project...
-- Install configuration: "Release"
-- Installing:
/home/cat/lubancat_ai_manual_code/example/ppyoloe/rknn_model_zoo/install/rk3588_linux_aarch64/r
knn_ppyoloe_demo/./rknn_ppyoloe_demo
-- Set runtime path of
"/home/cat/lubancat_ai_manual_code/example/ppyoloe/rknn_model_zoo/install/rk3588_linux_aarch64/
rknn_ppyoloe_demo/./rknn_ppyoloe_demo" to "$ORIGIN/../lib"
-- Installing:
/home/cat/lubancat_ai_manual_code/example/ppyoloe/rknn_model_zoo/install/rk3588_linux_aarch64/r
```

```
knn_ppyoloe_demo/model/bus.jpg
-- Installing:
/home/cat/lubancat_ai_manual_code/example/ppyoloe/rknn_model_zoo/install/rk3588_linux_aarch64/r
knn_ppyoloe_demo/model/coco_80_labels_list.txt
-- Installing:
/home/cat/lubancat_ai_manual_code/example/ppyoloe/rknn_model_zoo/install/rk3588_linux_aarch64/r
knn_ppyoloe_demo/model/ppyoloe.rknn
-- Installing:
/home/cat/lubancat_ai_manual_code/example/ppyoloe/rknn_model_zoo/install/rk3588_linux_aarch64/r
knn_ppyoloe_demo/lib/librknnrt.so
-- Installing:
/home/cat/lubancat_ai_manual_code/example/ppyoloe/rknn_model_zoo/install/rk3588_linux_aarch64/r
knn_ppyoloe_demo/lib/librga.so
```

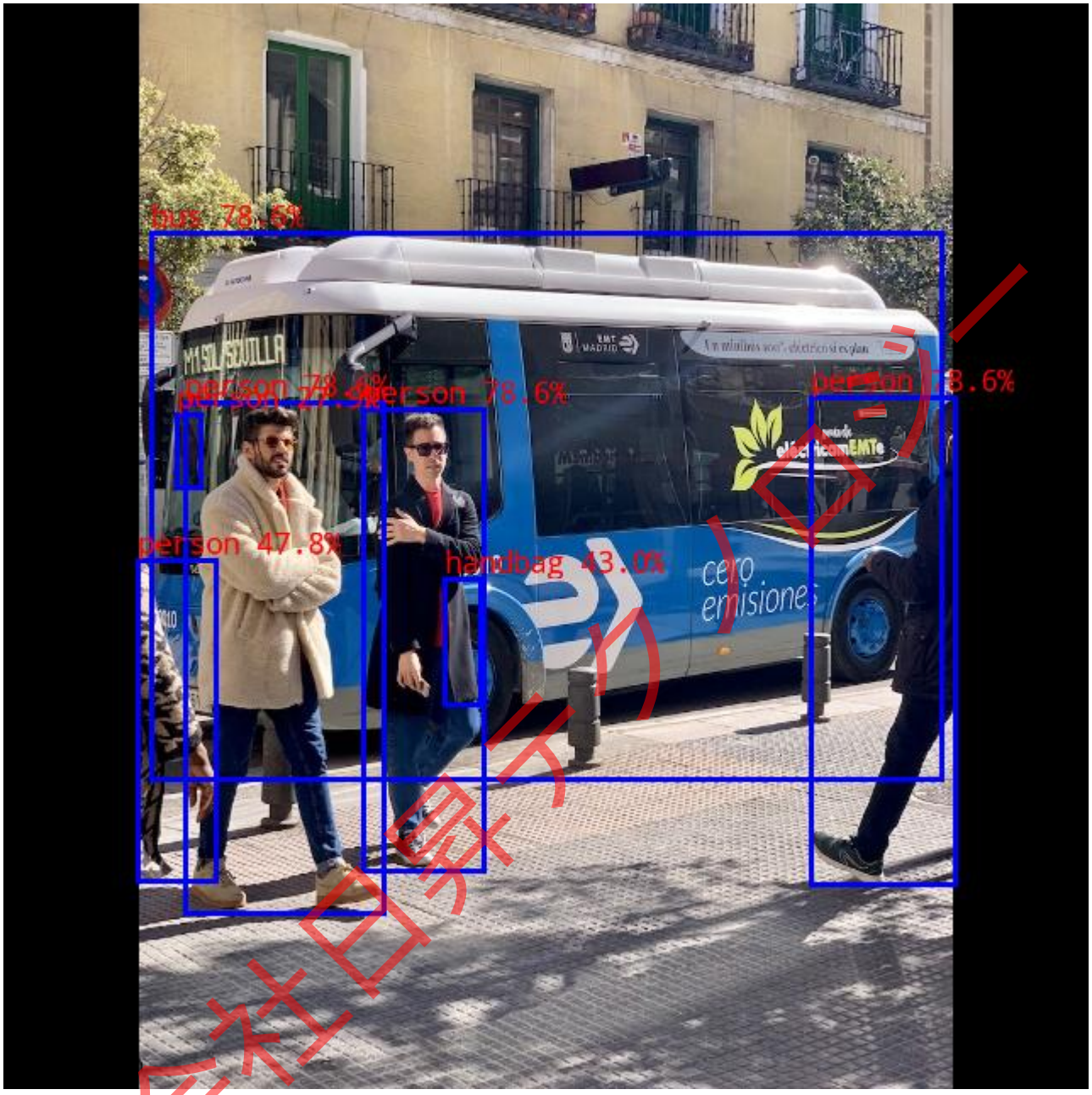
次のコマンドを使用してモデルを推論します：

```
  # install/rk3588_linux_aarch64/rknn_ppyoloe_demo ディレクトリに移動し、
rknn_v8n_rknnopt_RK3588_i8.rknn モデルファイルをこのディレクトリにコピーします
  cat@lubancat:~/lubancat_ai_manual_code/example/ppyoloe/rknn_model_zoo$ cd
install/rk3588_linux_aarch64/rknn_ppyoloe_demo/

  # 次のコマンドを実行します：
  export LD_LIBRARY_PATH=./lib
  cat@lubancat:~/lubancat_ai_manual_code/example/ppyoloe/rknn_model_zoo/install/rk3588_linux_aa
rch64/rknn_ppyoloe_demo$ ./rknn_ppyoloe_demo model/ppyoloe.rknn model/bus.jpg
  load lable ./model/coco_80_labels_list.txt
  model input num: 1, output num: 9
  input tensors:
    index=0, name=image, n_dims=4, dims=[1, 640, 640, 3], n_elems=1228800, size=1228800,
fmt=NHWC, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003922
  output tensors:
    index=0, name=save_infer_model/scale_0.tmp_0, n_dims=4, dims=[1, 68, 20, 20],
n_elems=27200, size=27200, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-47, scale=0.066592
    index=1, name=save_infer_model/scale_1.tmp_0, n_dims=4, dims=[1, 80, 20, 20],
n_elems=32000, size=32000, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003084
    index=2, name=save_infer_model/scale_2.tmp_0, n_dims=4, dims=[1, 1, 20, 20], n_elems=400,
size=400, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003922
    index=3, name=save_infer_model/scale_3.tmp_0, n_dims=4, dims=[1, 68, 40, 40],
n_elems=108800, size=108800, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-46, scale=0.076510
    index=4, name=save_infer_model/scale_4.tmp_0, n_dims=4, dims=[1, 80, 40, 40],
n_elems=128000, size=128000, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003492
    index=5, name=save_infer_model/scale_5.tmp_0, n_dims=4, dims=[1, 1, 40, 40], n_elems=1600,
size=1600, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003922
    index=6, name=save_infer_model/scale_6.tmp_0, n_dims=4, dims=[1, 68, 80, 80],
n_elems=435200, size=435200, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-42, scale=0.087638
    index=7, name=save_infer_model/scale_7.tmp_0, n_dims=4, dims=[1, 80, 80, 80],
n_elems=512000, size=512000, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003070
    index=8, name=save_infer_model/scale_8.tmp_0, n_dims=4, dims=[1, 1, 80, 80], n_elems=6400,
size=6400, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003922
  model is NHWC input fmt
```

```
  model input height=640, width=640, channel=3
  origin size=640x640 crop size=640x640
  input image: 640 x 640, subsampling: 4:2:0, colorspace: YCbCr, orientation: 1
  scale=1.000000 dst_box=(0 0 639 639) allow_slight_change=1 _left_offset=0 _top_offset=0
padding_w=0 padding_h=0
  src width=640 height=640 fmt=0x1 virAddr=0x0x55922ebe00 fd=0
  dst width=640 height=640 fmt=0x1 virAddr=0x0x559241c200 fd=0
  src_box=(0 0 639 639)
  dst_box=(0 0 639 639)
  color=0x72
  rga_api version 1.10.1_[0]
  rknn_run
  person @ (476 232 561 519) 0.786
  bus @ (87 135 553 457) 0.786
  person @ (107 235 224 536) 0.786
  person @ (212 239 283 511) 0.786
  person @ (79 328 125 517) 0.478
  handbag @ (260 339 281 413) 0.430
  person @ (102 242 116 285) 0.279
  write_image path: out.png width=640 height=640 channel=3 data=0x55922ebe00
```

結果は out.png として現在のディレクトリに保存されます。結果は以下の通りです：



16.3.5 ボードでの Python デプロイとテスト

```
# ボード側の操作
# ppyoloe/rknn_model_zoo/examples/ppyoloe/python ディレクトリに移動します。
cat@lubancat:~/lubancat_ai_manual_code/example/ppyoloe$ ls
      dataset.txt  ppyoloe_infer.py  ppyoloe_plus_crn_s_80e_coco.rknn  test.jpg  test.py
```

ソースコード 4: ppyoloe/ppyoloe_infer.py（一部のプログラム）

```
#...省略...

# model,image path
RK3566_RK3568_RKNN_MODEL = './model/RK356X/yolov3_quantization.rknn'
RK3588_RKNN_MODEL = './ppyoloe_plus_crn_s_80e_coco.rknn'

IMG_PATH = './test.jpg'
#IMG_PATH = './000000087038.jpg'
DATASET = './dataset.txt'

#...省略...

if __name__ == '__main__':

    host_name = get_host()
    if host_name == 'RK3566_RK3568':
        rknn_model = RK3566_RK3568_RKNN_MODEL
    elif host_name == 'RK3588':
        rknn_model = RK3588_RKNN_MODEL
    else:
        print("This demo cannot run on the current platform: {}".format(host_name))
        exit(-1)

    # Create RKNN object
    rknn_lite = RKNNLite()

    # load RKNN model
    print('--> Load RKNN model')
    ret = rknn_lite.load_rknn(rknn_model)
    if ret != 0:
        print('Load RKNN model failed')
        exit(ret)
    print('done')

    # Init runtime environment
    print('--> Init runtime environment')
    # run on RK356x/RK3588 with Debian OS, do not need specify target.
    if host_name == 'RK3588':
        ret = rknn_lite.init_runtime(core_mask=RKNNLite.NPU_CORE_0)
    else:
        ret = rknn_lite.init_runtime()
    if ret != 0:
        print('Init runtime environment failed!')
        exit(ret)
    print('done')

    # Set inputs
    img_src = cv2.imread(IMG_PATH)
    src_shape = img_src.shape[:2]
```

```
    img, ratio, (dw, dh) = letter_box(img_src, IMG_SIZE)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    img = np.expand_dims(img, 0)

    # Inference
    print('--> Running model')
    outputs = rknn_lite.inference(inputs=[img])
    print('done')

    # post process
    boxes, classes, scores = post_process(outputs)

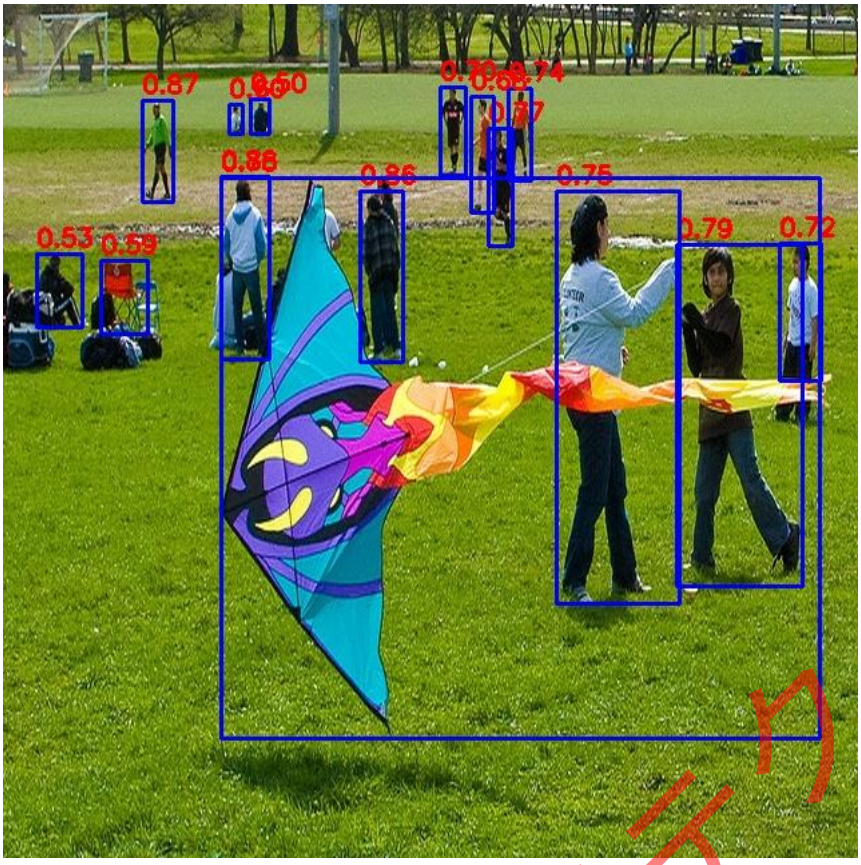
    img_p = img_src.copy()
    if boxes is not None:
        draw(img_p, get_real_box(src_shape, boxes, dw, dh, ratio), scores, classes)
        print('Save results to results.jpg!')
        cv2.imwrite("result.jpg", img_p)

    rknn_lite.release()
```

推論を実行：

```
# 推論を実行
cat@lubancat:~/lubancat_ai_manual_code/example/ppyoloe$ python ppyoloe_infer.py
--> Load RKNN model
done
--> Init runtime environment
I RKNN: [16:27:50.409] RKNN Runtime Information, librknnrt version: 2.0.0b0 (35a6907d79@2024-
03-24T10:31:14)
I RKNN: [16:27:50.409] RKNN Driver Information, version: 0.9.2
I RKNN: [16:27:50.409] RKNN Model Information, version: 6, toolkit version:
2.0.0b0+9bab5682(compiler version: 2.0.0b0 (35a6907d79@2024-03-24T02:34:11)), target: RKNPU v2,
target platform: rk3588, framework name: ONNX, framework layout: NCHW, model inference type:
static_shape
done
--> Running model
done
input_data shapes: [(1, 68, 20, 20), (1, 80, 20, 20), (1, 1, 20, 20), (1, 68, 40, 40), (1, 80,
40, 40), (1, 1, 40, 40), (1, 68, 80, 80), (1, 80, 80, 80), (1, 1, 80, 80)]
class: person, score: 0.7864068746566772
box coordinate left,top,right,down: [476, 232, 561, 519]
class: person, score: 0.7864068746566772
box coordinate left,top,right,down: [108, 233, 224, 536]
class: person, score: 0.7864068746566772
box coordinate left,top,right,down: [211, 240, 283, 512]
class: bus , score: 0.7864068746566772
box coordinate left,top,right,down: [85, 135, 552, 441]
Save results to results.jpg!
```

結果は result.png として現在のディレクトリに保存されます。結果は以下の通りです：



16.4 参考リンク

https://github.com/PaddlePaddle/PaddleDetection
https://github.com/PaddlePaddle/PaddleYOLO
https://github.com/airockchip/rknn_model_zoo/tree/main/examples/ppyoloe

第 17 章 YOLOv5(目標検知)

Yolov5 は、単段階オブジェクト検出法に属するオブジェクト検出アルゴリズムで、COCO データセットで事前にトレーニングされたオブジェクト検出アーキテクチャおよびモデルシリーズです。これは、未来のビジュアル AI 手法に対する Ultralytics のオープンソース研究を代表しており、数千時間に及ぶ研究開発から得られた教訓とベストプラクティスが含まれています。最新の YOLOv5 v7.0 には、YOLOv5n、YOLOv5s、YOLOv5m、YOLOv5l、YOLOv5x などがあり、オブジェクト検出だけでなく、セグメンテーションや分類などの応用シーンもあります。

YOLOv5 の基本原理は簡単に言うと、画像全体を複数のグリッドに分割し、各グリッドがその中のオブジェクトの種類と位置情報を予測し、予測ボックスと実際のボックスの交差オーバーユニオン（IoU）に基づいてオブジェクトボックスを選別し、最終的に予測ボックスを出力します。

本章では、YOLOv5 を簡単に使用し、Lubancat ボードで簡単にデプロイテストを行います。

ヒント:テスト環境：LubancatRK ボードのシステムは Debian11、PC Ubuntu20.04、PyTorch は 2.1.0、rknn-Toolkit2 バージョン 2.0.0beta、YOLOv5 v7.0、airockchip/yolov5 v6.2。

17.1 YOLOv5 環境のインストール

Python などの環境と関連ライブラリをインストールし、YOLOv5 リポジトリのソースコードをクローンします。

```
# 推論を実行
# anaconda インストールは前の関連内容を参考し、conda コマンドで仮想環境を作成
conda create -n yolov5 python=3.9
conda activate yolov5
# 最新の yolov5 ソースをコロン(チュートリアルの時に v7.0)，ブランチを指定できます# git clone
https://github.com/ultralytics/yolov5.git -b v7.0
git clone https://github.com/ultralytics/yolov5.git
cd yolov5
git checkout -b v7.0
# 必要のパッケージをインストール
pip3 install -r requirements.txt
# python コマンドから動作環境を確認
(yolov5) csun@CSUN-PC-
0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/yolov5$ python
Python 3.9.19 (main, May  6 2024, 19:43:03)
[GCC 11.2.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import torch
>>> import utils
>>> display=utils.notebook_init()
Checking setup...
YOLOv5    v7.0-331-gab364c98 Python-3.9.19 torch-2.3.1+cu121 CUDA:0 (NVIDIA GeForce GTX 1070
Ti, 8106MiB)
```

```
Setup complete ✅ (12 CPUs, 23.4 GB RAM, 69.0/853.7 GB disk)
>>>
```

17.2 YOLOv5 簡単使用

17.2.1 事前トレーニング済みの重みファイルの取得

`yolov5s.pt`、`yolov5m.pt`、`yolov5l.pt`、`yolov5x.pt`の重みファイルをダウンロードします。これらのファイルは[こちら](https://github.com/ultralytics/yolov5/releases)から直接取得できます。後ろの`n`、`s`、`m`、`l`、`x`は、ネットワークの幅と深さを示しており、最小は`n`で、速度が最も速く、精度が最も低いです。

yolov5s.pt をダウンロードします。

https://github.com/ultralytics/yolov5/releases/download/v7.0/yolov5s.pt

17.2.2 YOLOv5 簡単テスト

YOLOv5 のソースコードディレクトリに移動し、前述のダウンロードした重みファイルを現在のディレクトリに配置します。テスト用の画像 2 枚は`./data/images/`にあります。

```
# 簡単テスト
# --source テストデータを指定します。画像やビデオなどが使用可能です。
# --weights 重みファイルのパスを指定します。自分でトレーニングしたもの、または公式のものを使用できます。
wget https://github.com/ultralytics/yolov5/releases/download/v7.0/yolov5s.pt
# 具体的なコマンド例は以下の通りです：
(yolov5) csun@CSUN-PC-0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/yolov5$ python detect.py --source ./data/images --weights yolov5s.pt
detect: weights=['yolov5s.pt'], source=./data/images, data=data/coco128.yaml, imgsz=[640, 640], conf_thres=0.25, iou_thres=0.45, max_det=1000, device=, view_img=False, save_txt=False, save_csv=False, save_conf=False, save_crop=False, nosave=False, classes=None, agnostic_nms=False, augment=False, visualize=False, update=False, project=runs/detect, name=exp, exist_ok=False, line_thickness=3, hide_labels=False, hide_conf=False, half=False, dnn=False, vid_stride=1
YOLOv5 🚀 v7.0-331-gab364c98 Python-3.9.19 torch-2.3.1+cu121 CUDA:0 (NVIDIA GeForce GTX 1070 Ti, 8106MiB)

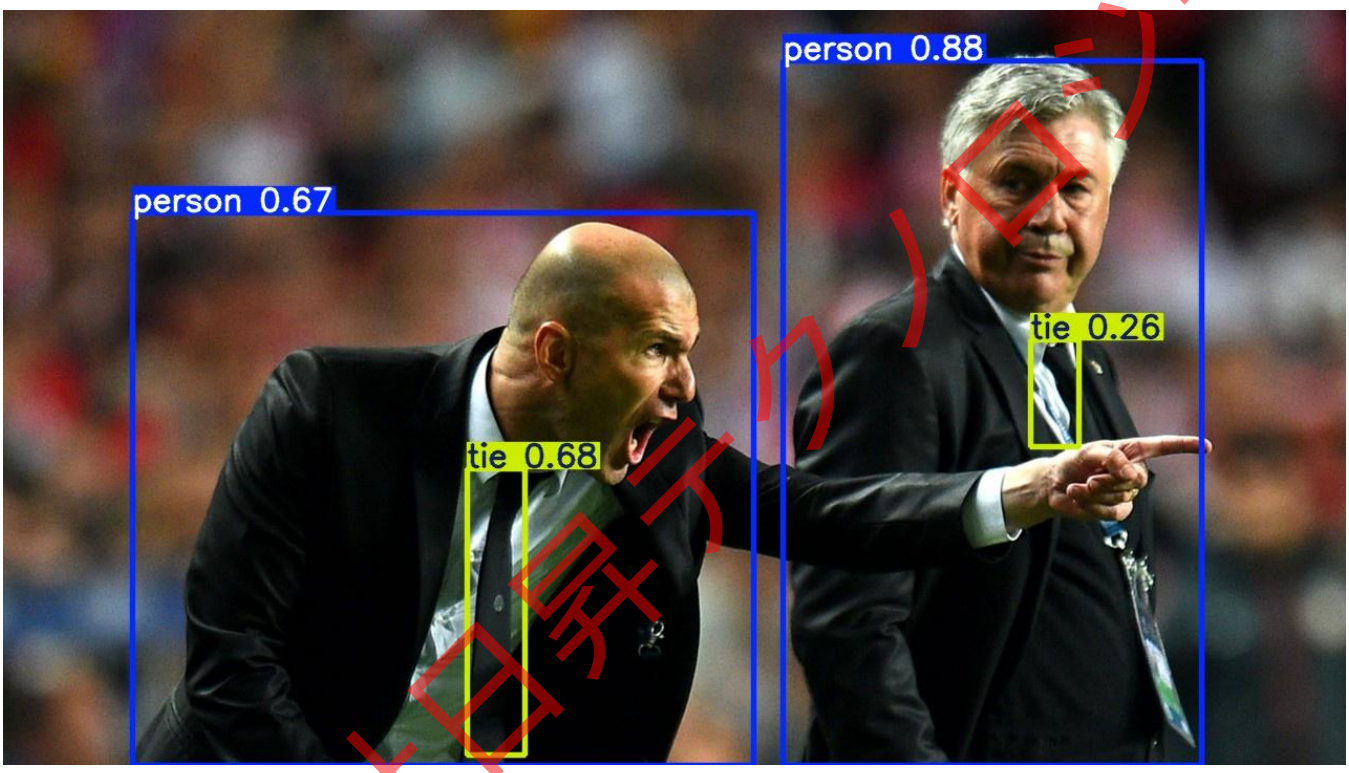
Fusing layers...
YOLOv5s summary: 213 layers, 7225885 parameters, 0 gradients, 16.4 GFLOPs
image 1/2 /home/csun/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/yolov5/data/images/bus.jpg: 640x480 4 persons, 1 bus, 26.3ms
image 2/2
```

```
/home/csun/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/yolov5/data/images/zidane
.jpg: 384x640 2 persons, 2 ties, 25.6ms
Speed: 0.3ms pre-process, 26.0ms inference, 117.0ms NMS per image at shape (1, 3, 640, 640)
Results saved to runs/detect/exp
```

このコマンドは、`./data/images`ディレクトリにある画像を使用し、`yolov5s.pt`重みファイルで検出を行います。

テスト結果：

順番に、基本設定、ネットワークパラメータ、検出結果、処理速度、最後に保存場所が表示されます。`runs/detect/exp2`ディレクトリに移動し、検出結果を確認します。



17.2.3 rknn モデルに変換

次に、`yolov5s.pt`に基づいて、rknn にエクスポートします：

1、`yolov5s.onnx`に変換します（torchscript などのモデルでも可）。まず onnx の依存環境をインストールします：

```
# onnx の環境をインストール
pip3 install -r requirements.txt coremltools onnx onnx-simplifier onnxruntime
# 重みファイルを取得し、`yolov5s.pt`を使用します。以下のコマンドを使用して、onnx モデルをエクスポートします
python3 export.py --weights yolov5s.pt --include onnx
# または以下のコマンドを使用して、torchscript をエクスポートします
python3 export.py --weights yolov5s.pt --include torchscript
# yolov5s.torchscript というファイルが生成されますが、「yolov5_relu.pt」に変更
```

```
mv yolov5s.torchscript yolov5_relu.pt
```

エクスポート結果：

```
  (yolov5) csun@CSUN-PC-
0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/yolov5$ python3 export.py --
weights yolov5s.pt --include onnx
  export: data=data/coco128.yaml, weights=['yolov5s.pt'], imgsz=[640, 640], batch_size=1,
device=cpu, half=False, inplace=False, keras=False, optimize=False, int8=False,
per_tensor=False, dynamic=False, simplify=False, opset=17, verbose=False, workspace=4,
nms=False, agnostic_nms=False, topk_per_class=100, topk_all=100, iou_thres=0.45,
conf_thres=0.25, include=['onnx']
  YOLOv5    v7.0-331-gab364c98 Python-3.9.19 torch-2.3.1+cu121 CPU

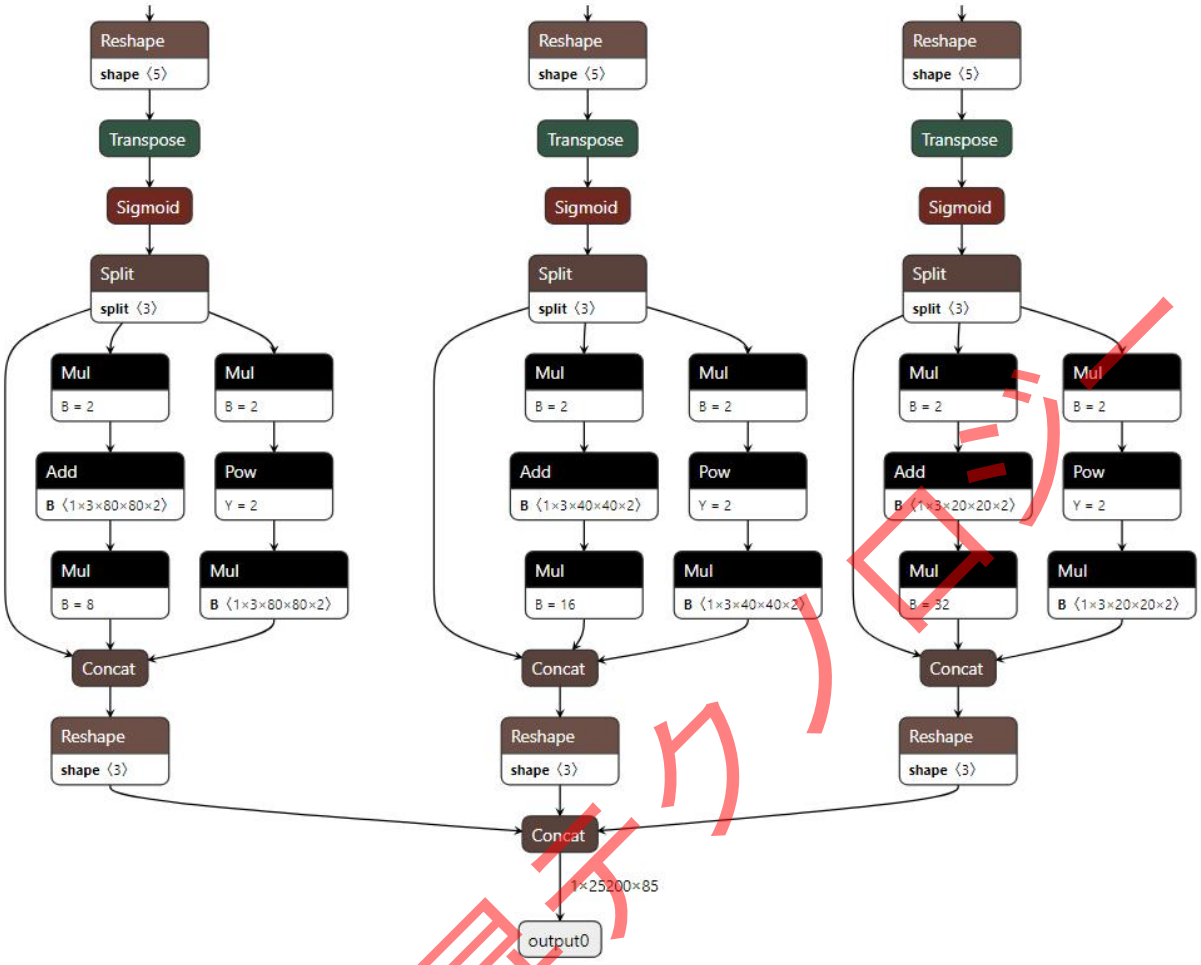
  Fusing layers...
  YOLOv5s summary: 213 layers, 7225885 parameters, 0 gradients, 16.4 GFLOPs

  PyTorch: starting from yolov5s.pt with output shape (1, 25200, 85) (14.1 MB)

  ONNX: starting export with onnx 1.16.1...
  ONNX: export success ✓ 0.6s, saved as yolov5s.onnx (28.0 MB)

  Export complete (1.0s)
  Results saved to /home/csun/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/yolov5
  Detect:          python detect.py --weights yolov5s.onnx
  Validate:        python val.py --weights yolov5s.onnx
  PyTorch Hub:     model = torch.hub.load('ultralytics/yolov5', 'custom', 'yolov5s.onnx')
  Visualize:       https://netron.app
```

現在のディレクトリに`yolov5s.onnx`ファイルが生成されます。Netron を使用してモデルを可視化できます。



2. 再エクスポート

上の画像を見て、onnx モデルの最後にある Detect 層を rknn に適応するために適切に処理します。このネットワーク構造を削除し（ただしモデル構造の最後の sigmoid 関数は削除されません）、直接 3 つの特徴マップを出力します。詳細は以下を参考にしてください。この処理のために yolov5 ファイルのソースコードを変更します：

ソースコード 1: models/yolo.py

```
    # def forward(self, x):
    #     """Processes input through YOLOv5 layers, altering shape for detection: `x(bs, 3, ny, nx, 85)`."""
    #     z = []  # inference output
    #     for i in range(self.nl):
    #         x[i] = self.m[i](x[i])  # conv
    #         bs, _, ny, nx = x[i].shape  # x(bs,255,20,20) to x(bs,3,20,20,85)
    #         x[i] = x[i].view(bs, self.na, self.no, ny, nx).permute(0, 1, 3, 4, 2).contiguous()

    #         if not self.training:  # inference
    #             if self.dynamic or self.grid[i].shape[2:4] != x[i].shape[2:4]:
    #                 self.grid[i], self.anchor_grid[i] = self._make_grid(nx, ny, i)

