AI エッジ基板 CSAIEG358803 で Python 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 2024/05/20

- **修正履歴**

NO	バージョン	修正内容	修正日
1	Ver1.0	新規作成	2024/05/20

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

第 1 章　AI エッジ基板と Python

1.1 Python の概要

Python とは何かについて、ここで簡単に紹介します：

Python はクロスプラットフォームのインタプリタ型プログラミング言語です・
Python は学びやすく、使いやすく、強力で、AI、データ処理、プログラミング教育などで幅広く使用されています。
多くの開発者がさまざまな Python ライブラリを提供しており、他の人はこれらのライブラリを使用して自分のプログラムを簡単に開発できます。

Python の基本知識については、次を参照してください：

Youtube ビデオ：（1 本ビデオより 6 時間で初心者も習得）
https://www.youtube.com/watch?v=YEr-cz9pAHU&list=PLavQwENTsEBXwIi_BiVwxeGYr_AYOdam1

オンライン Python チュートリアル：
https://docs.python.org/ja/3/tutorial/

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「python_lubancat_RK_tutorial_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 2 章　Python インストール

この章では、AI エッジ RK3588 基板に Python 環境をインストールする方法を説明します。

重要：特別な記載がない限り、本書のチュートリアルは Python 3.9.2 バージョン（イメージシステムは Debian11.9）に基づいて実験と説明を行います。

2.1 python、pip を APT ツールでインストール

AI エッジ RK3588 基板システムは APT ツールをサポートしているため、apt コマンドを使って直接インストールできます。Python3 バージョンを直接使用します。

```
# 基板上に下記コマンドを実行、インタネットに接続必要。
#（《Linux 基礎とアプリケーション開発実践ガイド》を参照して SSH 接続）
# 始めて apt を実行する前、update が必要
sudo apt update
# python3 インストール（出荷時すでにインストール済み、実行しなくても良い）
sudo apt -y install python3
# pip ツールインストール（出荷時すでにインストール済み）
sudo apt -y install python3-pip
```

2.2 Python のバージョンについて

AI エッジ RK3588 基板のイメージは Debian や Ubuntu などの Linux ディストリビューションに基づいており、これらのディストリビューションにはデフォルトで Python が含まれています。通常、古いバージョンの Python2* と最新のバージョンの Python3* が含まれています。デフォルトでインストールされているバージョンを確認するには、以下のコマンドを使用します：

```
# python3 を確認
cat@lubancat:~$ python3 --version

# 確認結果
Python 3.9.2

# python2 を確認
cat@lubancat:~$ python2 --version

# 出力
Python 2.7.18

# pip3 バージョンを確認
cat@lubancat:~$ pip3 --version

# 出力結果は下記通り
pip 20.3.4 from /usr/lib/python3/dist-packages/pip (python 3.9)
```

2.3 Python と pip のデフォルトバージョンの設定

上記のコマンドで Python3 と pip3 にバージョン番号が付いていることに気付くでしょう。これは主に Python の歴史的な理由で、Python2 との区別がしやすくなっています。しかし、Python2 は現在サポートが終了しているため、使用をお勧めしません。提供されたイメージには Python2 と Python3 がデフォルトでインストールされており、現在のシステムでデフォルトバージョンを設定できます。次のコマンドを使って、Python と pip コマンドが Python3 をデフォルトで使うように設定します：

```
#シンボリックリンクを設定し、Python のデフォルトを Python3 にする

sudo ln -sf /usr/bin/python3 /usr/bin/python

#シンボリックリンクを設定し、pip のデフォルトを pip3 にする

sudo ln -sf /usr/bin/pip3 /usr/bin/pip
```

設定後、python または pip コマンドを直接使用できます。現在のシステムで Python バージョンを確認してください：

```
# Python の場所を確認
cat@lubancat:~$ which python
/usr/bin/python

# Python バージョンを確認
cat@lubancat:~$ python --version
# 以下は出力例
Python 3.9.2

# pip バージョンを確認
cat@lubancat:~$ pip --version
# 以下は出力例
pip 20.3.4 from /usr/lib/python3/dist-packages/pip (python 3.9)
```

バージョンを明確に区別するため、本書では以降も引き続き python3 または pip3 コマンドを使用して説明します。

2.4 ソフトウェアライブラリのインストール方法

2.4.1 apt を使って pip の代わりにパッケージをインストールする

pip ツールでパッケージをインストールする場合、通常はローカルでのコンパイルが必要です。AI エッジ RK358 の性能が高性能の PC と比べて低いものがあり、コンパイルに非常に長い時間がかかるだけでなく、特定のライブラリが不足しているためにインストールに失敗することもあります。

したがって、性能の低いボードで依存パッケージをインストールする際は、まず apt ツールでインストール可能かを検索することをお勧めします。apt ではソフトウェアリポジトリから事前にコンパイルされたパッケージをダウンロードするため、インストール時間はほぼネットワーク速度のみに依存します。新しいバージョンが必要な場合や apt で見つからないパッケージが必要な場合のみ pip でのインストールを検討してください。

たとえば、Python の一般的なデータサイエンスライブラリである numpy をインストールするには、以下の apt コマンドを使用することで迅速に完了します：

```
# apt を使って Python の numpy パッケージをインストール
sudo apt -y install python3-numpy
```

他の具体的なソフトウェアパッケージが apt ツールでどのような名前になっているかについては、ネットで「apt インストール XXX（例：numpy）」のような内容を検索してください。Debian システムを使用している場合、Debian のソフトウェアリストリポジトリで「numpy」のようなキーワードを検索します：Debian 公式ソフトウェアパッケージを見つけるためです。

2.4.2 setuptools ツールでパッケージをインストールする

場合によっては、ユーザーが使用したい特定の Python ライブラリがあることがありますが、何らかの理由でライブラリのソースコードしか入手できない場合があります。その場合、pip ツールや apt ツールではインストールができません。

その際には Python の setuptools ツールを使用して、ソースコードから Python ライブラリをインストールすることができます。setuptools ツールのインストール方法は次のとおりです：

```
# ターミナルで以下のコマンドを入力します
sudo apt -y install python3-setuptools
```

第 3 章　Python 動作

この章では、AI エッジ RK3588 基板で Python を簡単に使用する方法を説明します。使い方は PC とほとんど同じです。

3.1 Python コーディング及び動作環境構築

本マニュアルは基本的に VS code の拡張機能 Remote-SSH を使って、ソースを基板の「/home/cat/python/code」フォルダにソースコードを直接作成して、VS code で動作するようにします。

1．Windows 上に VS code インストール

Download　URL：
https://code.visualstudio.com/download

2．日本語パッケージインストール

VS code はデフォルトで英語でインストールされます、拡張機能から「Japanese Language Pack for Visual Studio Code」検索すれば、インストールできます。

3．Python、Tabnine インストール

Python 言語サーポート用の拡張機能
Tabnine：AI プログラマ自動ツール

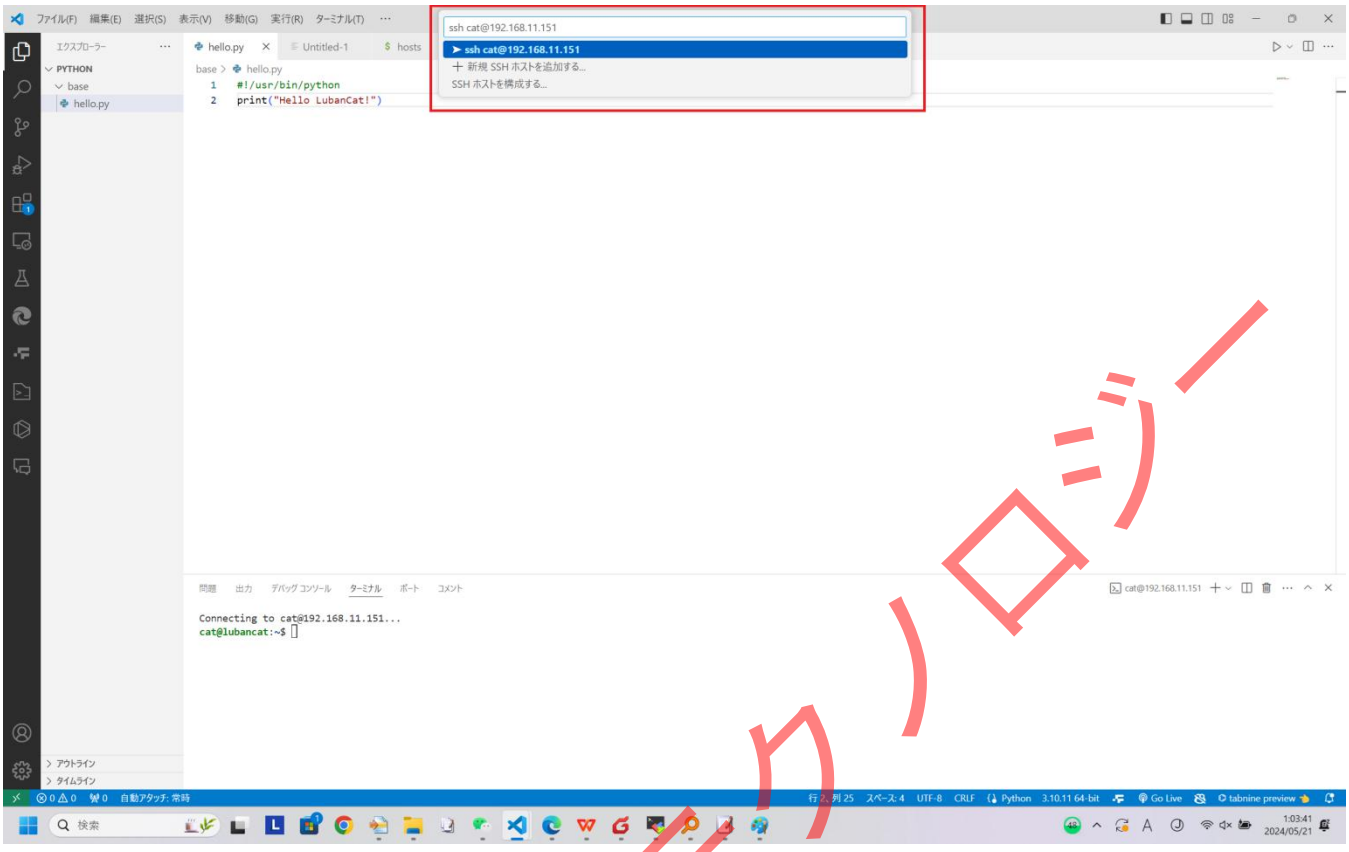
4．Remote-SSH 拡張インストール

VS code で基板と接続用の SSH 拡張機能：Remote-SSH をインストールしましょう。

5．基板の WIFI 或いは有線 LAN 設定

6．Remote-SSH 接続

接続前、VS code 画面の左下緑の「><」をクリックして、「ホストに接続する...」を選べ、「ssh cat@192.168.11.151」を入力（赤字が基板の IP）

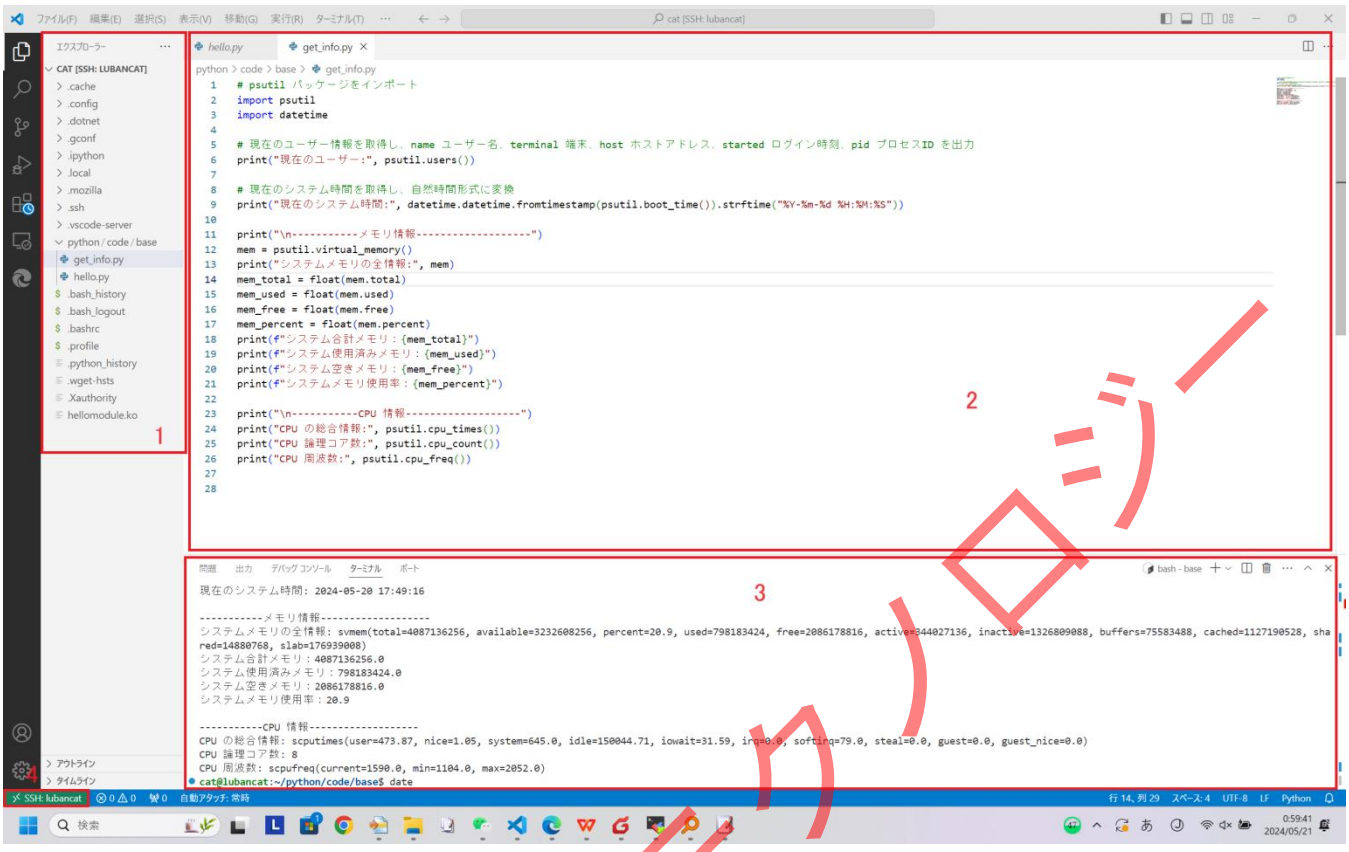


接続できた様子は下図通りです、

１：リモートエクスプローラー：要は基板上のフォルダーを Windows 上の VS code 画面に直接表示

２：編集画面　　３：ターミナル　　４：接続基（IP であれば IP を表示、こちらはホスト名を表示）

ファイル閲覧、コーディング、プログラム実行はすべて一つ画面で表示されます、特に、基板のリモート側を感じられなくてとても便利



3.2 Python 端末のインタラクティブ環境の使用

python コマンドを直接実行して Python のインタラクティブなプログラミング環境に入ることができます。インタラクティブプログラミングでは、スクリプトファイルを作成する必要がなく、Python インタプリタのインタラクティブモードでコードを書きます。

インタラクティブな環境で簡単な操作を行うことが可能です。たとえば、端末に「hello world!」を出力したり、簡単な算術演算を行ったりします。最後に exit() で終了します：

```
# ターミナルで入力
python3

# 出力内容
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.
>>>

# 入力してエンターを押します
>>> print("hello world!")

# 出力
hello world!
>>>
```

```
# 入力
>>> 8 + 2

# 出力
10
```