    #             if isinstance(self, Segment):  # (boxes + masks)
    #                 xy, wh, conf, mask = x[i].split((2, 2, self.nc + 1, self.no - self.nc - 5), 4)
    #                 xy = (xy.sigmoid() * 2 + self.grid[i]) * self.stride[i]  # xy
    #                 wh = (wh.sigmoid() * 2) ** 2 * self.anchor_grid[i]  # wh
    #                 y = torch.cat((xy, wh, conf.sigmoid(), mask), 4)
    #             else:  # Detect (boxes only)
```

```
        #                 xy, wh, conf = x[i].sigmoid().split((2, 2, self.nc + 1), 4)
        #                 xy = (xy * 2 + self.grid[i]) * self.stride[i]  # xy
        #                 wh = (wh * 2) ** 2 * self.anchor_grid[i]  # wh
        #                 y = torch.cat((xy, wh, conf), 4)
        #             z.append(y.view(bs, self.na * nx * ny, self.no))

        #     return x if self.training else (torch.cat(z, 1),) if self.export else (torch.cat(z, 1), x)

        def forward(self, x):
            z = []  # inference output
            for i in range(self.nl):
                z.append(torch.sigmoid(self.m[i](x[i])))

            return z
```

ソースコード 2: export.py

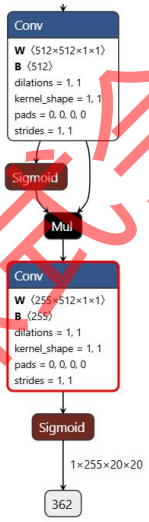
```
  # export.py の run()関数を修正

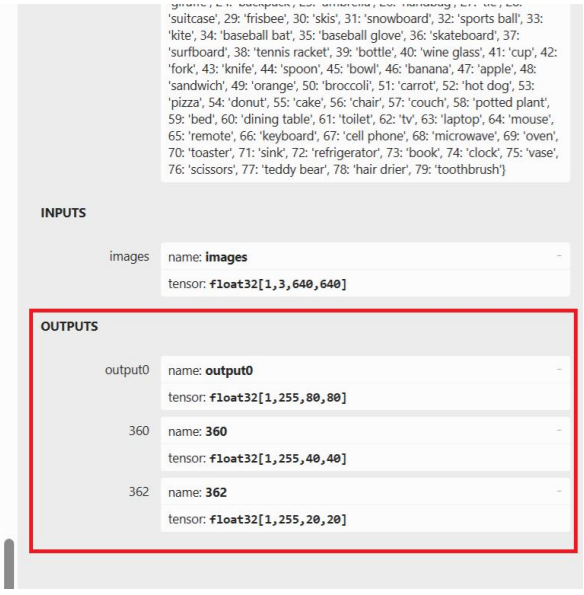
        if half and not coreml:
            im, model = im.half(), model.half()  # to FP16
  -     shape = tuple((y[0] if isinstance(y, tuple) else y).shape)  # model output shape
  +     shape = tuple((y[0] if (isinstance(y, tuple) or (isinstance(y, list))) else y).shape)  # model output
shape
        metadata = {'stride': int(max(model.stride)), 'names': model.names}  # model metadata
        LOGGER.info(f"¥n{colorstr('PyTorch:')} starting from {file} with output shape {shape}
({file_size(file):.1f} MB)")
```

変更後、再度エクスポートコマンドを実行します：

python export.py --weights yolov5s.pt --include onnx

Netron を使用してモデルを可視化します。





3. rknn モデルに変換

前の onnx モデルを rknn モデルに変換するため、rknn-Toolkit2 などの環境をインストールする必要

があります。詳細は《第 3 章 RKNN Toolkit2 紹介》章を参考にしてください。

サンプルソース 3:onnx2rknn.py

```
  # 詳しくはマニュアルを参考：https://www.dragonwake.com/download/LubanCat4/1-manual/CSAIEG358803_Embedded-AI-Development=guide.pdf
  import os
  import sys
  import numpy as np
  from rknn.api import RKNN

  # エクスポートする rknn モデルのパス
  RKNN_MODEL = './model/yolov5s.rknn'

  # ONNX モデルのパス
  MODEL_PATH = './yolov5s.onnx'

  DATASET_PATH = './dataset.txt'
  DEFAULT_QUANT = True

  # デフォルトは rk3588 プラットフォーム
  platform = "rk3588"

  if __name__ == '__main__':

      # RKNN オブジェクトの作成
      rknn = RKNN(verbose=False)

      # 前処理の設定
      print('--> モデルの設定')
      rknn.config(mean_values=[[0, 0, 0]], std_values=[
                      [255, 255, 255]], target_platform=platform)
      print('完了')

      # モデルの読み込み
      print('--> モデルの読み込み')
      ret = rknn.load_onnx(model=MODEL_PATH)
      #ret = rknn.load_pytorch(model=MODEL_PATH, input_size_list=[[1, 3, 640, 640]])
      if ret != 0:
          print('モデルの読み込みに失敗しました!')
          exit(ret)
      print('完了')

      # モデルのビルド
      print('--> モデルのビルド')
      ret = rknn.build(do_quantization=DEFAULT_QUANT, dataset=DATASET_PATH)
      if ret != 0:
          print('モデルのビルドに失敗しました!')
          exit(ret)
      print('完了')

      # rknn モデルのエクスポート
      print('--> rknn モデルのエクスポート')
      ret = rknn.export_rknn(RKNN_MODEL)
      if ret != 0:
          print('rknn モデルのエクスポートに失敗しました!')
```

```
        exit(ret)
    print('完了')

    # 精度分析、出力ディレクトリ./snapshot
    #print('--> 精度分析')
    #ret = rknn.accuracy_analysis(inputs=['./subset/000000052891.jpg'])
    #if ret != 0:
    #    print('精度分析に失敗しました!')
    #    exit(ret)
    #print('完了')

    # リリース
    rknn.release()
```

現在のディレクトリにエクスポートされた rknn モデルが生成され、accuracy_analysis インターフェースのコメントを解除すると、精度評価が実施され、簡単なデバッグが可能です。

モデル変換プログラムを実行：

```
  # メモ：rknn-toolkit2 がすでに.toolkit2_env にインストールされていますので、conda の yolov5 仮想環境から.toolkit2_env に切り替えて変換機能を実行、勿論、yolov5 で再インストールしてもよい
  #(yolov5) csun@CSUN-PC-0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/yolov5$ conda deactivate
  #source ~/project-Toolkit2/.toolkit2_env/bin/activate
  (.toolkit2_env) (base) csun@CSUN-PC-0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/yolov5$ python onnx2rknn.py
  I rknn-toolkit2 version: 2.0.0b0+9bab5682
  --> モデルの設定
  完了
  --> モデルの読み込み
  I It is recommended onnx opset 19, but your onnx model opset is 17!
  I Model converted from pytorch, 'opset_version' should be set 19 in torch.onnx.export for successful convert!
  I Loading : 100%|██████████| 134/134 [00:00<00:00, 23850.49it/s]
  完了
  --> モデルのビルド
  W build: found outlier value, this may affect quantization accuracy
                        const name              abs_mean    abs_std     outlier value
                        model.0.conv.weight     0.83        1.39        14.282
  I GraphPreparing : 100%|██████████| 145/145 [00:00<00:00, 6784.17it/s]
  I Quantizating : 100%|██████████| 145/145 [00:00<00:00, 361.30it/s]
  W build: The default input dtype of 'images' is changed from 'float32' to 'int8' in rknn model for performance!
                        Please take care of this change when deploy rknn model with Runtime API!
  W build: The default output dtype of 'output0' is changed from 'float32' to 'int8' in rknn model for performance!
                        Please take care of this change when deploy rknn model with Runtime API!
  W build: The default output dtype of '360' is changed from 'float32' to 'int8' in rknn model for performance!
                        Please take care of this change when deploy rknn model with Runtime API!
  W build: The default output dtype of '362' is changed from 'float32' to 'int8' in rknn model for performance!
```

```
                Please take care of this change when deploy rknn model with Runtime API!
I rknn building ...
I rknn buiding done.
完了
--> rknn モデルのエクスポート
完了
```

4. 基板と接続してテスト

rknn-Toolkit2 を使用してボード接続テスト、性能およびメモリ評価などを行うこともできます。ボードは USB を介して接続するか、PC に接続し、ADB 接続が正常であること、および rknn_server が起動していることを確認してください。（ここ LAN を経由テスト実施）

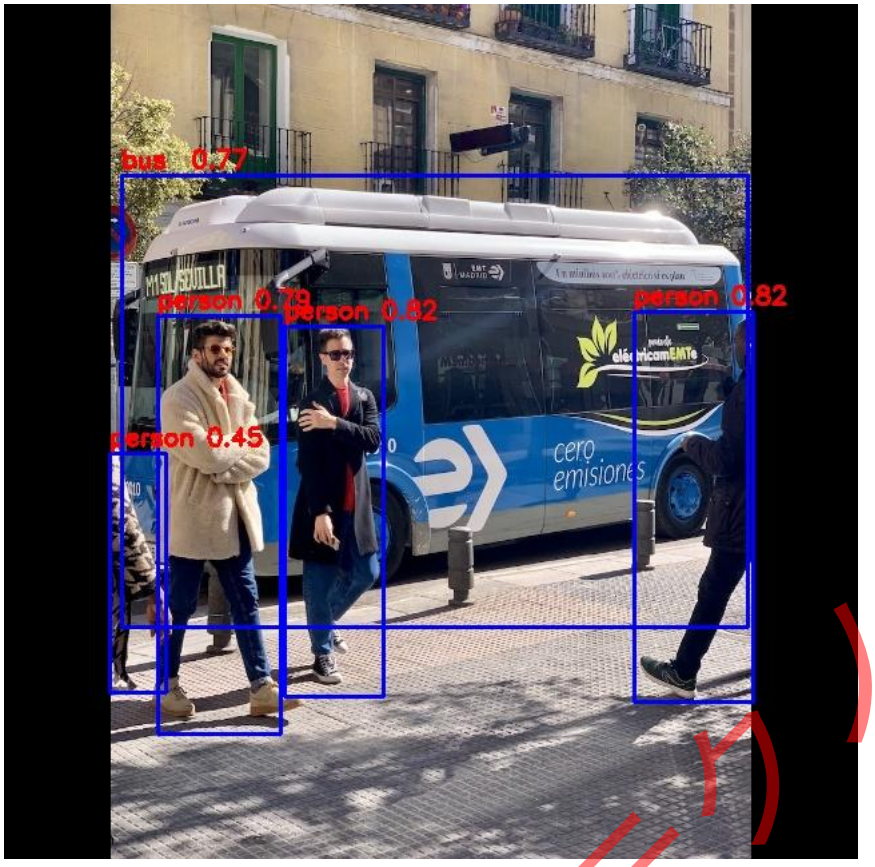
詳しくは≪AI エッジ基板 CSAIEG358803 で Python 開発及び実践≫の「22.1.3 性能とメモリ評価」を参照してください。

```
# メモ：test.py の「DEVICE_ID = "192.168.11.241:5555"」の IP がご自身環境と合わせて修正必要
# ボード側実施：
# rknn_server を起動
cat@lubancat:~/lubancat_ai_manual_code/example/yolov5$ restart_rknn.sh
cat@lubancat:~/lubancat_ai_manual_code/example/yolov5$ start rknn server, version:2.0.0b0
(18eacd0 build@2024-03-22T14:07:19)
I NPUTransfer: Starting NPU Transfer Server, Transfer version 2.1.0 (b5861e7@2020-11-
23T11:50:51)
# adbd サービスのステータスを確認し、active 状態であることを確認
下記コマンドを実行する前、
ps -ef | grep adbd
#動作している場合、飛ばします。
sudo /usr/bin/adbd &
# PC 側実施：
(.toolkit2_env) (base) csun@CSUN-PC-
0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/yolov5$ adb start-server
(.toolkit2_env) (base) csun@CSUN-PC-
0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/yolov5$ adb connect
192.168.11.241
connected to 192.168.11.241:5555
(.toolkit2_env) (base) csun@CSUN-PC-
0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/yolov5$ adb devices
List of devices attached
192.168.11.241:5555      device

(.toolkit2_env) (base) csun@CSUN-PC-
0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/yolov5$ python test.py
I rknn-toolkit2 version: 2.0.0b0+9bab5682
**************************
all device(s) with adb mode:
192.168.11.241:5555
**************************
adb: unable to connect for root: closed
I target set by user is: rk3588
I Get hardware info: target_platform = rk3588, os = Linux, aarch = aarch64
```

```
I Check RK3588 board npu runtime version
I Starting ntp or adb, target is RK3588
I Start adb...
I Connect to Device success!
I NPUTransfer: Starting NPU Transfer Client, Transfer version 2.1.0 (b5861e7@2020-11-23T11:50:36)
D RKNNAPI: ==========================================
D RKNNAPI: RKNN VERSION:
D RKNNAPI:   API: 2.0.0b0 (18eacd0 build@2024-03-22T06:07:59)
D RKNNAPI:   DRV: rknn_server: 2.0.0b0 (18eacd0 build@2024-03-22T14:07:19)
D RKNNAPI:   DRV: rknnrt: 2.0.0b0 (35a6907d79@2024-03-24T10:31:14)
D RKNNAPI: ==========================================
D RKNNAPI: Input tensors:
D RKNNAPI:   index=0, name=images, n_dims=4, dims=[1, 640, 640, 3], n_elems=1228800, size=1228800, w_stride = 0, size_with_stride = 0, fmt=NHWC, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003922
D RKNNAPI: Output tensors:
D RKNNAPI:   index=0, name=output0, n_dims=4, dims=[1, 255, 80, 80], n_elems=1632000, size=1632000, w_stride = 0, size_with_stride = 0, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003905
D RKNNAPI:   index=1, name=360, n_dims=4, dims=[1, 255, 40, 40], n_elems=408000, size=408000, w_stride = 0, size_with_stride = 0, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003917
D RKNNAPI:   index=2, name=362, n_dims=4, dims=[1, 255, 20, 20], n_elems=102000, size=102000, w_stride = 0, size_with_stride = 0, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003918
--> Running model
done
   class        score       xmin, ymin, xmax, ymax
-------------------------------------------------------
   person       0.816     [ 210,  241,  284,  518]
   person       0.816     [ 472,  230,  561,  522]
   person       0.790     [ 115,  233,  207,  546]
   person       0.453     [  79,  336,  121,  515]
    bus         0.770     [  88,  128,  557,  466]
Save results to result.jpg!
```

結果は現在のディレクトリの`result.jpg`に保存され、画像を確認します：



17.2.4 Lubancat ボードにデプロイ

rknn モデルをエクスポートした後、RKNN Toolkit Lite2 を使用してボード上に簡単にデプロイし、結果を取得した後に枠描画などの後処理を行います。詳細なファイルは以下を参照してください：

1.C++デプロイとテスト

```
# C++ソースコードをダウンロード：(1.全てのマニュアル付属ソース；2.AI のみ付属ソース)
# 1.全てのマニュアル付属ソース（大きい、前ダウンロードしたことであれば、飛ばす）
wget https://www.dragonwake.com/download/LubanCat4/7-srccode/tutorial/CSAIEG358803_rk_code_storage.zip
# 2.本マニュアルのみ付属ソース（小さい、ダウンロードしていない場合、こちらをお勧め）
wget https://www.dragonwake.com/download/LubanCat4/7-srccode/tutorial/CSAIEG358803_lubancat_ai_manual_code.zip
# コンパイル
pwd
/home/cat/lubancat_ai_manual_code/example/yolov5/cpp
# コンパイル
cat@lubancat:~/lubancat_ai_manual_code/example/yolov5/cpp$ ./build-linux.sh -t rk3588
./build-linux.sh -t rk3588
==================================
```

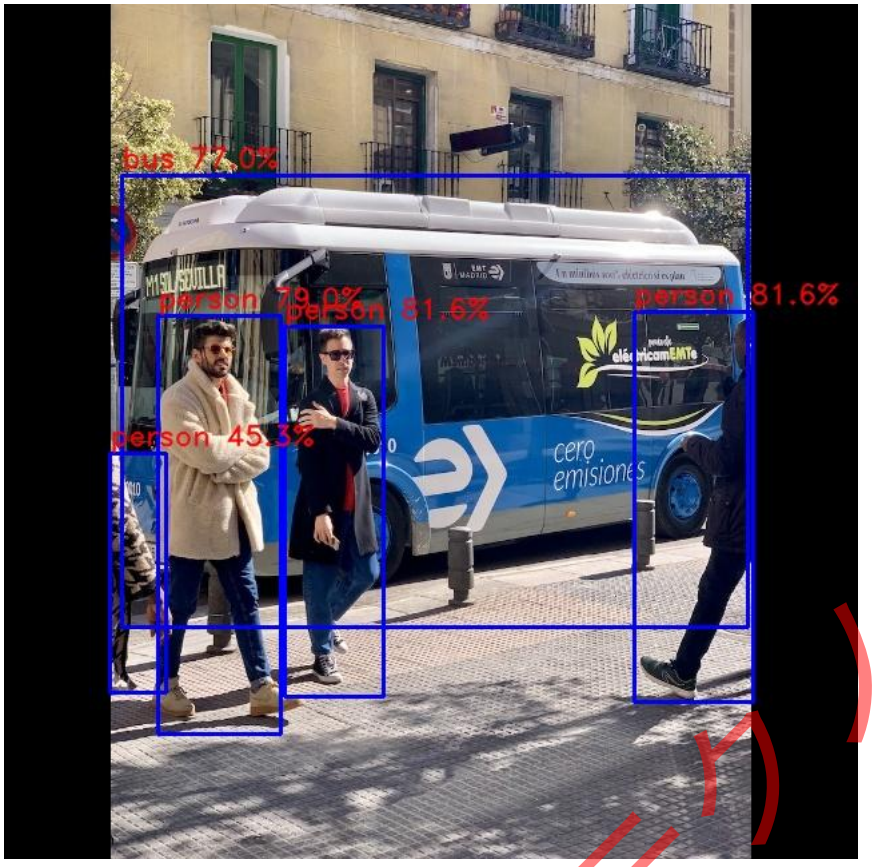
```
TARGET_SOC=rk3588
INSTALL_DIR=/home/cat/lubancat_ai_manual_code/example/yolov5/cpp/install/rk3588_linux
BUILD_DIR=/home/cat/lubancat_ai_manual_code/example/yolov5/cpp/build/build_rk3588_linux
ENABLE_ALLOC_DMA32=OFF
ENABLE_ZERO_COPY=OFF
CC=aarch64-linux-gnu-gcc
CXX=aarch64-linux-gnu-g++
==================================
-- The C compiler identification is GNU 10.2.1
-- The CXX compiler identification is GNU 10.2.1
# ...省略...
[ 50%] Built target yolov5_videocapture_demo
[100%] Built target rknn_yolov5_demo
Install the project...
-- Install configuration: ""
-- Installing:
/home/cat/lubancat_ai_manual_code/example/yolov5/cpp/install/rk3588_linux/lib/librga.so
-- Installing:
/home/cat/lubancat_ai_manual_code/example/yolov5/cpp/install/rk3588_linux/lib/librknnrt.so
-- Installing:
/home/cat/lubancat_ai_manual_code/example/yolov5/cpp/install/rk3588_linux/./rknn_yolov5_demo
-- Set runtime path of
"/home/cat/lubancat_ai_manual_code/example/yolov5/cpp/install/rk3588_linux/./rknn_yolov5_demo"
to "$ORIGIN/lib"
-- Installing:
/home/cat/lubancat_ai_manual_code/example/yolov5/cpp/install/rk3588_linux/./yolov5_videocapture
_demo
-- Set runtime path of
"/home/cat/lubancat_ai_manual_code/example/yolov5/cpp/install/rk3588_linux/./yolov5_videocaptur
e_demo" to "$ORIGIN/lib"
-- Installing:
/home/cat/lubancat_ai_manual_code/example/yolov5/cpp/install/rk3588_linux/model/bus.jpg
-- Installing:
/home/cat/lubancat_ai_manual_code/example/yolov5/cpp/install/rk3588_linux/model/coco_80_labels_
list.txt
```

前の変換された rknn モデルを「install/rk3588_linux/model/」に転送してから推論を実行

```
# PC側から rknn モデルを転送
(yolov5) csun@CSUN-PC-
0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/yolov5$ scp
model/yolov5s.rknn
cat@192.168.11.241:/home/cat/lubancat_ai_manual_code/example/yolov5/cpp/install/rk3588_linux/mo
del/
cat@192.168.11.241's password:
yolov5s.rknn                                  100% 8181KB  88.4MB/s   00:00
```

```
# ./rknn_yolov5_demo <model_path> <image_path>
# ボード側推論を実行
  cat@lubancat:~/lubancat_ai_manual_code/example/yolov5/cpp/install/rk3588_linux$ ./rknn_yolov5
_demo ./model/yolov5s.rknn ./model/bus.jpg
  load lable ./model/coco_80_labels_list.txt
  model input num: 1, output num: 3
  input tensors:
    index=0, name=images, n_dims=4, dims=[1, 640, 640, 3], n_elems=1228800, size=1228800,
fmt=NHWC, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003922
  output tensors:
    index=0, name=output0, n_dims=4, dims=[1, 255, 80, 80], n_elems=1632000, size=1632000,
fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003905
    index=1, name=360, n_dims=4, dims=[1, 255, 40, 40], n_elems=408000, size=408000, fmt=NCHW,
type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003917
    index=2, name=362, n_dims=4, dims=[1, 255, 20, 20], n_elems=102000, size=102000, fmt=NCHW,
type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003918
  model is NHWC input fmt
  model input height=640, width=640, channel=3
  scale=1.000000 dst_box=(0 0 639 639) allow_slight_change=1 _left_offset=0 _top_offset=0
padding_w=0 padding_h=0
  src width=640 height=640 fmt=0x1 virAddr=0x0x7f964e1040 fd=0
  dst width=640 height=640 fmt=0x1 virAddr=0x0x5582d24b40 fd=0
  src_box=(0 0 639 639)
  dst_box=(0 0 639 639)
  color=0x72
  rga_api version 1.10.0_[2]
  rknn_run
  person @ (210 241 284 518) 0.816
  person @ (472 230 561 522) 0.816
  person @ (115 233 207 546) 0.790
  bus @ (88 128 557 466) 0.770
  person @ (79 336 121 515) 0.453
  rknn run and process use 49.969000 ms
```

結果は現在のディレクトリの`out.jpg`に保存され、画像を確認します：



テストで RGA に関連する問題が発生した場合は、Rockchip_FAQ_RGA_EN.md を参照してください。

17.3 airockchip/yolov5 簡単テスト

上記は Ultralytics 公式の YOLOv5 v7.0 を使用して簡単なテストを行った結果です。次に、airockchip/yolov5 リポジトリを使用します。このリポジトリの yolov5 は rknpu デバイスに対して最適化されています：

- focus/SPPF ブロックを最適化し、同じ結果でより良い性能を得る。
- 出力ノードを変更し、モデルから post_process を削除する（後処理は量子化に不向き）。
- ReLU を SiLU の代わりに使用（新しいモデルをこのリポジトリで訓練する場合のみ置換）。

最新の airockchip/yolov5 リポジトリをクローンし、rk_opt@v6.2.1 ブランチに切り替えます。デフォルトの master ブランチを使用しても同じです。

```
# テスト時は v6.2 バージョン
git clone -b rk_opt@v6.2.1 https://github.com/airockchip/yolov5.git
```

```
# マニュアルを作成時、rk_opt@v6.2.1 サブブランチ
git clone -b rk_opt@v6.2.1 https://github.com/airockchip/yolov5.git
cd yolov5
```

重みファイルを取得してから、モデルを再トレーニングする必要があります。Ultralytics 公式リポジトリでトレーニングしたモデルも使用でき、その後、このリポジトリを使用して rknpu に適合するモデルをエクスポートします。

```
# カスタムデータセットを追加し、データセット設定ファイルやモデル設定ファイルを変更できます。最後に、このリポジトリを使用してモデルをトレーニングします。前にダウンロードしたファイルをここにコピーしてもよい
wget https://github.com/ultralytics/yolov5/releases/download/v7.0/yolov5s.pt
# チュートリアルのテストでは、coco128 を再トレーニングし、事前トレーニング済みの重み yolov5s.pt を使用します。
python3 train.py --data coco128.yaml --weights yolov5s.pt --img 640
# トレーニング中に自動的に重みとデータセットが取得されます。トレーニング中には多くの出力情報が表示され、その中には以下の情報も含まれます。
...
Optimizer stripped from runs/train/exp/weights/last.pt, 14.9MB
Optimizer stripped from runs/train/exp/weights/best.pt, 14.9MB

Validating runs/train/exp/weights/best.pt...
...
# トレーニングの分析結果と重みは runs/train/exp/ディレクトリに保存されます。
```

モデルは最終的に runs/train/exp/weights/ディレクトリに best.pt として保存されます。次に、このモデルを onnx モデルとしてエクスポートします。

```
# export.py のパラメータ説明
# --weights は重みファイルのパスを指定します。
# --rknpu はプラットフォームを指定します（rk_platform は rk1808, rv1109, rv1126, rk3399pro, rk3566, rk3568, rk3588, rv1103, rv1106 をサポート）。
```

```
# --include はエクスポートするモデルの形式を設定します。デフォルトでは torchscript 形式のモデルがエクスポートされますが、onnx を指定することもできます。
python3 export.py --weights runs/train/exp/weights/best.pt --rknpu rk3588

# または、以下のコマンドを使用して onnx モデルをエクスポートします。runs/train/exp/weights ディレクトリに best.onnx ファイルが生成されます。環境によっては onnx をインストールする必要があります。
python3 export.py --weights runs/train/exp/weights/best.pt --rknpu rk3588
```

torchscript をエクスポートし、現在のディレクトリに`yolov5s_relu.torchscript`ファイルが生成され、.pt ファイルとしてリネームされます。

cp runs/train/exp/weights/best.torchscript best.pt

17.3.1 rknn モデルに変換し、RK3588 ボードにデプロイ

前の変換ファイルを使用し、プログラムを直接実行して rknn ファイルをエクスポートします。または、ツールを使用してモデル変換、モデル評価、モデルデプロイなどを行います。

```
# 変換プログラムを実行して、torchscript または onnx モデルを rknn モデルに変換します。
cp runs/train/exp/weights/best.torchscript  best.pt

# 自分の環境に合わせて、pt2rknn.py ファイルのモデルのパスや実行するプラットフォームなどを変更してください。
 (.toolkit2_env) (base) csun@CSUN-PC-
0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/yolov5$ python pt2rknn.py
I rknn-toolkit2 version: 2.0.0b0+9bab5682
--> モデルの設定
完了
--> モデルの読み込み
I PtParse: 100%|████████████████████████████████████████| 728/728 [00:01<00:00,
459.79it/s]
I Loading : 100%|████████████████████████████████████████| 129/129 [00:00<00:00,
295.69it/s]
完了
--> モデルのビルド
W build: found outlier value, this may affect quantization accuracy
                    const name      abs_mean    abs_std     outlier value
                    weight.24       0.69        1.12        12.324
I GraphPreparing : 100%|████████████████████████████████████| 149/149 [00:00<00:00,
11469.10it/s]
I Quantizating : 100%|██████████████████████████████████████| 149/149 [00:00<00:00,
506.98it/s]
W build: The default input dtype of 'x.1' is changed from 'float32' to 'int8' in rknn model for
performance!
                    Please take care of this change when deploy rknn model with Runtime API!
W build: The default output dtype of '172' is changed from 'float32' to 'int8' in rknn model
for performance!
```

```
                    Please take care of this change when deploy rknn model with Runtime API!
W build: The default output dtype of 'l73' is changed from 'float32' to 'int8' in rknn model
for performance!
                    Please take care of this change when deploy rknn model with Runtime API!
W build: The default output dtype of 'l74' is changed from 'float32' to 'int8' in rknn model
for performance!
                    Please take care of this change when deploy rknn model with Runtime API!
I rknn building ...
I rknn buiding done.
完了
--> rknn モデルのエクスポート
完了
```

次に、rknn-toolkit2 ツールを使用してボードに接続し、モデルのテストや評価を行います。ボードは USB 経由で PC に接続し、adb 接続が正常であること、および rknn_server が起動していることを確認してください。

```
# メモ：test.py の「DEVICE_ID = "192.168.11.241:5555"」の IP がご自身環境と合わせて修正必要
# ボード側実施：
# rknn_server を起動
cat@lubancat:~/lubancat_ai_manual_code/example/yolov5$ restart_rknn.sh
cat@lubancat:~/lubancat_ai_manual_code/example/yolov5$ start rknn server, version:2.0.0b0
(18eacd0 build@2024-03-22T14:07:19)
I NPUTransfer: Starting NPU Transfer Server, Transfer version 2.1.0 (b5861e7@2020-11-
23T11:50:51)
# adbd サービスのステータスを確認し、active 状態であることを確認
下記コマンドを実行する前、
ps -ef | grep adbd
#動作している場合、飛ばします。
sudo /usr/bin/adbd &
# PC 側実施：
(.toolkit2_env) (base) csun@CSUN-PC-
0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/yolov5$ adb start-server
(.toolkit2_env) (base) csun@CSUN-PC-
0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/yolov5$ adb connect
192.168.11.241
connected to 192.168.11.241:5555
(.toolkit2_env) (base) csun@CSUN-PC-
0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/yolov5$ adb devices
List of devices attached
192.168.11.241:5555      device

(.toolkit2_env) (base) csun@CSUN-PC-
0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/yolov5$ python test.py
I rknn-toolkit2 version: 2.0.0b0+9bab5682
**************************
all device(s) with adb mode:
```

```
192.168.11.241:5555
**************************
adb: unable to connect for root: closed
I target set by user is: rk3588
I Get hardware info: target_platform = rk3588, os = Linux, aarch = aarch64
I Check RK3588 board npu runtime version
I Starting ntp or adb, target is RK3588
I Start adb...
I Connect to Device success!
I NPUTransfer: Starting NPU Transfer Client, Transfer version 2.1.0 (b5861e7@2020-11-
23T11:50:36)
D RKNNAPI: ==========================================
D RKNNAPI: RKNN VERSION:
D RKNNAPI:   API: 2.0.0b0 (18eacd0 build@2024-03-22T06:07:59)
D RKNNAPI:   DRV: rknn_server: 2.0.0b0 (18eacd0 build@2024-03-22T14:07:19)
D RKNNAPI:   DRV: rknnrt: 2.0.0b0 (35a6907d79@2024-03-24T10:31:14)
D RKNNAPI: ==========================================
D RKNNAPI: Input tensors:
D RKNNAPI:   index=0, name=x.1, n_dims=4, dims=[1, 640, 640, 3], n_elems=1228800, size=1228800,
w_stride = 0, size_with_stride = 0, fmt=NHWC, type=INT8, qnt_type=AFFINE, zp=-128,
scale=0.003922
D RKNNAPI: Output tensors:
D RKNNAPI:   index=0, name=172, n_dims=4, dims=[1, 255, 80, 80], n_elems=1632000, size=1632000,
w_stride = 0, size_with_stride = 0, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128,
scale=0.003904
D RKNNAPI:   index=1, name=173, n_dims=4, dims=[1, 255, 40, 40], n_elems=408000, size=408000,
w_stride = 0, size_with_stride = 0, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128,
scale=0.003922
D RKNNAPI:   index=2, name=174, n_dims=4, dims=[1, 255, 20, 20], n_elems=102000, size=102000,
w_stride = 0, size_with_stride = 0, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128,
scale=0.003922
--> Running model
done
   class        score       xmin, ymin, xmax, ymax
-------------------------------------------------
   person       0.949     [ 111,  230,  205,  523]
   person       0.918     [ 208,  234,  288,  512]
   person       0.340     [ 479,  206,  559,  518]
    bus         0.787     [  91,   89,  561,  494]
Save results to result.jpg!
```

7.3.2. ボードへのデプロイ

ボードにデプロイする際には、rknn-toolkit-lite2（Python インターフェース）や RKNPU（C/C++インターフェース）を使用できます。次に、rknn_model_zoo リポジトリで提供されるデプロイ例を参考にして、デプロイテストを行います。

1．C++テスト：