インタラクティブモードから退出するには、exit() や quit() を使うか、Ctrl+D の組み合わせで退出できます

```
>>> exit()
cat@lubancat:~$
```

また、端末で Python を使用する際に、ipython（拡張インタラクティブ Python シェル）をインストールすることも可能です。

```
# ipython のインストール
sudo pip3 install ipython

# 使用例
cat@lubancat:~$ ipython
Python 3.9.2 (default, Feb 28 2021, 17:03:44)
Type 'copyright', 'credits' or 'license' for more information
IPython 8.18.1 -- An enhanced Interactive Python. Type '?' for help.
```

8 + 2 の例：

```
In [1]: 8 + 2
Out[1]: 10

In [2]:
```

インタラクティブモードを終了するには、exit を使います。

3.3 Python スクリプトの使用

シンプルな Python コードを作成して、hello.py ファイルに保存し、実行することもできます。ボードシステムにシリアルポートでログインし、vim エディタでコードを書いたり、VS Code で SSH を使ってボードにリモートログインしコードを書いたりできます。

hello.py ファイルの内容は以下の通りです：

リスト 1：base/hello.py ファイルの内容

```
print("Hello CSUN AI Edge!")
```

次の手順で実行します：

```
# hello.py の作成
```

```
# `ls` コマンドでディレクトリにファイルがあることを確認
$ls
# 出力
hello.py

# 実行
$python3 hello.py


# 出力
Hello CSUN AI Edge!
```

スクリプトを使用する場合、スクリプト内でインタープリタのパスを指定することもできます。例えば：

リスト 2：base/hello.py ファイルの内容

```
#!/usr/bin/python
print("Hello CSUN AI Edge!")



# 実行
cat@lubancat:~/python/code/base$ ./hello.py
Hello CSUN AI Edge!
cat@lubancat:~/python/code/base$
```

3.4 他のパッケージ import

基本的な Python コードライブラリを実行するだけでなく、サードパーティのパッケージをコードにインポートして、より複雑なアプリケーションを実行することもできます。

例のプログラムを使って説明します。新しいファイル get_info.py を作成し、次のコードを入力します：

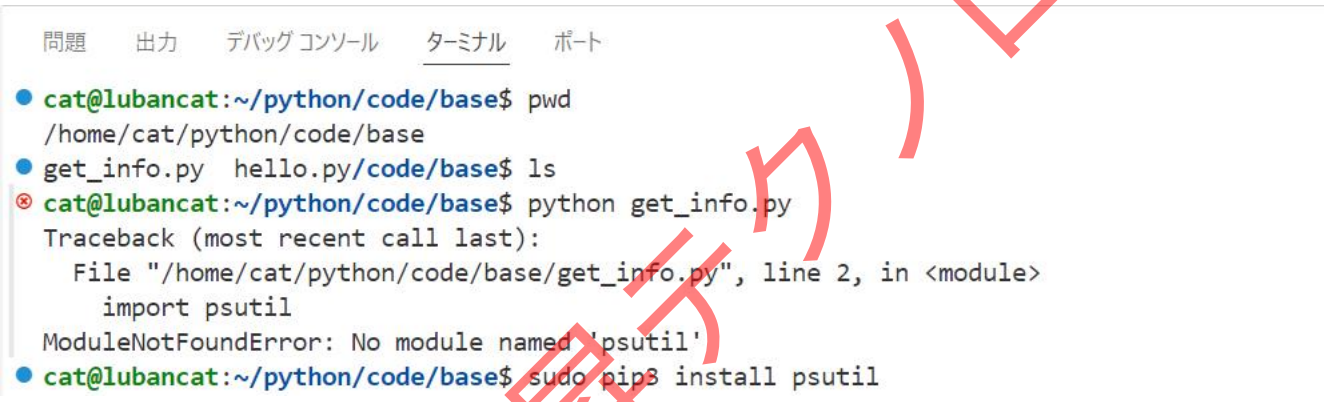
```
# psutil パッケージをインポート
import psutil
import datetime

# 現在のユーザー情報を取得し、name ユーザー名、terminal 端末、host ホストアドレス、started ログイン時刻、pid プロセス ID を出力
print("現在のユーザー:", psutil.users())

# 現在のシステム時間を取得し、自然時間形式に変換
print("現在のシステム時間:", datetime.datetime.fromtimestamp(psutil.boot_time()).strftime("%Y-%m-%d %H:%M:%S"))

print("¥n----------メモリ情報--------------------")
mem = psutil.virtual_memory()
print("システムメモリの全情報:", mem)
mem_total = float(mem.total)
mem_used = float(mem.used)
mem_free = float(mem.free)
```

```
mem_percent = float(mem.percent)
print(f"システム合計メモリ：{mem_total}")
print(f"システム使用済みメモリ：{mem_used}")
print(f"システム空きメモリ：{mem_free}")
print(f"システムメモリ使用率：{mem_percent}")

print("¥n----------CPU 情報--------------------")
print("CPU の総合情報:", psutil.cpu_times())
print("CPU 論理コア数:", psutil.cpu_count())
print("CPU 周波数:", psutil.cpu_freq())
```

$python get_info.py

実行すると、下図のようなエラーが出ます。

ModuleNotFoundError: No module named 'psutil'



pip3 ツールでパッケージ psutil をインストール

$sudo pip3 install psutil

第 4 章　Python 基本クイックガイド

Python はシンプルで学びやすく、強力な機能を持つプログラミング言語です。本章では Python の基本構文を簡単に見ていきます。詳細は参考文書或いは「1.1 Python の概要」に記載されている URL をご覧ください。

4.1 コメント

Python では、単一行コメントは # で始まります。複数行のコメントは、複数の # を使用するか、’’’ または ″″″ を使用します。例えば：

```
# これはコメントです。

'''
これもコメントです。
'''

"""
これはやはりコメントです。
"""
print ("Hello")
```

4.2 インデント

Python ではインデントを使用してコードブロックを表します。インデントのスペース数は可変ですが、同じコードブロック内のステートメントは同じインデントスペース数を含める必要があります。そうしないと、実行エラーが発生します。例は以下の通りです。

```
# for ループ
for i in range(5):
    print("in")
print("out")
# 出力結果
in
in
in
in
in
out
```

4.3 インデント

Python では変数と変数型の宣言は不要で、動的に決定されます。各変数は使用前に必ず値を代入する必要があり、代入されて初めてその変数が作成されます。変数の型は代入された値によって決まります。例えば：

```
#section 4.3
a = 10 # 整数変数
print(type(a))
a = 10.0 # 浮動小数点型変数
print(type(a))
a = "test" # 文字列変数
print(type(a))
# 出力結果
<class 'int'>
<class 'float'>
<class 'str'>
```

Python でよく使われる標準データ型には次のようなものがあります：不変データ型：Number（数値）、String（文字列）、Tuple（タプル）；可変データ型：List（リスト）、Dictionary（辞書）、Set（セット）。以下に簡単な例をいくつか挙げます：

4.3.1 String（文字列）

Python では、文字列（文字型はありません）はシングルクォート ’’ またはダブルクォート ″″ で囲みます。いくつかの特殊文字はバックスラッシュ ¥ でエスケープされます。例えば：

```
#4.3.1 String(文字列)
str0 = 'Python'

'''
字符串索引和截取
+---+---+---+---+---+---+
| P | y | t | h | o | n |
+---+---+---+---+---+---+
  0   1   2   3   4   5   （前からのインデックス）
 -6  -5  -4  -3  -2  -1   （後ろからのインデックス）
   :  1   2   3   4   5 :（前からのスライス）
   : -5  -4  -3  -2  -1 :（後ろからのスライス）

'''
print (str0)            #str0 文字列全体
print (str0[0])         #テキスト列の最初の項目は文字列の最初の文字を出力します
print (str0[-6])        #文字列の最初の文字を出力します
print (str0[:])         #str0 文字列全体
print (str0[2:])        # 3 文字目以降の全文字を出力
print (str0[:-2])       # 1 文字目から 4 文字目までの全文字を出力
print (str0 * 2)        # 文字列を 2 回出力します。print (2 * str0) と書くこともできます。
print (str0 + "TEST")   #接続文字列

# 出力結果
Python
P
P
Python
thon
Pyth
PythonPython
PythonTEST
```

文字列はインデックスまたはスライスすることができますが、インデックスを変更したり再代入することはできません。

4.3.2 List（リスト）

リストはほとんどの集合型データ構造を実現できます。リスト内の要素の型は異なっていても構いません。リストは数値、文字列、さらにはリスト（いわゆるネスト）を含むことができます。リストは角括弧 [ ] で囲まれた要素のリストで、要素はカンマで区切られます。文字列のインデックスと同様に、リストのインデックスは 0 から始まり、2 番目のインデックスは 1 です。

```
#4.3.2 List(リスト)
list0 = [ 'python', "test", 10, 10.0 ]
list1 = [ 123, 'Python' ]

print (list0)           # 完全なリストを出力
print (list0[0])        # リストの最初の要素を出力
print (list0[1:3])      # 2 番目から 3 番目までの要素を出力
print (list0[2:])       # 3 番目からの全要素を出力
print (list0*2)         # リストを 2 回出力
print (list0 + list1)   # リストを連結
# 出力結果
['python', 'test', 10, 10.0]
python
['test', 10]
[10, 10.0]
['python', 'test', 10, 10.0, 'python', 'test', 10, 10.0]
['python', 'test', 10, 10.0, 123, 'Python']
```

4.3.3 Tuple（タプル）

Python のタプルはリストに似ていますが、タプルの要素は変更できません。タプルは丸括弧（ ）を使用し、リストは角括弧 [ ] を使用します。タプルの作成は非常に簡単で、括弧内に要素を追加し、カンマで区切るだけです。

使用例：

```
#4.3.2 List(リスト)
tup1 = ('Python', 'Test', 1997, 2000)
tup2 = (1, 2, 3, 4, 5, 6, 7 )
print ("tup1[0]: ", tup1[0])
print ("tup2[1:5]: ", tup2[1:5])

# 出力結果
tup1[0]:  Python
tup2[1:5]:  (2, 3, 4, 5)
```

4.4 フロー制御関連

4.4.1 条件制御

Python の条件文は、1 つまたは複数の文の実行結果（True または False）に基づいてコードブロックの実行を決定します。

```
#4.4.1 条件制御

# 数字当てゲーム
number = 20
guess = 0
print("数字当てゲーム!")
while guess != number:
    guess = int(input("0-100 の間で数字を入力してください:"))

    if guess == number:
        print("おめでとうございます、正解です！")
    elif guess < number:
        print("数字が小さすぎます...")
    elif guess > number:
        print("数字が大きすぎます...")

# 出力結果
数字当てゲーム!
0-100 の間で数字を入力してください：10
数字が小さすぎます...
0-100 の間で数字を入力してください：30
数字が大きすぎます...
0-100 の間で数字を入力してください：20
おめでとうございます、正解です！
```

4.4.2 ループ文

Python には for ループと while ループがあります。例えば：

```
#4.4.2 ループ文

# while を使った合計計算
n = 100
sum = 0
counter = 1
while counter <= n:
    sum = sum + counter
    counter += 1

print("1 から %d までの合計は: %d" % (n, sum))

# 組み込み関数 range() を使用。数列を生成します。
for i in range(4):
    print(i)

# 出力結果
1 から 100 までの合計は: 5050
0
1
2
3
```

4.4.3 イテレーション

イテレータは、ループ位置を記憶できるオブジェクトです。イテレータオブジェクトは、コレクションの最初の要素からアクセスを開始し、すべての要素がアクセスされるまで続けます。イテレータは前方にのみ進み、後退することはありません。

例えば：

```
#4.4.3 イテレーション

list = [1, 2, 3, 4]
it = iter(list)  # イテレータオブジェクトを作成し、for 文でループ
for x in it:
    print(x, end=" ")

# 出力結果
1 2 3 4
```

4.5 関数

関数は、アプリケーションのモジュール性とコードの再利用性を向上させます。Python は多くの組み込み関数を提供しており、例えば print() など、ユーザーが定義した関数も使用できます。

Python では、def キーワードを使って関数を定義します。一般的な形式は次の通りです：

def 関数名（引数リスト）：関数本体

使用例：

```
#4.5 関数

# 関数 my_function を定義
def my_function(test):
    print("Hello!" + test)

# 関数 my_function を使用
my_function(' Python')

# 出力結果
Hello! Python
```

4.6 モジュールとパッケージ

4.6.1 モジュール

Python は、ファイルから定義を取得し、スクリプトまたは対話型インタープリタのインスタンスで使用する方法を提供します。このようなファイルはモジュールと呼ばれます。モジュール内の定義は、import 文を使用して他のモジュールやメインモジュールにインポートできます。

例えば：

```
#4.6.1 モジュール

# Python の標準モジュール: sys を使用し、import 文を使用
import sys
print('\nPython 環境パス:', sys.path)

# from … import 文を使用
from periphery import I2C

# 出力結果
```

Python 環境パス： ['/home/cat/python/code/base', '/usr/lib/python39.zip', '/usr/lib/python3.9', '/usr/lib/python3.9/lib-dynload', '/usr/local/lib/python3.9/dist-packages', '/usr/lib/python3/dist-packages', '/usr/lib/python3.9/dist-packages']

4.6.2 パッケージ

パッケージは、Python モジュールの名前空間を管理する形式で、「ドットモジュール名」を使用します。

例えば、モジュール名が A.B の場合、それはパッケージ A のサブモジュール B を表します。モジュールを使用する際に、異なるモジュール間のグローバル変数が互いに影響しないように心配する必要がないように、ドットモジュール名の形式を使用することで、異なるライブラリ間のモジュール名の重複を心配する必要もありません。

パッケージのインストールとダウンロードは、前の Python インストールセクションを参照してください。

4.7 仮想環境

Python アプリケーションは、通常、標準ライブラリに属さないパッケージやモジュールを使用しますが、特定のバージョンのパッケージやモジュールが必要な場合があります。なぜなら、アプリケーションは特定のバグを修正する必要があったり、使用するライブラリのインターフェースが以前のバージョンであるためです。例えば、アプリケーション A がバージョン 1.0 のモジュールを必要とし、アプリケーション B がバージョン 2.0 を必要とする場合、これらの要求は矛盾し、バージョン 1.0 または 2.0 をインストールすると、1 つのアプリケーションが動作しなくなります。この問題は、仮想環境を作成することで解決できます。

仮想環境は、特定のバージョンの Python 用の Python インストールを含む、自給自足のディレクトリツリーを作成し、多くの追加パッケージを含みます。異なるアプリケーションは、異なる仮想環境を使用し、異なるバージョンのパッケージとモジュールをインストールできます。

Python 3.3 バージョンから、標準の仮想環境 venv が付属しており、venv をインストールして使用し

ます：

```
# 仮想環境のインストール
cat@lubancat:~$ sudo apt-get install python3-venv

# project-test ディレクトリを作成して移動し、独立した Python 実行環境: test_env を作成
cat@lubancat:~$ mkdir project-test && cd project-test
cat@lubancat:~/project-test$ python3 -m venv .test_env

cat@lubancat:~/project-test$ ls -al
total 12
drwxr-xr-x 3 cat cat 4096 Dec 7 10:38 .
drwxr-xr-x 17 cat cat 4096 Dec 7 10:38 ..
drwxr-xr-x 6 cat cat 4096 Dec 7 10:38 .test_env
cat@lubancat:~/project-test$

# 環境をアクティブ化
cat@lubancat:~/project-test$ source .test_env/bin/activate
(.test_env) cat@lubancat:~/project-test$
(.test_env) cat@lubancat:~/project-test$ pip list
Package Version
------------- -------
pip 18.1
pkg-resources 0.0.0
setuptools 40.8.0
(.test_env) cat@lubancat:~/project-test$

# 現在の仮想環境のパッケージとモジュールのバージョンをエクスポート
(.test_env) cat@lubancat:~/project-test$ pip3 freeze > requirements.txt
# 必要なパッケージとモジュールのバージョンをインストール
(.test_env) cat@lubancat:~/project-test$ pip3 install -r requirements.txt

# 環境を終了
(.test_env) cat@lubancat:~/project-test$ deactivate
cat@lubancat:~/project-test$
```

仮想環境に関する詳細は、virtualenv、venv、pipenv というキーワードで検索してください。

4.8 参考文書

詳細とその他の情報については、次を参照してください：

https://docs.python.org/3.8/tutorial/index.html

第 5 章 GPIO の制御

重要：基板の外部インターフェースを制御するには root 権限が必要ですので、このチュートリアルのテストはすべて root ユーザーで Python モジュールをインストールして行います。

本基板上で純粋なソフトウェアタイプの Python アプリケーションを開発することは、PC 上での開発と変わりありません。次の章では、Python を使用して基板上の GPIO、PWM、ADC、I2C、および SPI などの純粋な PC プログラミングではあまり使用されない外部インターフェースを制御する方法に焦点を当てます。

これらの外部インターフェースを制御するために、主に以下の Python パッケージを使用します：

- python3-libgpiod：標準 GPIO libgpiod ライブラリの Python バージョンで、IO 入出力の制御のみをサポートします。
- python-periphery：GPIO、PWM、I2C、SPI、UART などのさまざまなインターフェースの基本的な制御をサポートします。
- Adafruit Blinka：GPIO、PWM、I2C、SPI、UART などをサポートし、一般的なセンサーや OLED 画面のアプリケーション例も含まれています。

これらはそれぞれ特徴があり、すべて試してみる価値があります。

基板の 40 ピン GPIO 図：（参考マニュアル：《AI エッジ基板 CSAIEG358803 ハードウェア仕様書》）

LuBanCat4 ピン図

拡張機能 3	拡張機能 2	拡張機能 1	共通機能	No.	GPIO	物理ピン		GPIO	No.	共通機能	拡張機能 1	拡張機能 2	拡張機能 3
					3.3V	1	2	5V					
			I2C5_SDA_M3	47	GPIO1_B7	3	4	5V					
			I2C5_SCL_M3	46	GPIO1_B6	5	6	GND					
	I2C2_SDA_M4	UART6_RX_M1		32	GPIO1_A0	7	8	GPIO4_A3	131	UART0_TX_M2			
					GND	9	10	GPIO4_A4	132	UART0_RX_M2			
	I2C2_SCL_M4	UART6_TX_M1		33	GPIO1_A1	11	12	GPIO1_D6	62	PWM14_M2	I2C8_SCL_M2		
	PDM1_SDI0_M1	PWM3_IR_M3		39	GPIO1_A7	13	14	GND					
	PDM1_SDI1_M1			40	GPIO1_B0	15	16	GPIO3_C1	113			UART7_RX_M1	SPI1_CLK_M1
					3.3V	17	18	GPIO3_D2	122				UART9_RTSN_M2
	PDM1_SDI3_M1	UART4_RX_M2	SPI0_MOSI_M2	42	GPIO1_B2	19	20	GND					
	PDM1_SDI2_M1		SPI0_MISO_M2	41	GPIO1_B1	21	22	GPIO3_D4	124		UART9_RX_M2		
	PDM1_CLK1_M1	UART4_TX_M2	SPI0_CLK_M2	43	GPIO1_B3	23	24	GPIO1_B4	44	SPI0_CS0_M2	UART7_RX_M2	PDM1_CLK0_M1	
					GND	25	26	GPIO1_B5	45	SPI0_CS1_M2	UART7_TX_M2		
			I2C6_SDA_M3	136	GPIO4_B0	27	28	GPIO4_B1	137	I2C6_SCL_M3			
				102	GPIO3_A6	29	30	GND					
SPI1_MOSI_M1		I2C3_SCL_M1		111	GPIO3_B7	31	32	GPIO1_D7	63	PWM15_IR_M3	I2C8_SDA_M2		
UART9_CTSN_M2			PWM10_M2	123	GPIO3_D3	33	34	GND					
		UART9_TX_M2	PWM11_IR_M3	125	GPIO3_D5	35	36	GPIO4_A0	128		SPI0_MISO_M1		UART9_RTSN_M1
SPI1_MISO_M1	UART7_TX_M1	I2C3_SDA_M1		112	GPIO3_C0	37	38	GPIO4_A1	129		SPI0_MOSI_M1		UART9_CTSN_M1
					GND	39	40	GPIO4_A2	130		SPI0_CLK_M1		

5.1 libgpiod の基本概念

GPIO は主に外部に高低電圧を出力するために使用され、GPIO を制御する際には基本的に libgpiod の制御が関係します。我々は主にボードのピンの命名方法を知っている必要があります。

CPU の GPIO ピンは（chip, line）の方式で命名されており、以下のコマンドを使用して確認できます：

```
# ボード上で以下のコマンドを実行します
sudo gpioinfo
# コマンドが見つからない場合は、以下の方法でインストールします

sudo apt -y install gpiod libgpiod-dev
# 以下は gpioinfo の出力例です
gpiochip0 - 32 lines:
line 0: unnamed unused input active-high
line 1: unnamed unused input active-high
line 2: unnamed unused input active-high
line 3: unnamed unused input active-high
line 4: unnamed unused input active-high
line 5: unnamed "headset_gpio" input active-high [used]
# ...
line 31: unnamed unused input active-high
gpiochip1 - 32 lines:
line 0: unnamed unused input active-high
line 1: unnamed unused input active-high
# ...
line 31: unnamed unused input active-high
```

ボードのピンヘッダと GPIO（chip, line）の対応方法については、ボードの具体的な説明を参照してください。

5.2 実験準備

ボード上の一部の GPIO は他の機能に使用されている可能性があります。その場合、デバイスツリーノードまたはデバイスツリープラグインの読み込みをコメントアウトし、システムを再起動して該当する GPIO ピンを解放します。

ほとんどのハードウェア外部機器を操作する機能には、root ユーザー権限が必要です。簡単な解決策は、コマンドの前に sudo を追加するか、root ユーザーとしてプログラムを実行することです。

5.3 方法一：python3-libgpiod を使用する

5.3.1 python3-libgpiod のインストール

python3-libgpiod パッケージを利用すると、Python を使用して GPIO ピンを簡単に制御できます。現在、python3-libgpiod には公式のドキュメントがありませんが、このパッケージをインポートして help を使用してヘルプを表示できます。インストールとヘルプの表示方法は以下の通りです：

```
# ボード上で以下のコマンドを実行してインストール
sudo apt -y install python3-libgpiod
# Python インタラクティブモードに入り、テストとヘルプの表示
python3
import gpiod
help(gpiod)
# 以下はヘルプの出力例です
NAME
gpiod - Python bindings for libgpiod.
DESCRIPTION
This module wraps the native C API of libgpiod in a set of python
classes.
# ...
```

5.3.2 libgpiod 出力

この python3-libgpiod パッケージを使用して GPIO を制御し、LED を点灯させる（出力）のサンプルコードは以下の通りです：

サンプルコード io/gpio/libgpiod_io0.py

```
#5.3.2 libgpiod 出力

import time
import gpiod

# 具体的なボードの LED ランプの接続に応じて、使用する Chip と Line を変更します。LED がない場合は外部に接続してください。
LINE_OFFSET = 1

chip1 = gpiod.Chip("1", gpiod.Chip.OPEN_BY_NUMBER)

gpio1_a1 = chip1.get_line(LINE_OFFSET)
```

```
gpio1_a1.request(consumer="gpio", type=gpiod.LINE_REQ_DIR_OUT, default_vals=[0])
print(gpio1_a1.consumer())
try:
    while True:
        gpio1_a1.set_value(1)
        time.sleep(0.5)
        print(gpio1_a1.get_value())
        gpio1_a1.set_value(0)
        time.sleep(0.5)
        print(gpio1_a1.get_value())
finally:
    gpio1_a1.set_value(1)
    gpio1_a1.release()

# 出力結果（設定値は 0.5s ごとに変わり、LED を接続されなくてもレベル値を確認すれば検証できる）
gpio
1
0
1
0
1
0
1
0
1
0
1
0
1
0
1
```

具体的なボードの LED ランプの接続に応じて、使用する Chip と Line を変更します。LED がない場合は外部に接続してください。

コードの説明：

- 9 行目では、chip ID が 0 の gpiod.Chip オブジェクト chip1 を作成しています。
- 11 行目では、chip1 オブジェクトの line0 をピンとして設定しています。
- 13 行目では、GPIO を申請し、出力として設定し、デフォルトで低電圧を出力します。

その後のコードでは、`gpio1_a1` オブジェクトを使用して選択された GPIO の高低電圧を制御し、LED ランプの点灯を制御します。

5.3.2.1 操作手順

*ボードに二つ LED しかありません、赤：電源 LED、緑：点滅、システム動作指示 LED、他の LED がないので、40PIN の GPIO から外部 LED を使いましょう。例：GPIO1_A1 を使い、LED の＋が 40 ピンインタフェースのピン 11 に、-がピン 9（GND）を接続するようにします。

GPIO インデックス：（バンク：IO1→1）32×1+（ポート：A→0）8×0+（ピン→1）1＝33

GPIO1_A1 を有効にする

```
#su root
```

#echo 33 > /sys/class/gpio/export

ピンの値を読み込み

#cat /sys/class/gpio/gpio33/value

1

ピンに出力モードを設定

#echo out > /sys/class/gpio/gpio33/direction

ローレベルを設定（現状：ハイレベル→1）

#echo 0 > /sys/class/gpio/gpio33/value

設定結果を確認

#cat /sys/class/gpio/gpio33/value

0

GPIO を開放

#echo 33 > /sys/class/gpio/unexport

上記コマンドで操作しますが、前のページで作成した Python プログラムでも実行しましょう。

実行したら、新しいターミナルで sudo gpioinfo コマンドで確認できます。

問題　出力　デバッグ コンソール　ターミナル　ポート

```
        line  21:      unnamed       "irq"   input  active-high [used]
        line  22:      unnamed      unused   input  active-high
        line  23:      unnamed      unused   input  active-high
        line  24:      unnamed      unused   input  active-high
        line  25:      unnamed      unused   input  active-high
        line  26:      unnamed      unused   input  active-high
        line  27:      unnamed      unused   input  active-high
        line  28:      unnamed      unused   input  active-high
        line  29:      unnamed      unused   input  active-high
        line  30:      unnamed      unused   input  active-high
        line  31:      unnamed      unused   input  active-high
gpiochip1 - 32 lines:
        line   0:      unnamed      unused  output  active-high
        line   1:      unnamed      "gpio"  output  active-high [used]
        line   2:      unnamed      unused   input  active-high
        line   3:      unnamed      unused   input  active-high
        line   4:      unnamed      unused   input  active-high
        line   5:      unnamed      unused   input  active-high
        line   6:      unnamed    "hp-con"  output  active-high [used]
        line   7:      unnamed      unused   input  active-high
        line   8:      unnamed      unused   input  active-high
        line   9:      unnamed      unused   input  active-high
        line  10:      unnamed      unused   input  active-high
        line  11:      unnamed      unused   input  active-high
        line  12:      unnamed      unused   input  active-high
        line  13:      unnamed      unused   input  active-high
        line  14:      unnamed      unused   input  active-high
```

実際外部 LED は点滅しています、GPIO 制御は成功したことを検証できました。

5.3.3 libgpiod 入力出力

同様に、この python3-libgpiod パッケージを使用して GPIO 入力（ボタン）を検出するサンプルコードは以下の通りです：

サンプルコード io/gpio/libgpiod_io1.py

```python
#5.3.3 libgpiod 入力出力

import time
import gpiod

# 具体的なボードの LED とボタンの接続に応じて使用する Chip と Line を変更します
# ここでは GPIO1_B0 に LED、GPIO3_C1 にボタンを接続します
LED_LINE_OFFSET = 8
BUTTON_LINE_OFFSET = 17

chip1_led = gpiod.Chip("1", gpiod.Chip.OPEN_BY_NUMBER)
chip3_button = gpiod.Chip("3", gpiod.Chip.OPEN_BY_NUMBER)
```

```
led = chip1_led.get_line(LED_LINE_OFFSET)
led.request(consumer="LED", type=gpiod.LINE_REQ_DIR_OUT, default_vals=[0])

button = chip3_button.get_line(BUTTON_LINE_OFFSET)
button.request(consumer="BUTTON", type=gpiod.LINE_REQ_DIR_IN)

print(led.consumer())
print(button.consumer())

try:
    while True:
        led.set_value(button.get_value())
        print("BUTTON INPUT:" + str(button.get_value()))
        print("LED OUT:" + str(led.get_value()))
        time.sleep(1)                                    #added in order to print,should be deleted in production
finally:
    led.set_value(1)
    led.release()
    button.release()



# 出力結果（設定値は 0.5s ごとに変わる）
```

コードの説明：

- 15 行目では、LED の GPIO 制御方向を出力として設定しています。
- 18 行目では、ボタンの GPIO 制御方向を入力として設定しています。
- 25 行目では、ボタンピンの入力値を読み取り、LED を制御します。

5.4 方法二：python-periphery を使用する

5.4.1 python-periphery のインストール

python-periphery は python3-libgpiod と似た機能を持っていますが、periphery は GPIO 入出力制御の他に、I2C、SPI などのバスプロトコルもサポートしています。

python-periphery のインストール方法は以下の通りです：

```
# ボード上で以下のコマンドを実行してインストール
sudo pip3 install python-periphery
```

5.4.2 periphery 入力出力

この python-periphery を使用して入出力を行うサンプルコードは以下の通りです：

サンプルコード io/gpio/periphery_io.py

```
#5.4.2 periphery 入力出力

import time
from periphery import GPIO

LED_CHIP = "/dev/gpiochip0"
LED_LINE_OFFSET = 8

BUTTON_CHIP = "/dev/gpiochip3"
```

```
BUTTON_LINE_OFFSET = 17

led = GPIO(LED_CHIP, LED_LINE_OFFSET, "out")
button = GPIO(BUTTON_CHIP, BUTTON_LINE_OFFSET, "in")
try:
    while True:
        led.write(button.read())
        print("BUTTON INPUT:" + str(button.read()))
        print("LED OUT:" + str(led.read()))
        time.sleep(1)                              #added in order to print,should be deleted in production
finally:
    led.write(True)
    led.close()
    button.close()


# 出力結果（設定値は 0.5s ごとに変わる）
```

（次ページに続く）

コードの説明：
- 6~10 行目では、LED とボタンの chip と line 番号を定義しています。
- 12~13 行目では、それぞれ led と button の GPIO 出力、入力オブジェクトを作成しています。
- 16 行目では、ボタンピンの入力値を読み取り、LED を制御します。