```
# 例をクローン（前のテストと同じデプロイプログラム）、もしまだコンパイルしていなければ、以下のコマンドを実行してプログラムをコンパイルします。チュートリアルテストは lubancat-4（パラメータ rk3588 を指定）

# C++ソースコードをダウンロード：(1.全てのマニュアル付属ソース；2.AI のみ付属ソース)
# 1.全てのマニュアル付属ソース（大きい、前ダウンロードしたことであれば、飛ばす）
wget https://www.dragonwake.com/download/LubanCat4/7-srccode/tutorial/CSAIEG358803_rk_code_storage.zip
# 2.本マニュアルのみ付属ソース（小さい、ダウンロードしていない場合、こちらをお勧め）
wget https://www.dragonwake.com/download/LubanCat4/7-srccode/tutorial/CSAIEG358803_lubancat_ai_manual_code.zip
# コンパイル
pwd
/home/cat/lubancat_ai_manual_code/example/yolov5/cpp
# コンパイル
cat@lubancat:~/lubancat_ai_manual_code/example/yolov5/cpp$ ./build-linux.sh -t rk3588
```

コンパイルが完了すると（或いは 17.2.4 節にコンパイルされたものをそのまま使う）、複数の実行ファイルが生成されます（install/rk3588_linux ディレクトリに保存）。チュートリアルテストでは、画像推論を行い、そのディレクトリに移動してプログラムを実行します。

```
# PC 側から変換後の rknn モデルを転送
(yolov5) csun@CSUN-PC-
0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/yolov5$ scp
model/yolov5s.rknn
cat@192.168.11.241:/home/cat/lubancat_ai_manual_code/example/yolov5/cpp/install/rk3588_linux/mo
del/
cat@192.168.11.241's password:
yolov5s.rknn                                  100% 8181KB  88.4MB/s   00:00

# ./rknn_yolov5_demo <model_path> <image_path>
# ボード側推論を実行
  cat@lubancat:~/lubancat_ai_manual_code/example/yolov5/cpp/install/rk3588_linux$ ./rknn_yolov5
_demo ./model/yolov5s.rknn ./model/bus.jpg
  load lable ./model/coco_80_labels_list.txt
  model input num: 1, output num: 3
  input tensors:
    index=0, name=x.1, n_dims=4, dims=[1, 640, 640, 3], n_elems=1228800, size=1228800,
fmt=NHWC, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003922
  output tensors:
    index=0, name=172, n_dims=4, dims=[1, 255, 80, 80], n_elems=1632000, size=1632000,
```

```
fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003904
    index=1, name=173, n_dims=4, dims=[1, 255, 40, 40], n_elems=408000, size=408000, fmt=NCHW,
type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003922
    index=2, name=174, n_dims=4, dims=[1, 255, 20, 20], n_elems=102000, size=102000, fmt=NCHW,
type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003922
  model is NHWC input fmt
  model input height=640, width=640, channel=3
  scale=1.000000 dst_box=(0 0 639 639) allow_slight_change=1 _left_offset=0 _top_offset=0
padding_w=0 padding_h=0
  src width=640 height=640 fmt=0x1 virAddr=0x0x7faaeef040 fd=0
  dst width=640 height=640 fmt=0x1 virAddr=0x0x556d9dbe90 fd=0
  src_box=(0 0 639 639)
  dst_box=(0 0 639 639)
  color=0x72
  rga_api version 1.10.0_[2]
  rknn_run
  person @ (111 230 205 523) 0.949
  person @ (208 234 288 512) 0.918
  bus @ (91 89 561 494) 0.787
  person @ (479 206 559 518) 0.340
  rknn run and process use 36.382000 ms
```

結果は現在のディレクトリの out.jpg に保存されます。より多くのモデルテストプログラムについては、rknn_model_zoo リポジトリを参照してください。

2.Python テスト

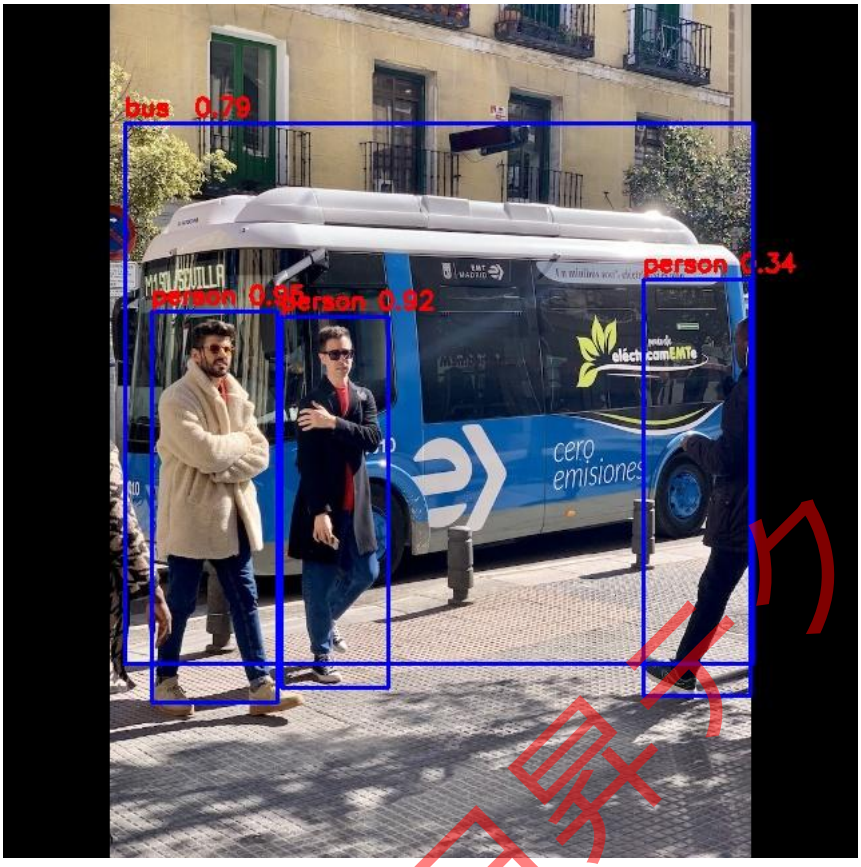
サンプルソース 4：yolov5/test.py

```python
# platform.machine()により OS を判定してから、RKNN 推論オブジェクトは異なる
if __name__ == '__main__':

    if is_architecture('x86_64'):
        print("Current platform is x86_64")
        from rknn.api import RKNN
        # RKNN オブジェクト作成
        rknn = RKNN()
        rknn.list_devices()

        # rknn モデルをロード
        rknn.load_rknn(path=RKNN_MODEL)

        # 動作環境を設定、デフォルトは rk3588
        ret = rknn.init_runtime(target=TARGET, device_id=DEVICE_ID)
        if ret != 0:
            print('Init runtime environment failed!')
            exit(ret)

    elif is_architecture('aarch64'):
        print("Current platform is aarch64")
        from rknnlite.api import RKNNLite
        # Create RKNN object
        rknn_lite = RKNNLite()
        # load RKNN model
```

```
        print('--> Load RKNN model')
        ret = rknn_lite.load_rknn(RKNN_MODEL)
        if ret != 0:
            print('Load RKNN model failed!')
            exit(ret)
        print('done')

        # Init runtime environment
        print('--> Init runtime environment')
        ret = rknn_lite.init_runtime(core_mask=RKNNLite.NPU_CORE_0)
        if ret != 0:
            print('Init runtime environment failed!')
            exit(ret)
        print('done')
    else:
        print(f"Current platform is {platform.machine()}")
        exit(0)



    # 入力設定
    img_src = cv2.imread(IMG_PATH)
    src_shape = img_src.shape[:2]
    img, ratio, (dw, dh) = letterbox(img_src, new_shape=(IMG_SIZE, IMG_SIZE))
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    #img = cv2.resize(img, (IMG_SIZE, IMG_SIZE))
    img = np.expand_dims(img, 0)

    # モデル推論
    print('--> Running model')
    if is_architecture('x86_64'):
        outputs = rknn.inference(inputs=[img], data_format=['nhwc'])
    elif is_architecture('aarch64'):
        outputs = rknn_lite.inference(inputs=[img])
    print('done')

    # 後の処理
    input0_data = outputs[0]
    input1_data = outputs[1]
    input2_data = outputs[2]

    # 1*255*h*w -> 3*85*h*w
    input0_data = input0_data.reshape([3, -1]+list(input0_data.shape[-2:]))
    input1_data = input1_data.reshape([3, -1]+list(input1_data.shape[-2:]))
    input2_data = input2_data.reshape([3, -1]+list(input2_data.shape[-2:]))

    input_data = list()
    input_data.append(np.transpose(input0_data, (2, 3, 0, 1)))
    input_data.append(np.transpose(input1_data, (2, 3, 0, 1)))
    input_data.append(np.transpose(input2_data, (2, 3, 0, 1)))

    boxes, classes, scores = yolov5_post_process(input_data)

    if boxes is not None:
        boxes = get_real_box(src_shape, boxes, dw, dh, ratio)
        draw(img_src, boxes, scores, classes)
        cv2.imwrite('result.jpg', img_src)
        print('Save results to result.jpg!')

    if is_architecture('x86_64'):
```

```
        rknn.release()
    elif is_architecture('aarch64'):
        rknn_lite.release()
```

結果は現在のディレクトリの result.jpg に保存されます。



17.4 参考リンク

https://github.com/ultralytics/yolov5
https://github.com/airockchip/yolov5/tree/master
https://github.com/airockchip/rknn_model_zoo

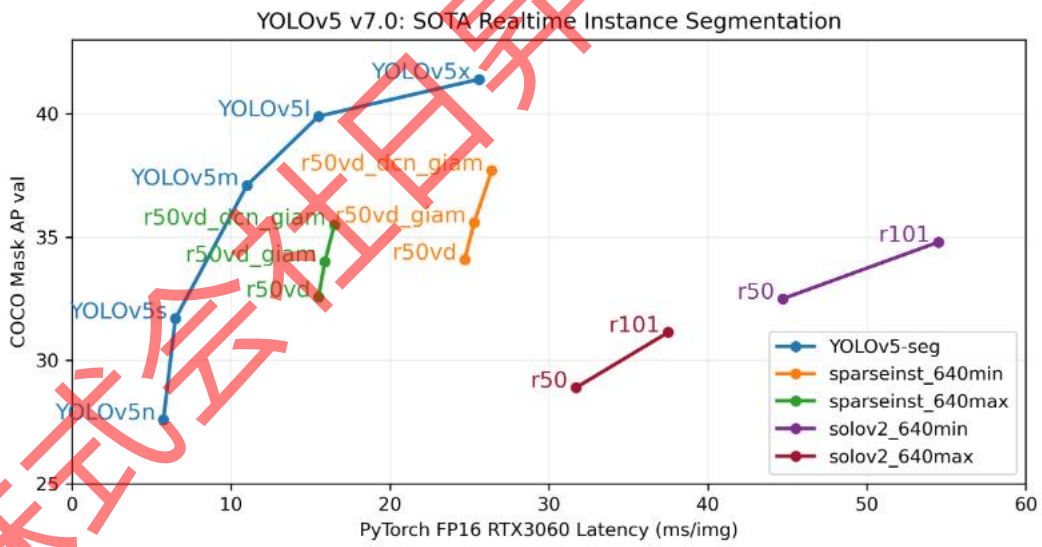
第 18 章 YOLOv5(画像セグメンテーション)

YOLOv5 はインスタンスセグメンテーションと分類のサポートを追加しました。インスタンスセグメンテーション（画像セグメンテーション）は、画像中のオブジェクトとその関連する形状を認識するコンピュータビジョンタスクです。オブジェクトの位置だけでなく、オブジェクトの輪郭もラベル付けする必要があります。インスタンスセグメンテーションは、オブジェクトのサイズを検出したり、背景からオブジェクトを切り取ったり、回転するオブジェクトを検出したりする際に非常に有用です。

YOLOV5 のインスタンスセグメンテーションには 5 つの異なるサイズのモデルがあり、最も小さいインスタンスセグメンテーションモデルは yolov5n-seg で、パラメータが 200 万個しかありません（マスク精度は 23.4 しかありません）。エッジやモバイルデバイスへのデプロイに適しています。最も正確なモデルは yolov5x-seg ですが、速度も最も遅いです。

Model	size (pixels)	mAP^{box} 50-95	mAP^{mask} 50-95	Train time 300 epochs A100 (hours)	Speed ONNX CPU (ms)	Speed TRT A100 (ms)	params (M)	FLOPs @640 (B)
YOLOv5n-seg	640	27.6	23.4	80:17	**62.7**	**1.2**	**2.0**	**7.1**
YOLOv5s-seg	640	37.6	31.7	88:16	173.3	1.4	7.6	26.4
YOLOv5m-seg	640	45.0	37.1	108:36	427.0	2.2	22.0	70.8
YOLOv5l-seg	640	49.0	39.9	66:43 (2x)	857.4	2.9	47.9	147.7
YOLOv5x-seg	640	**50.7**	**41.4**	62:56 (3x)	1579.2	4.5	88.8	265.7

COCO データセット上での YOLOv5 インスタンスセグメンテーションと他のモデルの精度と速度の比較：



この章では、YOLOv5-seg モデルの簡単なテストを行い、yolov5-seg モデルの出力結果について説明します。その後、rknpu デプロイメントの最適化が施された airockchip/yolov5 リポジトリを紹介し、最後に yolov5-seg モデルをボードにデプロイしてテストします。

18.1 YOLOV5 インスタンスセグメンテーション

環境を作成し、yolov5 リポジトリのソースコードをクローンし、依存関係をインストールします。

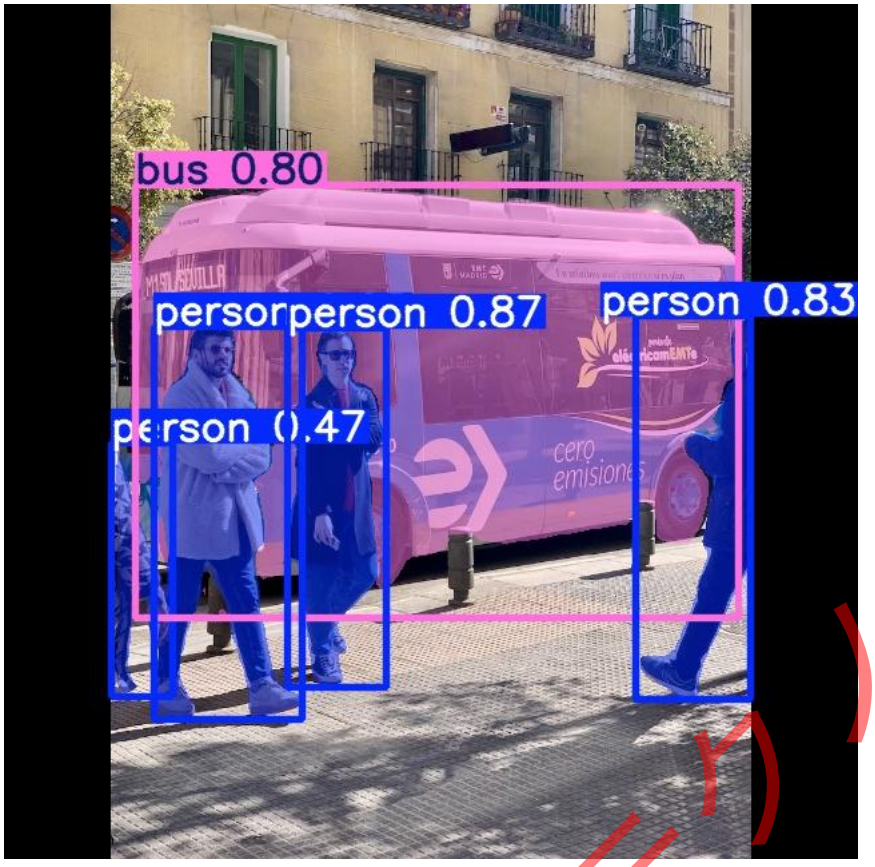
```
# anaconda のインストールは前の環境構築チュートリアルを参照し、conda コマンドを使用して環境を作成します
conda create -n yolov5 python=3.9
conda activate yolov5

# 最新の yolov5（チュートリアルテスト時は v7.0）をクローンします。バージョンブランチを指定することもできます
# git clone https://github.com/ultralytics/yolov5.git -b v7.0
git clone https://github.com/ultralytics/yolov5
cd yolov5
git checkout -b v7.0

# 依存パッケージをインストールします
pip3 install -r requirements.txt
```

18.1.1 インスタンスセグメンテーションの簡単なテスト

ソースコードをクローンした後、segment/predict.py を使用して画像やビデオに対してインスタンスセグメンテーションの簡単なテストを行います。

```
# yolov5s-seg.pt ファイルをダウンロード
wget https://github.com/ultralytics/yolov5/releases/download/v7.0/yolov5s-seg.pt
# segment/predict.py のパラメータ説明
# --weights モデルの重みファイルを指定します。デフォルトでは公式の重みファイルを使用し、初回使用時に自動でダウンロードされます
# --source テストする画像またはビデオファイルを指定します
(.toolkit2_env) (base) csun@CSUN-PC-
0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/yolov5-seg$ python
segment/predict.py --weights yolov5s-seg.pt --source ./bus.jpg
segment/predict: weights=['yolov5s-seg.pt'], source=./bus.jpg, data=data/coco128.yaml,
imgsz=[640, 640], conf_thres=0.25, iou_thres=0.45, max_det=1000, device=, view_img=False,
save_txt=False, save_conf=False, save_crop=False, nosave=False, classes=None,
agnostic_nms=False, augment=False, visualize=False, update=False, project=runs/predict-seg,
name=exp, exist_ok=False, line_thickness=3, hide_labels=False, hide_conf=False, half=False,
dnn=False, vid_stride=1, retina_masks=False
YOLOv5    v7.0-331-gab364c98 Python-3.8.10 torch-2.1.0+cu121 CPU

Fusing layers...
YOLOv5s-seg summary: 224 layers, 7611485 parameters, 0 gradients, 26.4 GFLOPs
image 1/1 /home/csun/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/yolov5-
seg/bus.jpg: 640x640 4 persons, 1 bus, 128.5ms
Speed: 0.6ms pre-process, 128.5ms inference, 0.9ms NMS per image at shape (1, 3, 640, 640)
Results saved to runs/predict-seg/exp
```

テスト画像の結果は runs/predict-seg/exp ディレクトリに保存されます。



18.1.2 yolov5-seg モデルの出力

yolov5-seg は物体検出のコードフレームワークに基づいており、トレーニング、テスト、検証などのコードエントリはソースコード segment フォルダにあります。ソースコードのアーキテクチャは物体検出と同じであり、主にデータセット、ネットワーク部分、損失、および評価指標の部分が変更されています。

ネットワーク構造では、yaml ファイルを使用してネットワークを構築します。セグメンテーションと検出のモデルネットワークは最後のヘッド層が異なるだけで、前の部分は同じです。yolov5s-seg.yaml と yolov5s.yaml を例にとって比較すると（YOLOv5 V7.0）、以下の図のようになります。

```
# yolov5n-seg.pt ファイルをダウンロード
(.toolkit2_env) (base) csun@CSUN-PC-
0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/yolov5$ diff
models/yolov5s.yaml models/segment/yolov5s-seg.yaml
6c6
< width_multiple: 0.50 # layer channel multiple
---
> width_multiple: 0.5 # layer channel multiple
48c48
<     [[17, 20, 23], 1, Detect, [nc, anchors]], # Detect(P3, P4, P5)
---
>     [[17, 20, 23], 1, Segment, [nc, anchors, 32, 256]], # Detect(P3, P4, P5)
```

ネットワーク内の Segment 分割モジュールは物体検出の Detect モジュールを継承しており、各検

出ボックスのマスク係数を追加しています。また、検出に加えて「Proto」というニューラルネットワークを追加し、検出対象のマスクを出力します。

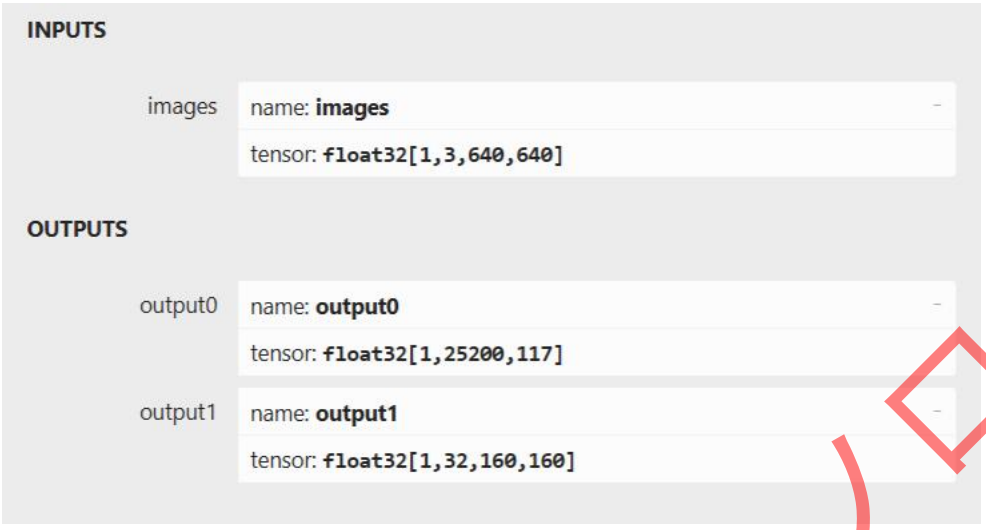
```
129 class Segment(Detect):
130     # YOLOv5 Segment head for segmentation models
131     def __init__(self, nc=80, anchors=(), nm=32, npr=256, ch=(), inplace=True):
132         """Initializes YOLOv5 Segment head with options for mask count, protos, and channel adjustments."""
133         super().__init__(nc, anchors, ch, inplace)
134         self.nm = nm  # number of masks
135         self.npr = npr  # number of protos
136         self.no = 5 + nc + self.nm  # number of outputs per anchor
137         self.m = nn.ModuleList(nn.Conv2d(x, self.no * self.na, 1) for x in ch)  # output conv
138         self.proto = Proto(ch[0], self.npr, self.nm)  # protos
139         self.detect = Detect.forward
```

詳細についてはソースコード yolov5/models/yolo.py を参照してください。

次に yolov5-seg モデルの出力を見てみましょう。まず onnx モデルをエクスポートし、以下のコマンドを使用します。

```
# --weights モデルの重みファイルを指定します。デフォルトでは公式の重みファイルを使用し、初回使用時に自動でダウンロードされます
(.toolkit2_env) (base) csun@CSUN-PC-0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/yolov5$ python export.py --weights yolov5s-seg.pt --include onnx
export: data=data/coco128.yaml, weights=['yolov5s-seg.pt'], imgsz=[640, 640], batch_size=1, device=cpu, half=False, inplace=False, keras=False, optimize=False, int8=False, per_tensor=False, dynamic=False, simplify=False, opset=17, verbose=False, workspace=4, nms=False, agnostic_nms=False, topk_per_class=100, topk_all=100, iou_thres=0.45, conf_thres=0.25, include=['onnx']
YOLOv5    v7.0-331-gab364c98 Python-3.8.10 torch-2.1.0+cu121 CPU

Fusing layers...
YOLOv5s-seg summary: 224 layers, 7611485 parameters, 0 gradients, 26.4 GFLOPs

PyTorch: starting from yolov5s-seg.pt with output shape (1, 25200, 117) (14.9 MB)

ONNX: starting export with onnx 1.14.1...
ONNX: export success ✓ 0.7s, saved as yolov5s-seg.onnx (29.5 MB)

Export complete (1.0s)
Results saved to /home/csun/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/yolov5-seg
Detect:          python segment/predict.py --weights yolov5s-seg.onnx
Validate:        python segment/val.py --weights yolov5s-seg.onnx
PyTorch Hub:     model = torch.hub.load('ultralytics/yolov5', 'custom', 'yolov5s-seg.onnx')  #
WARNING ⚠ SegmentationModel not yet supported for PyTorch Hub AutoShape inference
Visualize:       https://netron.app
```

モデルはカレントディレクトリに yolov5s-seg.onnx として保存され、Netron を使用してモデルを可視化できます。



yolov5-seg の結果は 2 部分に分かれています。「output0」は検出結果で、次元は 1*25200*177 です。最初の次元はバッチサイズ、2 番目の次元は 25200 個の出力（3*[80*80+40*40+20*20]）、3 番目の次元は 117（85+32）です。最初の 85 フィールドには 4 つの座標属性（cx、cy、w、h）、信頼度、および 80 個のクラススコアが含まれています。後の 32 フィールドは各検出ボックスのマスク係数です。「output1」はセグメンテーション結果（マスク）で、次元は 32*160*160 です。

物体検出部分の後処理は yolov5 の検出と一致しており、インスタンスセグメンテーション部分の後処理は、目標ボックス内のマスク係数とプロトマスクを加重平均することでインスタンスセグメンテーションの効果を得ることができます。

18.2 airockchip/yolov5 のテスト

rknn に yolov5-seg モデルをデプロイするテストでは、ultralytics/yolov5 ソースコードを使用してモデル後処理ネットワーク構造を変更するか、rknn 公式の airockchip/yolov5 リポジトリを使用します。このリポジトリの yolov5 は rknpu デバイスに最適化されています:

・focus/SPPF ブロックを最適化し、同じ結果でより良いパフォーマンスを得る
・出力ノードを変更し、モデルから post_process を削除（後処理量化が難しいため）
・ReLU を SiLU の代わりにアクティベーション層として使用（このリポジトリを使用して新しいモデルをトレーニングする場合にのみ置き換えられます）

このリポジトリを使用して直接 RKNPU に適合するモデルをトレーニングしてエクスポートするか、ultralytics/yolov5 でトレーニングしたモデルを使用してこのリポジトリを使用して RKNPU に適合するモデルをエクスポートします。

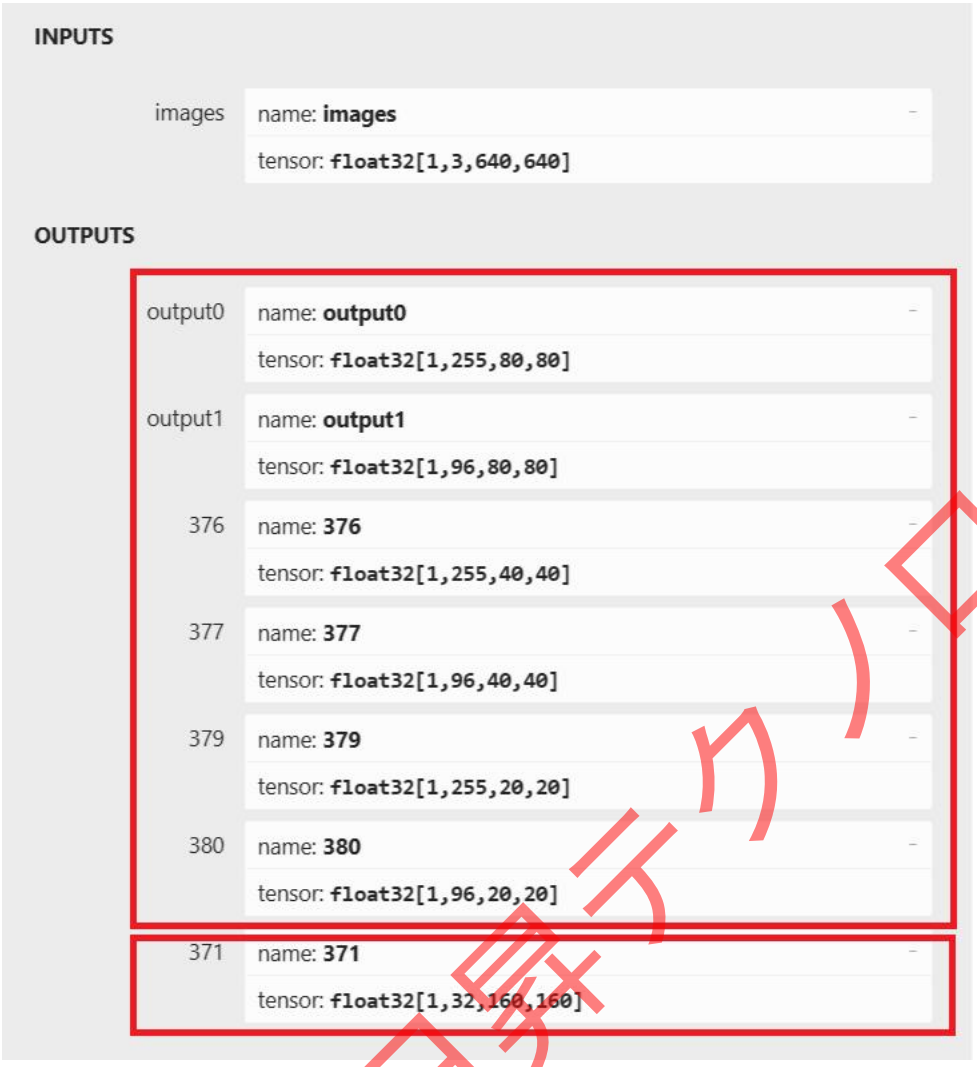
18.2.1 モデルのエクスポート

ここでは、ultralytics/yolov5 リポジトリの重みファイルを使用し、airockchip/yolov5 リポジトリの export.py を通じて onnx モデルをエクスポートします。

```
# 重みファイルを取得
# wget https://github.com/ultralytics/yolov5/releases/download/v7.0/yolov5s-seg.pt
# onnx モデルをエクスポートし、--include を指定しないとデフォルトで onnx モデルがエクスポートされます
(.toolkit2_env) (base) csun@CSUN-PC-
0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/yolov5-seg$mkdir airockchip
&& cd airockchip
git clone https://github.com/airockchip/yolov5.git
(.toolkit2_env) (base) csun@CSUN-PC-
0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/yolov5-
seg/airockchip/yolov5$ python export.py --rknpu --weights yolov5s-seg.pt --include onnx
export: data=data/coco128.yaml, weights=['yolov5s-seg.pt'], imgsz=[640, 640], batch_size=1,
device=cpu, half=False, inplace=False, keras=False, optimize=False, int8=False, dynamic=False,
simplify=False, opset=12, verbose=False, workspace=4, nms=False, agnostic_nms=False,
topk_per_class=100, topk_all=100, iou_thres=0.45, conf_thres=0.25, include=['onnx'], rknpu=True
YOLOv5    v4.0-1657-gd25a075 Python-3.8.10 torch-2.1.0+cu121 CPU

Fusing layers...
YOLOv5s-seg summary: 224 layers, 7611485 parameters, 0 gradients, 26.4 GFLOPs
---> save anchors for RKNN
[[10.0, 13.0], [16.0, 30.0], [33.0, 23.0], [30.0, 61.0], [62.0, 45.0], [59.0, 119.0], [116.0,
90.0], [156.0, 198.0], [373.0, 326.0]]
export segment model for RKNPU

PyTorch: starting from yolov5s-seg.pt with output shape (1, 255, 80, 80) (14.9 MB)

ONNX: starting export with onnx 1.14.1...
ONNX: export success ✓ 0.8s, saved as yolov5s-seg.onnx (29.1 MB)