実験操作は前のセクションと同様です。別のターミナルを使用して gpioinfo コマンドを実行すると、2 つのピンが使用中であることが確認できます：

```
cat@lubancat:~/python/code$ sudo gpioinfo
gpiochip0 - 32 lines:
        line   0:      unnamed       unused   input  active-high
        line   1:      unnamed       unused   input  active-high
        line   2:      unnamed       unused   input  active-high
        line   3:      unnamed       unused   input  active-high
        line   4:      unnamed       unused   input  active-high
        line   5:      unnamed       unused   input  active-high
        line   6:      unnamed       unused   input  active-high
        line   7:      unnamed       unused   input  active-high
        line   8:      unnamed  "periphery"  output  active-high [used]
        line   9:      unnamed       unused   input  active-high
        line  10:      unnamed       unused   input  active-high
        line  11:      unnamed       unused   input  active-high
        line  12:      unnamed       unused   input  active-high
        line  13:      unnamed       unused   input  active-high
        line  14:      unnamed       unused   input  active-high
        line  15:      unnamed      "reset"  output   active-low [used]
        line  16:      unnamed      "reset"   input   active-low [used]
```

```
gpiochip3 - 32 lines:
        line   0:      unnamed       unused   input  active-high
        line   1:      unnamed       unused   input  active-high
        line   2:      unnamed       unused   input  active-high
        line   3:      unnamed       unused   input  active-high
        line   4:      unnamed       unused   input  active-high
        line   5:      unnamed       unused   input  active-high
        line   6:      unnamed       unused   input  active-high
        line   7:      unnamed       unused   input  active-high
        line   8:      unnamed       unused   input  active-high
        line   9:      unnamed       unused   input  active-high
        line  10:      unnamed "snps,reset"  output   active-low [used]
        line  11:      unnamed       unused   input  active-high
        line  12:      unnamed       unused   input  active-high
        line  13:      unnamed       unused   input  active-high
        line  14:      unnamed       unused   input  active-high
        line  15:      unnamed       unused   input  active-high
        line  16:      unnamed       unused   input  active-high
        line  17:      unnamed  "periphery"   input  active-high [used]
        line  18:      unnamed       unused   input  active-high
        line  19:      unnamed       unused   input  active-high
        line  20:      unnamed       unused   input  active-high
```

5.5 方法三：Adafruit Blinka を使用する

5.5.1 Adafruit Blinka のインストール

Adafruit Blinka は python-periphery と似た機能を持ち、GPIO 入出力制御の他に、I2C、SPI などのバスプロトコルもサポートしていますが、Adafruit は blinka を基に、画面の制御などの豊富なアプリケーションデモも提供しています。

Adafruit Blinka のインストール方法は以下の通りです：

```
# ボード上で以下のコマンドを実行してインストール
sudo apt -y install python3-libgpiod
# 注：方法一で既にインストールしている場合、このステップは省略します。
sudo pip3 install Adafruit-Blinka
```

5.5.2 使用前に必ずお読みください

LubanCat RK シリーズのボードはすでに Adafruit-Blinka ライブラリに適合していますが、ボードによって Adafruit-Blinka ライブラリで利用できるリソースが異なります。手元の LubanCat ボードに応じて、対応するサンプルプログラムで使用されているリソース名を変更すれば、対応する実験を完了できます。

注意すべき点として、いくつかのボードはリソースが限られているため、Adafruit-Blinka ライブラリの機能がすべてのボードでサポートされているわけではありません。自分のボードが特定の機能をサポートしているかどうかを知りたい場合は、ボードのリソース定義を確認してください。以下の手順を参考にしてください：

確認前に Adafruit-Blinka ライブラリがインストールされており、ボードが検出可能であることを確認します。

1. python3 を入力して sudo python3 ターミナルに入ります。
2. `from adafruit_platformdetect import Detector` を入力して adafruit_platformdetect パッケージを追加します。
3. `detector = Detector()` を入力し、続いて `print("Chip id: ", detector.chip.id)` と `print("Board id: ", detector.board.id)` を入力してボードとチップの ID を検出します。
4. python3 ターミナルで `import board` を入力して board パッケージをインポートします。
5. `board.` と入力して Tab キーを 2 回押すと、補完情報が表示され、ボードに定義されているリソースの内容が表示されます。

```
cat@lubancat:~/python/code/io/gpio$ sudo python3
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.
>>> import board
>>> board.
  File "<stdin>", line 1
    board.
          ^
SyntaxError: invalid syntax
>>> from adafruit_platformdetect import Detector
>>> detector = Detector()
>>> print("Chip id: ", detector.chip.id)
Chip id:  RK3588
>>> print("Board id: ", detector.board.id)
Board id:  LUBANCAT4
>>> board.
board.CS0       board.GPIO13    board.GPIO21    board.GPIO27    board.GPIO32    board.GPIO38    board.I2C5_SCL  board.MOSI      board.SCLK      board.board_id
board.CS1       board.GPIO15    board.GPIO22    board.GPIO28    board.GPIO33    board.GPIO40    board.I2C5_SDA  board.PWM10     board.SPI(      board.detector
board.GPIO10    board.GPIO16    board.GPIO23    board.GPIO29    board.GPIO35    board.GPIO5     board.I2C6_SCL  board.PWM11     board.UART0_RX  board.pin
board.GPIO11    board.GPIO18    board.GPIO24    board.GPIO3     board.GPIO36    board.GPIO7     board.I2C6_SDA  board.PWM14     board.UART0_TX  board.sys
board.GPIO12    board.GPIO19    board.GPIO26    board.GPIO31    board.GPIO37    board.GPIO8     board.MISO      board.PWM15     board.ap_board
>>>
```

注意：ここで表示されるリソースは、Adafruit-Blinka ライブラリに適合させるために追加されたリソースにすぎません。具体的なリソースが利用可能かどうかは、ボードの状況によります。ユーザーは自分で試して確認してください。

ヒント：追加されたボードリソースは 40 ピンの引き出しピンで、第何番目のピンが第何番目の GPIO に対応するかのルールは、物理的なピンが接地または電源でない限り適用されます。例えば、GPIO3 は 40 ピンの第 3 番目の物理ピンに対応しますが、物理ピンが接地または電源の場合は対応しません。

5.5.3 Adafruit-Blinka 入力出力

この Adafruit Blinka を使用して入出力を行うサンプルコードは以下の通りです：

サンプルコード io/gpio/digital_io.py の内容

```
#5.5.3 Adafruit-Blinka 入力出力

import board
import digitalio
import time

# 具体的なボードに応じて使用する GPIO を変更します
# ここでは LubanCat 4 ボードを例に、board.GPIO11 は 40Ppin の第 11 番目の物理ピンです

#board.GPIO11 = GPIO1_A1 = Pin 33
led = digitalio.DigitalInOut(board.GPIO11)
led.direction = digitalio.Direction.OUTPUT

try:
```

```
    while True:
        led.value=1
        time.sleep(0.5)
        led.value=0
        time.sleep(0.5)
finally:
    led.value = True
    led.deinit()



# 出力結果（設定値は 0.5s ごとに変わる）
```

コードの説明：
- 3~4 行目では、board と digitalio は Blinka パッケージが提供するライブラリです。
- 10 行目では、使用するピンを board.GPIO11 として定義し、出力方向に設定しています。
- 14~19 行目では、ピンを高低電圧に設定します。

コード内の board.GPIO11 ピンは、Blinka ライブラリで定義されている LubanCat 4 のピンです。以下のコマンドを使用します：

sudo python3 digital_io.py

対応するピンに LED ランプを接続している場合、LED が点滅するのが確認できます。

ボードのリソースを申請するとき、ボードのモデルが検出され、ボード情報に基づいて対応するリソース定義がロードされます。そのため、定義されたリソース名を直接使用することができ、python-periphery よりも直感的に使用できます。

より詳細なピン定義は、Adafruit Blinka のソースコードを直接参照してください：
- Adafruit Blinka における LubanCat シリーズボードのピン定義
- Adafruit Blinka における LubanCat RK シリーズチップのピン定義

5.6 参考資料

実際には、他にも多くの Python ライブラリが外部ハードウェアの制御をサポートしています。興味があれば、github や pypi で検索してみてください。
- 《CircuitPython》
- python-periphery ソースコード
- Adafruit Blinka ソースコード

第 6 章 PWM 出力

前の章では、python3-libgpiod や python-periphery などの Python ライブラリを使用して、GPIO 外部機器を操作する Python アプリケーションを作成し、LubanCat ボード上で Python アプリケーションを通じてボード上の LED を点灯させ、ボタンリソースを使用しました。本節では、これらのライブラリが提供するもう一つの機能である PWM 波形出力について紹介します。

本節の実験では、LED の GPIO ピンを使用して PWM 出力を行います。LED の明暗変化を通じて実験結果を確認できます（ボードに外部 LED を接続する必要がある）。

6.1 PWM 実験

GPIO ピンの高低電圧の変化を制御することで、LED の点灯や消灯などの簡単な制御ができます。さらに GPIO ピンの電圧変化の周波数や持続時間を調整することで、制御はさらに複雑になります。これが PWM（パルス幅変調）です。

通常、PWM 波形出力を利用して、LED の明るさの調整、無線ブザーの音量調整、さらには小型サーボの制御など多くの機能を実現できます。本節の実験では、PWM を出力して LED を制御し、LED の呼吸灯効果を実現します（LubanCat 4 ボードを例に、外部 LED を接続します）。

PWM がどのように機能するかについては詳しくは触れませんが、PWM の原理については各自で調べてください。本節の重点は、PWM 出力の例を提供することです。もちろん、これは前述の Python ライブラリを使用して行います。

重要: 特に記載がない限り、本書のチュートリアルは Python 3.9.2 バージョン（extboot パーティションの Debian11 イメージ）に基づいて実験および解説を行います。

6.2 実験準備

ボード上の一部の GPIO はシステムによって使用されている場合があります。その場合、特定のデバイスツリーノードまたはデバイスツリープラグインのロードをコメントアウトし、システムを再起動して該当する GPIO ピンを解放します。また、他のピンをテストすることもできます。LubanCat_RK シリーズボードのピンは、次の図を参照してください：「LubanCat-RK シリーズ-40pin ピン配置図」

「Permission denied」などのエラーメッセージが表示された場合は、ユーザー権限に注意してください。ほとんどのハードウェア外部機器を操作する機能には root ユーザー権限が必要です。簡単な解決策として、コマンドの前に sudo を追加するか、root ユーザーとしてプログラムを実行してください。

PWM デバイスツリープラグインの有効化方法（LubanCat 4 を例に）：

```
# ボードごとに設定ファイルと PWM が異なります。
  LubanCat 4 を例に、設定ファイルは uEnv.txt です。
# pwm3、pwm10、pwm11、pwm14、pwm15 を有効にできます。
# システムターミナルにログインし、/boot/uEnv/uEnv.txt ファイルを開きます。
sudo nano /boot/uEnv/uEnv.txt
```

pwm の前のコメントを解除し、ファイルを保存して終了し、システムを再起動します。

ターミナルで以下のコマンドを入力して、現在のボードの PWM リソースを確認します：

ls /sys/class/pwm/

pwmchip0 pwmchip1 pwmchip2 pwmchip3 pwmchip4 pwmchip5 pwmchip6 pwmchip7

チップ番号と PWM の関係：

pwmchip0 PWM2 裏面 MIPI DSI

pwmchip1 PWM6 正面 MIPI DSI

pwmchip2 PWM3_IR_M2

pwmchip3 PWM7 赤外線受信用

pwmchip4 PWM10_M2

pwmchip5 PWM11_IR_M3

pwmchip6 PWM14_M2

pwmchip7 PWM15_IR_M3

ボード上の PWM リソースの例は以下の通りです。参考にしてください：

pwmchip0 が pwm2 、pwmchip1 が pwm6 で、バックライト調整に使用されます。pwmchip3 は、赤外線受信用なので、ハードウェア上の pwm7 に対応します。同様に進めていきます。

6.3 python-periphery を使用する

python-periphery ライブラリの PWM 出力は、Linux の PWM サブシステムに基づいて実装されています。このため、PWM 出力を行うにはボードがサポートしている必要があります。LubanCat ボードは、python-periphery ライブラリを使用して PWM 出力を完璧に実現できます。このように、ソフトウェア層で GPIO を使用して PWM 出力をシミュレートする必要はありません。

PWM サブシステムに関する詳細は下記通りです。(Shell でコマンドより制御)

#外部 LED を PWM3 で接続

GPIO1_A7 を使い、LED の＋が 40 ピンインタフェースのピン 13 に、

-がピン 9（GND）を接続するようにします。

#su root

パスワード：root

pwm3 をユーザースペースにエクスポート

```
echo 0 > /sys/class/pwm/pwmchip2/export

# pwm 周期を設定 単位は ns
echo 1000000 > /sys/class/pwm/pwmchip2/pwm0/period

# デューティサイクルを設定
echo 500000 > /sys/class/pwm/pwmchip2/pwm0/duty_cycle

# pwm の極性を設定
echo "normal" > /sys/class/pwm/pwmchip2/pwm0/polarity

# pwm を有効化
echo 1 > /sys/class/pwm/pwmchip2/pwm0/enable

# pwm3 をユーザースペースからアンエクスポート
echo 0 > /sys/class/pwm/pwmchip1/unexport
```

6.3.1 python-periphery のインストール

重要: 前のセクションでこのライブラリをインストールした場合は、この操作をスキップしてください。
python-periphery のインストール方法は以下の通りです:

```
# ボード上で以下のコマンドを実行してインストール
sudo pip3 install python-periphery
```

6.3.2 periphery で PWM を出力

python-periphery ライブラリを使用して PWM を出力するサンプルコードは以下の通りです：

サンプルコード io/pwm/pwm_test_periphery.py の内容

```
#6.3.2 periphery で PWM を出力

from periphery import PWM
import time


try:
    # デューティサイクルの増加ステップを定義
    step = 0.05
    # 最大範囲を定義
    rangeMax = int(1/0.05)
    # PWM 10, チャンネル 0 を開く。開発ボード上の PWM10、GPIO インタフェース兼用の GPIO3_D3 ピン 33 に対応、チップは/sys/class/pwm/pwmchip4
    pwm = PWM(4, 0)
    # PWM 出力周波数を 1 kHz に設定
    pwm.frequency = 1e3
    # デューティサイクルを 0% に設定。1 サイクル内の高電圧時間の全周期に対する比率
    pwm.duty_cycle = 0.00
    # PWM 出力を有効にする
    pwm.enable()
    while True:
        for i in range(0,rangeMax):
            # step 秒間休止
            time.sleep(step)
            # デューティサイクルを毎回 step% 増加。浮動小数点演算誤差を避けるために round を使用
            pwm.duty_cycle = round(pwm.duty_cycle+step,2)
        # 完全に消灯 1 秒
        if pwm.duty_cycle == 0.0:
            time.sleep(1)
        for i in range(0,rangeMax):
            time.sleep(step)
            pwm.duty_cycle = round(pwm.duty_cycle-step,2)
except:
    print("Some errors occur!\n")
finally:
    # 終了時に LED を消灯
    pwm.duty_cycle = 0.0
    # リソースを解放
    pwm.close()
```

コードの説明：

- 12 行目では、パラメータがチップ番号、チャネルにより PWM オブジェクトを作成しています。
- 14、16 行目では、PWM オブジェクトのパラメータを初期化しています。これには PWM 出力波形の周波数とデューティサイクルが含まれます。
- 18 行目では、対応するパラメータに基づいて PWM 波形出力を有効にしています。

ヒント：PWM の Python モジュールの使用方法に不慣れな場合は、Python ターミナルのインタラクティブモードに入り、from periphery import PWM と help(PWM) コマンドを使用して PWM モジュールに関するヘルプを参照してください。

6.3.2.1 実験操作

サンプルコードは LubanCat 4 ボードを使用しており、操作は以下の通りです：

```
# python-periphery パッケージがインストールされていることを確認します。
# PWM デバイスツリープラグインを有効にし、有効にした PWM を確認します。
# ボード上の pwm_test.py があるディレクトリで以下のコマンドを実行します。
sudo python3 pwm_test_periphery.py
```

権限の問題についてプロンプトが表示された場合は、root ユーザーに切り替えて次を実行できます。

```
# root に切り替えます。これはパスワード root です
スールート
```

外部 LED が接続されている場合は、オンボード LED ライトの明暗の変化を確認したり、オシロスコープで波形を表示したりできます。

第 7 章 UART 通信

本章では、一般的な通信プロトコルである UART について紹介します。LubanCat ボードでは、しばしば端末を使用してボードとやり取りを行いますが、これらの接続方法の中で、UART シリアル通信は最も頻繁に使用される方法の一つです。

ここでは UART プロトコル自体の詳細には触れません。UART 通信プロトコルについては各自で調べて学習してください。本章では、Python ライブラリを使用してボードのシリアルポートリソースを呼び出すことに重点を置きます。ハードウェア側では、ボードと USB-to-TTL モジュールを使用してコンピュータに接続し、シリアルポート上位機で実験の現象をデモンストレーションおよび操作します。

重要: 特に明記されていない限り、本書のチュートリアルは Python 3.9.2 バージョン（extboot パーティション Debian11 イメージ）に基づいて実験および解説を行っています。

7.1 実験準備

7.1.1 UART リソースの追加

ボード上の一部の GPIO は有効になっていない可能性があります。デバイスツリーノードまたはデバイスツリープラグインの読み込みをコメントアウトして、システムを再起動して有効にします。基板の 40 ピン GPIO 図：（参考マニュアル：《AI エッジ基板 CSAIEG358803 ハードウェア仕様書》）

筆者が使用している LubanCat 4 ボードの場合、引き出された 40 ピンのピン配置図に基づいて、シリアルポートリソースは UART0 を使用できます。以下に UART0 のデバイスツリープラグインを有効にする手順を示します。

ピン８：TX；
ピン１０：RX；
ピン１４：GND

```bash
# ボードごとに設定ファイルと PWM は異なります。LubanCat 4 を例に取ると、設定ファイルは uEnvLubanCat4.txt です。
# 以下のコマンドを使用します：
sudo nano /boot/uEnv/uEnvLubanCat4.txt
# uart0 の前のコメントを外して、ファイルを保存して終了し、システムを再起動します。
```


```bash
# ターミナルで以下のコマンドを入力して UART リソースを確認できます：
ls /dev/ttyS*
```

ボード上の UART リソースの例は以下の通りです（参考）。

```bash
/dev/ttyS0
```

ここで、`/dev/ttyS0`がボード上の uart0 リソースに対応しています。

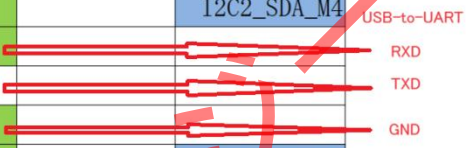
7.1.2 ハードウェアの接続

注：ハードウェア接続のセクションは、自身の開発環境などに応じて調整して参考にしてください。

本章では、上述の Python ライブラリを使用して LubanCat 4 ボード上の UART0 リソースを利用するコードを作成し、USB-to-TTL モジュールとシリアルポートホスト PC を使用してコードの正確性をテストします。

筆者のハードウェア接続は以下の図を参考にしてください：

Pin	GPIO	番号	説明	PWM	UART	SPI	I2C/PDM
1	3V3	-	3.3V				
2	5V	-	5V				
3	GPIO1_B7	47	GPIO、I2C				I2C5_SDA_M3
4	5V	-	5V				
5	GPIO1_B6	46	GPIO、I2C				I2C5_SCL_M3
6	GND	-	GND				
7	GPIO 1_A0	32	GPIO、UART、I2C		UART6_RX_M1		I2C2_SDA_M4
8	GPIO4_A3	131	GPIO、UART		UART0_TX_M2		
9	GND	-	GND				
10	GPIO4_A4	132	GPIO、UART		UART0_RX_M2		
11	GPIO 1_A1	33	GPIO、UART、I2C		UART6_TX_M1		I2C2_SCL_M4
12	GPIO1_D6	62	GPIO、PWM、I2C	PWM14_M2			I2C8_SCL_M2
13	GPIO1_A7	39	GPIO、PWM、PDM	PWM3_IR_M2			PDM1_SDI0_M1
14	GND	-	GND				
15	GPIO 1_B0	40	GPIO、PDM				PDM1_SDI1_M1
16	GPIO 3_C1	113	GPIO、UART、SPI		UART7_RX_M1	SPI1_CLK_M1	
17	3V3	-	3.3V				
18	GPIO3_D2	122	GPIO、UART		UART9_RTSN_M2		
19	GPIO1_B2	42	GPIO、UART、SPI、PDM		UART4_RX_M2	SPI0_MOSI_M2	PDM1_SDI3_M1
20	GND	-	GND				



USB-to-TTL モジュールの接続は上図の通りです。

7.2 方法 1：pyserial ライブラリの使用

pyserial ライブラリはシリアルポートリソースへのアクセス方法をカプセル化しており、このライブラリは複数のプラットフォームでシリアルポートリソースを使用できるようにします。プラットフォーム固有の機能に関する多くの方法があります。公式の使用説明書は pyserial ライブラリに置き換えて参照してください。

Linux システムでは、UART サブシステムに関連する内容については『シリアル通信』を参照してください。

7.2.1 pyserial のインストール

重要：前のセクションの実験で既にこのライブラリをインストールしている可能性があります。既にインストールしている場合はこの操作をスキップしてください。

pyserial のインストール方法は以下の通りです：

```bash
```

```
# ボードで以下のコマンドを使用してインストール
sudo pip3 install pyserial
```

7.2.2 pyserial の使用

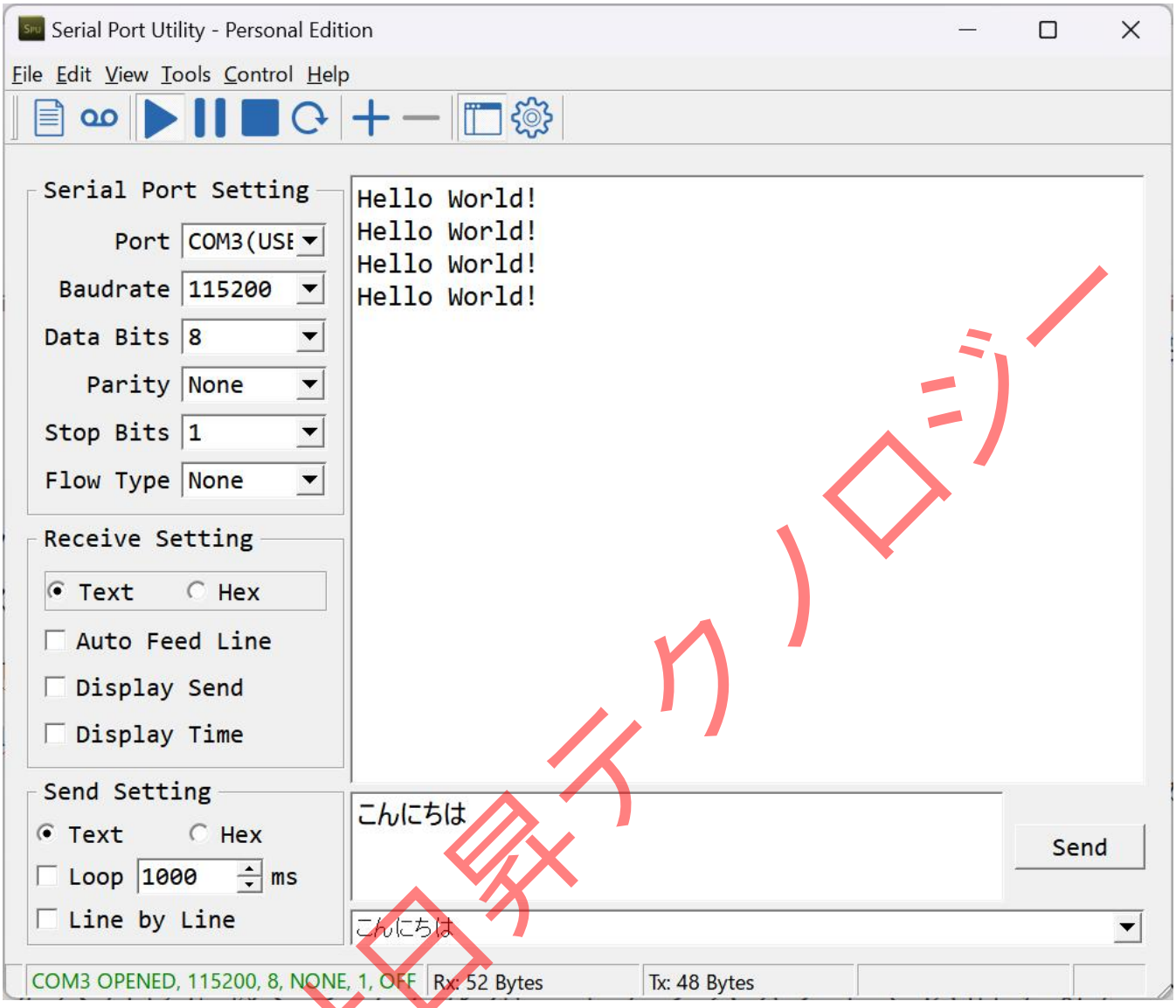
サンプルコード：io/uart/uart_test0.py の内容

```
#7.2.2 pyserial の使用
""" pyserial uart テスト """
import serial

# uart0 を開き、ボーレートを 115200、データビットを 8、パリティなし、ストップビットを 1 に設定し、フロー制御を使用せず、
#ノンブロッキングモードでシリアルポートを開き、タイムアウトを 3 秒に設定
with serial.Serial(
    "/dev/ttyS0",
    baudrate=115200,
    bytesize=serial.EIGHTBITS,
    stopbits=serial.STOPBITS_ONE,
    parity=serial.PARITY_NONE,
    timeout=3,
) as uart0:

    # 申請したシリアルポートを使用してバイトストリームデータ"Hello World!\n"を送信
    uart0.write(b"Hello World!\n")

    # ノンブロッキングモードで開いたシリアルポートでデータを読み取る場合、この関数は 128 バイトを読み取るか、読み取り時間が 1 秒を超えると終了
    buf = uart0.read(128)

    # 注:Python で読み取ったデータのタイプは bytes です
    # 元のデータをプリント
    print("元のデータ:\n", buf)
    # shift_jis 文字列にデコードし、日本語を表示できるようにする
    data_strings = buf.decode("shift_jis")
    # 読み取ったデータ量とデータ内容をプリント
    print("読み取った{:d}バイトのデータを文字列形式でプリント:\n {:s}".format(len(buf), data_strings))
```

7.2.2.1 実験操作

ハードウェア接続時に、uart0 に対応する TX/RX インターフェースを USB-to-TTL モジュールの RXD/TXD インターフェースに接続したため、実験現象を確認するには、シリアルポートホスト PC を使用してデータを表示し、実験現象を観察します。

本資料を作成する際に、下記のシリアルツールを使用しています。
https://www.dragonwake.com/download/LubanCat4/6-tools.zip
上記圧縮パッケージに「serial_port_utility_538_1001.exe」というツールを含まれます。



上図のように、シリアルポートアシスタントでシリアルポートの動作モードをプログラム設定と一致させ、対応するポートを開きます。次に、プログラムを実行します。

サンプルコードは日昇 CSAIEG358803 基板（Lubancat4）を使用しており、操作は以下の通りです：

```bash
# pyserial パッケージがインストールされていることを確認
# UART デバイスツリープラグインが有効になっていることを確認
# ボードの uart_test0.py があるディレクトリで以下のコマンドを実行
sudo python3 uart_test.py
# プログラムで設定されたデータ：Hello World!¥n がシリアルポートアシスタントに表示されます
```

端末でコマンドを入力した後、シリアルポートアシスタントで送信するデータを編集し、送信します。

待機タイムアウト後、端末に受信したデータ：「こんにちは」が表示されます。

```
cat@lubancat:~/python/code/io$ python3 uart_test0.py
元のデータ:
 b'\x82\xb1\x82\xf1\x82\xc9\x82\xbf\x82\xcd\r\n'
読み取った12バイトのデータを文字列形式でプリント:
 こんにちは
```

注：待機時間は 3 秒しかないので、シリアルポートアシスタントで早く送信しないと、基板側の Python プログラムは終了します。

このライブラリについての詳細な使用方法は、pyserial API を参照してください。

7.3 方法 2：python-periphery の使用

python-periphery ライブラリがサポートする UART 機能は、Linux の UART システムに基づいて実装されています。このライブラリを使用するためには、ボードがサポートを提供する必要があります。LubanCat ボードは、python-periphery ライブラリの UART 通信機能を完璧に使用できます。

7.3.1 python-periphery のインストール

重要：前のセクションでこのライブラリをインストールしている場合は、この操作を必ずスキップしてください。

python-periphery のインストール方法は以下の通りです：

```bash
# ボードで以下のコマンドを使用してインストール
sudo pip3 install python-periphery
```

7.3.2 periphery の UART 機能の使用

python-periphery ライブラリを使用した UART 機能のサンプルコードは以下の通りです：

サンプルコード：io/uart/uart_test1.py の内容

```
#7.3.2 periphery の UART 機能の使用
""" python-periphery uart テスト """
from periphery import Serial

try:
    # シリアルポートリソース/dev/ttyS3 を申請し、ボーレートを 115200、データビットを 8、パリティなし、ストップビットを 1 に設定し、フロー制御を使用しない
    serial = Serial(
        "/dev/ttyS0",
        baudrate=115200,
        databits=8,
        parity="none",
        stopbits=1,
        xonxoff=False,
        rtscts=False,
    )
    # 申請したシリアルポートを使用してバイトストリームデータ"python-periphery!\n"を送信
    serial.write(b"python-periphery!\n")

    # シリアルポートでデータを読み取り、この関数は 128 バイトを読み取るか、読み取り時間が 3 秒を超えると終了
    buf = serial.read(128, 3)

    # 注:Python で読み取ったデータのタイプは bytes です
    # 元のデータをプリント
    print("受信した元のデータ:\n", buf)

    # shift_jis 文字列にデコードし、日本語を表示できるようにする
    data_strings = buf.decode("shift_jis")

    # 読み取ったデータ量とデータ内容をプリント
    print("読み取った{:d}バイトのデータを文字列形式でプリント:\n {:s}".format(len(buf), data_strings))
finally:
    # 申請したシリアルポートリソースを解放
    serial.close()
```

コードの説明：

- 6 行目：UART リソースを申請し、uart3 を占有し、対応する動作モードを設定します。
- 16 行目：申請したシリアルポートリソースを使用してデータを送信します。
- 19 行目：申請したシリアルポートリソースを使用してデータを受信し、受信方法はブロッキング受信です。
- 23～29 行目：読み取ったデータを印刷します。
- 32 行目：UART リソースを解放し、uart3 を解放します。