Export complete (1.2s)
Results saved to /home/csun/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/yolov5-
seg/airockchip/yolov5
Detect:          python segment/detect.py --weights yolov5s-seg.onnx
Validate:        python segment/val.py --weights yolov5s-seg.onnx
PyTorch Hub:     model = torch.hub.load('ultralytics/yolov5', 'custom', 'yolov5s-seg.onnx')  #
WARNING    SegmentationModel not yet supported for PyTorch Hub AutoShape inference
Visualize:       https://netron.app
```


yolov5s-seg.onnx モデルはカレントディレクトリに保存され、RK_anchors.txt ファイルが生成されます。このファイルにはアンカーの値が保存され、後でボードの C++ デプロイメントに使用されます。

Netron を使用して yolov5s-seg.onnx モデルを確認できます。そのモデルの出力は次の図のようになっています。



出力は主に 2 部分に分かれています。一部は検出結果で、もう一部はセグメンテーション結果です。後処理部分の量化があまり良くないため、モデルの後処理部分は CPU で処理され、モデルは 3 種類のサイズの特徴マップを直接出力します。

特徴マップサイズ 80*80 を例にとると、関連する出力ノードは 2 つ（インスタンスセグメンテーションマスクデータと物体検出が分かれています）です。1*255*80*80 と 1*96*80*80 で、前者のノード 1*[3*(80+5)]*80*80 では、3 は各グリッドの 3 つのアンカーを表し、（5+80）は各アンカーに含まれる情報を表します。5 は 4 つの位置座標（x、y、w、h）と 1 つの信頼度を示し、信頼度はこのグリッド内にオブジェクトが存在する確率を示します。80 は coco データセットの 80 クラスのスコアを表します。後者のノード 1*[3*32]*80*80 は検出ボックスのマスク係数を示します。

最後の「371」ノードはモデルのセグメンテーション結果で、後処理は目標ボックス内のマスク係数マトリックスとの乗算を行い、最終的なセグメンテーション効果を得ます。

18.2.2 rknn モデルへの変換と簡単なテスト

次に、rknn-toolkit2 ツール（rknn-toolkit2 環境のインストールは前の章を参照）を使用して onnx モデルを rknn モデルに変換し、簡単なテストを行います。

サンプルソース 1：onnx2rknn.py（参考用のサンプルプログラム）

```
  # 詳しくはマニュアルを参考：https://www.dragonwake.com/download/LubanCat4/1-manual/CSAIEG358803_Embedded-AI-Development=guide.pdf
  import os
  import sys
  import numpy as np
  from rknn.api import RKNN

  # エクスポートする rknn モデルのパス
  RKNN_MODEL = './model/yolov5s-seg.rknn'

  # ONNX モデルのパス
  MODEL_PATH = './yolov5s-seg.onnx'

  DATASET_PATH = './dataset.txt'
  DEFAULT_QUANT = True

  # デフォルトは rk3588 プラットフォーム
  platform = "rk3588"

  if __name__ == '__main__':

      # RKNN オブジェクトの作成
      rknn = RKNN(verbose=False)

      # 前処理の設定
      print('--> モデルの設定')
      rknn.config(mean_values=[[0, 0, 0]], std_values=[
                      [255, 255, 255]], target_platform=platform)
      print('完了')

      # モデルの読み込み
      print('--> モデルの読み込み')
      ret = rknn.load_onnx(model=MODEL_PATH)
      #ret = rknn.load_pytorch(model=MODEL_PATH, input_size_list=[[1, 3, 640, 640]])
      if ret != 0:
          print('モデルの読み込みに失敗しました!')
          exit(ret)
      print('完了')

      # モデルのビルド
      print('--> モデルのビルド')
      ret = rknn.build(do_quantization=DEFAULT_QUANT, dataset=DATASET_PATH)
      if ret != 0:
          print('モデルのビルドに失敗しました!')
          exit(ret)
      print('完了')

      # rknn モデルのエクスポート
      print('--> rknn モデルのエクスポート')
      ret = rknn.export_rknn(RKNN_MODEL)
      if ret != 0:
          print('rknn モデルのエクスポートに失敗しました!')
          exit(ret)
      print('完了')

      # 精度分析、出力ディレクトリ./snapshot
      #print('--> 精度分析')
      #ret = rknn.accuracy_analysis(inputs=['./subset/000000052891.jpg'])
```

```
    #if ret != 0:
    #    print('精度分析に失敗しました!')
    #    exit(ret)
    #print('完了')

    # リリース
    rknn.release()
```

onnx2rknn.py プログラム内のモデルパス、エクスポートモデルパス、およびプラットフォームを変更し、プログラムを実行して rknn モデルをエクスポートします（デフォルトで量化を有効）。デフォルトではカレントディレクトリの model/yolov5s-seg.rknn に保存されます。

```
# airockchip/yolov5/export.py により出力された yolov5s-seg.onnx を yolov5-seg フォルダーにコピー
# yolov5-seg フォルダーに移動し、python onnx2rknn.py を実行
(.toolkit2_env) (base) csun@CSUN-PC-0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/yolov5-seg$ python onnx2rknn.py
I rknn-toolkit2 version: 2.0.0b0+9bab5682
--> モデルの設定
完了
--> モデルの読み込み
I It is recommended onnx opset 19, but your onnx model opset is 12!
I Model converted from pytorch, 'opset_version' should be set 19 in torch.onnx.export for successful convert!
I Loading : 100%|████████████████████████████████████████| 139/139 [00:00<00:00, 24309.23it/s]
完了
--> モデルのビルド
W build: found outlier value, this may affect quantization accuracy
                        const name          abs_mean    abs_std     outlier value
                        onnx::Conv_382      0.81        1.29        -13.844
I GraphPreparing : 100%|████████████████████████████████████| 159/159 [00:00<00:00, 9545.88it/s]
I Quantizating : 100%|████████████████████████████████████| 159/159 [00:00<00:00, 289.84it/s]
W build: The default input dtype of 'images' is changed from 'float32' to 'int8' in rknn model for performance!
                    Please take care of this change when deploy rknn model with Runtime API!
W build: The default output dtype of 'output0' is changed from 'float32' to 'int8' in rknn model for performance!
                    Please take care of this change when deploy rknn model with Runtime API!
W build: The default output dtype of 'output1' is changed from 'float32' to 'int8' in rknn model for performance!
                    Please take care of this change when deploy rknn model with Runtime API!
W build: The default output dtype of '376' is changed from 'float32' to 'int8' in rknn model for performance!
                    Please take care of this change when deploy rknn model with Runtime API!
W build: The default output dtype of '377' is changed from 'float32' to 'int8' in rknn model for performance!
                    Please take care of this change when deploy rknn model with Runtime API!
W build: The default output dtype of '379' is changed from 'float32' to 'int8' in rknn model for performance!
                    Please take care of this change when deploy rknn model with Runtime API!
W build: The default output dtype of '380' is changed from 'float32' to 'int8' in rknn model for performance!
                    Please take care of this change when deploy rknn model with Runtime API!
W build: The default output dtype of '371' is changed from 'float32' to 'int8' in rknn model for performance!
                    Please take care of this change when deploy rknn model with Runtime API!
I rknn building ...
I rknn buiding done.
完了
--> rknn モデルのエクスポート
完了
```

rknn-toolkit2 ツールの連携ボードテスト機能を使用するには、ボードを USB adb またはネットワーク adb でホストに接続し、adb 接続が正常であることを確認してから、ボード側の rknn_server を起動します。詳細は前の章の操作手順を参照してください。

サンプルソース 2: test.py（参考用のサンプルプログラム）

```
if __name__ == '__main__':

    if is_architecture('x86_64'):
        print("Current platform is x86_64")
        from rknn.api import RKNN
        # RKNN オブジェクト作成
        rknn = RKNN()
        rknn.list_devices()

        # rknn モデルをロード
        rknn.load_rknn(path=rknn_model_path)

        # 動作環境を設定、デフォルトは rk3588
        ret = rknn.init_runtime(target=target, device_id=device_id)
        if ret != 0:
            print('Init runtime environment failed!')
            exit(ret)

    elif is_architecture('aarch64'):
        print("Current platform is aarch64")
        from rknnlite.api import RKNNLite
        # Create RKNN object
        rknn_lite = RKNNLite()
        # load RKNN model
        print('--> Load RKNN model')
        ret = rknn_lite.load_rknn(rknn_model_path)
        if ret != 0:
            print('Load RKNN model failed!')
            exit(ret)
        print('done')

        # Init runtime environment
        print('--> Init runtime environment')
        ret = rknn_lite.init_runtime(core_mask=RKNNLite.NPU_CORE_0)
        if ret != 0:
            print('Init runtime environment failed!')
            exit(ret)
        print('done')
    else:
        print(f"Current platform is {platform.machine()}")
        exit(0)



    # 入力設定
    # img_src = cv2.imread(IMG_PATH)
    # src_shape = img_src.shape[:2]
    # img, ratio, (dw, dh) = letterbox(img_src, new_shape=(IMG_SIZE, IMG_SIZE))
    # img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    # #img = cv2.resize(img, (IMG_SIZE, IMG_SIZE))
    # img = np.expand_dims(img, 0)

    img_src = cv2.imread(img_path)
```

```
    if img_src is None:
        print("Load image failed!")
        exit()
    src_shape = img_src.shape[:2]
    img, ratio, (dw, dh) = letter_box(img_src, IMG_SIZE)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    # モデル推論
    print('--> Running model')
    if is_architecture('x86_64'):
        outputs = rknn.inference(inputs=[img])
    elif is_architecture('aarch64'):
        outputs = rknn_lite.inference(inputs=[img] , data_format=['nhwc'])
    print('done')

    boxes, classes, scores, seg_img = post_process(outputs, anchors)

    if boxes is not None:
        real_boxs = get_real_box(src_shape, boxes, dw, dh, ratio)
        real_segs = get_real_seg(src_shape, IMG_SIZE, dw, dh, seg_img)
        img_p = merge_seg(img_src, real_segs, classes)
        draw(img_p, real_boxs, scores, classes)
        cv2.imwrite("result.jpg", img_p)

    if is_architecture('x86_64'):
        rknn.release()
    elif is_architecture('aarch64'):
        rknn_lite.release()
```

テストプログラムは、デバイス ID、モデルパス、およびターゲットプラットフォームを変更する必要があります。プログラムを実行すると、結果は result.jpg に保存されます。

rknn-toolkit2 ツールを使用してボードに接続してテストを行い、精度分析、メモリ評価などの操作も行えます。詳細は rknn-toolkit2 マニュアルを参照してください。

18.2.3 ボード上のデプロイメントテスト

次に、rknn_model_zoo リポジトリの提供する C++ デプロイメント例を参照して、ボード上でデプロイメントテストを行います。チュートリアルテストの例は、参考用のサンプルプログラムを参照してください。

1.C++デプロイとテスト

チュートリアルテストは lubancat-4 ボード上で行われます：

```
# 参考用のサンプルプログラムまたは rknn_model_zoo リポジトリのプログラムをボードにダウンロードします
wget https://www.dragonwake.com/download/LubanCat4/7-srccode/tutorial/CSAIEG358803_lubancat_ai_manual_code.zip
# 上記 PC 側に変換された yolov5-seg.rknn をボード側に転送
cat@lubancat:~/lubancat_ai_manual_code/example/yolov5_seg/cpp$ ls
CMakeLists.txt  image_utils.c  postprocess.cc  rknpu2           yolov5_seg.h
yolov5seg_videocapture_demo.cc
build-linux.sh  image_utils.h  postprocess.h   yolov5-seg.rknn  yolov5seg_demo.cc
# ボード上で直接サンプルプログラムをコンパイルします（cmake と gcc のインストールが必要です）。build-linux.sh スクリプトを実行し、-t オプションでターゲットボードを指定します（チュートリアルテストは lubancat-4、rk3588）、-d オプションは rga2 メモリ割り当てを 4G 未満のアドレスで有効にし、-z オプションはゼロコピーインターフェイス（rknn モデル i8 量化）を使用することを有効にします
cat@lubancat:~/lubancat_ai_manual_code/example/yolov5_seg/cpp$ ./build-linux.sh -t rk3588 -d -z
./build-linux.sh -t rk3588 -d -z
==================================
TARGET_SOC=rk3588
INSTALL_DIR=/home/cat/lubancat_ai_manual_code/example/yolov5_seg/cpp/install/rk3588_linux
BUILD_DIR=/home/cat/lubancat_ai_manual_code/example/yolov5_seg/cpp/build/build_rk3588_linux
ENABLE_ALLOC_DMA32=TRUE
ENABLE_ZERO_COPY=TRUE
CC=aarch64-linux-gnu-gcc
CXX=aarch64-linux-gnu-g++
==================================
-- The C compiler identification is GNU 10.2.1
-- The CXX compiler identification is GNU 10.2.1
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /usr/bin/aarch64-linux-gnu-gcc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/aarch64-linux-gnu-g++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Found OpenCV: /usr (found version "4.5.1")
-- Configuring done
-- Generating done
-- Build files have been written to:
/home/cat/lubancat_ai_manual_code/example/yolov5_seg/cpp/build/build_rk3588_linux
Scanning dependencies of target rknn_yolov5seg_demo
Scanning dependencies of target yolov5seg_videocapture_demo
[ 10%] Building CXX object CMakeFiles/rknn_yolov5seg_demo.dir/rknpu2/yolov5_seg.cc.o
[ 40%] Building CXX object CMakeFiles/rknn_yolov5seg_demo.dir/postprocess.cc.o
```

```
[ 40%] Building CXX object CMakeFiles/yolov5seg_videocapture_demo.dir/yolov5seg_videocapture_demo.cc.o
[ 40%] Building CXX object CMakeFiles/rknn_yolov5seg_demo.dir/yolov5seg_demo.cc.o
[ 50%] Building CXX object CMakeFiles/yolov5seg_videocapture_demo.dir/postprocess.cc.o
# ...省略...
[100%] Linking CXX executable yolov5seg_videocapture_demo
[100%] Built target rknn_yolov5seg_demo
[100%] Built target yolov5seg_videocapture_demo
[ 50%] Built target yolov5seg_videocapture_demo
[100%] Built target rknn_yolov5seg_demo
Install the project...
-- Install configuration: ""
-- Installing: /home/cat/lubancat_ai_manual_code/example/yolov5_seg/cpp/install/rk3588_linux/lib/librga.so
-- Installing: /home/cat/lubancat_ai_manual_code/example/yolov5_seg/cpp/install/rk3588_linux/lib/librknnrt.so
-- Installing:
/home/cat/lubancat_ai_manual_code/example/yolov5_seg/cpp/install/rk3588_linux/./rknn_yolov5seg_demo
-- Set runtime path of
"/home/cat/lubancat_ai_manual_code/example/yolov5_seg/cpp/install/rk3588_linux/./rknn_yolov5seg_demo" to
"$ORIGIN/lib"
-- Installing:
/home/cat/lubancat_ai_manual_code/example/yolov5_seg/cpp/install/rk3588_linux/./yolov5seg_videocapture_demo
-- Set runtime path of
"/home/cat/lubancat_ai_manual_code/example/yolov5_seg/cpp/install/rk3588_linux/./yolov5seg_videocapture_demo"
to "$ORIGIN/lib"
-- Installing: /home/cat/lubancat_ai_manual_code/example/yolov5_seg/cpp/install/rk3588_linux/model/bus.jpg
-- Installing:
/home/cat/lubancat_ai_manual_code/example/yolov5_seg/cpp/install/rk3588_linux/model/coco_80_labels_list.txt
-- Installing: /home/cat/lubancat_ai_manual_code/example/yolov5_seg/cpp/install/rk3588_linux/model/yolov5s-
seg.rknn
```

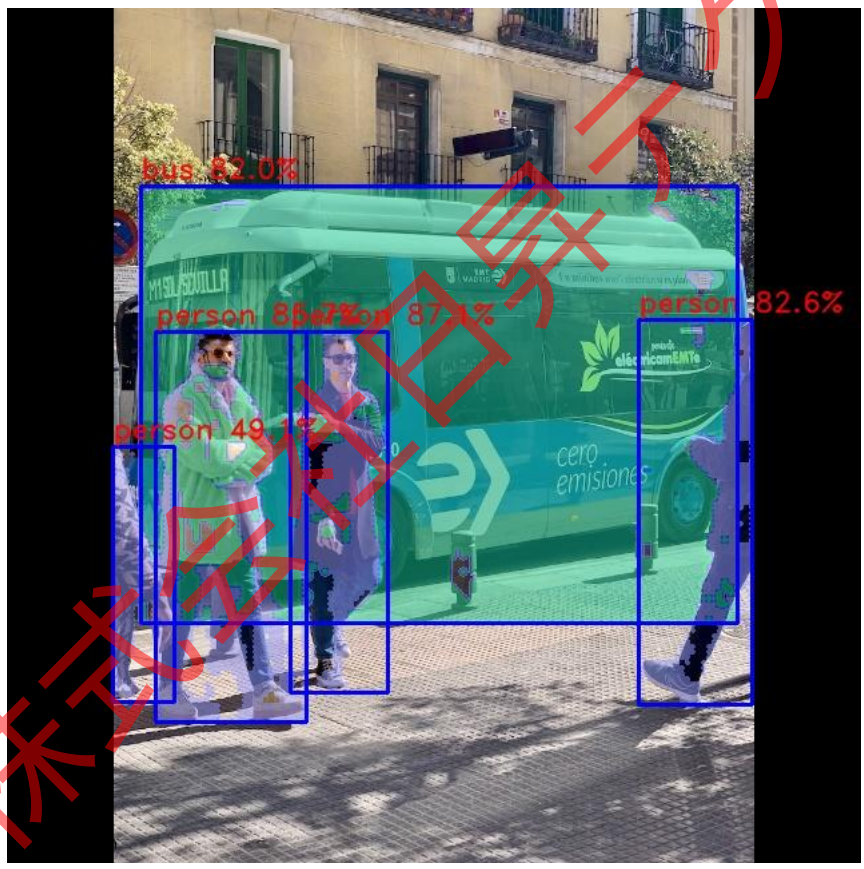
最終的にプログラムは install/rk3588_linux ディレクトリに保存されます。デフォルトでは、rknn_yolov5seg_demo と yolov5seg_videocapture_demo の 2 つのプログラムが生成されます。

rknn_yolov5seg_demo サンプルプログラムを実行すると、プログラムは npu 推論を使用して処理し、結果を保存します。

```
# インストールディレクトリのファイルを確認します
cat@lubancat:~/lubancat_ai_manual_code/example/yolov5_seg/cpp/install/rk3588_linux$ ls
lib  model  rknn_yolov5seg_demo  yolov5seg_videocapture_demo
# プログラムを実行します。rknn_yolov5seg_demo <model_path> <image_path>
cat@lubancat:~/lubancat_ai_manual_code/example/yolov5_seg/cpp/install/rk3588_linux$ ./rknn_yolov5seg_demo ./mod
el/yolov5s-seg.rknn ./model/bus.jpg
load lable ./model/coco_80_labels_list.txt
rknn_api/rknnrt version: 1.6.0 (9a7b5d24c@2023-12-13T17:31:11), driver version: 0.9.2
model input num: 1, output num: 7
input tensors:
  index=0, name=images, n_dims=4, dims=[1, 640, 640, 3], n_elems=1228800, size=1228800, fmt=NHWC, type=INT8,
qnt_type=AFFINE, zp=-128, scale=0.003922
output tensors:
  index=0, name=output0, n_dims=4, dims=[1, 255, 80, 80], n_elems=1632000, size=1632000, fmt=NCHW, type=INT8,
qnt_type=AFFINE, zp=-128, scale=0.003902
  index=1, name=output1, n_dims=4, dims=[1, 96, 80, 80], n_elems=614400, size=614400, fmt=NCHW, type=INT8,
qnt_type=AFFINE, zp=18, scale=0.018347
  index=2, name=376, n_dims=4, dims=[1, 255, 40, 40], n_elems=408000, size=408000, fmt=NCHW, type=INT8,
qnt_type=AFFINE, zp=-128, scale=0.003922
  index=3, name=377, n_dims=4, dims=[1, 96, 40, 40], n_elems=153600, size=153600, fmt=NCHW, type=INT8,
qnt_type=AFFINE, zp=32, scale=0.019994
  index=4, name=379, n_dims=4, dims=[1, 255, 20, 20], n_elems=102000, size=102000, fmt=NCHW, type=INT8,
qnt_type=AFFINE, zp=-128, scale=0.003917
  index=5, name=380, n_dims=4, dims=[1, 96, 20, 20], n_elems=38400, size=38400, fmt=NCHW, type=INT8,
```

```
qnt_type=AFFINE, zp=35, scale=0.021878
  index=6, name=371, n_dims=4, dims=[1, 32, 160, 160], n_elems=819200, size=819200, fmt=NCHW, type=INT8,
qnt_type=AFFINE, zp=-110, scale=0.015716
model is NHWC input fmt
model input height=640, width=640, channel=3
scale=1.000000 dst_box=(0 0 639 639) allow_slight_change=1 _left_offset=0 _top_offset=0 padding_w=0 padding_h=0
src width=640 height=640 fmt=0x1 virAddr=0x0x7fa545b040 fd=0
dst width=640 height=640 fmt=0x1 virAddr=0x0x7f9ebd6000 fd=17
src_box=(0 0 639 639)
dst_box=(0 0 639 639)
color=0x72
rga_api version 1.10.0_[2]
rknn_run
person @ (212 242 285 512) 0.871
person @ (111 242 224 534) 0.857
person @ (473 233 558 521) 0.826
bus @ (100 133 547 460) 0.820
person @ (79 328 125 519) 0.491
rknn run and process use 104.323000 ms
```

結果はカレントディレクトリの out.jpg に保存されます。サンプルプログラムのテスト結果は次の図のようになります：



デスクトップ端末で yolov5seg_videocapture_demo サンプルプログラムを実行すると、カメラでキャプチャした画像を推論処理して表示します。

```
# yolov5seg_videocapture_demo <model path> </path/xxxx.mp4> / <0/1/2>
# ここカメラ/dev/video11 を呼び出す

cat@lubancat:~/lubancat_ai_manual_code/example/yolov5_seg/cpp/install/rk3588_linux$ ./yolov5seg_videocapture_demo ./model/yolov5s-seg.rknn 11
```

テスト前にカメラが正常に接続されていることを確認してください。また、サンプルプログラムは opencv を使用します。システムに opencv がインストールされていることを確認してください。自分でコンパイルした opencv を使用する場合は、サンプルプログラムの CMakeLists.txt ファイルを修正し、opencv ライブラリのパスを正しく設定してください。例えば：

yolov5_seg/cpp/CMakeLists.txt

```
cmake_minimum_required(VERSION 3.10)

project(rknn_yolov5seg_demo)

set(CMAKE_INSTALL_RPATH "$ORIGIN/lib")

SET(CMAKE_CXX_FLAGS "${CMAKE_CXX_FLAGS} -pthread")

# システムのデフォルトの OpenCV を探す
#find_package(OpenCV REQUIRED)

# 自己コンパイルしてインストールした OpenCV を探す、下記の OpenCV_DIR パスを変更してください
set(OpenCV_DIR /path/opencv/lib/cmake/opencv4)
find_package(OpenCV REQUIRED)
file(GLOB OpenCV_FILES "${OpenCV_DIR}/../../libopencv*")
install(PROGRAMS ${OpenCV_FILES} DESTINATION lib)

# 以下省略
# .....
```

2. Python デプロイとテスト

```
# ボード側で実行
cat@lubancat:~/lubancat_ai_manual_code/example/yolov5_seg$ python test.py
Current platform is aarch64
--> Load RKNN model
done
--> Init runtime environment
I RKNN: [22:17:29.851] RKNN Runtime Information, librknnrt version: 2.0.0b0 (35a6907d79@2024-03-24T10:31:14)
I RKNN: [22:17:29.851] RKNN Driver Information, version: 0.9.2
I RKNN: [22:17:29.851] RKNN Model Information, version: 6, toolkit version: 2.0.0b0+9bab5682(compiler version: 2.0.0b0 (35a6907d79@2024-03-24T02:34:11)), target: RKNPU v2, target platform: rk3588, framework name: ONNX, framework layout: NCHW, model inference type: static_shape
done
--> Running model
done
input_data shapes: [(1, 255, 80, 80), (1, 96, 80, 80), (1, 255, 40, 40), (1, 96, 40, 40), (1, 255, 20, 20), (1, 96, 20, 20), (1, 32, 160, 160)]
```

```
person @ (744 41 1152 711) 0.841
person @ (143 203 847 714) 0.806
tie @ (434 439 499 709) 0.704
```

テストプログラムは、モデルパスなどを変更する必要があります。プログラムを実行すると、結果は result.jpg に保存されます。

18.3 参考リンク

https://github.com/ultralytics/yolov5
https://github.com/airockchip/yolov5
https://github.com/airockchip/rknn_model_zoo

第 19 章 YOLOv8

Ultralytics YOLOv8 は、以前の YOLO バージョンの成功を基にして、新しい機能と改善を導入し、性能と柔軟性をさらに向上させました。YOLOv8 は、迅速で正確かつ使いやすい設計であり、物体検出と追跡、インスタンス分割、画像分類、姿勢推定タスクに最適です。

YOLOv8 には、n、s、m、l、および x の 5 つの異なるモデルサイズの事前トレーニングされたモデルがあります。異なるサイズのモデルの物体検出の正確度は以下の通りです：

本章では、YOLOv8 の物体検出と画像分割を簡単にテストし、Lubancat4 ボード上でのデプロイメントをテストします。

ヒント: テスト環境：ボードのシステムは Debian11、PC は ubuntu20.04、PyTorch は 2.1.0、rknn-Toolkit2 バージョン 2.0.0b

19.1. YOLOv8 関連環境インストール

conda を使用して仮想環境を作成し、ultralytics 関連環境をインストールして yolo コマンドをテストします。これにより、後で airockchip/ultralytics_yolov8 を使用してモデルをエクスポートするための環境が整います。 あるいは、モデルエクスポートのセクションに直接進んで airockchip/ultralytics_yolov8 をクローンし、requirements.txt に基づいて環境をインストールすることもできます。

```
# conda を使用して仮想環境を作成
conda create -n yolov8 python=3.8
conda activate yolov8

# pytorch などをインストール、前の節を参考

# YOLOv8 をインストール、コマンドを直接使用してインストール
pip install ultralytics -i https://pypi.org/simple

# あるいはリポジトリをクローンしてインストール
git clone https://github.com/ultralytics/ultralytics.git
cd ultralytics

pip install -e .

# インストールが成功したら、yolo コマンドを使用してバージョンを確認
(.toolkit2_env) (base) csun@CSUN-PC-
0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/yolov8/ultralytics$ yolo version
8.2.48
```

インストール後、yolo コマンドまたは Python プログラムを使用してモデルのトレーニングなどの操作を行うことができます。

19.2. YOLOv8 物体検出

19.2.1. 物体検出の簡単なテスト

環境を簡単にテストするために、まず公式サイトから必要な重みファイルを取得します（ダウンロードしなくても、yolo コマンドで公式の事前トレーニング済みの重みを指定すると自動的にダウンロードされます）。その後、モデルを使用して予測を行います。

```
# 公式リポジトリの重みを取得するためのディレクトリを作成し、
# 重みをダウンロード（https://github.com/ultralytics/ultralytics/tree/main から）
# または手動でダウンロードせず、yolo コマンドで自動的にダウンロードさせます。
wget https://github.com/ultralytics/assets/releases/download/v0.0.0/yolov8n.pt

# テスト画像を取得できますが、以下の場所から取得するか、失敗した場合はサンプルから取得してください
wget https://ultralytics.com/images/bus.jpg
# あるいはリポジトリをクローンしてインストール
git clone https://github.com/ultralytics/ultralytics.git
cd ultralytics

pip install -e .

# インストールが成功したら、yolo コマンドを使用してバージョンを確認
(.toolkit2_env) (base) csun@CSUN-PC-
0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/yolov8/ultralytics$ yolo version
8.2.48
```

その後、yolo コマンドを使用してテストを行います：

```
# 最初のパラメータはタスクを指定します[detect, segment, classify]、ここでは物体検出（detect）をテストします。このパラメータは任意です。
# 第二のパラメータはモードを指定します[train, val, predict, export, track]、トレーニング、評価、予測などを選択します。
# その他のパラメータは、model でモデルを設定し、source で予測する画像のパスを指定し、imgsz で画像のサイズを指定します。詳細なパラメータは、以下を参照してください：https://docs.ultralytics.com/usage/cfg/

# 簡単な物体検出
(.toolkit2_env) (base) csun@CSUN-PC-
0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/yolov8/ultralytics$ yolo
detect predict model=./yolov8n.pt source=./bus.jpg
Ultralytics YOLOv8.2.48    Python-3.8.10 torch-2.1.0+cu121 CUDA:0 (NVIDIA GeForce GTX 1070 Ti,
8106MiB)
YOLOv8n summary (fused): 168 layers, 3151904 parameters, 0 gradients, 8.7 GFLOPs

image 1/1
/home/csun/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/yolov8/ultralytics/bus.jp
g: 640x480 4 persons, 1 bus, 1 stop sign, 71.5ms
Speed: 28.0ms preprocess, 71.5ms inference, 827.5ms postprocess per image at shape (1, 3, 640,
480)
Results saved to runs/detect/predict
   Learn more at https://docs.ultralytics.com/modes/predict
```

```sh
```

予測された画像結果は現在の runs ディレクトリに保存され、具体的なパスは ./runs/detect/predict/bus.jpg です。

テスト結果：

19.2.2. モデルトレーニング

ここでは coco128 をテストし、モデルをトレーニングします：

```
yolo detect train data=coco128.yaml model=yolov8n.pt epochs=100 imgsz=640
```

また、自分のモデルをトレーニングすることもできます。

19.2.3. モデルエクスポート（airockchip/ultralytics_yolov8）

airockchip/ultralytics_yolov8 を使用して、rknpu に適したモデルをエクスポートし、NPU 上でより高い推論効率を得ることができます。このリポジトリでは、モデルが最適化されています：

●dfl 構造は NPU 処理においてパフォーマンスが低いため、モデルの外部に移動されました。
●例えば、6000 個の候補ボックスがある場合、元のモデルは dfl 構造を「ボックス信頼度フィルタリング」の前に配置し、6000 個の候補ボックスすべてに対して dfl 計算を行います。しかし、dfl 構造を「ボックス信頼度フィルタリング」の後に配置することで、フィルタリング後に 100 個の候補ボックスが残ると仮定すると、dfl 部分の計算量は 100 個に減少し、計算リソースと帯域幅リソースの使用が大幅に削減されます。

例えば、6000 個の候補ボックスがあり、検出クラスが 80 クラスの場合、しきい値検索操作は 6000 * 80 = 約 48 万回繰り返され、多くの時間を占有します。したがって、モデルをエクスポートする際に、モデル内に追加の 80 クラス検出目標の合計操作を追加して、信頼度を迅速にフィルタリングします。（この構造は特定の状況で有効であり、モデルのトレーニング結果に依存します）./ultralytics/nn/modules/head.py の 52 行目から 54 行目の位置にあるこの最適化部分のコメントアウトを行うことができます。対応するコードは：

```
cls_sum = torch.clamp(y[-1].sum(1, keepdim=True), 0, 1)
y.append(cls_sum)
```

具体的には、RKOPT_README.md を参照してください。

airockchip/ultralytics_yolov8 モデルのエクスポートをテストします。チュートリアルでは rk_opt_v1 ブランチをテストしており、このブランチは torchscript モデルをエクスポートします。main ブランチを使用するとデフォルトで onnx モデルがエクスポートされるため、rknn に変換する際にモデルロード関数を変更する必要があります。

```
# airockchip/ultralytics_yolov8 をクローン、rk_opt_v1 ブランチ
git clone -b rk_opt_v1 https://github.com/airockchip/ultralytics_yolov8.git
cd ultralytics_yolov8

# トレーニング済みのモデル yolov8n.pt を ultralytics_yolov8 ディレクトリにコピー
# その後、./ultralytics/cfg/default.yaml ファイルを変更し、モデルのパスを設定します：
model: ./yolov8n.pt # (str, optional) モデルファイルのパス、例: yolov8n.pt, yolov8n.yaml
data: ./dataset/coco128.yaml # (str, optional) データファイルのパス、例: coco128.yaml
epochs: 100  # (int) トレーニングエポック数

# モデルをエクスポート
(.toolkit2_env) (base) csun@CSUN-PC-
```

```
0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/yolov8/ultralytics_yolov8$ export PYTHONPATH=./
(.toolkit2_env) (base) csun@CSUN-PC-
0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/yolov8/ultralytics_yolov8$ python ./ultralytics/engine/exporter.py
WARNING    Ultralytics settings reset to default values. This may be due to a possible problem with your settings or a recent ultralytics package update.
View settings with 'yolo settings' or at '/home/csun/.config/Ultralytics/settings.yaml'
Update settings with 'yolo settings key=value', i.e. 'yolo settings runs_dir=path/to/dir'.
Downloading https://github.com/ultralytics/assets/releases/download/v0.0.0/yolov8n.pt to 'yolov8n.pt'...
100%|████████████████████████████████████████████████████████████████████████████████| 6.23M/6.23M [00:00<00:00, 64.1MB/s]
Ultralytics YOLOv8.0.151    Python-3.8.10 torch-2.1.0+cu121 CPU (Intel Core(TM) i7-8700 3.20GHz)
YOLOv8n summary (fused): 168 layers, 3151904 parameters, 0 gradients, 8.7 GFLOPs

PyTorch: starting from 'yolov8n.pt' with input shape (16, 3, 640, 640) BCHW and output shape(s) ((16, 64, 80, 80), (16, 80, 80, 80), (16, 1, 80, 80), (16, 64, 40, 40), (16, 80, 40, 40), (16, 1, 40, 40), (16, 64, 20, 20), (16, 80, 20, 20), (16, 1, 20, 20)) (6.2 MB)

RKNN: starting export with torch 2.1.0+cu121...