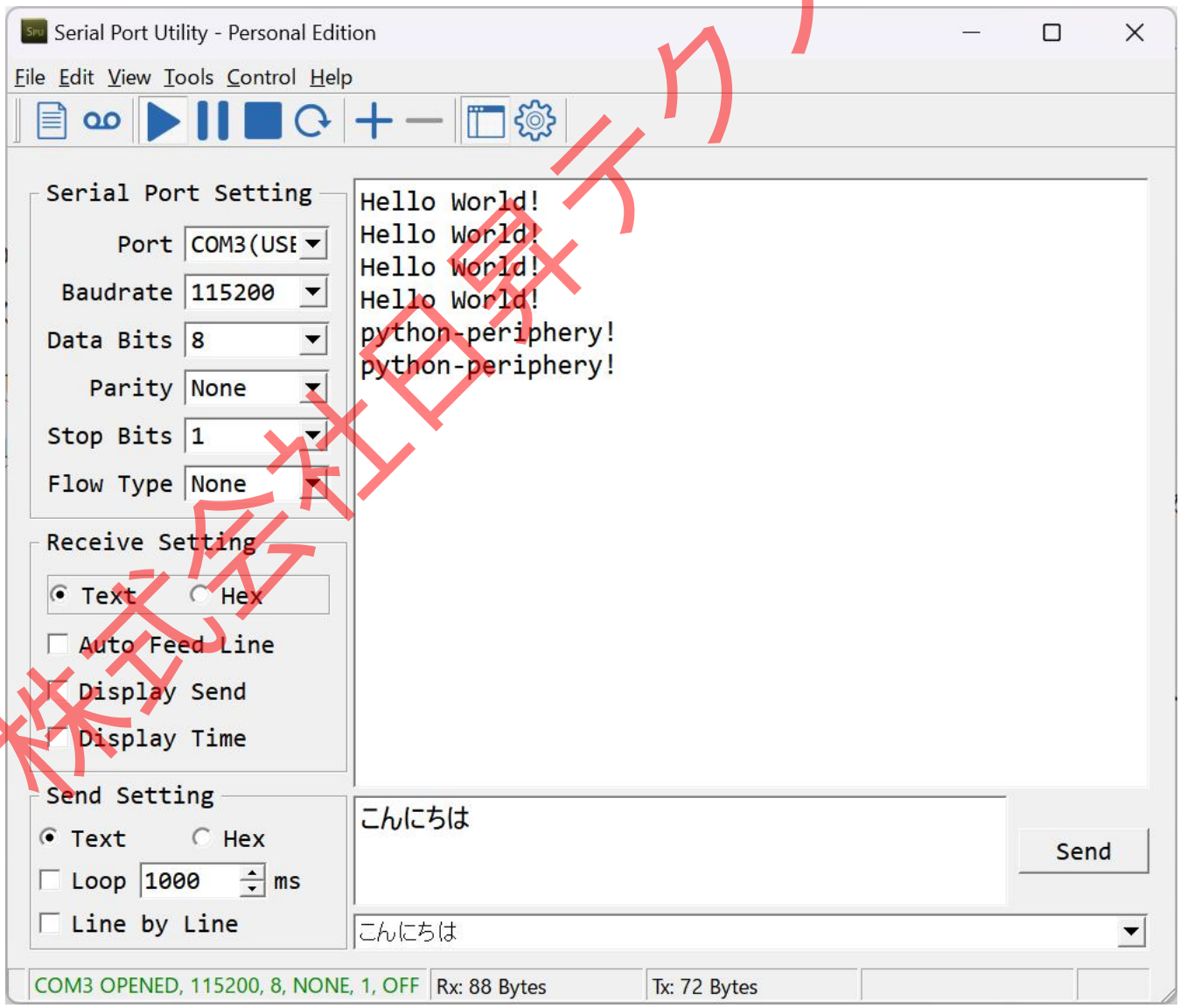
7.3.2.1 実験操作

実験操作は上記と同じです。

サンプルコードは LubanCat4 ボードを使用しており、操作は以下の通りです：

```bash
# python-periphery パッケージがインストールされていることを確認
# UART デバイスツリープラグインが有効になっていることを確認
# ボードの uart_test1.py があるディレクトリで以下のコマンドを実行
sudo python3 uart_test1.py
# プログラムで設定されたデータ：python-periphery!¥n がシリアルポートアシスタントに表示されます
```

端末でコマンドを入力した後、シリアルポートアシスタントで送信するデータを編集し、送信します。



待機タイムアウト後、端末に受信したデータ：こんにちはが表示されます。

```
cat@lubancat:~/python/code/io/uart$ python3 uart_test1.py
受信した元のデータ:
 b'\x82\xb1\x82\xf1\x82\xc9\x82\xbf\x82\xcd\r\n'
読み取った12バイトのデータを文字列形式でプリント:
 こんにちは
```

これで、実験は完了です。

python-periphery ライブラリの UART 機能の詳細な使用方法については、Periphery UART を参照してください。

第 8 章 I2C 通信

I2C（IIC）プロトコルは電子デバイスでよく使用される通信プロトコルであり、これを使用してさまざまな電子機器を制御することができます（注：通常、制御されるデバイスはスレーブです）。例えば、一般的な姿勢センサー MPU6050、温度イメージセンサー MLX90640、0.96 インチ OLED ディスプレイ SSD1306 などは、I2C 通信プロトコルを使用してボード（注：通常、制御を開始するのはマスター）と通信します。

I2C プロトコルの具体的な内容についてはここでは詳しく説明しませんが、I2C 通信プロトコルに関する知識を自分で調べて学んでください。

本章では、前述の python-periphery および Adafruit Blinka の 2 つの Python ライブラリを使用して、日昇 CSAIEG358803(LubanCat4 ボード)上の I2C マスターを使用することに焦点を当てます。

8.1 I2C 実験

一般的には、特定の通信対象を持ってこの実験を完了する必要があります。前述のさまざまなタイプの電子機器などが該当します。

読者の手元のモジュールや I2C デバイスが異なることを考慮し、上記の Python ライブラリの使用紹介は 2 つの部分に分かれます。一つはライブラリの公式デモの紹介、もう一つは実際のモジュールでの API 使用の紹介です。

著者は 0.96 インチ OLED ディスプレイモジュール SSD1306 を使用して、上記のライブラリの使用例を作成しました。実際の使用では、LubanCat4 ボード上の I2C マスターが通信できるデバイスタイプに制限はありません。そのため、著者はコード内に関連するコードの詳細なコメントを付けます。同じモジュールを持っていない場合でも、同様の操作で異なるモジュールに対して実験コードを作成することができます。

*手元に 0.96 インチ OLED ディスプレイモジュール SSD1306 がない場合、弊社通販サイトから購入頂ければと思います。

小型 OLED 液晶モジュール（0.96inch、SPI シリアル通信式）（csun.co.jp)

8.2 実験準備

8.2.1 ボードに I2C リソースを追加する

ボード上の GPIO が I2C として再利用されていない場合、デバイスツリープラグインを設定してロードすることができます。

基板の 40 ピン GPIO 図：（参考マニュアル：《AI エッジ基板 CSAIEG358803 ハードウエア仕様書》）

LuBanCat4 ピン図

拡張機能3	拡張機能2	拡張機能1	共通機能	No.	GPIO	物理ピン		GPIO	No.	共通機能	拡張機能1	拡張機能2	拡張機能3
					3.3V	1	2	5V					
			I2C5_SDA_M3	47	GPIO1_B7	3	4	5V					
			I2C5_SCL_M3	46	GPIO1_B6	5	6	GND					
	I2C2_SDA_M4	UART6_RX_M1		32	GPIO1_A0	7	8	GPIO4_A3	131	UART0_TX_M2			
					GND	9	10	GPIO4_A4	132	UART0_RX_M2			
	I2C2_SCL_M4	UART6_TX_M1		33	GPIO1_A1	11	12	GPIO1_D6	62	PWM14_M2	I2C8_SCL_M2		
	PDM1_SDI0_M1	PWM3_IR_M3		39	GPIO1_A7	13	14	GND					
	PDM1_SDI1_M1			40	GPIO1_B0	15	16	GPIO3_C1	113			UART7_RX_M1	SPI1_CLK_M1
					3.3V	17	18	GPIO3_D2	122				UART9_RTSN_M2
	PDM1_SDI3_M1	UART4_RX_M2	SPI0_MOSI_M2	42	GPIO1_B2	19	20	GND					
	PDM1_SDI2_M1		SPI0_MISO_M2	41	GPIO1_B1	21	22	GPIO3_D4	124		UART9_RX_M2		
	PDM1_CLK1_M1	UART4_TX_M2	SPI0_CLK_M2	43	GPIO1_B3	23	24	GPIO1_B4	44	SPI0_CS0_M2	UART7_RX_M2	PDM1_CLK0_M1	
					GND	25	26	GPIO1_B5	45	SPI0_CS1_M2	UART7_TX_M2		
			I2C6_SDA_M3	136	GPIO4_B0	27	28	GPIO4_B1	137	I2C6_SCL_M3			
				102	GPIO3_A6	29	30	GND					
SPI1_MOSI_M1		I2C3_SCL_M1		111	GPIO3_B7	31	32	GPIO1_D7	63	PWM15_IR_M3	I2C8_SDA_M2		
UART9_CTSN_M2			PWM10_M2	123	GPIO3_D3	33	34	GND					
		UART9_TX_M2	PWM11_IR_M3	125	GPIO3_D5	35	36	GPIO4_A0	128		SPI0_MISO_M1		UART9_RTSN_M1
SPI1_MISO_M1	UART7_TX_M1	I2C3_SDA_M1		112	GPIO3_C0	37	38	GPIO4_A1	129		SPI0_MOSI_M1		UART9_CTSN_M1
					GND	39	40	GPIO4_A2	130		SPI0_CLK_M1		

以下は LubanCat 4 を例にして i2c のデバイスツリープラグインを有効にする方法です（LubanCat 4 の引き出しピンには 2 つの i2c があります）：

ボードごとに設定ファイルと i2c が異なります。LubanCat 2 を例にして、設定ファイルは uEnvLubanCat 2.txt です。

I2c3、i2c5 を有効にすることができます。

システムターミナルにログインし、/boot/uEnv/uEnvLubanCat4.txt ファイルを開きます。

sudo vim /boot/uEnv/uEnvLubanCat4.txt

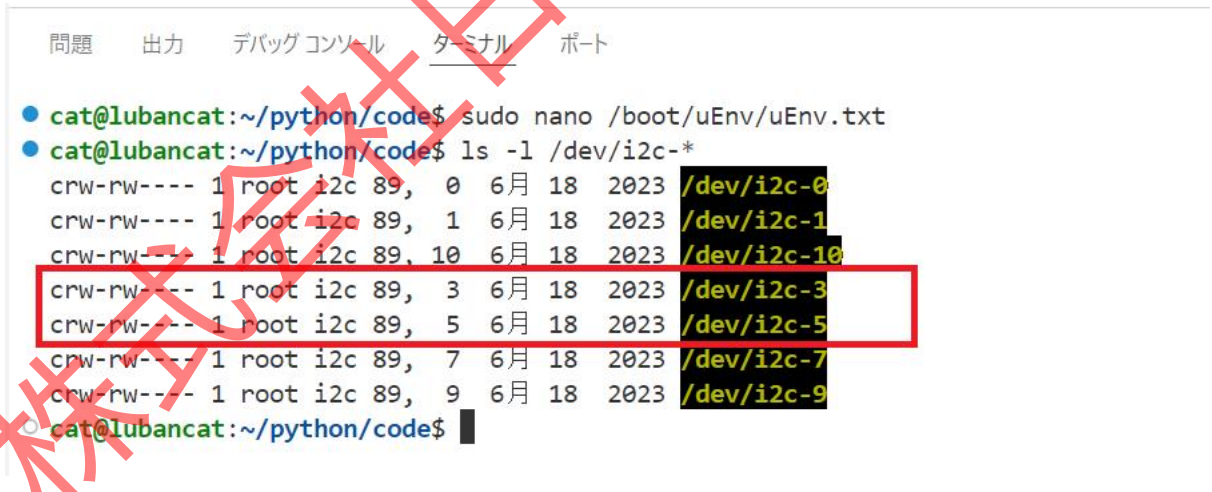
編集モードに入り、I2c3 の前のコメントを解除し、ファイルを保存して終了し、システムを再起動します。

再起動後、ボードの I2C リソースがロードされたかどうかを確認します：

ターミナルに次のコマンドを入力して、I2C リソースを確認します：

ls -l /dev/i2c-*

ボード上の I2C リソースの例は以下の通りです。参考にしてください：



簡単な i2c 関連のテストについてはマニュアル《Linux 基礎とアプリケーション開発実践ガイド.pdf》「第 22 章 I2C 通信」を参照してください。

8.2.2 ハードウェア接続

LubanCat4 ボードには I2C デバイスがないため、実験現象を展示するために、ハードウェア層で 0.96 インチ OLED ディスプレイ SSD1306 を接続します。使用するのは I2c5 です。

接続は次の図を参考にしてください（LubanCat4 の I2c5）。oled ディスプレイとボードの 40pin ピンの接続表は次の通りです：

OLED ピン	GPIO	Lubancat4 ボードピン
GND	GND	6 ピン
VCC	VCC	1 ピン
SCL	GPIO1_B6	5 ピン
SDA	GPIO1_B7	3 ピン

8.3 python-periphery を使用する

python-periphery ライブラリの I2C 機能は Linux の I2C システムに基づいて実装されているため、このライブラリを使用するにはボードがサポートしている必要があります。LubanCat ボードは、python-periphery ライブラリを使用して I2C 通信機能を完璧に実現できます。これにより、ソフトウェア層で GPIO を使用して I2C マスター機能をシミュレートする必要がなくなります。

8.3.1 python-periphery のインストール

重要: 前のセクションでこのライブラリをインストールした場合は、この操作をスキップしてください。

python-periphery のインストール方法は以下の通りです：

```
# ボード上で以下のコマンドを実行してインストールします。
sudo pip3 install python-periphery
```

8.3.2 periphery を使用して I2C 機能を使用する

まず、python-periphery ライブラリの I2C 機能を使用するサンプルコードを参照してください：

具体的なデバイスに対してプログラミングを行うことができます。

ここでは、前述の 0.96 インチ OLED ディスプレイ SSD1306 モジュールを使用してプログラミングを行います：

サンプルコード io/i2c/i2c_test_periphery.py の内容

```python
#8.3.2 periphery を使用して I2C 機能を使用する
""" periphery i2c テストで 0.96 インチ OLED モジュールを使用します。"""
import time
from periphery import I2C

# i2c-3 コントローラーを開く
try:
    i2c = I2C("/dev/i2c-5")
    print("I2C バスが正常に開かれました。")
except Exception as e:
    print(f"I2C バスのオープンに失敗しました: {e}")

# デバイススレーブアドレス 0x3c、つまり OLED モジュールのアドレス

I2CSLAVEADDR = 0x3C
```

```
# periphery i2c ライブラリの読み取り機能テスト
def i2c_read_reg(devregaddr):
    """
    periphery i2c ライブラリの読み取り機能テスト
    """
    # データ構造を作成
    # 最初のメッセージは読み取り対象のデバイスアドレスです。
    # 2 番目のメッセージには読み取ったメッセージが格納されます。read=True が追加されていることに注意してください。
    msgs = [I2C.Message([devregaddr]), I2C.Message([0x00], read=True)]
    # メッセージを送信し、i2cSlaveAddr に送信します。
    try:
        i2c.transfer(I2CSLAVEADDR, msgs)
        print("レジスタ 0x{:02x}から読み取りました: 0x{:02x}".format(devregaddr, msgs[1].data[0]))
    except Exception as e:
        print(f"I2C 転送エラー: {e}")



def oled_write_cmd(cmd):
    """
    periphery i2c ライブラリを使用した送信機能テスト
    """
    msgs = [I2C.Message([0x00, cmd])]

    try:
        i2c.transfer(I2CSLAVEADDR, msgs)
    except Exception as e:
        print(f"oled_write_cmd:I2C 転送エラー: {e}")



def oled_write_data(data):
    """
    periphery i2c ライブラリを使用した送信機能テスト
    """
    msgs = [I2C.Message([0x40, data])]
    #i2c.transfer(I2CSLAVEADDR, msgs)
    try:
        i2c.transfer(I2CSLAVEADDR, msgs)
    except Exception as e:
        print(f"oled_write_data:I2C 転送エラー: {e}")

def oled_init():
    """
    OLED モジュールを使用してコード機能をテストする 0.96 インチ OLED モジュールの初期化
    """
    time.sleep(1)
    oled_write_cmd(0xAE)
    oled_write_cmd(0x20)
    oled_write_cmd(0x10)
    oled_write_cmd(0xB0)
    oled_write_cmd(0xC8)
    oled_write_cmd(0x00)
    oled_write_cmd(0x10)
    oled_write_cmd(0x40)
    oled_write_cmd(0x81)
    oled_write_cmd(0xFF)
    oled_write_cmd(0xA1)
    oled_write_cmd(0xA6)
    oled_write_cmd(0xA8)
    oled_write_cmd(0x3F)
    oled_write_cmd(0xA4)
    oled_write_cmd(0xD3)
    oled_write_cmd(0x00)
    oled_write_cmd(0xD5)
    oled_write_cmd(0xF0)
    oled_write_cmd(0xD9)
    oled_write_cmd(0x22)
    oled_write_cmd(0xDA)
    oled_write_cmd(0x12)
    oled_write_cmd(0xDB)
    oled_write_cmd(0x20)
    oled_write_cmd(0x8D)
```

```
    oled_write_cmd(0x14)
    oled_write_cmd(0xAF)

def oled_fill(filldata):
    """
    OLED ディスプレイをクリアします。
    """
    for i in range(8):
        oled_write_cmd(0xB0 + i)
        # ページ 0～ページ 1
        oled_write_cmd(0x00)
        # 低コラム開始アドレス
        oled_write_cmd(0x10)
        # 高コラム開始アドレス
        for j in range(128):
            oled_write_data(filldata)

# コードテスト
try:
    # OLED ディスプレイを初期化し、SSD1306 が必要です。
    oled_init()
    # OLED ディスプレイをクリアします。
    oled_fill(0xFF)
    # レジスタの読み取りテスト
    i2c_read_reg(0x10)
finally:
    print("テストが正常に終了しました。")
    # リソースを解放します。
    i2c.close()
```

I2C マスターを使用して 0.96 インチ OLED ディスプレイ SSD1306 モジュールを制御するためにどのようなコマンドを送信するかは、このチュートリアルの重点ではありません。このチュートリアルの目的は、提供されたサンプルコードを通じて、関連するライブラリの API 機能の使用についての参考資料を提供することです。

8.3.2.1 実験操作

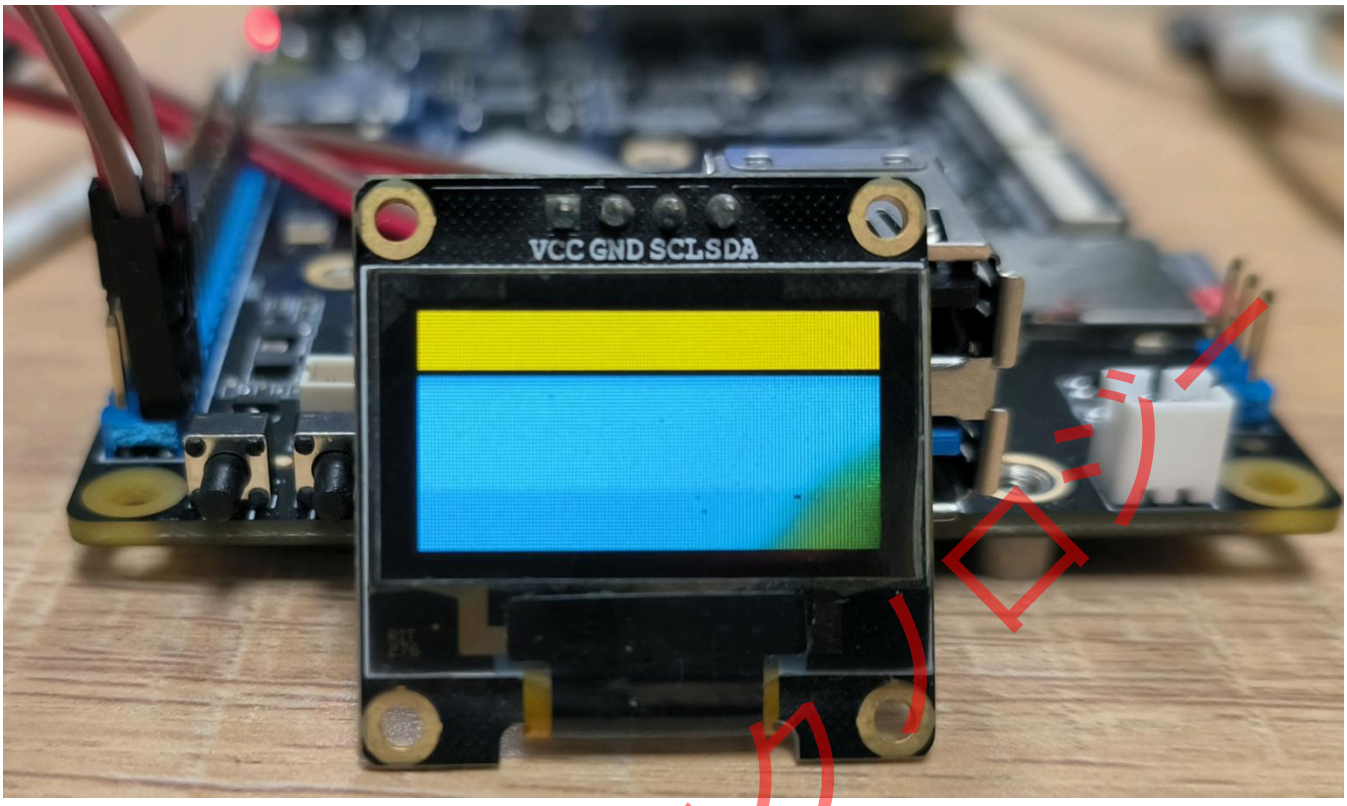
サンプルコードは LubanCat4 ボードを使用しており、操作は以下の通りです：

ボード上の i2c_test_periphery.py があるディレクトリで以下のコマンドを実行します。

sudo python3 i2c_test_periphery.py

接続された OLED ディスプレイがクリアされ、色ブロックが表示され、ターミナルにレジスタ情報が表示されるのが確認できます。

表示参考（手持ちの古い画面を使用）：



8.4 方式二：Adafruit Blinka を使用する

Adafruit Blinka は python-periphery と似た機能を持ち、I2C 通信も Linux の I2C サブシステムに基づいています。ただし、コードの使い方には大きな違いがあります。

8.4.1 Adafruit Blinka のインストール

重要: 前のセクションでこのライブラリをインストールした場合は、この操作をスキップしてください。

Adafruit Blinka のインストール方法は以下の通りです：

```
# ボード上で以下のコマンドを実行してインストールします。
sudo apt -y install python3-libgpiod python-periphery # 注：すでにインストールされている場合は、このステップをスキップします。
sudo pip3 install Adafruit-Blinka
sudo pip3 install machine
```

8.4.2 Adafruit-Blinka を使用した I2C マスター

ここでは、I2C1 マスターと 0.96 インチ OLED ディスプレイモジュール SSD1306 の通信プログラムを直接紹介します：

サンプルコード io/i2c/i2c_test_adafruit.py の内容

```
#8.4.2 Adafruit-Blinka を使用した I2C マスター
""" blinka i2c テストで 0.96 インチ OLED モジュールを使用します。"""
import board
import busio

# OLED 初期化コマンドバイト配列
OledInitBuf = bytes(
    [
        0xAE,
        0x20,
        0x10,
        0xB0,
        0xC8,
        0x00,
        0x10,
        0x40,
        0x81,
        0xFF,
        0xA1,
        0xA6,
        0xA8,
        0x3F,
        0xA4,
        0xD3,
        0x00,
        0xD5,
        0xF0,
        0xD9,
        0x22,
        0xDA,
        0x12,
        0xDB,
        0x20,
        0x8D,
        0x14,
        0xAF,
    ]
)


# データ送信、受信バッファ
OutBuffer = bytearray(1)
InBuffer = bytearray(1)

try:
    # I2C リソースを要求します。
    i2c = busio.I2C(board.I2C5_SCL, board.I2C5_SDA)
    # I2C デバイスアドレスをスキャンしてテストします。
    print("I2C バスに接続されている I2C デバイスのアドレス:")
    for i in i2c.scan():
        print("0x%02x " % i)

    # IIC 送信コマンドテスト (writeto)、OLED の初期化
    # OLED を初期化し、初期化コマンドをバイト配列 OledInitBuf に格納します。
    i2c.writeto(0x3C, OledInitBuf)

    # IIC 読み取りコマンドテスト (writeto_then_readfrom)、OLED の 0x10 レジスタを読み取ります。


    # データを送信し、送信完了後に停止信号を生成せず、新たに開始信号を生成して読み取ります。
```

```
    # 送信データ内容をバイト配列 OutBuffer に格納します。内容は OLED レジスタアドレス:0x10 です。
    OutBuffer[0] = 0x10
    # 送信し、レジスタアドレス 0x10 を送信し、レジスタアドレス 0x10 の内容を InBuffer に読み取ります。
    i2c.writeto_then_readfrom(0x3C, OutBuffer, InBuffer)
    # データ情報を表示します。
    print("(writeto_then_readfrom) 読み取った内容:0x%02x" % InBuffer[0])

    # IIC 読み取りテスト (readfrom_into)、デバイス内部のアドレスから読み取るため、読み取りアドレスを指定する必要はありません。
    i2c.readfrom_into(0x3C, InBuffer)
    # データ情報を表示します。
    print("(readfrom_into) 読み取った内容:0x%02x" % InBuffer[0])

    # OLED クリア画面
    for i in range(8):
        i2c.writeto(0x3C, bytearray([0x00, 0xB0 + i])) # ページ 0～ページ 1
        i2c.writeto(0x3C, bytearray([0x00, 0x00])) # ローカラム開始アドレス
        i2c.writeto(0x3C, bytearray([0x00, 0x10])) # ハイカラム開始アドレス
    for j in range(128):
        i2c.writeto(0x3C, bytearray([0x40, 0xFF]))

finally:
    # I2C バスリソースを解放します。
    i2c.deinit()
```

コードの説明：

- 47 行目では、I2C リソースを要求し、I2C1 マスターを占有します。
- 55 行目では、バイト配列 OledInitBuf のデータを、I2C1 マスターに接続されたスレーブデバイスに送信します。スレーブデバイスのアドレスは 0x3C です。
- 63～65 行目では、バイト配列 OutBuffer のデータを I2C1 マスターに接続されたスレーブデバイスに送信し、レジスタアドレス 0x10 で 1 バイトのデータを読み取ります。スレーブデバイスのアドレスは 0x3C です。
- 70 行目では、スレーブデバイス内部の現在のアドレスから 1 バイトのデータを読み取り、バイト配列 InBuffer に格納します。
- 84 行目では、I2C リソースを解放し、I2C1 マスターを解放します。

実験操作は前のセクションと同様で、プログラムをボード上で実行するだけです。

コードスニペットの busio.I2C(SCL, SDA) の SCL、SDA は Blinka ライブラリで定義された LubanCat ボードのリソースです。

より詳細なピン定義は Adafruit Blinka のソースコードを直接参照してください：

- Adafruit Blinka における LubanCat シリーズボードのピン定義
- Adafruit Blinka における LubanCat RK シリーズチップのピン定義

8.4.3 Adafruit_CircuitPython_SSD1306

前述の Blink ライブラリを使用して I2C マスターで 0.96 インチ OLED ディスプレイ SSD1306 モジュールを制御する方法を実現しました。Blink ライブラリは CircuitPython に基づいて再実装されたリアルタイムコンポーネントオブジェクトです。つまり、Blink ライブラリを使用して開発された Python プログラムは、リアルタイムでボード上のハードウェアリソースを使用できます。CircuitPython に詳しい方は、CircuitPython ライブラリを基に実装されたさまざまな電子デバイスの制御デモが非常に多いことを知っているでしょう。

つまり、LubanCat ボードが Blink ライブラリに適合していれば、CircuitPython の豊富なさまざまなセンサーデモを利用できるということです。著者が上記の実験で使用したモジュールを例にとると、0.96 インチ OLED ディスプレイ SSD1306 モジュールのデモは正常に使用できます。

次に、著者は SSD1306 モジュールを例に、CircuitPython が 0.96 インチ OLED ディスプレイ SSD1306 モジュールをサポートするプログラムを LubanCat ボード上で実行する方法を示します。

まず対応するリポジトリを見つけます。ここでは 0.96 インチ OLED ディスプレイ SSD1306 を例に説明します。Adafruit の Github でキーワード SSD1306 を検索すると、Adafruit_CircuitPython_SSD1306 が表示されます。これが使用するリポジトリです。もちろん、他の一般的なモジュール（例えば dht11、mpu6050 など）も検索することができます。これらは各自で探索してください。

総じて、Adafruit の Github で多くのデモリポジトリを探すことができます。

対応するリポジトリの下に、ライブラリの追加方法が記載されています：

ボード上で以下のコマンドを実行して対応するセンサーライブラリをインストールします。以下のコマンドのいずれかを選択してください。

root ユーザーの場合：

```
pip3 install adafruit-circuitpython-ssd1306
```

通常ユーザーの場合：

```
sudo pip3 install adafruit-circuitpython-ssd1306
```

または、他の方法でダウンロードしたリポジトリをボードにアップロードし、リポジトリディレクトリに入り以下のコマンドを実行してインストールします：

```
sudo python3 setup.py install
```

ボードの性能とネットワークの理由から、インストールプロセスは非常に時間がかかることがあります。忍耐強くお待ちください。インストール中にエラーが発生した場合は、上記の別の方法を試してください。

インストールが完了したら、リポジトリに提供されているさまざまなデモを使用できます。一部のデモでスレーブデバイスのアドレスが手持ちのモジュールと完全に適合しない場合、エラーメッセージが表示されることがあります。その場合は、デモのコード内のスレーブデバイスのアドレスを変更する必要があります。

CircuitPython が 0.96 インチ OLED ディスプレイ SSD1306 モジュールに提供するコードデモの一例は以下の通りです：

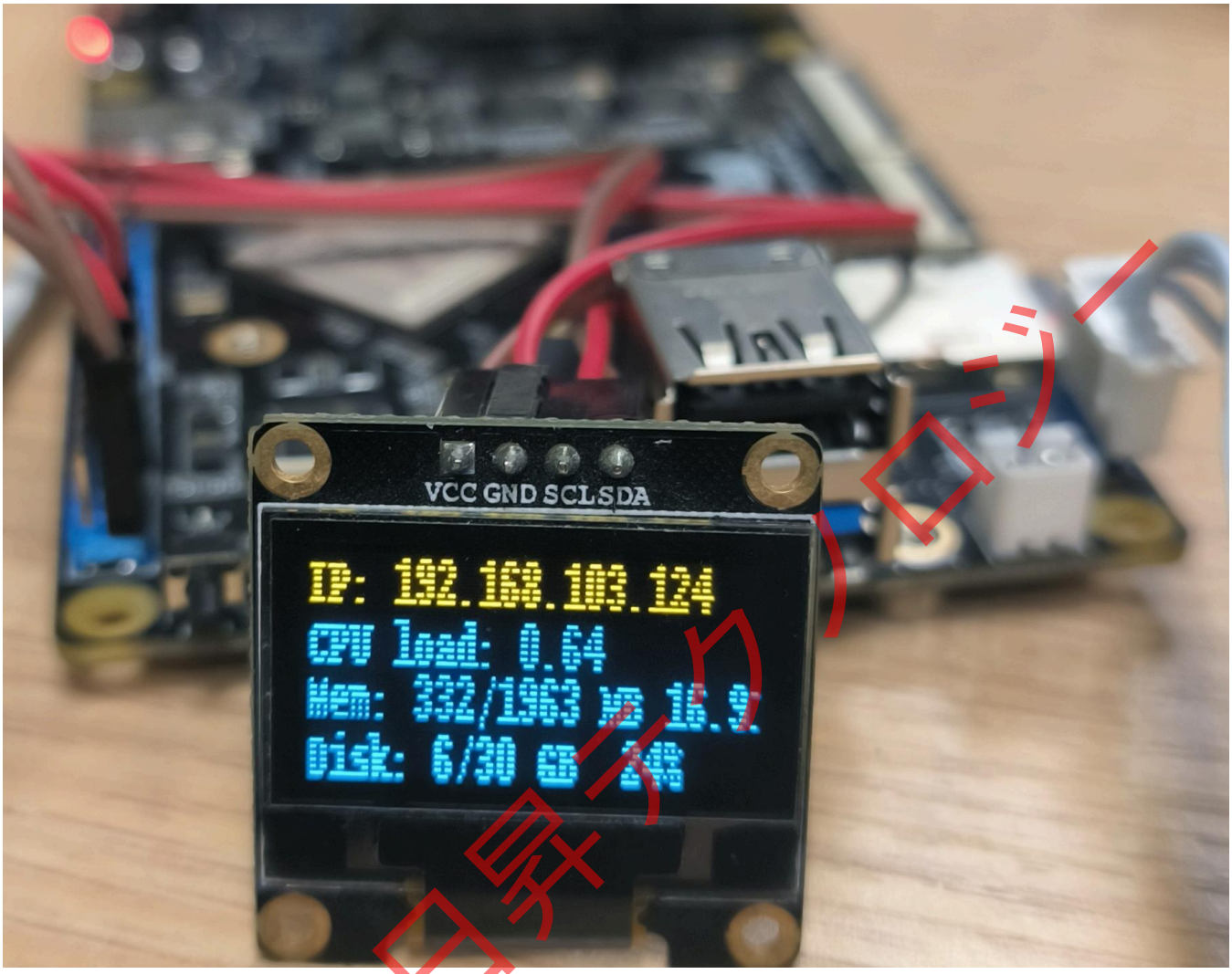
Pillow ライブラリは必要ですので、インストールしましょう。

```
sudo pip3 install Pillow
```