RKNN: feed yolov8n_rknnopt.torchscript to RKNN-Toolkit or RKNN-Toolkit2 to generate RKNN model.
Refer https://github.com/airockchip/rknn_model_zoo/tree/main/models/CV/object_detection/yolo
RKNN: export success ✓ 4.9s, saved as 'yolov8n_rknnopt.torchscript' (12.3 MB)

Export complete (7.8s)
Results saved to /home/csun/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/yolov8/ultralytics_yolov8
Predict:         yolo predict task=detect model=yolov8n_rknnopt.torchscript imgsz=640
Validate:        yolo val task=detect model=yolov8n_rknnopt.torchscript imgsz=640 data=coco.yaml
Visualize:       https://netron.app
```

エクスポートされたモデルは、現在のディレクトリ内の yolov8n_rknnopt.torchscript に保存されます。

19.2.4. モデル変換

前述のエクスポートされたモデルは、toolkit2 を使用して rknn モデルに変換する必要があります。toolkit2 のインストールについては前述の環境インストールセクションを参照してください。
rknn_model_zoo リポジトリの例を参考に、モデル変換プログラムをコンパイルし、torchscript モデルを rknn モデルに変換します。

サンプルソース：pt2rknn.py

```
import os
import sys
import numpy as np
from rknn.api import RKNN
```

```
  DATASET_PATH = '../dataset/coco_subset_20.txt'
  # デフォルトで量子化を有効にする
  DEFAULT_QUANT = True

  def parse_arg():
      if len(sys.argv) < 3:
          print("Usage: python3 {} [torchscript_model_path] [platform] [dtype(optional)]
[output_rknn_path(optional)]".format(sys.argv[0]));
          print("       platform choose from [rk3562,rk3566,rk3568,rk3588]")
          print("       dtype choose from    [i8, fp]")
          print("Example: python pt2rknn.py ./yolov8n.onnx rk3588")
          exit(1)

      model_path = sys.argv[1]
      platform = sys.argv[2]

      do_quant = DEFAULT_QUANT
      if len(sys.argv) > 3:
          model_type = sys.argv[3]
          if model_type not in ['i8', 'fp']:
              print("ERROR: Invalid model type: {}".format(model_type))
              exit(1)
          elif model_type == 'i8':
              do_quant = True
          else:
              do_quant = False

      if len(sys.argv) > 4:
          output_path = sys.argv[4]
      else:
          output_path = "./model/yolov8_"+platform+".rknn"

      return model_path, platform, do_quant, output_path

  if __name__ == '__main__':
      model_path, platform, do_quant, output_path = parse_arg()

      # RKNN オブジェクトを作成
      rknn = RKNN(verbose=False)

      # 前処理の設定
      print('--> Config model')
      rknn.config(mean_values=[[0, 0, 0]], std_values=[
                  [255, 255, 255]], target_platform=platform)
      print('done')

      # モデルをロード
      print('--> Loading model')
      #ret = rknn.load_onnx(model=model_path)
      ret = rknn.load_pytorch(model=model_path, input_size_list=[[1, 3, 640, 640]])
      if ret != 0:
          print('Load model failed!')
          exit(ret)
      print('done')

      # モデルをビルド
      print('--> Building model')
      ret = rknn.build(do_quantization=do_quant, dataset=DATASET_PATH)
      if ret != 0:
          print('Build model failed!')
```

```
        exit(ret)
    print('done')

    # rknn モデルをエクスポート
    print('--> Export rknn model')
    ret = rknn.export_rknn(output_path)
    if ret != 0:
        print('Export rknn model failed!')
        exit(ret)
    print('done')

    # 精度分析、出力ディレクトリ./snapshot
    #print('--> Accuracy analysis')
    #ret = rknn.accuracy_analysis(inputs=['./subset/000000052891.jpg'])
    #if ret != 0:
    #    print('Accuracy analysis failed!')
    #    exit(ret)
    #print('done')

    # 解放
    rknn.release()
```

以下のコマンドを実行して、rknn モデルをエクスポートします：

```
# python3 pt2rknn.py を実行してパラメータの説明を確認できます
(.toolkit2_env) (base) csun@CSUN-PC-
0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/yolov8/ultralytics_yolov8$ python3 pt2rknn.py
Usage: python3 pt2rknn.py [torchscript_model_path] [platform] [dtype(optional)] [output_rknn_path(optional)]
       platform choose from [rk3562,rk3566,rk3568,rk3588]
       dtype choose from    [i8, fp]
Example: python pt2rknn.py ./yolov8n.onnx rk3588
# モデルとターゲットを指定して、rknn モデルに変換します。デフォルトで int8 量子化が有効
# サンプルソースから lubancat_ai_manual_code/example/yolov8/dataset をコピーしてください。
(.toolkit2_env) (base) csun@CSUN-PC-
0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/yolov8/ultralytics_yolov8$ mkdir model
(.toolkit2_env) (base) csun@CSUN-PC-
0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/yolov8/ultralytics_yolov8$ python
pt2rknn.py ./yolov8n_rknnopt.torchscript rk3588
I rknn-toolkit2 version: 2.0.0b0+9bab5682
--> Config model
done
--> Loading model
I PtParse: 100%|████████████████████████████████████████| 768/768 [00:00<00:00, 1079.29it/s]
I Loading : 100%|████████████████████████████████████████| 147/147 [00:00<00:00, 840.59it/s]
done
--> Building model
W build: found outlier value, this may affect quantization accuracy
                    const name      abs_mean    abs_std     outlier value
                    weight.92       2.44        2.47        -17.494
                    weight.4        0.09        0.14        -10.215
                    weight.23       0.12        0.19        13.361, 13.317
                    weight.11       0.18        0.20        -11.216
I GraphPreparing : 100%|████████████████████████████████| 161/161 [00:00<00:00, 10972.00it/s]
I Quantizating : 100%|██████████████████████████████████| 161/161 [00:04<00:00, 37.25it/s]
W build: The default input dtype of 'x.1' is changed from 'float32' to 'int8' in rknn model for performance!
                    Please take care of this change when deploy rknn model with Runtime API!
W build: The default output dtype of '189' is changed from 'float32' to 'int8' in rknn model for performance!
                    Please take care of this change when deploy rknn model with Runtime API!
```

```
W build: The default output dtype of 'l90' is changed from 'float32' to 'int8' in rknn model for performance!
                    Please take care of this change when deploy rknn model with Runtime API!
W build: The default output dtype of 'l91' is changed from 'float32' to 'int8' in rknn model for performance!
                    Please take care of this change when deploy rknn model with Runtime API!
W build: The default output dtype of 'l92' is changed from 'float32' to 'int8' in rknn model for performance!
                    Please take care of this change when deploy rknn model with Runtime API!
W build: The default output dtype of 'l93' is changed from 'float32' to 'int8' in rknn model for performance!
                    Please take care of this change when deploy rknn model with Runtime API!
W build: The default output dtype of 'l94' is changed from 'float32' to 'int8' in rknn model for performance!
                    Please take care of this change when deploy rknn model with Runtime API!
W build: The default output dtype of 'l95' is changed from 'float32' to 'int8' in rknn model for performance!
                    Please take care of this change when deploy rknn model with Runtime API!
W build: The default output dtype of 'l96' is changed from 'float32' to 'int8' in rknn model for performance!
                    Please take care of this change when deploy rknn model with Runtime API!
W build: The default output dtype of 'l97' is changed from 'float32' to 'int8' in rknn model for performance!
                    Please take care of this change when deploy rknn model with Runtime API!
I rknn building ...
I rknn buiding done.
done
--> Export rknn model
done
```

エクスポートされた rknn モデルを使用して、toolkit2 でボードに接続し、簡単なモデルテストやメモリ、パフォーマンス評価を行います。rknn_model_zoo リポジトリが提供する例を使用することができます。ここではその例に基づいて説明します：

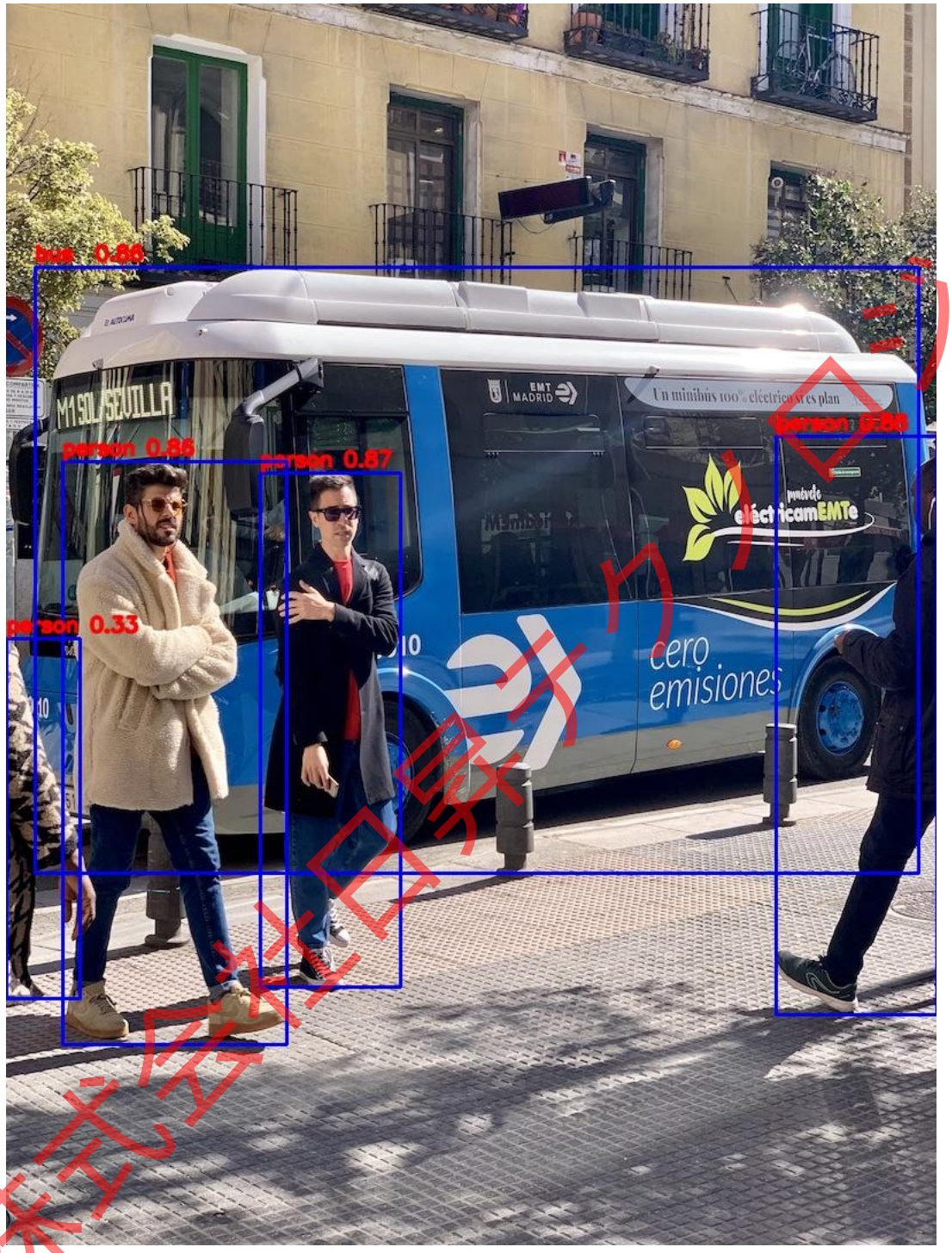
サンプルソース test.py：

```python
if __name__ == '__main__':

    rknn = RKNN()
    rknn.list_devices()

    # rknn モデルをロード
    rknn.load_rknn(path=rknn_model_path)

    # 実行環境を設定し、デフォルトターゲットは rk3588
    ret = rknn.init_runtime(target=target, device_id=device_id)

    # 入力画像
    img_src = cv2.imread(img_path)
    src_shape = img_src.shape[:2]
    img, ratio, (dw, dh) = letter_box(img_src, IMG_SIZE)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    #img = cv2.resize(img_src, IMG_SIZE)

    # 推論を実行
    print('--> Running model')
    outputs = rknn.inference(inputs=[img])
    print('done')

    # 後処理
    boxes, classes, scores = post_process(outputs)

    img_p = img_src.copy()
    if boxes is not None:
        draw(img_p, get_real_box(src_shape, boxes, dw, dh, ratio), scores, classes)
        cv2.imwrite("result.jpg", img_p)
```

コマンドを実行する前に、PC が USB または Ethernet ケーブルを介してボードに接続されていることを確認し、adb デバイスが接続されていることを確認してください。ボードで rknn_server または restart_rknn.sh コマンドを実行し、rknn_server を起動します。PC の Ubuntu でコマンドを実行します（チュートリアルでテストされたボードは lubancat-4 です）：

```
(.toolkit2_env) (base) csun@CSUN-PC-
0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/yolov8/ultralytics_yolov8$ python test.py
I rknn-toolkit2 version: 2.0.0b0+9bab5682
**************************
all device(s) with adb mode:
192.168.11.241:5555
**************************
adb: unable to connect for root: closed
I target set by user is: rk3588
I Get hardware info: target_platform = rk3588, os = Linux, aarch = aarch64
I Check RK3588 board npu runtime version
I Starting ntp or adb, target is RK3588
I Start adb...
I Connect to Device success!
I NPUTransfer: Starting NPU Transfer Client, Transfer version 2.1.0 (b5861e7@2020-11-23T11:50:36)
D RKNNAPI: ==========================================
D RKNNAPI: RKNN VERSION:
D RKNNAPI:   API: 2.0.0b0 (18eacd0 build@2024-03-22T06:07:59)
D RKNNAPI:   DRV: rknn_server: 2.0.0b0 (18eacd0 build@2024-03-22T14:07:19)
D RKNNAPI:   DRV: rknnrt: 2.0.0b0 (35a6907d79@2024-03-24T10:31:14)
D RKNNAPI: ==========================================
D RKNNAPI: Input tensors:
D RKNNAPI:   index=0, name=x.1, n_dims=4, dims=[1, 640, 640, 3], n_elems=1228800, size=1228800, w_stride = 0,
size_with_stride = 0, fmt=NHWC, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003922
D RKNNAPI: Output tensors:
D RKNNAPI:   index=0, name=189, n_dims=4, dims=[1, 64, 80, 80], n_elems=409600, size=409600, w_stride = 0,
size_with_stride = 0, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-58, scale=0.117659
D RKNNAPI:   index=1, name=190, n_dims=4, dims=[1, 80, 80, 80], n_elems=512000, size=512000, w_stride = 0,
size_with_stride = 0, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003104
D RKNNAPI:   index=2, name=191, n_dims=4, dims=[1, 1, 80, 80], n_elems=6400, size=6400, w_stride = 0,
size_with_stride = 0, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003922
D RKNNAPI:   index=3, name=192, n_dims=4, dims=[1, 64, 40, 40], n_elems=102400, size=102400, w_stride = 0,
size_with_stride = 0, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-45, scale=0.093747
D RKNNAPI:   index=4, name=193, n_dims=4, dims=[1, 80, 40, 40], n_elems=128000, size=128000, w_stride = 0,
size_with_stride = 0, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003594
D RKNNAPI:   index=5, name=194, n_dims=4, dims=[1, 1, 40, 40], n_elems=1600, size=1600, w_stride = 0,
size_with_stride = 0, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003922
D RKNNAPI:   index=6, name=195, n_dims=4, dims=[1, 64, 20, 20], n_elems=25600, size=25600, w_stride = 0,
size_with_stride = 0, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-34, scale=0.083036
D RKNNAPI:   index=7, name=196, n_dims=4, dims=[1, 80, 20, 20], n_elems=32000, size=32000, w_stride = 0,
size_with_stride = 0, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003874
D RKNNAPI:   index=8, name=197, n_dims=4, dims=[1, 1, 20, 20], n_elems=400, size=400, w_stride = 0,
size_with_stride = 0, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003922
--> Running model
W inference: The 'data_format' is not set, and its default value is 'nhwc'!
done
person @ (670 376 810 879) 0.883
person @ (221 408 343 855) 0.872
person @ (49 398 244 905) 0.864
person @ (0 552 64 865) 0.327
bus  @ (25 229 794 756) 0.860
```

最終的な結果は現在のディレクトリ内の result.jpg に保存されます。



さらに詳しいテストについては、rknn_model_zoo リポジトリを参照してください。

19.2.5. ボード上でのデプロイ推論

1. rknn_model_zoo リポジトリが提供する RKNN_C_demo を使用し、ボードにデプロイします。テストは lubancat-4 ボードで行い、Debian11 システムで、デフォルトのシステムクロックです。

1.C++デプロイとテスト

コマンドを実行してモデル推論を行います：

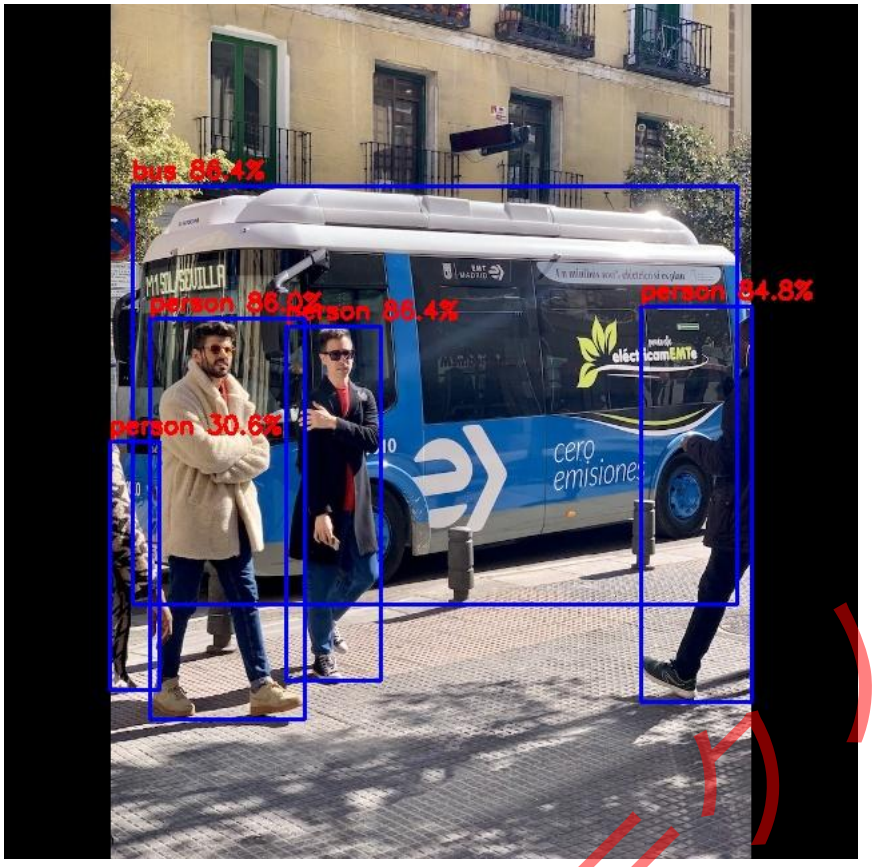
```
# サンプルソースから lubancat_ai_manual_code/example/yolov8/yolov5_det をボードにコピー
# build-linux_RK3588.sh スクリプトを実行し、プロジェクトをビルドします（システムのデフォルトコンパイラを使用）。
最終的に生成されたファイルは build/ディレクトリにインストールされます。
cat@lubancat:~/lubancat_ai_manual_code/example/yolov8/yolov8_det/cpp$ ./build-linux.sh -t rk3588
./build-linux.sh -t rk3588
==================================
TARGET_SOC=rk3588
INSTALL_DIR=/home/cat/lubancat_ai_manual_code/example/yolov8/yolov8_det/cpp/install/rk3588_linu
x
BUILD_DIR=/home/cat/lubancat_ai_manual_code/example/yolov8/yolov8_det/cpp/build/build_rk3588_li
nux
ENABLE_DMA32=OFF
CC=aarch64-linux-gnu-gcc
CXX=aarch64-linux-gnu-g++
==================================
-- The C compiler identification is GNU 10.2.1
-- The CXX compiler identification is GNU 10.2.1
-- Detecting C compiler ABI info
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /usr/bin/aarch64-linux-gnu-gcc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/aarch64-linux-gnu-g++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Found OpenCV: /usr (found version "4.5.1")
-- Configuring done
-- Generating done
-- Build files have been written to:
/home/cat/lubancat_ai_manual_code/example/yolov8/yolov8_det/cpp/build/build_rk3588_linux
Scanning dependencies of target yolov8_videocapture_demo
Scanning dependencies of target rknn_yolov8_demo
# ...省略...
[100%] Built target yolov8_videocapture_demo
[ 50%] Built target yolov8_videocapture_demo
[100%] Built target rknn_yolov8_demo
Install the project...
-- Install configuration: ""
-- Installing:
/home/cat/lubancat_ai_manual_code/example/yolov8/yolov8_det/cpp/install/rk3588_linux/lib/librga.so
-- Installing:
/home/cat/lubancat_ai_manual_code/example/yolov8/yolov8_det/cpp/install/rk3588_linux/lib/librknnrt.so
-- Installing:
/home/cat/lubancat_ai_manual_code/example/yolov8/yolov8_det/cpp/install/rk3588_linux/./rknn_yolov8_demo
-- Set runtime path of
"/home/cat/lubancat_ai_manual_code/example/yolov8/yolov8_det/cpp/install/rk3588_linux/./rknn_yolov8_demo" to
"$ORIGIN/lib"
-- Installing:
/home/cat/lubancat_ai_manual_code/example/yolov8/yolov8_det/cpp/install/rk3588_linux/./yolov8_videocapture_demo
-- Set runtime path of
```

```
"/home/cat/lubancat_ai_manual_code/example/yolov8/yolov8_det/cpp/install/rk3588_linux/./yolov8_videocapture_demo" to "$ORIGIN/lib"
-- Installing:
/home/cat/lubancat_ai_manual_code/example/yolov8/yolov8_det/cpp/install/rk3588_linux/model/bus.jpg
-- Installing:
/home/cat/lubancat_ai_manual_code/example/yolov8/yolov8_det/cpp/install/rk3588_linux/model/coco_80_labels_list.txt
-- Installing:
/home/cat/lubancat_ai_manual_code/example/yolov8/yolov8_det/cpp/install/rk3588_linux/model/yolov8_rk3588.rknn
```

現在のディレクトリ install/rk3588_linux に切り替え、以下のコマンドを実行します：

```
cat@lubancat:~/lubancat_ai_manual_code/example/yolov8/yolov8_det/cpp/install/rk3588_linux$ ./rknn_yolov8_demo ./model/yolov8_rk3588.rknn ./model/bus.jpg
load lable ./model/coco_80_labels_list.txt
model input num: 1, output num: 9
input tensors:
  index=0, name=x.1, n_dims=4, dims=[1, 640, 640, 3], n_elems=1228800, size=1228800, fmt=NHWC, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003922
output tensors:
  index=0, name=189, n_dims=4, dims=[1, 64, 80, 80], n_elems=409600, size=409600, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-58, scale=0.117659
  index=1, name=190, n_dims=4, dims=[1, 80, 80, 80], n_elems=512000, size=512000, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003104
  index=2, name=191, n_dims=4, dims=[1, 1, 80, 80], n_elems=6400, size=6400, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003922
  index=3, name=192, n_dims=4, dims=[1, 64, 40, 40], n_elems=102400, size=102400, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-45, scale=0.093747
  index=4, name=193, n_dims=4, dims=[1, 80, 40, 40], n_elems=128000, size=128000, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003594
  index=5, name=194, n_dims=4, dims=[1, 1, 40, 40], n_elems=1600, size=1600, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003922
  index=6, name=195, n_dims=4, dims=[1, 64, 20, 20], n_elems=25600, size=25600, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-34, scale=0.083036
  index=7, name=196, n_dims=4, dims=[1, 80, 20, 20], n_elems=32000, size=32000, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003874
  index=8, name=197, n_dims=4, dims=[1, 1, 20, 20], n_elems=400, size=400, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003922
model is NHWC input fmt
model input height=640, width=640, channel=3
scale=1.000000 dst_box=(0 0 639 639) allow_slight_change=1 _left_offset=0 _top_offset=0 padding_w=0 padding_h=0
src width=640 height=640 fmt=0x1 virAddr=0x0x55757f4d40 fd=0
dst width=640 height=640 fmt=0x1 virAddr=0x0x5575920d70 fd=0
src_box=(0 0 639 639)
dst_box=(0 0 639 639)
color=0x72
rga_api version 1.10.0_[2]
rknn_run
person @ (211 241 282 506) 0.864
bus @ (96 136 549 449) 0.864
person @ (109 235 225 535) 0.860
person @ (477 226 560 522) 0.848
person @ (79 327 116 513) 0.306
rknn run and process use 42.041000 ms
```

結果は現在のディレクトリ内の result.jpg に保存されます。結果の表示：



上記の C++ デプロイメント例は、OpenCV を使用して画像を読み込み、保存し、librga を使用して画像を処理します。他の画像読み込みおよび処理方法については、rknn_model_zoo リポジトリを参照してください。上記のプログラムが RGA 関連の問題を報告する場合は、Rockchip_FAQ_RGA_EN.md を参照してください。

2. Python デプロイとテスト

サンプルソース：yolov8/yolov8_det/test.py

```
if __name__ == '__main__':
    if is_architecture('x86_64'):
        print("Current platform is x86_64")
        from rknn.api import RKNN
        # RKNN オブジェクト作成
        rknn = RKNN()
        rknn.list_devices()

        # rknn モデルをロード
        rknn.load_rknn(path=rknn_model_path)

        # 動作環境を設定、デフォルトは rk3588
        ret = rknn.init_runtime(target=target, device_id=device_id)
        if ret != 0:
            print('Init runtime environment failed!')
            exit(ret)

    elif is_architecture('aarch64'):
        print("Current platform is aarch64")
        from rknnlite.api import RKNNLite
        # Create RKNN object
        rknn_lite = RKNNLite()
        # load RKNN model
```

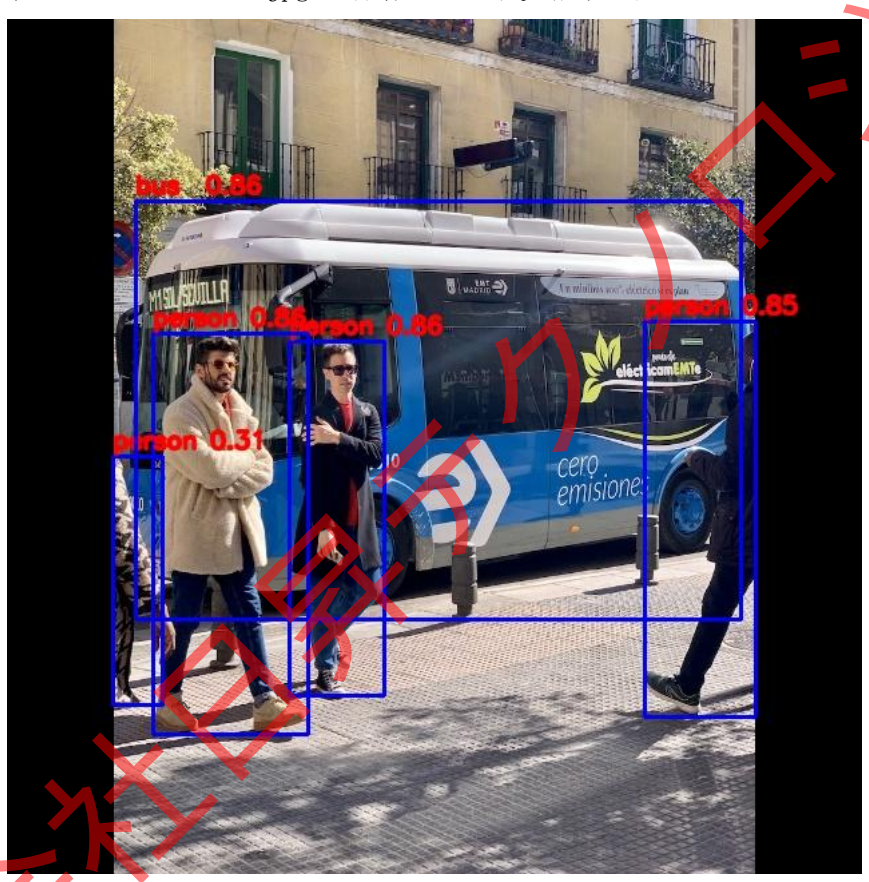
```
        print('--> Load RKNN model')
        ret = rknn_lite.load_rknn(rknn_model_path)
        if ret != 0:
            print('Load RKNN model failed!')
            exit(ret)
        print('done')

        # Init runtime environment
        print('--> Init runtime environment')
        ret = rknn_lite.init_runtime(core_mask=RKNNLite.NPU_CORE_0)
        if ret != 0:
            print('Init runtime environment failed!')
            exit(ret)
        print('done')
    else:
        print(f"Current platform is {platform.machine()}")
        exit(0)

    # 入力画像
    img_src = cv2.imread(img_path)
    src_shape = img_src.shape[:2]
    img, ratio, (dw, dh) = letter_box(img_src, IMG_SIZE)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    #img = cv2.resize(img_src, IMG_SIZE)

    # 推論を実行
    print('--> Running model')
    if is_architecture('x86_64'):
        outputs = rknn.inference(inputs=[img])
    elif is_architecture('aarch64'):
        img = np.expand_dims(img, 0)
        outputs = rknn_lite.inference(inputs=[img])
    print('done')

    # 後処理
    boxes, classes, scores = post_process(outputs)

    img_p = img_src.copy()
    if boxes is not None:
        draw(img_p, get_real_box(src_shape, boxes, dw, dh, ratio), scores, classes)
        cv2.imwrite("result.jpg", img_p)

    if is_architecture('x86_64'):
        rknn.release()
    elif is_architecture('aarch64'):
        rknn_lite.release()
```

ボード側で推論を実行：

```
cat@lubancat:~/lubancat_ai_manual_code/example/yolov8/yolov8_det$ python test.py
Current platform is aarch64
--> Load RKNN model
done
--> Init runtime environment
I RKNN: [15:32:41.993] RKNN Runtime Information, librknnrt version: 2.0.0b0 (35a6907d79@2024-03-24T10:31:14)
I RKNN: [15:32:41.994] RKNN Driver Information, version: 0.9.2
I RKNN: [15:32:41.994] RKNN Model Information, version: 6, toolkit version: 2.0.0b0+9bab5682(compiler
version: 2.0.0b0 (35a6907d79@2024-03-24T02:34:11)), target: RKNPU v2, target platform: rk3588, framework name:
PyTorch, framework layout: NCHW, model inference type: static_shape
done
```

```
--> Running model
done
person @ (211 241 282 506) 0.864
person @ (109 235 225 535) 0.860
person @ (477 226 560 522) 0.848
person @ (79 327 116 513) 0.306
bus  @ (96 136 549 449) 0.864
```

結果は現在のディレクトリ内の out.jpg に保存されます。結果の表示：



19.3. YOLOv8 インスタンスセグメンテーション

19.3.1. インスタンスセグメンテーションテスト

YOLOv8 はインスタンスセグメンテーション（画像セグメンテーション）をサポートしており、yolo コマンドを使用して直接テストできます：

```
# yolo コマンドのパラメータ説明:
# 最初のパラメータはタスクを指定します[detect, segment, classify]、ここではインスタンスセグメンテーション（segment）をテストします。このパラメータは任意です。
# 第二のパラメータはモデルを設定します。このパラメータは必須です。
# その他のパラメータは、source で予測する画像のパスまたはビデオを指定し、imgsz で画像のサイズを指定します。詳細なパラメータは以下を参照してください：https://docs.ultralytics.com/usage/cfg/

# 簡単な画像セグメンテーションテスト、以下のコマンドは公式リポジトリから事前トレーニングされたモデル yolov8n-seg.pt を取得し、推論を行います
```

```
(.toolkit2_env) (base) csun@CSUN-PC-0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/yolov8/ultralytics$ yolo segment predict model=yolov8n-seg.pt source='https://ultralytics.com/images/bus.jpg'
Downloading https://github.com/ultralytics/assets/releases/download/v8.2.0/yolov8n-seg.pt to 'yolov8n-seg.pt'...
100%|████████████████████████████████████████████████████████████████████████| 6.74M/6.74M [00:00<00:00, 16.9MB/s]
Ultralytics YOLOv8.2.48    Python-3.8.10 torch-2.1.0+cu121 CUDA:0 (NVIDIA GeForce GTX 1070 Ti, 8106MiB)
YOLOv8n-seg summary (fused): 195 layers, 3404320 parameters, 0 gradients, 12.6 GFLOPs

Found https://ultralytics.com/images/bus.jpg locally at bus.jpg
image 1/1 /home/csun/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/yolov8/ultralytics/bus.jpg: 640x480 4 persons, 1 bus, 1 skateboard, 45.7ms
Speed: 1.5ms preprocess, 45.7ms inference, 248.0ms postprocess per image at shape (1, 3, 640, 480)
Results saved to /home/csun/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/yolov8/ultralytics_yolov8/runs/segment/predict
   Learn more at https://docs.ultralytics.com/modes/predict
```

結果は現在のディレクトリ内の runs/segment/predict に保存されます：

19.3.2. モデルトレーニング

coco128-seg.yaml を使用してトレーニングをテストすることも、自分のデータセットを使用してトレーニングすることもできます。

```
# 簡単なトレーニング、事前トレーニングされたモデル yolov8n-seg.pt に基づいて、coco128 を使用
yolo segment train data=coco128-seg.yaml model=yolov8n-seg.pt epochs=100 imgsz=640
Speed: 0.2ms preprocess, 7.3ms inference, 0.0ms loss, 0.7ms postprocess per image
Results saved to
/home/csun/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/yolov8/ultralytics_yolov8
```

```
/runs/segment/train
(.toolkit2_env) (base) csun@CSUN-PC-
0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/yolov8/ultralytics_yolov8$ ls
runs/segment/train
BoxF1_curve.png  MaskF1_curve.png  args.yaml                          labels_correlogram.jpg
train_batch1.jpg     train_batch722.jpg     val_batch1_pred.jpg
BoxPR_curve.png  MaskPR_curve.png  confusion_matrix.png               results.csv
train_batch2.jpg     val_batch0_labels.jpg  val_batch2_labels.jpg
BoxP_curve.png   MaskP_curve.png   confusion_matrix_normalized.png  results.png
train_batch720.jpg   val_batch0_pred.jpg    val_batch2_pred.jpg
BoxR_curve.png   MaskR_curve.png   labels.jpg                         train_batch0.jpg
train_batch721.jpg   val_batch1_labels.jpg  weights
# レーニングされたモデルを簡単に評価する
yolo segment val model=runs/segment/train/weights/best.pt
(.toolkit2_env) (base) csun@CSUN-PC-
0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/yolov8/ultralytics_yolov8$ yolo segment val
model=runs/segment/train/weights/best.pt
WARNING    'data' is missing. Using default 'data=coco8-seg.yaml'.
Ultralytics YOLOv8.0.151    Python-3.8.10 torch-2.1.0+cu121 CUDA:0 (NVIDIA GeForce GTX 1070 Ti, 8106MiB)
YOLOv8n-seg summary (fused): 195 layers, 3404320 parameters, 0 gradients, 12.6 GFLOPs

Dataset 'coco8-seg.yaml' images not found   , missing path
'/home/csun/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/yolov8/datasets/coco8-seg/images/val'
Downloading https://ultralytics.com/assets/coco8-seg.zip to
'/home/csun/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/yolov8/datasets/coco8-seg.zip'...
100%|████████████████████████████████████████████████████████████████████████████████████████████
████████████████████████| 439k/439k [00:00<00:00, 4.36MB/s]
Unzipping /home/csun/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/yolov8/datasets/coco8-seg.zip
to /home/csun/linuxshare/work/AI/jupyter/lubancat_ai_manual_code
Dataset download success ✅ (2.1s), saved to
/home/csun/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/yolov8/datasets

val: Scanning /home/csun/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/yolov8/datasets/coco8-
seg/labels/val... 4 images, 0 backgrounds, 0 corrupt: 100%|████████
val: New cache created:
/home/csun/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/yolov8/datasets/coco8-
seg/labels/val.cache
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95)     Mask(P          R
mAP50  mAP50-95): 100%|██████████| 1/1 [00:00<00:00,  2.28
                   all          4         17      0.705      0.962      0.961      0.825      0.705      0.962
0.958      0.716
                person          4         10      0.794      0.774      0.793      0.521      0.794      0.774
0.771       0.34
                   dog          4          1      0.624          1      0.995      0.895      0.624          1
0.995      0.895
                 horse          4          2      0.574          1      0.995      0.946      0.574          1
0.995      0.424
              elephant          4          2      0.985          1      0.995      0.797      0.985          1
0.995      0.747
              umbrella          4          1      0.626          1      0.995      0.995      0.626          1
0.995      0.995
          potted plant          4          1      0.624          1      0.995      0.796      0.624          1
0.995      0.895
Speed: 0.3ms preprocess, 15.1ms inference, 0.0ms loss, 71.4ms postprocess per image
Results saved to
/home/csun/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/yolov8/ultralytics_yolov8/runs/segment/va
l
```

19.3.3. モデルエクスポート

airockchip/ultralytics_yolov8 を使用して、rknpu にデプロイするためのモデルをエクスポートします。このモデルは NPU 上でより高い推論効率を得ることができます。

```
# 現在のディレクトリで、airockchip/ultralytics_yolov8 の main ブランチをクローン
git clone https://github.com/airockchip/ultralytics_yolov8.git
cd ultralytics_yolov8

# ultralytics/cfg/default.yaml 内のモデルファイルのパスを変更します。前述のトレーニングされたモデル、または事前トレーニングされたモデル yolov8n-seg.pt を使用できます。
model: ./runs/segment/train/weights/best.pt # (str, optional) モデルファイルのパス
data: # (str, optional) データファイルのパス、例: coco128.yaml
epochs: 100  # (int) トレーニングエポック数

# python ./ultralytics/engine/exporter.py を実行してモデルをエクスポートします
(.toolkit2_env) (base) csun@CSUN-PC-
0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/yolov8/ultralytics_yolov8$ export
PYTHONPATH=./
(.toolkit2_env) (base) csun@CSUN-PC-
0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/yolov8/ultralytics_yolov8$ python ./ultralyti
cs/engine/exporter.py
Ultralytics YOLOv8.0.151    Python-3.8.10 torch-2.1.0+cu121 CPU ()
YOLOv8n-seg summary (fused): 195 layers, 3404320 parameters, 0 gradients, 12.6 GFLOPs

PyTorch: starting from 'runs/segment/train/weights/best.pt' with input shape (16, 3, 640, 640) BCHW and output
shape(s) ((16, 64, 80, 80), (16, 80, 80, 80), (16, 1, 80, 80), (16, 32, 80, 80), (16, 64, 40, 40), (16, 80, 40,
40), (16, 1, 40, 40), (16, 32, 40, 40), (16, 64, 20, 20), (16, 80, 20, 20), (16, 1, 20, 20), (16, 32, 20, 20),
(16, 32, 160, 160)) (6.7 MB)

RKNN: starting export with torch 2.1.0+cu121...

RKNN: feed runs/segment/train/weights/best.onnx to RKNN-Toolkit or RKNN-Toolkit2 to generate RKNN model.
Refer https://github.com/airockchip/rknn_model_zoo/tree/main/models/CV/object_detection/yolo
RKNN: export success ✅ 0.4s, saved as 'runs/segment/train/weights/best.onnx' (13.0 MB)

Export complete (2.8s)
Results saved to
/home/csun/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/yolov8/ultralytics_yolov8/runs/segment/tr
ain/weights
Predict:         yolo predict task=segment model=runs/segment/train/weights/best.onnx imgsz=640
Validate:        yolo val task=segment model=runs/segment/train/weights/best.onnx imgsz=640
data=/home/csun/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/yolov8/ultralytics_yolov8/ultralytic
s/cfg/datasets/coco128-seg.yaml
Visualize:       https://netron.app
```

モデルは対応するディレクトリ内（../runs/segment/train/weights/best.onnx）に保存され、識別しやすいように yolov8n-seg.onnx にリネームできます

エクスポートされた onnx モデルは、netron を使用してそのネットワーク構造を確認できます。

19.3.4. rknn モデルへの変換

エクスポートされた onnx モデルは、toolkit2 を使用して rknn モデルに変換する必要があります。ここでは、onnx モデルを rknn モデルに変換するために簡単にモデル変換プログラムをコンパイルします。

サンプルソース onnx2rknn.py

```
if __name__ == '__main__':
    model_path, platform, do_quant, output_path = parse_arg()

    # Create RKNN object
    rknn = RKNN(verbose=False)

    # Pre-process config
    print('--> Config model')
    rknn.config(mean_values=[[0, 0, 0]], std_values=[
                    [255, 255, 255]], target_platform=platform)
    print('done')

    # Load model
    print('--> Loading model')
    ret = rknn.load_onnx(model=model_path)
    #ret = rknn.load_pytorch(model=model_path, input_size_list=[[1, 3, 640, 640]])
    if ret != 0:
        print('Load model failed!')
        exit(ret)
    print('done')

    # Build model
    print('--> Building model')
    ret = rknn.build(do_quantization=do_quant, dataset=DATASET_PATH)
    if ret != 0:
        print('Build model failed!')
        exit(ret)
    print('done')

    # Export rknn model
    print('--> Export rknn model')
    ret = rknn.export_rknn(output_path)
    if ret != 0:
        print('Export rknn model failed!')
        exit(ret)
    print('done')

    # 精度分析、出力ディレクトリ./snapshot
    #print('--> Accuracy analysis')
    #ret = rknn.accuracy_analysis(inputs=['./subset/000000052891.jpg'])
    #if ret != 0:
    #    print('Accuracy analysis failed!')
    #    exit(ret)
    #print('done')

    # Release
    rknn.release()
```

サンプルコードをダウンロードし、以下のコマンドを実行して onnx モデルを rknn モデルに変換

します：

```
  (.toolkit2_env) (base) csun@CSUN-PC-
0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/yolov8/yolov8_seg$ mkdir model
  (.toolkit2_env) (base) csun@CSUN-PC-
0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/yolov8/yolov8_seg$ python3 onnx2rknn.py
yolov8n-seg.onnx rk3588
  I rknn-toolkit2 version: 2.0.0b0+9bab5682
  --> Config model
  done
  --> Loading model
  I It is recommended onnx opset 19, but your onnx model opset is 12!
  I Model converted from pytorch, 'opset_version' should be set 19 in torch.onnx.export for successful convert!
  I Loading : 100%|████████████████████████████████████████| 162/162 [00:00<00:00, 50436.26it/s]
  done
  --> Building model
  W build: found outlier value, this may affect quantization accuracy
                        const name                      abs_mean    abs_std     outlier value
                        model.22.cv3.1.1.conv.weight    0.12        0.18        -12.212
  I GraphPreparing : 100%|████████████████████████████████████| 183/183 [00:00<00:00, 10852.55it/s]
  I Quantizating : 100%|████████████████████████████████████| 183/183 [00:05<00:00, 32.21it/s]
  W build: The default input dtype of 'images' is changed from 'float32' to 'int8' in rknn model for
performance!
                         Please take care of this change when deploy rknn model with Runtime API!
  W build: The default output dtype of '375' is changed from 'float32' to 'int8' in rknn model for performance!
                         Please take care of this change when deploy rknn model with Runtime API!
  W build: The default output dtype of 'onnx::ReduceSum_383' is changed from 'float32' to 'int8' in rknn model
for performance!
                         Please take care of this change when deploy rknn model with Runtime API!
  W build: The default output dtype of '388' is changed from 'float32' to 'int8' in rknn model for performance!
                         Please take care of this change when deploy rknn model with Runtime API!
  W build: The default output dtype of '354' is changed from 'float32' to 'int8' in rknn model for performance!
                         Please take care of this change when deploy rknn model with Runtime API!
  W build: The default output dtype of '395' is changed from 'float32' to 'int8' in rknn model for performance!
                         Please take care of this change when deploy rknn model with Runtime API!
  W build: The default output dtype of 'onnx::ReduceSum_403' is changed from 'float32' to 'int8' in rknn model
for performance!
                         Please take care of this change when deploy rknn model with Runtime API!
  W build: The default output dtype of '407' is changed from 'float32' to 'int8' in rknn model for performance!
                         Please take care of this change when deploy rknn model with Runtime API!
  W build: The default output dtype of '361' is changed from 'float32' to 'int8' in rknn model for performance!
                         Please take care of this change when deploy rknn model with Runtime API!
  W build: The default output dtype of '414' is changed from 'float32' to 'int8' in rknn model for performance!
                         Please take care of this change when deploy rknn model with Runtime API!
  W build: The default output dtype of 'onnx::ReduceSum_422' is changed from 'float32' to 'int8' in rknn model
for performance!
                         Please take care of this change when deploy rknn model with Runtime API!
  W build: The default output dtype of '426' is changed from 'float32' to 'int8' in rknn model for performance!
                         Please take care of this change when deploy rknn model with Runtime API!
  W build: The default output dtype of '368' is changed from 'float32' to 'int8' in rknn model for performance!
                         Please take care of this change when deploy rknn model with Runtime API!
  W build: The default output dtype of '347' is changed from 'float32' to 'int8' in rknn model for performance!
                         Please take care of this change when deploy rknn model with Runtime API!
  I rknn building ...
  I rknn buiding done.
  done
  --> Export rknn model
  done
```

rknn モデルに変換された後、toolkit2 を使用してボードに接続し、モデルの簡単なテストやメモリ、パフォーマンス評価を行います。

サンプルソース test.py：

```
if __name__ == '__main__':
    model_path, platform, do_quant, output_path = parse_arg()

    # Create RKNN object
    rknn = RKNN(verbose=False)

    # Pre-process config
    print('--> Config model')
    rknn.config(mean_values=[[0, 0, 0]], std_values=[
                    [255, 255, 255]], target_platform=platform)
    print('done')

    # Load model
    print('--> Loading model')
    ret = rknn.load_onnx(model=model_path)
    #ret = rknn.load_pytorch(model=model_path, input_size_list=[[1, 3, 640, 640]])
    if ret != 0:
        print('Load model failed!')
        exit(ret)
    print('done')

    # Build model
    print('--> Building model')
    ret = rknn.build(do_quantization=do_quant, dataset=DATASET_PATH)
    if ret != 0:
        print('Build model failed!')
        exit(ret)
    print('done')

    # Export rknn model
    print('--> Export rknn model')
    ret = rknn.export_rknn(output_path)
    if ret != 0:
        print('Export rknn model failed!')
        exit(ret)
    print('done')

    # 精度分析、出力ディレクトリ./snapshot
    #print('--> Accuracy analysis')
    #ret = rknn.accuracy_analysis(inputs=['./subset/000000052891.jpg'])
    #if ret != 0:
    #    print('Accuracy analysis failed!')
    #    exit(ret)
    #print('done')

    # Release
    rknn.release()
```

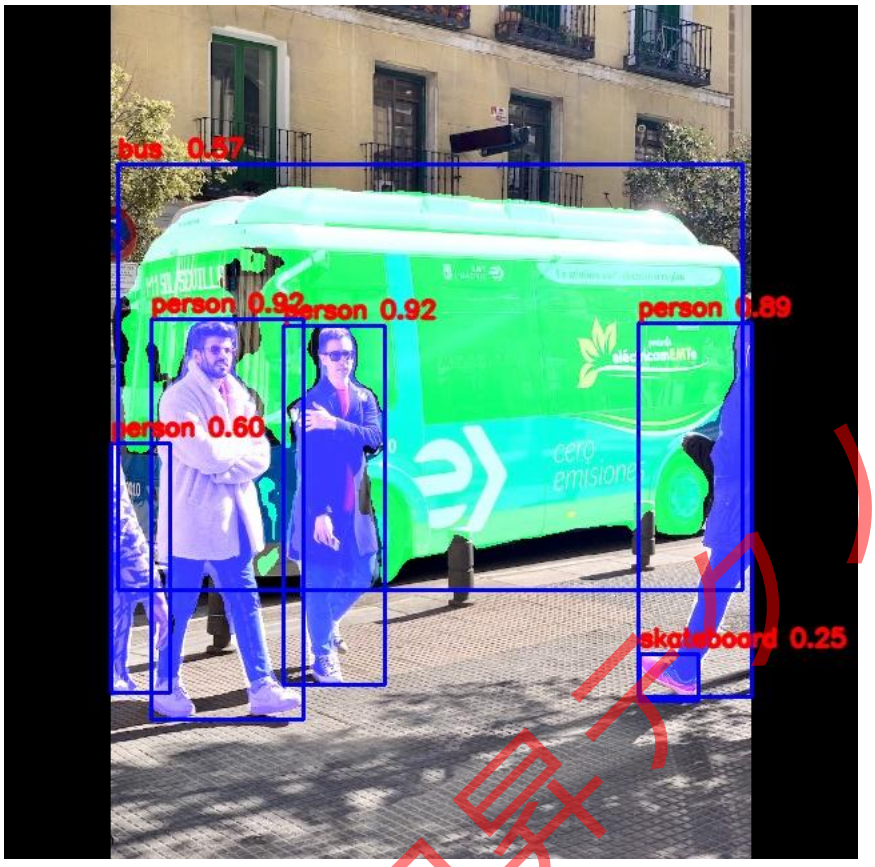
usb またはイーサネットケーブルを介してボードに接続し、adb 接続が成功していることを確認します。ボード上で rknn_server コマンドを実行し、rknn_server を起動します。PC 端末で test.py を実

行します。

```
  (.toolkit2_env) (base) csun@CSUN-PC-
0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/yolov8/yolov8_seg$ python test.py
  I rknn-toolkit2 version: 2.0.0b0+9bab5682
  ************************
  all device(s) with adb mode:
  192.168.11.241:5555
  ************************
  adb: unable to connect for root: closed
  I target set by user is: rk3588
  I Get hardware info: target_platform = rk3588, os = Linux, aarch = aarch64
  I Check RK3588 board npu runtime version
  I Starting ntp or adb, target is RK3588
  I Start adb...
  I Connect to Device success!
  I NPUTransfer: Starting NPU Transfer Client, Transfer version 2.1.0 (b5861e7@2020-11-23T11:50:36)
  D RKNNAPI: ==========================================
  D RKNNAPI: RKNN VERSION:
  D RKNNAPI:   API: 2.0.0b0 (18eacd0 build@2024-03-22T06:07:59)
  D RKNNAPI:   DRV: rknn_server: 2.0.0b0 (18eacd0 build@2024-03-22T14:07:19)
  D RKNNAPI:   DRV: rknnrt: 2.0.0b0 (35a6907d79@2024-03-24T10:31:14)
  D RKNNAPI: ==========================================
  D RKNNAPI: Input tensors:
  D RKNNAPI:   index=0, name=images, n_dims=4, dims=[1, 640, 640, 3], n_elems=1228800, size=1228800, w_stride =
0, size_with_stride = 0, fmt=NHWC, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003922
  D RKNNAPI: Output tensors:
  D RKNNAPI:   index=0, name=375, n_dims=4, dims=[1, 64, 80, 80], n_elems=409600, size=409600, w_stride = 0,
size_with_stride = 0, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-52, scale=0.134740
  D RKNNAPI:   index=1, name=onnx::ReduceSum_383, n_dims=4, dims=[1, 80, 80, 80], n_elems=512000, size=512000,
w_stride = 0, size_with_stride = 0, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003224
  D RKNNAPI:   index=2, name=388, n_dims=4, dims=[1, 1, 80, 80], n_elems=6400, size=6400, w_stride = 0,
size_with_stride = 0, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003922
  D RKNNAPI:   index=3, name=354, n_dims=4, dims=[1, 32, 80, 80], n_elems=204800, size=204800, w_stride = 0,
size_with_stride = 0, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=23, scale=0.026157
  D RKNNAPI:   index=4, name=395, n_dims=4, dims=[1, 64, 40, 40], n_elems=102400, size=102400, w_stride = 0,
size_with_stride = 0, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-41, scale=0.105550
  D RKNNAPI:   index=5, name=onnx::ReduceSum_403, n_dims=4, dims=[1, 80, 40, 40], n_elems=128000, size=128000,
w_stride = 0, size_with_stride = 0, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003693
  D RKNNAPI:   index=6, name=407, n_dims=4, dims=[1, 1, 40, 40], n_elems=1600, size=1600, w_stride = 0,
size_with_stride = 0, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003922
  D RKNNAPI:   index=7, name=361, n_dims=4, dims=[1, 32, 40, 40], n_elems=51200, size=51200, w_stride = 0,
size_with_stride = 0, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=28, scale=0.024372
  D RKNNAPI:   index=8, name=414, n_dims=4, dims=[1, 64, 20, 20], n_elems=25600, size=25600, w_stride = 0,
size_with_stride = 0, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-47, scale=0.085976
  D RKNNAPI:   index=9, name=onnx::ReduceSum_422, n_dims=4, dims=[1, 80, 20, 20], n_elems=32000, size=32000,
w_stride = 0, size_with_stride = 0, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003819
  D RKNNAPI:   index=10, name=426, n_dims=4, dims=[1, 1, 20, 20], n_elems=400, size=400, w_stride = 0,
size_with_stride = 0, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003922
  D RKNNAPI:   index=11, name=368, n_dims=4, dims=[1, 32, 20, 20], n_elems=12800, size=12800, w_stride = 0,
size_with_stride = 0, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=46, scale=0.024216
  D RKNNAPI:   index=12, name=347, n_dims=4, dims=[1, 32, 160, 160], n_elems=819200, size=819200, w_stride = 0,
size_with_stride = 0, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-119, scale=0.032402
  --> モデルを実行中
  W inference: The 'data_format' is not set, and its default value is 'nhwc'!
  完了
  person @ (209 240 285 509) 0.920
  person @ (110 235 224 535) 0.916
  person @ (475 238 560 518) 0.886
  person @ (80 328 124 515) 0.602
```

```
bus  @ (85 119 553 438) 0.573
skateboard @ (476 486 520 521) 0.251
```

結果は result.jpg に保存されます。



19.3.5. ボードへのデプロイ

1.C++デプロイとテスト

rknn_model_zoo リポジトリが提供するデプロイメントサンプルを参考にして、例を作成し、yolov8n-seg.rknn をボードにデプロイします。

```
# サンプルソースから lubancat_ai_manual_code/example/yolov8/yolov8_seg をボードにコピー
  cat@lubancat:~/lubancat_ai_manual_code/example/yolov8/yolov8_seg/cpp$ ./build-linux.sh -t rk3588
  ./build-linux.sh -t rk3588
  ==================================
  TARGET_SOC=rk3588
  INSTALL_DIR=/home/cat/lubancat_ai_manual_code/example/yolov8/yolov8_seg/cpp/install/rk3588_linux
  BUILD_DIR=/home/cat/lubancat_ai_manual_code/example/yolov8/yolov8_seg/cpp/build/build_rk3588_linux
  ENABLE_DMA32=OFF
  CC=aarch64-linux-gnu-gcc
  CXX=aarch64-linux-gnu-g++
  ==================================
  -- The C compiler identification is GNU 10.2.1
  -- The CXX compiler identification is GNU 10.2.1
  -- Detecting C compiler ABI info
  -- Detecting C compiler ABI info - done
  -- Check for working C compiler: /usr/bin/aarch64-linux-gnu-gcc - skipped
  -- Detecting C compile features
```

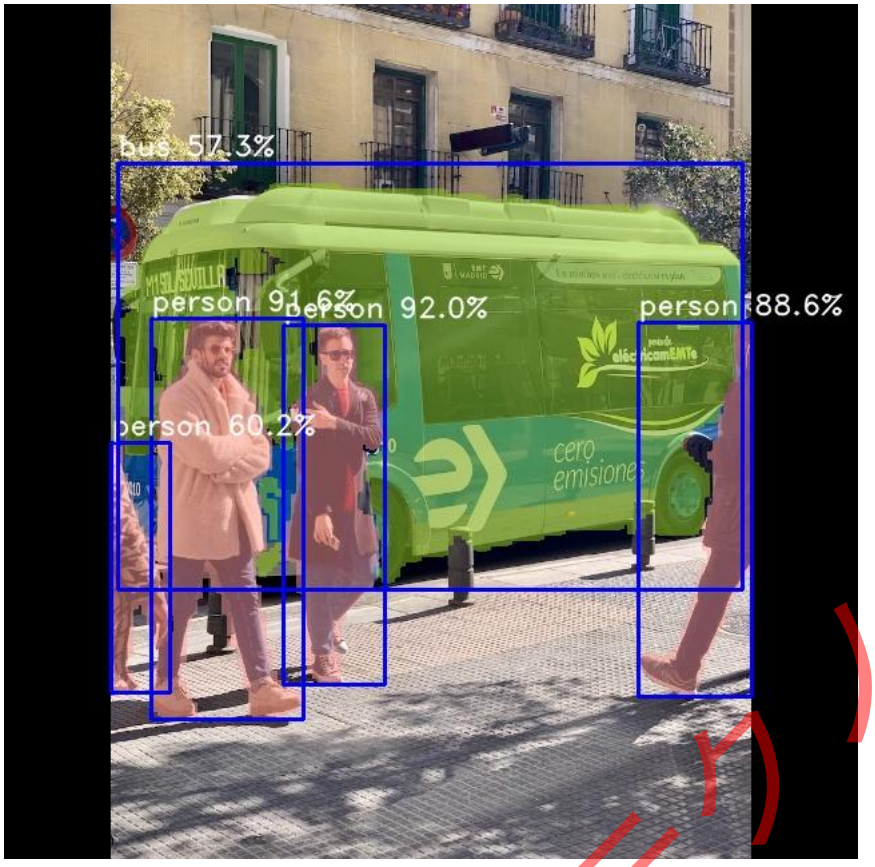
```
  -- Detecting C compile features - done
  -- Detecting CXX compiler ABI info
  -- Detecting CXX compiler ABI info - done
  -- Check for working CXX compiler: /usr/bin/aarch64-linux-gnu-g++ - skipped
  -- Detecting CXX compile features
  -- Detecting CXX compile features - done
  -- Found OpenCV: /usr (found version "4.5.1")
  -- Configuring done
  -- Generating done
  -- Build files have been written to:
/home/cat/lubancat_ai_manual_code/example/yolov8/yolov8_seg/cpp/build/build_rk3588_linux
  Scanning dependencies of target rknn_yolov8_seg_demo
  Scanning dependencies of target yolov8seg_videocapture_demo
# ...省略...
[100%] Built target yolov8seg_videocapture_demo
[ 50%] Built target yolov8seg_videocapture_demo
[100%] Built target rknn_yolov8_seg_demo
Install the project...
-- Install configuration: ""
-- Installing:
/home/cat/lubancat_ai_manual_code/example/yolov8/yolov8_seg/cpp/install/rk3588_linux/lib/librga.so
-- Installing:
/home/cat/lubancat_ai_manual_code/example/yolov8/yolov8_seg/cpp/install/rk3588_linux/lib/librknnrt.so
-- Installing:
/home/cat/lubancat_ai_manual_code/example/yolov8/yolov8_seg/cpp/install/rk3588_linux/./rknn_yolov8_seg_demo
-- Set runtime path of
"/home/cat/lubancat_ai_manual_code/example/yolov8/yolov8_seg/cpp/install/rk3588_linux/./rknn_yolov8_seg_demo"
to "$ORIGIN/lib"
-- Installing:
/home/cat/lubancat_ai_manual_code/example/yolov8/yolov8_seg/cpp/install/rk3588_linux/./yolov8seg_videocapture_d
emo
-- Set runtime path of
"/home/cat/lubancat_ai_manual_code/example/yolov8/yolov8_seg/cpp/install/rk3588_linux/./yolov8seg_videocapture_
demo" to "$ORIGIN/lib"
-- Installing:
/home/cat/lubancat_ai_manual_code/example/yolov8/yolov8_seg/cpp/install/rk3588_linux/model/bus.jpg
-- Installing:
/home/cat/lubancat_ai_manual_code/example/yolov8/yolov8_seg/cpp/install/rk3588_linux/model/coco_80_labels_list.
txt
-- Installing:
/home/cat/lubancat_ai_manual_code/example/yolov8/yolov8_seg/cpp/install/rk3588_linux/model/yolov8_seg_rk3588.rk
nn
```

現在のディレクトリ install/rk3588_linux に切り替え、以下のコマンドを実行します：

```
# ./rknn_yolov8_seg_demo <model_path> <image_path>
cat@lubancat:~/lubancat_ai_manual_code/example/yolov8/yolov8_seg/cpp/install/rk3588_linux$ ./rknn_yolov8_seg_de
mo ./model/yolov8_seg_rk3588.rknn ./model/bus.jpg
load lable ./model/coco_80_labels_list.txt
model input num: 1, output num: 13
input tensors:
  index=0, name=images, n_dims=4, dims=[1, 640, 640, 3], n_elems=1228800, size=1228800, fmt=NHWC, type=INT8,
qnt_type=AFFINE, zp=-128, scale=0.003922
output tensors:
  index=0, name=375, n_dims=4, dims=[1, 64, 80, 80], n_elems=409600, size=409600, fmt=NCHW, type=INT8,
qnt_type=AFFINE, zp=-52, scale=0.134740
  index=1, name=onnx::ReduceSum_383, n_dims=4, dims=[1, 80, 80, 80], n_elems=512000, size=512000, fmt=NCHW,
type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003224
  index=2, name=388, n_dims=4, dims=[1, 1, 80, 80], n_elems=6400, size=6400, fmt=NCHW, type=INT8,
```

```
qnt_type=AFFINE, zp=-128, scale=0.003922
  index=3, name=354, n_dims=4, dims=[1, 32, 80, 80], n_elems=204800, size=204800, fmt=NCHW, type=INT8,
qnt_type=AFFINE, zp=23, scale=0.026157
  index=4, name=395, n_dims=4, dims=[1, 64, 40, 40], n_elems=102400, size=102400, fmt=NCHW, type=INT8,
qnt_type=AFFINE, zp=-41, scale=0.105550
  index=5, name=onnx::ReduceSum_403, n_dims=4, dims=[1, 80, 40, 40], n_elems=128000, size=128000, fmt=NCHW,
type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003693
  index=6, name=407, n_dims=4, dims=[1, 1, 40, 40], n_elems=1600, size=1600, fmt=NCHW, type=INT8,
qnt_type=AFFINE, zp=-128, scale=0.003922
  index=7, name=361, n_dims=4, dims=[1, 32, 40, 40], n_elems=51200, size=51200, fmt=NCHW, type=INT8,
qnt_type=AFFINE, zp=28, scale=0.024372
  index=8, name=414, n_dims=4, dims=[1, 64, 20, 20], n_elems=25600, size=25600, fmt=NCHW, type=INT8,
qnt_type=AFFINE, zp=-47, scale=0.085976
  index=9, name=onnx::ReduceSum_422, n_dims=4, dims=[1, 80, 20, 20], n_elems=32000, size=32000, fmt=NCHW,
type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003819
  index=10, name=426, n_dims=4, dims=[1, 1, 20, 20], n_elems=400, size=400, fmt=NCHW, type=INT8,
qnt_type=AFFINE, zp=-128, scale=0.003922
  index=11, name=368, n_dims=4, dims=[1, 32, 20, 20], n_elems=12800, size=12800, fmt=NCHW, type=INT8,
qnt_type=AFFINE, zp=46, scale=0.024216
  index=12, name=347, n_dims=4, dims=[1, 32, 160, 160], n_elems=819200, size=819200, fmt=NCHW, type=INT8,
qnt_type=AFFINE, zp=-119, scale=0.032402
model is NHWC input fmt
model input height=640, width=640, channel=3
scale=1.000000 dst_box=(0 0 639 639) allow_slight_change=1 _left_offset=0 _top_offset=0 padding_w=0 padding_h=0
src width=640 height=640 fmt=0x1 virAddr=0x0x7fa464a040 fd=0
dst width=640 height=640 fmt=0x1 virAddr=0x0x55a0223190 fd=0
src_box=(0 0 639 639)
dst_box=(0 0 639 639)
color=0x72
rga_api version 1.10.0_[2]
rknn_run
matmul_by_cpu_uint8 use: 42.492001 ms
resize_by_opencv_uint8 use: 6.234000 ms
crop_mask_uint8 use: 6.762000 ms
seg_reverse use: 0.297000 ms
person @ (209 240 285 509) 0.920
person @ (110 235 224 535) 0.916
person @ (475 238 560 518) 0.886
person @ (80 328 124 515) 0.602
bus @ (85 119 553 438) 0.573
rknn run and process use use: 115.386002 ms
```

テストされた yolov8s-seg モデルの結果は out.jpg に保存されます。結果は以下の通りです。



上記のように、YOLOv8 の物体検出と分割テストを行いました。より多くのモデルテストについては、rknn_model_zoo リポジトリを参照してください。

2. Python デプロイとテスト

サンプルソース：yolov8/yolov8_seg/test.py

```
if __name__ == '__main__':

    if is_architecture('x86_64'):
        print("Current platform is x86_64")
        from rknn.api import RKNN
        # RKNN オブジェクト作成
        rknn = RKNN()
        rknn.list_devices()

        # rknn モデルをロード
        rknn.load_rknn(path=rknn_model_path)

        # 動作環境を設定、デフォルトは rk3588
        ret = rknn.init_runtime(target=target, device_id=device_id)
        if ret != 0:
            print('Init runtime environment failed!')
            exit(ret)

    elif is_architecture('aarch64'):
        print("Current platform is aarch64")
        from rknnlite.api import RKNNLite
        # Create RKNN object
        rknn_lite = RKNNLite()
        # load RKNN model
        print('--> Load RKNN model')
        ret = rknn_lite.load_rknn(rknn_model_path)
```

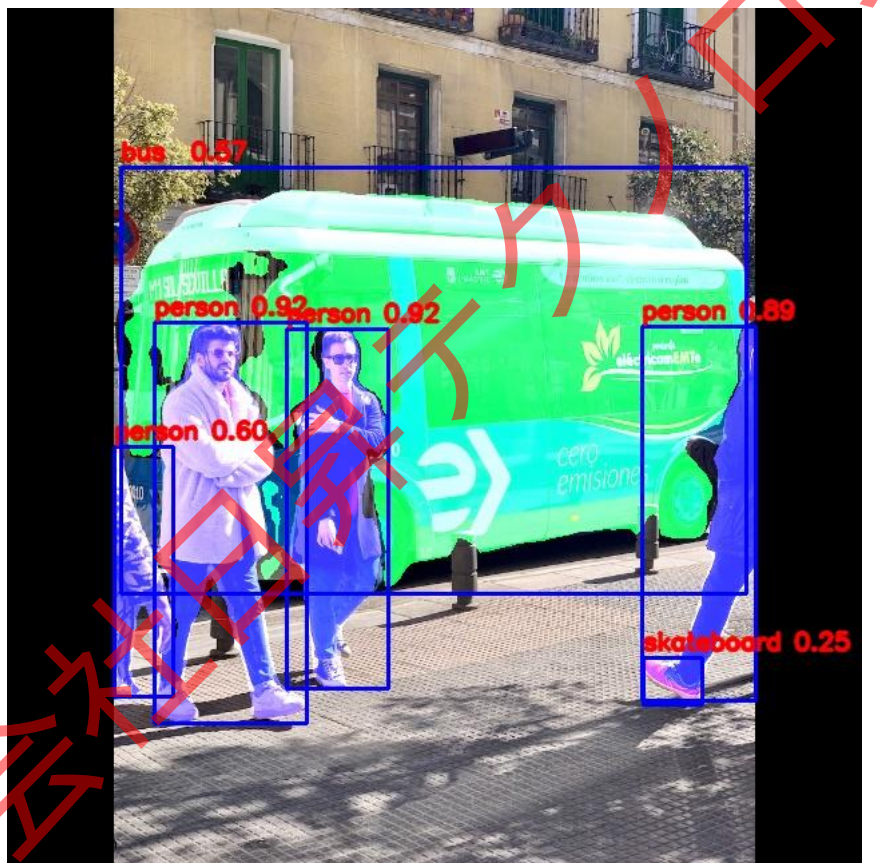
```
        if ret != 0:
            print('Load RKNN model failed!')
            exit(ret)
        print('done')

        # Init runtime environment
        print('--> Init runtime environment')
        ret = rknn_lite.init_runtime(core_mask=RKNNLite.NPU_CORE_0)
        if ret != 0:
            print('Init runtime environment failed!')
            exit(ret)
        print('done')
    else:
        print(f"Current platform is {platform.machine()}")
        exit(0)


    # 画像を入力する
    img_src = cv2.imread(img_path)
    if img_src is None:
        print("画像の読み込みに失敗しました！")
        exit()
    src_shape = img_src.shape[:2]
    img, ratio, (dw, dh) = letter_box(img_src, IMG_SIZE)
    img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

    # 推論を実行
    print('--> Running model')
    if is_architecture('x86_64'):
        outputs = rknn.inference(inputs=[img])
    elif is_architecture('aarch64'):
        img = np.expand_dims(img, 0)
        outputs = rknn_lite.inference(inputs=[img])
    print('done')

    # 後処理
    boxes, classes, scores, seg_img = post_process(outputs)

    if boxes is not None:
        real_boxs = get_real_box(src_shape, boxes, dw, dh, ratio)
        real_segs = get_real_seg(src_shape, IMG_SIZE, dw, dh, seg_img)
        img_p = merge_seg(img_src, real_segs, classes)
        draw(img_p, real_boxs, scores, classes)
        cv2.imwrite("result.jpg", img_p)

    if is_architecture('x86_64'):
        rknn.release()
    elif is_architecture('aarch64'):
        rknn_lite.release()
```

ボード側で推論を実行：

```
cat@lubancat:~/lubancat_ai_manual_code/example/yolov8/yolov8_seg$ python test.py
Current platform is aarch64
--> Load RKNN model
done
--> Init runtime environment
I RKNN: [17:37:06.140] RKNN Runtime Information, librknnrt version: 2.0.0b0 (35a6907d79@2024-03-24T10:31:14)
I RKNN: [17:37:06.140] RKNN Driver Information, version: 0.9.2
```

```
  I RKNN: [17:37:06.140] RKNN Model Information, version: 6, toolkit version: 2.0.0b0+9bab5682(compiler
version: 2.0.0b0 (35a6907d79@2024-03-24T02:34:11)), target: RKNPU v2, target platform: rk3588, framework name:
ONNX, framework layout: NCHW, model inference type: static_shape
  done
  --> Running model
  done
  person @ (209 240 285 509) 0.920
  person @ (110 235 224 535) 0.916
  person @ (475 238 560 518) 0.886
  person @ (80 328 124 515) 0.602
  bus  @ (85 119 553 438) 0.573
  skateboard @ (476 486 520 521) 0.251
```

結果は現在のディレクトリ内の result.jpg に保存されます。結果の表示：



19.4. 参考リンク

- https://docs.ultralytics.com/quickstart/
- https://github.com/ultralytics/ultralytics
- https://github.com/airockchip/ultralytics_yolov8
- https://github.com/airockchip/rknn_model_zoo

20.総合サンプル１：ゴミ検出と識別

この章では、TACO データセットに基づいて YOLOv8 セグメンテーションモデルを使用してゴミの検出と識別を行い、LubanCat ボードにデプロイして実行します。

20.1 Yolov8-seg

YOLOv8 は、Ultralytics がオープンソースで提供する最新のアルゴリズムであり、現在、物体検出、インスタンスセグメンテーション、ポーズ推定などのタスクをサポートしています。これらのモデルをボードにデプロイする方法については、前の章を参照してください。

YOLOv8 関連環境のインストール：

```
# conda を使用して仮想環境を作成
conda create -n yolov8 python=3.8
conda activate yolov8

# リポジトリをクローンしてインストール、検証用のバージョン v8.1.9
cd garbage_detection
# git clone https://github.com/ultralytics/ultralytics.git
git clone --branch v8.1.9 https://github.com/ultralytics/ultralytics.git
cd ultralytics

pip install -e .
```

20.2. TACO データセット

TACO は、室内、森林、道路、ビーチなどの異なる環境で撮影されたゴミの画像データセットであり、これらの画像のゴミオブジェクトはボックスとポリゴンで詳細にアノテーションされています。アノテーション情報は COCO データセットと同じフォーマットを使用し、全体で 60 のカテゴリがありますが、一部のカテゴリはアノテーションが少ない、または全くない場合があります（トレーニングデータセットではカテゴリを減らすことをお勧めします）。

データセットのダウンロード：

```
git clone https://github.com/pedropro/TACO.git
cd TACO

# スクリプトを実行して画像データをダウンロード、デフォルトでは公式データセット
（annotations.json）は 1500 枚ダウンロードされますが、非公式データ
（annotations_unofficial.json）を指定してダウンロードすることもできます
python3 download.py
```

ダウンロードに失敗した場合は再度コマンドを実行し、再度失敗する場合は、付属のリポジトリからデータセットをダウンロードして、TACO/data ディレクトリに解凍してください。

TACO データセットには多くのプログラムが含まれており、データセットに関する情報を確認できます：

example/garbage_detection/ultralytics/TACO/dataset_info.py（サンプルソースの中にある）

```
# demo.ipynb を参考し、公式データセットの全体情報を確認
import json
import numpy as np

dataset_path = './data'
anns_file_path = dataset_path + '/' + 'annotations.json'

# Read annotations
with open(anns_file_path, 'r') as f:
    dataset = json.loads(f.read())

categories = dataset['categories']
anns = dataset['annotations']
imgs = dataset['images']
nr_cats = len(categories)
nr_annotations = len(anns)
nr_images = len(imgs)

# Load categories and super categories
cat_names = []
super_cat_names = []
super_cat_ids = {}
super_cat_last_name = ''
nr_super_cats = 0
for cat_it in categories:
    cat_names.append(cat_it['name'])
    super_cat_name = cat_it['supercategory']
    # Adding new supercat
    if super_cat_name != super_cat_last_name:
        super_cat_names.append(super_cat_name)
        super_cat_ids[super_cat_name] = nr_super_cats
        super_cat_last_name = super_cat_name
        nr_super_cats += 1

print('Number of super categories:', nr_super_cats)
print('Number of categories:', nr_cats)
print('Number of annotations:', nr_annotations)
print('Number of images:', nr_images)
```

```
(.toolkit2_env) (base) csun@CSUN-PC-
0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/garbage_detection/ultralytics
/TACO$ python dataset_info.py
Number of super categories: 28
Number of categories: 60
Number of annotations: 4784
Number of images: 1500
```

公式データセットは合計 1500 枚の画像、60 のカテゴリ、4784 のアノテーションがあります。

20.2.1. データセットの処理

TACO データセットのアノテーション情報は COCO フォーマットと同じです。我々は COCO データセットの API を使用して TACO データセットを処理し、データセットを分割します。Yolov8-seg をトレーニングするために、yolo フォーマットに変換する必要があります。

TACO リポジトリが提供する detector/split_dataset.py を利用して、TACO データセットのアノテーション情報 annotations.json を train、val、test の 3 つに分割し、対応する json ファイルを生成します。

```
# TACO プロジェクトディレクトリで detector/split_dataset.py を実行してデータセットを分割、--dataset_dir は json パスを指定し、--test_percentage はテストセットの割合を設定、--val_percentage は検証セットの割合を設定、--nr_trials は分割数を設定
python  detector/split_dataset.py --dataset_dir ./data  --test_percentage 5 --val_percentage 10 --nr_trials 1
```

これにより、data ディレクトリに annotations_0_train.json、annotations_0_val.json、annotations_0_test.json の 3 つのファイルが生成され、テストセットは 5%、評価データは 10%、残りはトレーニングデータになります。

yolo データセットは、画像ディレクトリとラベルディレクトリで構成されており、画像のすべてのアノテーション情報は、画像と同じ名前の txt ファイルに保存されます。YOLOv8 セグメンテーションモデルのトレーニングのアノテーションフォーマットは次のようになります：

```
<id> <x_1> <y_1> ... <x_n> <y_n>
```

1 つのオブジェクトのアノテーション情報は 1 行に保存され、行の最初にはカテゴリ ID があり、その後に多角形の各点のピクセル座標の x と y の値が順番に並びます。x と y の値は画像の幅と高さで正規化される必要があります。具体的な形式は、yolov8-seg テストデータセット COCO8-Seg データセットを参照してください。

付属のリポジトリのプログラムを使用して、前述の分割データセットの json ファイルを読み込み、yolo データセットの要件に従ってフォーマットに変換します：

example/garbage_detection/ultralytics/TACO/taco2yolo.py

```
    # ...省略...
    # クラス ID
    catIds = data_source.getCatIds()
    # クラス名を取得
    categories = data_source.loadCats(catIds)
    categories.sort(key=lambda x: x['id'])

    # クラスを保存
    class_path = args.save_path + '/classes.txt'
    with open(class_path, "w") as file:
        for item in categories:
            file.write(f"{item['id']}: {item['name']}¥n")

    # 各画像をループ
    img_ids = data_source.getImgIds()
    for index, img_id in tqdm.tqdm(enumerate(img_ids)):
```

```
        img_info = data_source.loadImgs(img_id)[0]
        file_name = img_info['file_name'].replace('/', '_')
        save_name = file_name.split('.')[0]

        height = img_info['height']
        width = img_info['width']

        save_label_path = yolo_label_path + '/' + save_name + '.txt'
        with open(save_label_path, mode='w') as fp:
            annotation_id = data_source.getAnnIds(img_id)
            if len(annotation_id) == 0:
                fp.write('')
                shutil.copy('data/{}'.format(img_info['file_name']), os.path.join(yolo_image_path,
file_name))
                continue

            annotations = data_source.loadAnns(annotation_id)

            for annotation in annotations:
                category_id = annotation["category_id"]
                seg_labels = []
                for segmentation in annotation["segmentation"]:
                    points = np.array(segmentation).reshape((int(len(segmentation) / 2), 2))
                    for point in points:
                        x = point[0] / width
                        y = point[1] / height
                        seg_labels.append(x)
                        seg_labels.append(y)
                fp.write(str(category_id) + " " + " ".join([str(a) for a in seg_labels]) + "¥n")

        shutil.copy('data/{}'.format(img_info['file_name']), os.path.join(yolo_image_path, file_name))
```

TACO プロジェクトディレクトリでプログラムを実行します：

```
# テストデータセット、プログラムを実行するとラベルファイルが生成され、画像が対応するディレクトリにコピーされ、YOLO フォーマットに変換されます
python taco2yolo.py --annotation_path ./data/annotations_0_test.json --subset test

# 評価データセット
python taco2yolo.py --annotation_path ./data/annotations_0_val.json --subset val

# トレーニングデータセット
python taco2yolo.py --annotation_path ./data/annotations_0_train.json --subset train
```

プログラムは TACO データの画像を指定されたディレクトリにコピーし、対応するディレクトリにラベルファイルおよびカテゴリ名ファイルを生成します。処理後のデータセットのディレクトリ構造は次のようになります：

フォルダー：TACO/taco

```
├─── images
│    └─── val
│    │    ├─── 0000000001.jpg
│    │    ├─── 0000000002.jpg
│    │   .......
│    └─── test
│    │    ├─── 0000000001.jpg
│    │    ├─── 0000000002.jpg|    |    .......
│    └─── train
│         ├─── 0000000001.jpg
│         ├─── 0000000002.jpg
│        .......
├─── labels
│    └─── val
│    │    ├─── 0000000001.txt
│    │    ├─── 0000000002.txt
│    │   .......
│    └─── test
│    │    ├─── 0000000001.txt
│    │    ├─── 0000000002.txt
│    │   .......
│    └─── train|   ..............
```

20.3. モデルトレーニング

Yolov8 をトレーニングするには、データセット構成ファイルとモデル構成ファイルを作成する必要があります。

ultralytics/cfg/datasets/coco8-seg.yaml をコピーしてデータセット構成ファイルを作成し、taco-seg.yaml という名前に変更します。内容は次の通りです：

example/garbage_detection/ultralytics/TACO/taco-seg.yaml

```
path:
/home/csun/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/garbage_detection/ultralytics/TACO/taco
    train: train.txt  # 訓練セットパス、path ディレクトリに対して相対パス
    val: val.txt  # 検証セットパス、path ディレクトリに対して相対パス
    test: test.txt # テストセットパス、path ディレクトリに対して相対パス、記入しなくてもよい

    names:
    0: Aluminium foil
    1: Battery
    2: Aluminium blister pack
    3: Carded blister pack
    4: Other plastic bottle
```

```
    5: Clear plastic bottle
    6: Glass bottle
    7: Plastic bottle cap
    8: Metal bottle cap
    9: Broken glass
    10: Food Can
    # 省略......
```

実際のデータセット保存ディレクトリに応じて修正し、カテゴリ ID とカテゴリ名を前述のデータセット処理で生成された example/garbage_detection/ultralytics/TACO/taco/classes.txt ファイルからコピーします。

上記訓練設定ファイル「taco-seg.yaml」の中、train.txt,val.txt,test.txt は gen-trainlist.py により自動生成できます。(data_dir というパスは実行環境により修正必要)

example/garbage_detection/ultralytics/TACO/gen-trainlist.py

```
import os

def create_image_list(data_dir, subsets):
    base_path = os.path.join(data_dir, 'images')
    for subset in subsets:
        subset_path = os.path.join(base_path, subset)
        image_files = [os.path.join('./images', subset, img) for img in os.listdir(subset_path) if
img.endswith('.jpg')]
        with open(os.path.join(data_dir, f'{subset}.txt'), 'w') as f:
            f.write('¥n'.join(image_files) + '¥n')

if __name__ == "__main__":
    data_dir =
'/home/csun/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/garbage_detection/ultralytics/TACO/taco'
    subsets = ['train', 'val', 'test']
    create_image_list(data_dir, subsets)
```

訓練用の写真設定リストファイルを生成

```
(.toolkit2_env) (base) csun@CSUN-PC-
0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/garbage_detection/ultralytics/TACO$ python
gen-trainlist.py
```

example/garbage_detection/ultralytics/TACO/taco に train.txt,val.txt,test.txt を生成されます。

ultralytics/cfg/models/v8/yolov8-seg.yaml をコピーしてモデル構成ファイルを作成し、yolov8n-seg-taco.yaml という名前に変更します（デフォルトで n サイズのモデルを使用します）。名前を yolov8s-seg-taco.yaml に変更すると、s サイズのモデルが使用されます。

```
(.toolkit2_env) (base) csun@CSUN-PC-
0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/garbage_detection/ultralytics
/TACO$ cp ../ultralytics/cfg/models/v8/yolo
    v8-seg.yaml ./yolov8n-seg-taco.yaml
```

example/garbage_detection/taco/yolov8n-seg-taco.yaml

```
  # Ultralytics YOLO   , AGPL-3.0 license
  # YOLOv8-seg instance segmentation model. For Usage examples see https://docs.ultralytics.com/tasks/segment

  # Parameters
  nc: 60 # number of classes
  scales: # model compound scaling constants, i.e. 'model=yolov8n-seg.yaml' will call yolov8-seg.yaml with
scale 'n'
    # [depth, width, max_channels]
    n: [0.33, 0.25, 1024]
    s: [0.33, 0.50, 1024]
    m: [0.67, 0.75, 768]
    l: [1.00, 1.00, 512]
    x: [1.00, 1.25, 512]

  # YOLOv8.0n backbone
  backbone:
    # [from, repeats, module, args]
  #...省略...
```

モデルトレーニングはコマンドまたはPythonプログラムを使用して行いますが、チュートリアルではyoloコマンドを使用してテストします。前述の構成ファイルを同じディレクトリに配置し、yolov8環境で次のコマンドを実行します：

```
# 最初のパラメータはタスクを示し[detect, segment, classify]、ここでは物体検出をテストするため
detectを使用します（オプション）；
# 二番目のパラメータはモードを示し[train, val, predict, export, track]、トレーニング、評価、推
論などを選択します；
# その他のパラメータdataはデータセット構成ファイルを指定し、modelは修正したモデル構成ファイル
パスを指定します
cd taco
pwd
/home/csun/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/garbage_detection/ultraly
tics/TACO/taco
cp ../taco-seg.yaml ./
cp ../yolov8n-seg-taco.yaml ./
yolo segment train data=taco-seg.yaml model=yolov8n-seg-taco.yaml epochs=100 batch=16 imgsz=640
```

訓練時、以下のエラーが出るかもしれません。

```
(.toolkit2_env) (base) csun@CSUN-PC-
0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/garbage_detection/ultralytics
/TACO/taco$ yolo segment train data=taco-seg.yaml model=yolov8n-seg-taco.yaml epochs=100
batch=16 imgsz=640
project=/home/csun/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/garbage_detection
/ultralytics/runs
Ultralytics YOLOv8.1.9    Python-3.8.10 torch-2.1.0+cu121 CUDA:0 (NVIDIA GeForce GTX 1070 Ti,
8106MiB)
# ...省略...
```

```
Image sizes 640 train, 640 val
Using 8 dataloader workers
Logging results to
/home/csun/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/garbage_detection/ultraly
tics/runs/train3
Starting training for 100 epochs...

      Epoch    GPU_mem   box_loss   seg_loss   cls_loss   dfl_loss  Instances       Size
      1/100      3.19G          0          0      130.4          0          0        640:
100%|██████████| 45/45 [00:21<00:00,  2.09it/s]
                 Class     Images  Instances      Box(P          R      mAP50  mAP50-95)
Mask(P          R      mAP50  mAP50-95): 100%|██████████| 3/3 [00:00<00:00,  5.04
Traceback (most recent call last):
  File "/home/csun/project-Toolkit2/.toolkit2_env/bin/yolo", line 8, in <module>
    sys.exit(entrypoint())
  File
"/home/csun/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/garbage_detection/ultral
ytics/ultralytics/cfg/__init__.py", line 697, in entrypoint
    getattr(model, mode)(**overrides)  # default args from model
  File
"/home/csun/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/garbage_detection/ultral
ytics/ultralytics/engine/model.py", line 650, in train
    self.trainer.train()
  File
"/home/csun/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/garbage_detection/ultral
ytics/ultralytics/engine/trainer.py", line 204, in train
    self._do_train(world_size)
  File
"/home/csun/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/garbage_detection/ultral
ytics/ultralytics/engine/trainer.py", line 429, in _do_train
    self.metrics, self.fitness = self.validate()
  File
"/home/csun/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/garbage_detection/ultral
ytics/ultralytics/engine/trainer.py", line 570, in validate
    metrics = self.validator(self)
  File "/home/csun/project-Toolkit2/.toolkit2_env/lib/python3.8/site-
packages/torch/utils/_contextlib.py", line 115, in decorate_context
    return func(*args, **kwargs)
  File
"/home/csun/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/garbage_detection/ultral
ytics/ultralytics/engine/validator.py", line 195, in __call__
    stats = self.get_stats()
  File
"/home/csun/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/garbage_detection/ultral
ytics/ultralytics/models/yolo/detect/val.py", line 172, in get_stats
    stats = {k: torch.cat(v, 0).cpu().numpy() for k, v in self.stats.items()}  # to numpy
  File
"/home/csun/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/garbage_detection/ultral
ytics/ultralytics/models/yolo/detect/val.py", line 172, in <dictcomp>
    stats = {k: torch.cat(v, 0).cpu().numpy() for k, v in self.stats.items()}  # to numpy
RuntimeError: torch.cat(): expected a non-empty list of Tensors
```

解決方法：

1）ラベルファイルを見つからないという原因です。

train.txt,val.txt,test.txt に生成されたパスは YoloV8 の関数と合わないですので、gen-trainlist.py を修正します。例：images/train/batch_9_000077.jpg という写真パスからラベルパスを生成する際に、下記関数を「labels/train/batch_9_000077.txt」正しく変換できません。

[./images/train/batch_9_000077.jpg]に変更すれば、[./labels/train/batch_9_000077.txt]に正常に変更できました。（サンプルソースの中、すでに gen-trainlist.py を修正済み）

```
def img2label_paths(img_paths):
    """Define label paths as a function of image paths."""
    sa, sb = f"{os.sep}images{os.sep}", f"{os.sep}labels{os.sep}"  # /images/, /labels/ substrings
    return [sb.join(x.rsplit(sa, 1)).rsplit(".", 1)[0] + ".txt" for x in img_paths]
```

2）YoloV8 が新しいバージョンを使う場合、TACO データセットをうまく処理できないですので、8.1.9 に限定にしましょう。

これで、yolov8n-seg モデルを簡単にトレーニングできました。トレーニングの出力ファイルは runs/segment/train ディレクトリに保存され、最終モデルは runs/segment/train/weights/best.pt に保存されます。

トレーニングしたモデルを評価します：

```
(.toolkit2_env) (base) csun@CSUN-PC-0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/garbage_detection/ultralytics/TACO/taco$ yolo segment val model=xxx/runs/segment/train/weights/last.pt
```

出力ファイルは runs/segment/val ディレクトリに保存されます。トレーニングしたモデルの精度が低い場合、トレーニング戦略の変更、データセットの追加と最適化、他のサイズのモデルの使用などをテストしてみてください。

20.4. モデルのエクスポート

トレーニングした yolov8n-seg モデルを LubanCat ボードに推論デプロイするには、いくつかの最適化と変更が必要です。我々は、airockchip/ultralytics_yolov8 リポジトリのプログラムを使用して、rknpu に適したモデルをエクスポートします。

リポジトリプログラムをクローンします：

```
# カレントディレクトリに、airockchip/ultralytics_yolov8 の main ブランチをクローンします
git clone https://github.com/airockchip/ultralytics_yolov8.git
cd ultralytics_yolov8
```

ultralytics/cfg/default.yaml ファイルを修正します：

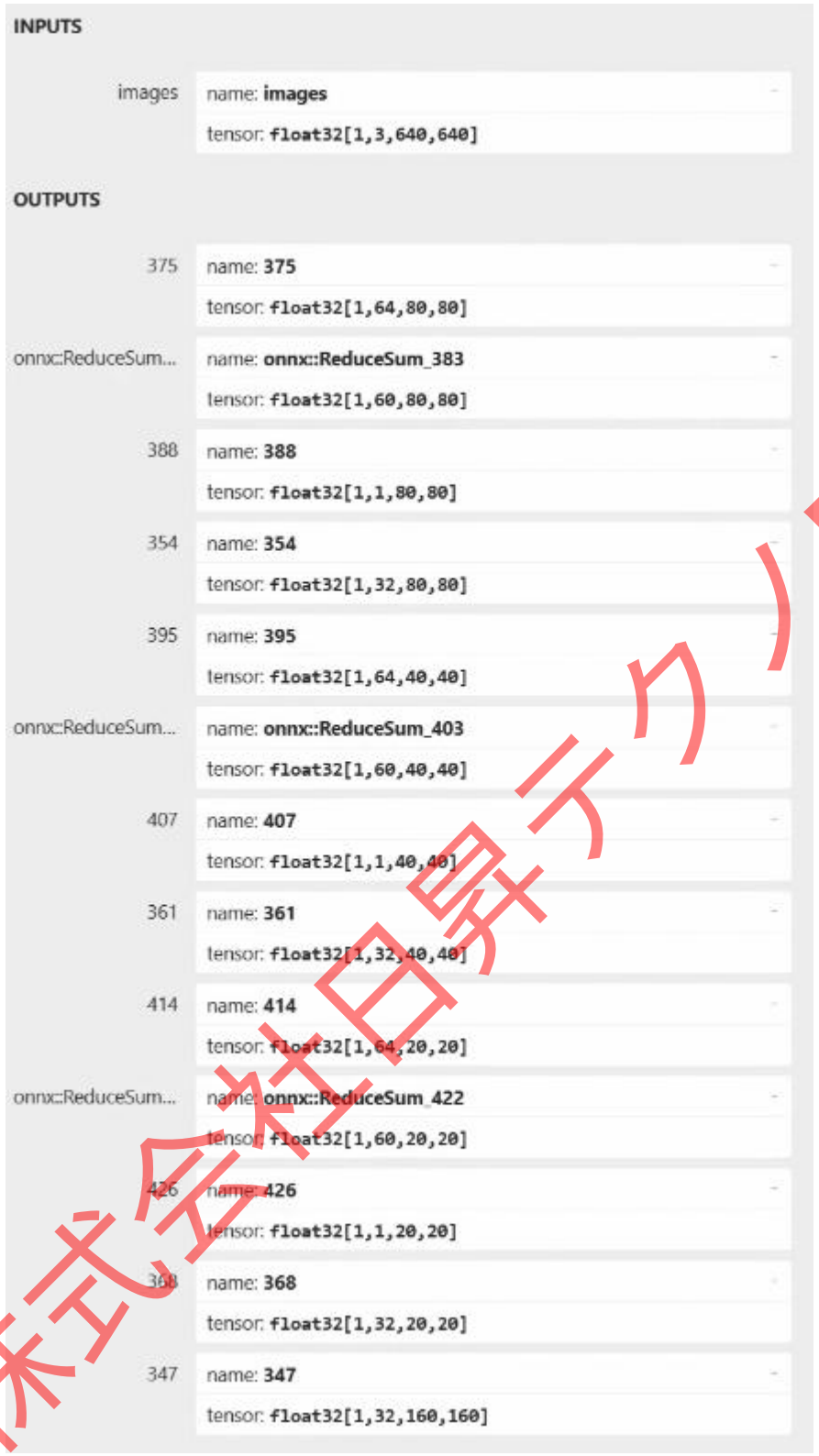
```
# ultralytics/cfg/default.yaml の model ファイルパスを修正し、前述のトレーニングしたモデルを指定します
```

```
# Train settings
model: xxxxx/runs/segment/train/weights/best.pt # (str, optional) path to model file, i.e.
yolov8n.pt, yolov8n.yaml
data:  # (str, optional) path to data file, i.e. coco128.yaml
epochs: 100  #(int) number of epochs to train for
```

プログラムを実行して onnx モデルをエクスポートします：

```
# プログラムを実行して onnx モデルをエクスポート
(yolov8) csun@CSUN-PC-
0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/yolov8/ultralytics_yolov8$ ex
port PYTHONPATH=./
(yolov8) csun@CSUN-PC-
0013:~/linuxshare/work/AI/jupyter/lubancat_ai_manual_code/example/yolov8/ultralytics_yolov8$ py
thon ultralytics/engine/exporter.py
Ultralytics YOLOv8.0.151    Python-3.8.18 torch-2.1.0+cu121 CPU ()
YOLOv8n-seg-taco summary (fused): 195 layers, 3269764 parameters, 0 gradients, 12.0 GFLOPs

PyTorch: starting from '/mnt/f/wsl_file/wsl_ai/yolov8/ultralytics_yolov8/best.pt' with
input shape (16, 3, 640, 640) BCHW and output shape(s) ((16, 64, 80, 80), (16, 60, 80, 80),
(16, 1, 80, 80),
(16, 32, 80, 80), (16, 64, 40, 40), (16, 60, 40, 40), (16, 1, 40, 40), (16, 32, 40, 40), (16,
64, 20, 20),
 (16, 60, 20, 20), (16, 1, 20, 20), (16, 32, 20, 20), (16, 32, 160, 160)) (6.5 MB)

RKNN: starting export with torch 2.1.0+cu121...

RKNN: feed /mnt/f/wsl_file/wsl_ai/yolov8/ultralytics_yolov8/best.onnx to RKNN-Toolkit or RKNN-
Toolkit2 to generate RKNN model.
Refer https://github.com/airockchip/rknn_model_zoo/tree/main/models/CV/object_detection/yolo
RKNN: export success ✓ 0.6s, saved as
'/mnt/f/wsl_file/wsl_ai/yolov8/ultralytics_yolov8/best.onnx' (12.5 MB)

Export complete (3.7s)
Results saved to /mnt/f/wsl_file/wsl_ai/yolov8/ultralytics_yolov8
Predict:         yolo predict task=segment
model=/mnt/f/wsl_file/wsl_ai/yolov8/ultralytics_yolov8/best.onnx imgsz=640
Validate:        yolo val task=segment
model=/mnt/f/wsl_file/wsl_ai/yolov8/ultralytics_yolov8/best.onnx imgsz=640 data=taco-seg.yaml
Visualize:       https://netron.app
```

エクスポートされた onnx モデルは best.pt モデルパスに保存され、名前は best.onnx です。Netron を使用してこのモデルの出力を確認します：



ultralytics_yolov8 リポジトリを使用してエクスポートされた onnx モデル（yolov8n-seg）の入力は 1*3*640*640 で、出力は合計 13 個あり、特徴マップ関連が 3 つ（12 個）とセグメントマスクが 1 つ（1*32*160*160）です。

例として特徴マップ 80*80 を挙げると、関連する出力は 1*64*80*80、1*60*80*80、1*1*80*80、

1*32*80*80 です。

●最初の 1*64*80*80 は検出ボックスの座標に関連し、関連する後処理によってボックスの座標（x1, y1, w, h）を取得します；
●二番目の 1*60*80*80 は、「80*80」は検出ボックスの数を示し、「60」は 60 カテゴリの信頼度を示します；
●三番目の 1*1*80*80 は、60 カテゴリの信頼度の合計を示し、後処理でボックスを高速にフィルタリングするために使用されます；
●四番目の 1*32*80*80 はセグメントマスク関連の重みです。

20.5. デプロイテスト

rknn モデルをエクスポートした後、LubanCat ボードに推論デプロイするには、rknpu2 を使用します。rknpu2 の使用方法については前の章を参照してください。yolov8-seg モデルの後処理については、rknn_model_zoo のデプロイ例を参考にします。

●推論プログラムの主な手順は次のとおりです：
●画像を読み込み、モデルを初期化し、画像を前処理する
●モデルを推論する
●後処理（ボックスデコードと NMS、マスクの処理）

結果を可視化または画像に保存する

ボード上でチュートリアルの付属サンプルソースをダウンロードし、example/garbage_detection/cpp ディレクトリに切り替えてプログラムをコンパイルします：

```
# 本マニュアル付属サンプルソースダウンロード URL：
https://www.dragonwake.com/download/LubanCat4/7-srccode/tutorial/CSAIEG358803_lubancat_ai_manual_code.zip
git clone https://github.com/airockchip/ultralytics_yolov8.git
cd ultralytics_yolov8

# -t テストプラントフォーム rk3588 を指定
cat@lubancat:~/lubancat_ai_manual_code/example/garbage_detection/cpp$./build-linux.sh -t rk3588
./build-linux.sh -t rk3588
==================================
TARGET_SOC=rk3588
INSTALL_DIR=/home/cat/lubancat_ai_manual_code/example/garbage_detection/cpp/install/rk3588_linux
BUILD_DIR=/home/cat/lubancat_ai_manual_code/example/garbage_detection/cpp/build/build_rk3588_linux
ENABLE_DMA32=OFF
CC=aarch64-linux-gnu-gcc
CXX=aarch64-linux-gnu-g++
==================================
-- The C compiler identification is GNU 10.2.1
-- The CXX compiler identification is GNU 10.2.1
-- Detecting C compiler ABI info
```

```
-- Detecting C compiler ABI info - done
-- Check for working C compiler: /usr/bin/aarch64-linux-gnu-gcc - skipped
-- Detecting C compile features
-- Detecting C compile features - done
-- Detecting CXX compiler ABI info
-- Detecting CXX compiler ABI info - done
-- Check for working CXX compiler: /usr/bin/aarch64-linux-gnu-g++ - skipped
-- Detecting CXX compile features
-- Detecting CXX compile features - done
-- Found OpenCV: /usr (found version "4.5.1")
-- Configuring done
-- Generating done
-- Build files have been written to:
/home/cat/lubancat_ai_manual_code/example/garbage_detection/cpp/build/build_rk3588_linux
Scanning dependencies of target taco_yolov8_seg
Scanning dependencies of target taco_yolov8seg_videocapture
# ...省略...
[100%] Built target taco_yolov8_seg
Install the project...
-- Install configuration: ""
-- Installing:
/home/cat/lubancat_ai_manual_code/example/garbage_detection/cpp/install/rk3588_linux/lib/librga
.so
-- Installing:
/home/cat/lubancat_ai_manual_code/example/garbage_detection/cpp/install/rk3588_linux/lib/librkn
nrt.so
-- Installing:
/home/cat/lubancat_ai_manual_code/example/garbage_detection/cpp/install/rk3588_linux/./taco_yol
ov8_seg
-- Set runtime path of
"/home/cat/lubancat_ai_manual_code/example/garbage_detection/cpp/install/rk3588_linux/./taco_yo
lov8_seg" to "$ORIGIN/lib"
-- Installing:
/home/cat/lubancat_ai_manual_code/example/garbage_detection/cpp/install/rk3588_linux/./taco_yol
ov8seg_videocapture
-- Set runtime path of
"/home/cat/lubancat_ai_manual_code/example/garbage_detection/cpp/install/rk3588_linux/./taco_yo
lov8seg_videocapture" to "$ORIGIN/lib"
-- Installing:
/home/cat/lubancat_ai_manual_code/example/garbage_detection/cpp/install/rk3588_linux/model/batc
h_1_000001.jpg
-- Installing:
/home/cat/lubancat_ai_manual_code/example/garbage_detection/cpp/install/rk3588_linux/model/taco
_labels.txt
-- Installing:
/home/cat/lubancat_ai_manual_code/example/garbage_detection/cpp/install/rk3588_linux/model/yolo
v8n_seg_rk3588.rknn
```

コンパイルされた出力ファイルは install/rk3588_linux ディレクトリにあり、taco_yolov8_seg と taco_yolov8seg_videocapture があります。最初のファイルは単一画像のテストに使用され、二番目のファイルはカメラまたはビデオファイルのテストに使用されます。

```
# 前述のテストセットデータから画像を取得し、単一画像の簡単なテストを行います
cat@lubancat:~/lubancat_ai_manual_code/example/garbage_detection/cpp/install/rk3588_linux$ ./taco_yolov8_seg
./taco_yolov8_seg <model_path> <image_path>
cat@lubancat:~/lubancat_ai_manual_code/example/garbage_detection/cpp/install/rk3588_linux$ ./taco_yolov8_seg ./model/yolov8n_seg_rk3588.rknn ./model/batch_1_000001.jpg
load lable ./model/taco_labels.txt
load lable ./model/taco_labels.txt
model input num: 1, output num: 13
input tensors:
  index=0, name=images, n_dims=4, dims=[1, 640, 640, 3], n_elems=1228800, size=1228800, fmt=NHWC, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003922
output tensors:
  index=0, name=375, n_dims=4, dims=[1, 64, 80, 80], n_elems=409600, size=409600, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=51, scale=0.222927
  index=1, name=onnx::ReduceSum_383, n_dims=4, dims=[1, 60, 80, 80], n_elems=384000, size=384000, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.002898
  index=2, name=388, n_dims=4, dims=[1, 1, 80, 80], n_elems=6400, size=6400, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003922
  index=3, name=354, n_dims=4, dims=[1, 32, 80, 80], n_elems=204800, size=204800, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-8, scale=0.020550
  index=4, name=395, n_dims=4, dims=[1, 64, 40, 40], n_elems=102400, size=102400, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=34, scale=0.149211
  index=5, name=onnx::ReduceSum_403, n_dims=4, dims=[1, 60, 40, 40], n_elems=96000, size=96000, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.002205
  index=6, name=407, n_dims=4, dims=[1, 1, 40, 40], n_elems=1600, size=1600, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.003922
  index=7, name=361, n_dims=4, dims=[1, 32, 40, 40], n_elems=51200, size=51200, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=35, scale=0.013169
  index=8, name=414, n_dims=4, dims=[1, 64, 20, 20], n_elems=25600, size=25600, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=18, scale=0.093956
  index=9, name=onnx::ReduceSum_422, n_dims=4, dims=[1, 60, 20, 20], n_elems=24000, size=24000, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.001961
  index=10, name=426, n_dims=4, dims=[1, 1, 20, 20], n_elems=400, size=400, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-128, scale=0.002739
  index=11, name=368, n_dims=4, dims=[1, 32, 20, 20], n_elems=12800, size=12800, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=38, scale=0.012272
  index=12, name=347, n_dims=4, dims=[1, 32, 160, 160], n_elems=819200, size=819200, fmt=NCHW, type=INT8, qnt_type=AFFINE, zp=-118, scale=0.029067
model is NHWC input fmt
model input height=640, width=640, channel=3
scale=0.312347 dst_box=(0 80 639 559) allow_slight_change=1 _left_offset=0 _top_offset=80 padding_w=0 padding_h=160
src width=2049 height=1537 fmt=0x1 virAddr=0x0x7f98e43040 fd=0
dst width=640 height=640 fmt=0x1 virAddr=0x0x556f42b3f0 fd=0
src_box=(0 0 2048 1536)
dst_box=(0 80 639 559)
color=0x72
rga_api version 1.10.0_[2]
fill dst image (x y w h)=(0 0 640 640) with color=0x72727272
Error on improcess STATUS=-1
```

```
RGA error message: Unsupported function: src unsupport width stride 2049, rgb888 width stride should be 16 aligned!
try convert image use cpu
finish
rknn_run
Clear plastic bottle @ (822 739 1290 950) 0.562
rknn run and process use: 110.355003 ms
```

結果画像：

20.6. 参考リンク

https://github.com/ultralytics/ultralytics/tree/main

https://github.com/pedropro/TACO

https://github.com/airockchip/rknn_model_zoo

第 21 章 総合サンプル２：カメラ監視検出

本章では、カメラ監視検出の例を簡単に紹介します。ユーザーはブラウザで監視ページにログインし、ログイン後にボタンをクリックしてビデオ録画とターゲット検出を行います。Web アプリケーションは Python の Flask フレームワークを使用しており、ストリーミングライブを実現します。画像は OpenCV を使用してカメラから取得し、画像検出処理には NPU を使用します。

- テストプラットフォーム：Lubancat 4
- ボードシステム：Debian11（デスクトップ付き）
- Python バージョン：Python 3.9
- OpenCV バージョン：4.10.0.82
- Toolkit Lite2：2.0.0beta
- Flask：3.0.3

21.1 依存ツールおよびライブラリのインストール

実験テストでは、Lubancat 4 を使用し、システムは Debian11 です。以下の手順でツールやライブラリをインストールします（一部のツールはシステムにすでにインストールされているため、再度インストールする必要はありません）：

```
# ツールのインストール
sudo apt update
sudo apt -y install git wget

# Python 関連のライブラリをインストール、デフォルトで Python3 を使用
sudo apt -y install python3-flask python3-pil python3-numpy python3-pip

# OpenCV-Python 関連のライブラリをインストール、テストでは 4.7.0.68 バージョンを使用
sudo pip3 install opencv-contrib-python
# RKNN-Toolkit-Lite2 のインストールについては前章の NPU 使用章を参照
```

21.2 ビデオストリームサーバーとカメラフレームの取得

21.2.1 ビデオストリームサーバーのデプロイ

この例では、Flask アプリケーションフレームワークを使用して、リアルタイムビデオストリームサーバーのある Web ページを構築します。

ヒント：Flask の簡単な使用法については前のチュートリアルや Flask の公式ドキュメントを参照してください。

Flask は`/video_viewer`ルートを通じて、ジェネレータを引数にとる Response オブジェクトを返します。Flask はジェネレータを呼び出し、ループに入り、カメラから取得したフレームデータをレスポンスブロックとしてクライアントに送信します。

サンプルソースコードを取得：

git clone -b main https://gitee.com/LubanCat/lubancat-flask-opencv-rknn.git

サンプルソースコード 1:lubancat-flask-opencv-rknn/controller/modules/home/views.py

```
# ホームページ
@home_blu.route('/')
def index():
    # テンプレートレンダリング
    username = session.get("username")
    if not username:
        return redirect(url_for("user.login"))
    return render_template("index.html")

# ビデオストリームを取得
def video_stream():
    global video_camera
    global global_frame


    if video_camera is None:
        video_camera = VideoCamera()


    while True:
        # start_time = time.time()
        frame = video_camera.get_frame()
        # end_time = time.time()
        # print('get_frame cost %f second' % (end_time - start_time))
        # time.sleep(0.01)
        if frame is not None:
            global_frame = frame
            yield (b'--frame¥r¥n'
                   b'Content-Type: image/jpeg¥r¥n¥r¥n' + frame + b'¥r¥n¥r¥n')
        else:
            yield (b'--frame¥r¥n'
                   b'Content-Type: image/jpeg¥r¥n¥r¥n' + global_frame + b'¥r¥n¥r¥n')

# ビデオビューア
@home_blu.route('/video_viewer')
def video_viewer():
    # テンプレートレンダリング
    username = session.get("username")
    if not username:
        return redirect(url_for("user.login"))
    return Response(video_stream(),
                    mimetype='multipart/x-mixed-replace; boundary=frame')
```

HTML の`index.html`ページでは、画像タグ`<img id="video" src="{{ url_for('home.video_viewer') }}">`を使用して、`url_for`で`/video_viewer`ルートを指し、ブラウザは自動的にストリーム中の画像を表示し、画像要素を更新します。

サンプルソースコード 2:lubancat-flask-opencv-rknn/controller/templates/index.html（一部）

```
<!DOCTYPE html>
<html lang="ja">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <meta http-equiv="X-UA-Compatible" content="ie=edge">
    <title>カメラ監視検出</title>
    <style>
        body {
            background-color: #484856;
        }
    </style>
</head>
<body>
<h1 align="center" style="color: whitesmoke;">Flask+OpenCV+Rknn</h1>
<div class="top">
    <div class="recorder" id="recorder" align="center">
        <button id="record" class="btn">ビデオを録画</button>
        <button id="stop" class="btn">録画を停止</button>
        <button id="process" class="btn">検出を開始</button>
        <button id="pause" class="btn">検出を一時停止</button>
        <input type="button" class="btn" value="ログアウト"
            onclick="javascrtpt:window.location.href='{{ url_for('user.logout') }}'">
        <a id="download"></a>
        <script type="text/javascript" src="{{ url_for('static', filename='button_process.js') }}"></script>
    </div>
</div>
<img id="video" src="{{ url_for('home.video_viewer') }}">
</body>
</html>
```

28.2.2 カメラからフレームを取得

カメラからフレームを取得するには、ラップされた`VideoCamera`クラスをインポートします：

サンプルソースコード 3:lubancat-flask-opencv-rknn/controller/utils/camera.py（一部省略）

```
class VideoCamera(object):
    def __init__(self):
        # システムのデフォルトカメラを開く
        self.cap = cv2.VideoCapture(0)
        if not self.cap.isOpened():
            raise RuntimeError('カメラを開けませんでした。')

        # RKNN オブジェクトを作成
        self.rknn_lite = RKNNLite()

        # フレームの幅と高さを設定
```

```
            self.cap.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
            self.cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 640)

            # ビデオ録画スレッド
            self.recordingThread = None
            self.is_record = False
            self.out = None

            # 画像処理
            self.image = None
            self.rknn_frame = None
            self.frame = None
            self.outputs = None

            # 認識スレッドの状態
            self.is_process = False

            # RKNN をロード
            self.load_rknn()

        # プログラム終了時にカメラを解放
        def __del__(self):
            self.cap.release()
            self.rknn_lite.release()

        def get_frame(self):
            ret, self.frame = self.cap.read()
            if ret:
                if self.is_process:
                    self.image = cv2.cvtColor(self.frame, cv2.COLOR_BGR2RGB)
                    self.outputs = self.rknn_lite.inference(inputs=[self.image])
                    self.frame = process_image(self.image, self.outputs)

                if self.frame is not None:
                    ret, image = cv2.imencode('.jpg', self.frame)
                    return image.tobytes()
            else:
                return None
......
```

21.3 NPU で画像を処理

NPU を使用して画像検出処理を行います。この例では追加のモデルトレーニングは行わず、公式の Toolkit Lite2 ツールの`examples/onnx/yolov5`サンプルを使用します。

ヒント: ボード上での RKNN-Toolkit-Lite2 のインストールと使用方法については、前章のチュートリアルや公式 GitHub ドキュメントを参照してください。

例プログラム処理フロー:

サンプルソースコード 4:カメラ監視検出のプログラム「controller/utils/camera.py」から抜粋

```
class VideoCamera(object):
    def __init__(self):
        # システムのデフォルトカメラを開く
        self.cap = cv2.VideoCapture(0)
        if not self.cap.isOpened():
            raise RuntimeError('カメラを開けませんでした。')

        # RKNN オブジェクトを作成
        self.rknn_lite = RKNNLite()

        # フレームの幅と高さを設定
        self.cap.set(cv2.CAP_PROP_FRAME_WIDTH, 640)
        self.cap.set(cv2.CAP_PROP_FRAME_HEIGHT, 640)

        # ビデオ録画スレッド
        self.recordingThread = None
        self.is_record = False
        self.out = None

        # 画像処理
        self.image = None
        self.rknn_frame = None
        self.frame = None
        self.outputs = None

        # 認識スレッドの状態
        self.is_process = False

        # RKNN をロード
        self.load_rknn()

    # プログラム終了時にカメラを解放
    def __del__(self):
        self.cap.release()
        self.rknn_lite.release()

    def get_frame(self):
        ret, self.frame = self.cap.read()
        if ret:
            if self.is_process:
                self.image = cv2.cvtColor(self.frame, cv2.COLOR_BGR2RGB)
                self.outputs = self.rknn_lite.inference(inputs=[self.image])
                self.frame = process_image(self.image, self.outputs)

            if self.frame is not None:
                ret, image = cv2.imencode('.jpg', self.frame)
                return image.tobytes()
        else:
            return None

    def start_record(self):
        self.is_record = True
        self.recordingThread = RecordingThread("ビデオ録画スレッド", self.cap)
        self.recordingThread.start()

    def stop_record(self):
        self.is_record = False

        if self.recordingThread is not None:
            self.recordingThread.stop()

    def load_rknn(self):
```

```
        # RKNN モデルをロード
        print('--> RKNN モデルをロード')
        ret = self.rknn_lite.load_rknn(RKNN_MODEL)
        if ret != 0:
            print('RKNN モデルのロードに失敗しました')
            exit(ret)
        # ランタイム環境を初期化
        print('--> ランタイム環境を初期化')
        ret = self.rknn_lite.init_runtime()
        if ret != 0:
            print('ランタイム環境の初期化に失敗しました！')
            exit(ret)

    def start_process(self):
        self.is_process = True

    def stop_process(self):
        self.is_process = False

    def rknn_process(self):
        if self.is_process:
            print('--> is_process true')
            self.outputs = self.rknn_lite.inference(inputs=[self.frame])
        else:
            self.outputs = None
```

NPU を使用せずに画像検出と識別処理を行う場合は、OpenCV を直接使用できます。興味のある方はコードを研究し、データセットを追加してトレーニングするか、他のモデルを使用して実験を行ってください。OpenCV ライブラリを使用した画像処理と数字認識機能のコードは、実験コードディレクトリ `controller/utils/opencvtest.py` にあります。

28.4 動作環境設定

```
# lubancat-flask-opencv-rknn ディレクトリに移動
cd ./lubancat-flask-opencv-rknn

# main.py ファイルの起動関数パラメータを変更
vim main.py

# デフォルト設定は host="0.0.0.0"で、これはデフォルトネットワークインターフェースの IP です
app.run(threaded=True, host="0.0.0.0", port=5000)

# ボードの環境に応じて、具体的な IP、例えば 192. を設定
app.run(threaded=True, host="192.168.11.241", port=5000)
# 現在、サーバーは IP アドレス 192.168.11.241、ポート 5000 でリッスンしています。

    # サンプルソースコードのディレクトリに移動
    cd ./controller/utils/
    # camera.py ファイルの VideoCamera(object)関数を編集
    vim camera.py
    # self.cap = cv2.VideoCapture(11)行の 11 をあなたのカメラのデバイス番号またはデバイスファ
```

イル("/dev/video11")に変更

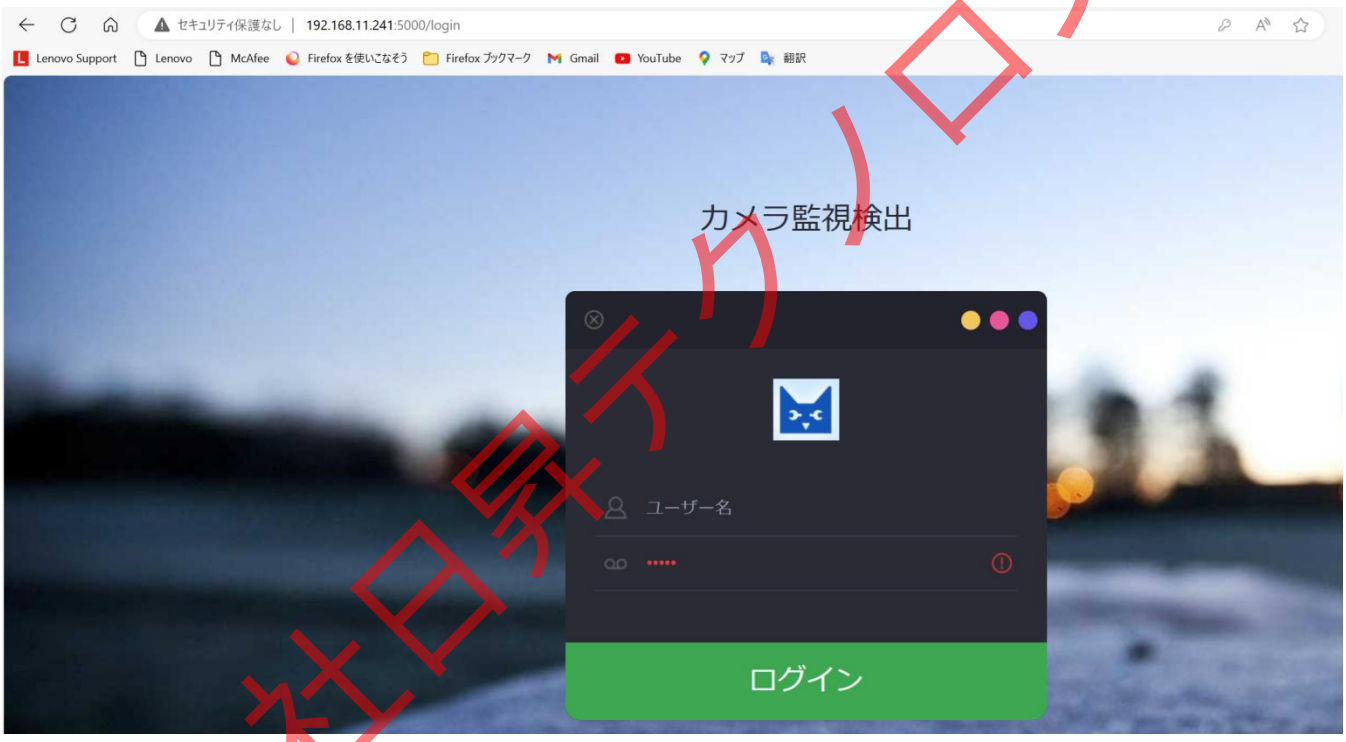
```
self.cap = cv2.VideoCapture(11)
```

28.5 実験のテスト

前節のチュートリアルに従って修正した後、実験コードを実行して実験現象を確認します：

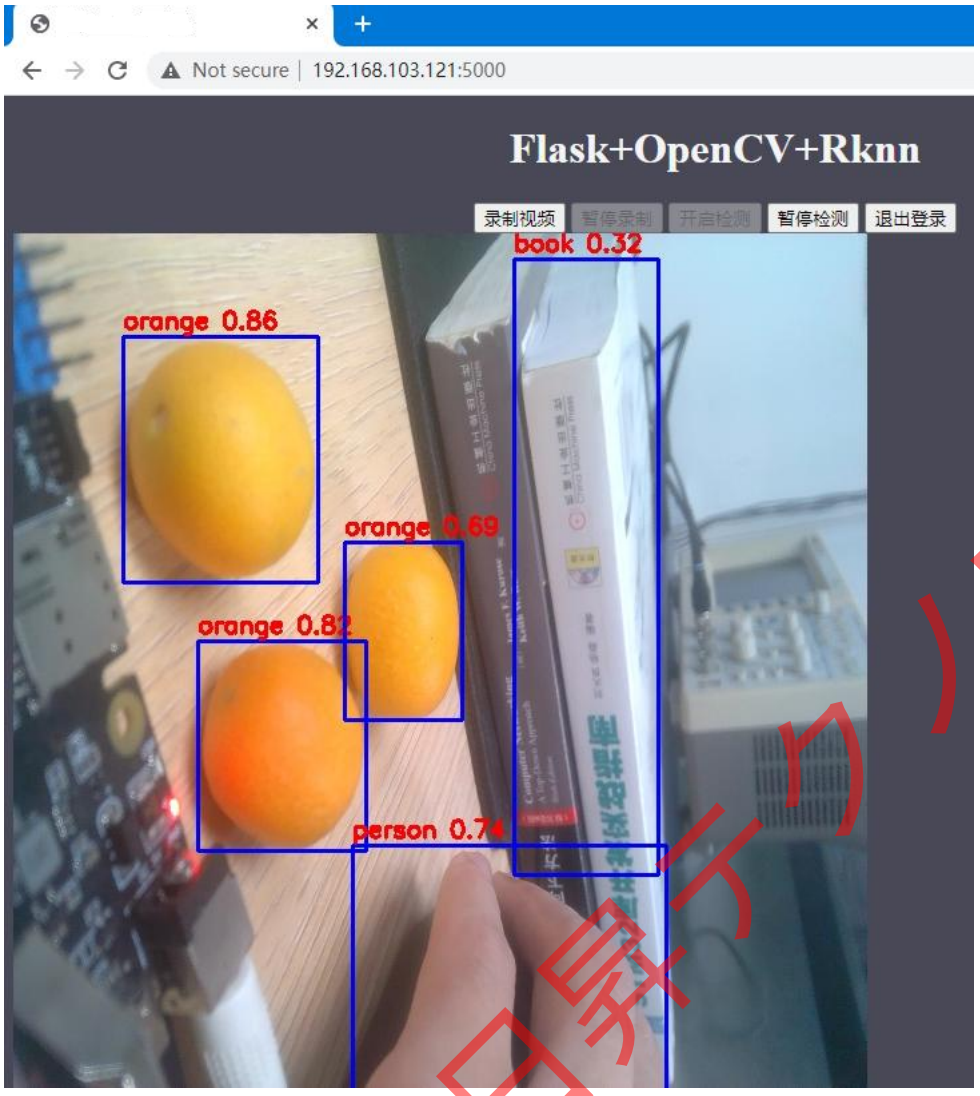
```
# プロジェクトコードディレクトリ lubancat-flask-opencv-rknn で以下のコマンドを実行
sudo python3 main.py
```

実験の現象を以下の URL でブラウザから確認できます: `http://192.168.11.241:5000`



実験現象の例：

1. ログイン後、監視画面に入り、検出を開始するには「検出開始」ボタンをクリックします。
   Login id/pwd:cat/temppwd　　基板の SSH ログイン ID とパスワードが同じです。



2. フレーム取得時間のテスト

Python の`time`モジュールを使用してプログラムの実行時間を計測し、1 フレームの取得にかかる時間と NPU 処理にかかる時間を簡単にテストします。

サンプルソースコード 5: controller/modules/home/views.py

```
# ビデオストリームを取得
def video_stream():
    global video_camera
```

```
    global global_frame

    if video_camera is None:
        video_camera = VideoCamera()

    while True:
        start_time = time.time()
        frame = video_camera.get_frame()
        end_time = time.time()
        print('get_frame cost %f second' % (end_time - start_time))
        # time.sleep(0.01)
        if frame is not None:
            global_frame = frame
            yield (b'--frame¥r¥n'
                   b'Content-Type: image/jpeg¥r¥n¥r¥n' + frame + b'¥r¥n¥r¥n')
        else:
            yield (b'--frame¥r¥n'
                   b'Content-Type: image/jpeg¥r¥n¥r¥n' + global_frame + b'¥r¥n¥r¥n')
```

テストは Lubancat 4 を使用し、デフォルト設定の Debian11 システム、Toolkit Lite2 ツールの YOLOv5 サンプルモデルを使用しました。以下のテストデータは参考用です：

- NPU を使用せずに画像検出を行う場合、1 フレームの取得に約 65ms かかり、最速で 23ms、最遅で 121ms。
- NPU を使用して画像処理を行う場合、NPU デフォルト 600MHz でのテストでは、1 フレームの取得に約 345ms かかります（カメラ画像の取得、前処理、NPU 処理、後処理を含む）。
- NPU の周波数を 1000MHz に設定したテストでは、1 フレームの取得に約 314ms かかります。

```
# Lubancat4 NPU 周波数を 1000MHz に設定
echo 1000000000 > /sys/class/devfreq/fdab0000.npu/cur_freq
```

ルーバンキャット監視検出の例は、簡単な監視表示とターゲット検出機能を実現し、学習の参考として使用できます。

21.6 参考リンク

- https://github.com/miguelgrinberg/flask-video-streaming
- https://github.com/rockchip-linux/rknn-toolkit2