最後に、任意のデモをターミナルで実行して効果を確認します：

```
sudo python3 ssd1306_stats.py
```

これで、LubanCat ボードの動作状態などの情報がリアルタイムで表示されます。



第 9 章 SPI 通信

前の章では、I2C 通信プロトコルの使用に関する内容を紹介しました。この章では、もう一つの通信プロトコルである SPI 通信プロトコルの使用について説明します。SPI 通信プロトコルも電子機器の通信でよく使われるプロトコルです。しかし、SPI 通信プロトコルは通信速度が高く、全二重通信方式を採用しています。そのため、通信速度が要求される電子機器では、SPI 通信プロトコルを使用したインターフェースをよく見かけます。例えば、小型 LCD スクリーン、AD 変換チップ、nRF シリーズの RF モジュール、SPI-FLASH などです。

SPI プロトコルの詳細については、ここでは詳しく説明しませんので、皆さん自身で SPI 通信プロトコルに関する知識を検索して学んでください。

この章では、前に紹介した python-periphery ライブラリを使用して、AI エッジ基板 CSAIEG358803 (LubanCat4 ボード)上の SPI マスターを使用することに重点を置きます。

9.1 SPI 実験

SPI 通信プロトコルは全二重通信方式をサポートしているため、ループバックテストを行うことで実験を検証できます。実験では、SPI インターフェースの MOSI と MISO を接続するだけです。

9.2 実験準備

9.2.1 SPI リソースの追加

ボード上の GPIO は I2C として再利用されていない可能性があるため、デバイスツリープラグインを設定してロードすることができます。

基板の 40 ピン GPIO 図：（参考マニュアル：《AI エッジ基板 CSAIEG358803 ハードウエア仕様書》）

LuBanCat4 ピン図

拡張機能3	拡張機能2	拡張機能1	共通機能	No.	GPIO	物理ピン		GPIO	No.	共通機能	拡張機能1	拡張機能2	拡張機能3
					3.3V	1	2	5V					
			I2C5_SDA_M3	47	GPIO1_B7	3	4	5V					
			I2C5_SCL_M3	46	GPIO1_B6	5	6	GND					
	I2C2_SDA_M4	UART6_RX_M1		32	GPIO1_A0	7	8	GPIO4_A3	131	UART0_TX_M2			
					GND	9	10	GPIO4_A4	132	UART0_RX_M2			
	I2C2_SCL_M4	UART6_TX_M1		33	GPIO1_A1	11	12	GPIO1_D6	62	PWM14_M2	I2C8_SCL_M2		
	PDM1_SDIO_M1	PWM3_IR_M3		39	GPIO1_A7	13	14	GND					
	PDM1_SDI1_M1			40	GPIO1_B0	15	16	GPIO3_C1	113			UART7_RX_M1	SPI1_CLK_M1
					3.3V	17	18	GPIO3_D2	122				UART9_RTSN_M2
	PDM1_SDI3_M1	UART4_RX_M2	SPI0_MOSI_M2	42	GPIO1_B2	19	20	GND					
	PDM1_SDI2_M1		SPI0_MISO_M2	41	GPIO1_B1	21	22	GPIO3_D4	124		UART9_RX_M2		
	PDM1_CLK1_M1	UART4_TX_M2	SPI0_CLK_M2	43	GPIO1_B3	23	24	GPIO1_B4	44	SPI0_CS0_M2	UART7_RX_M2	PDM1_CLK0_M1	
					GND	25	26	GPIO1_B5	45	SPI0_CS1_M2	UART7_TX_M2		
			I2C6_SDA_M3	136	GPIO4_B0	27	28	GPIO4_B1	137	I2C6_SCL_M3			
				102	GPIO3_A6	29	30	GND					
SPI1_MOSI_M1		I2C3_SCL_M1		111	GPIO3_B7	31	32	GPIO1_D7	63	PWM15_IR_M3	I2C8_SDA_M2		
UART9_CTSN_M2			PWM10_M2	123	GPIO3_D3	33	34	GND					
		UART9_TX_M2	PWM11_IR_M3	125	GPIO3_D5	35	36	GPIO4_A0	128		SPI0_MISO_M1		UART9_RTSN_M1
SPI1_MISO_M1	UART7_TX_M1	I2C3_SDA_M1		112	GPIO3_C0	37	38	GPIO4_A1	129		SPI0_MOSI_M1		UART9_CTSN_M1
					GND	39	40	GPIO4_A2	130		SPI0_CLK_M1		

ピン 19 がピン 20 と接続にします。

ここでは、LubanCat4 を例にして SPI デバイスツリープラグインを有効にする方法を説明します (LubanCat4 の引き出しピンは spi0-m2 を使用できます) 。

ボードごとの設定ファイルは spi と異なります。LubanCat4 を例にすると、設定ファイルは uEnvLubanCat4.txt です。

システム端末にログインし、/boot/uEnv/uEnvLubanCat4.txt ファイルを開きます。

sudo nano /boot/uEnv/uEnvLubanCat4.txt

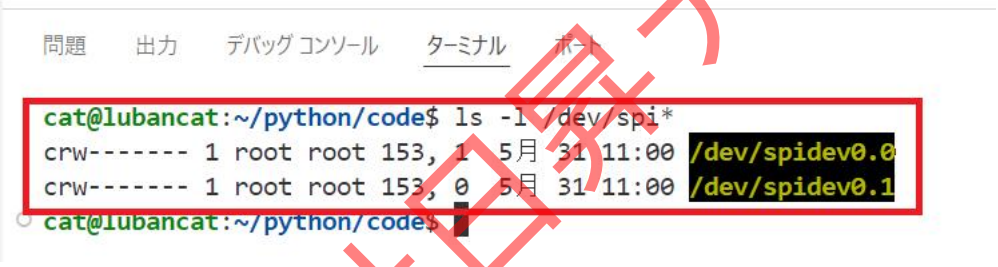
編集モードに入り、spi0-m2 の前のコメントを解除して、ファイルを保存して終了し、システムを再起動します。

再起動後、ボードの SPI リソースがロードされているか確認します：

端末で次のコマンドを入力すると、SPI リソースを確認できます：

ls -l /dev/spi*

ボード上の SPI リソースの例は、参考のために示されています：



Linux での SPI テストについては、マニュアル≪Linux 基礎とアプリケーション開発実践ガイド≫「第 23 章 SPI 通信」を参照してください。

9.2.2 ハードウェア接続

注：ハードウェア接続セクションは、自身の開発環境などに応じて調整してください。

ループバックテストを行うため、次のように接続します（LubanCat4）。

9.3 python-periphery の使用

python-periphery ライブラリがサポートする SPI 機能は、Linux の SPI システムに基づいています。そのため、このライブラリを使用するためには、ボードが対応している必要があります。LubanCat ボードのように、python-periphery ライブラリの SPI 通信機能を完全に使用できます。これにより、ソフトウェアレイヤーで GPIO を使用して SPI マスター機能をエミュレートする必要がなくなります。

9.3.1 python-periphery のインストール

重要：前のセクションでこのライブラリをインストールした場合、この手順をスキップしてください。
python-periphery のインストール方法は次のとおりです：
# ボード上で次のコマンドを使用してインストールします
sudo pip3 install python-periphery

9.3.2 periphery の SPI 機能の使用

python-periphery ライブラリを使用して SPI 機能を使用するサンプルコードは次のとおりです：
サンプルコード io/spi/spi_test_periphery.py ファイルの内容

```
""" periphery spi テスト """
from periphery import SPI

# 送信データリスト
data_out = [0xAA, 0xBB, 0xCC, 0xDD]

try:
    # SPI リソースを取得し、spidev3.0 コントローラを開き、SPI マスターを動作モード 0、動作速度を 1MHz に設定します
    spi = SPI("/dev/spidev0.1", 0, 1000000)

    # データを送信しながら data_in リストにデータを受信します
    data_in = spi.transfer(data_out)

    # 送信データの内容を表示します
    print("送信データ: [0x{:02x}, 0x{:02x}, 0x{:02x}, 0x{:02x}]".format(*data_out))
    print("受信データ: [0x{:02x}, 0x{:02x}, 0x{:02x}, 0x{:02x}]".format(*data_in))
finally:
    # 取得した SPI リソースを閉じます
    spi.close()
```

コードの説明：
- 第 5 行：送信するメッセージリストを構築します。
- 第 9 行：SPI リソースを取得し、SPI3 マスターを占有し、対応する動作モードを設定します。
- 第 12 行：SPI 通信を開始し、SPI は全二重通信方式のため、データを送信しながらデータを受信できます。
- 第 19 行：SPI リソースを解放し、SPI3 マスターを解放します。

9.3.2.1 実験操作

サンプルコードは LubanCat4 ボードを使用し、操作は次のとおりです：
# ボード上の spi_test_periphery.py があるディレクトリで次のコマンドを実行します
sudo python3 spi_test_periphery.py
# 端末に表示される受信データが送信データと一致していることがわかります。

```
cat@lubancat:~/python/code/io/spi$ sudo python3 spi_test_periphery.py
送信データ: [0xaa, 0xbb, 0xcc, 0xdd]
受信データ: [0xaa, 0xbb, 0xcc, 0xdd]
```

ハードウェア接続時に、SPI インターフェースの MOSI と MISO を接続しているため、SPI コントローラがデータを送信すると同時にデータを受信します。これがループバックテストです。

9.4 方法二：Adafruit Blinka を使用する

Adafruit Blinka は python-periphery と同様に、SPI 通信も Linux の SPI サブシステムに基づいて実現しています。ただし、コードの使用方法には大きな違いがあります。

9.4.1 Adafruit Blinka のインストール

重要: 前のセクションでこのライブラリをインストールした場合、この手順をスキップしてください。
Adafruit Blinka のインストール方法は次のとおりです：

```
# ボード上で次のコマンドを使用してインストールします
sudo apt -y install python3-libgpiod python-periphery
# 注：すでにインストールされている場合、このステップをスキップしてください
sudo pip3 install Adafruit-Blinka
```

9.4.2 Adafruit-Blinka の SPI マスターの使用
Adafruit Blinka を使用して入出力を行うサンプルコードは次のとおりです：

サンプルコード io/spi/spi_test_adafruit.py ファイルの内容

```python
""" blinka spi テスト """
import busio
from board import *

# データ送信、受信バッファ
OutBuffer = [0xAA, 0xBB, 0xCC, 0xDD]
InBuffer = bytearray(4)

try:
    # spi リソースを取得
    spi = busio.SPI(SCLK, MOSI, MISO)

    # SPI を設定する際に最初にロック
    spi.try_lock()
    # SPI マスターの動作速度を 1MHz、クロック極性を 0、クロック位相を 0、1 データビットを 8 ビットに設定
    spi.configure(1000000, 0, 0, 8)
    # 設定操作が完了したらロック解除
    spi.unlock()

    # SPI 通信、送信しながら読み取りを行う。送信データは OutBuffer に、読み取りデータは InBuffer に格納
    spi.write_readinto(OutBuffer, InBuffer)

    print("(write_readinto) 受信データ: [0x{:02x}, 0x{:02x}, 0x{:02x}, 0x{:02x}]".format(*InBuffer))

    # SPI 通信、送信のみの方法のデモ
    spi.write(OutBuffer)
    print("(OutBuffer) 送信データ: [0x{:02x}, 0x{:02x}, 0x{:02x}, 0x{:02x}]".format(*OutBuffer))

    InBuffer = [0, 0, 0, 0]
    # SPI 通信、読み取りのみの方法のデモ
    spi.readinto(InBuffer)
    print("(readinto) 受信データ: [0x{:02x}, 0x{:02x}, 0x{:02x}, 0x{:02x}]".format(*InBuffer))
finally:
    # spi リソースを解放
    spi.deinit()
```

コードの説明：

- 第 11 行：SPI リソースを取得し、SPI3 マスターを占有します。
- 第 14~18 行：SPI3 マスターの動作モードを設定し、設定前にリソースをロックします。
- 第 21 行：SPI3 マスターを使用してバイト配列 OutBuffer のデータを送信し、同時に InBuffer にデータを読み取ります。
- 第 30 行：SPI3 を使用してデータを送信するのみで、読み取りは行いません。
- 第 37 行：SPI3 を使用してデータを読み取るのみで、送信は行いません。

実行後の表示：

```
cat@lubancat:~/python/code/io/spi$ sudo python3 spi_test_adafruit.py
(write_readinto) 受信データ: [0xaa, 0xbb, 0xcc, 0xdd]
(OutBuffer) 送信データ: [0xaa, 0xbb, 0xcc, 0xdd]
(readinto) 受信データ: [0x00, 0x00, 0x00, 0x00]
```

さらに、io/spi/ssd1306_spi_status.py という補助的な例もあり、これは spi を使用して画面に表示します。具体的には、前の i2c で Adafruit_CircuitPython_SSD1306 ライブラリを使用して画面を制御するセクションを参照してください。

Adafruit Blinka ライブラリは SPI オブジェクトの使用例を提供していませんが、関連する API 機能については、CircuitPython SPI の紹介を参照してください。

第 10 章 スクリーン表示 - Pygame

私たちの LubanCat ボードは、HDMI インターフェースや MIPI DSI インターフェースなど、さまざまな表示デバイスをサポートしています。

10.1 Pygame ライブラリの紹介

Pygame は、Python ライブラリの中でゲーム開発に使用されるツールですが、Pygame の用途はそれだけに限りません。Pygame はゲーム開発だけでなく、表示処理やマルチメディアデバイスの処理など、多くの機能を提供しています。このライブラリには、ディスプレイデバイスを操作するための多くの機能が含まれています。特に display モジュールは、Pygame の表示インターフェースを制御するためのさまざまな関数を提供しており、この章ではこのモジュールを使用してスクリーン表示を行います。

10.2 実験の準備

10.2.1 スクリーンリソースの追加

端末で以下のコマンドを入力して、表示デバイスリソースを確認します：

```bash
cat@lubancat:~$ ls /dev/dri/card*
/dev/dri/card0 /dev/dri/card1
```

スクリーンを接続すると、以下のように表示されます：

```bash
cat@lubancat:~/lcd$ ls /dev/fb*
/dev/fb0
```

ボード上の表示デバイスリソースの例として、LubanCat4 と Debian イメージを使用します：

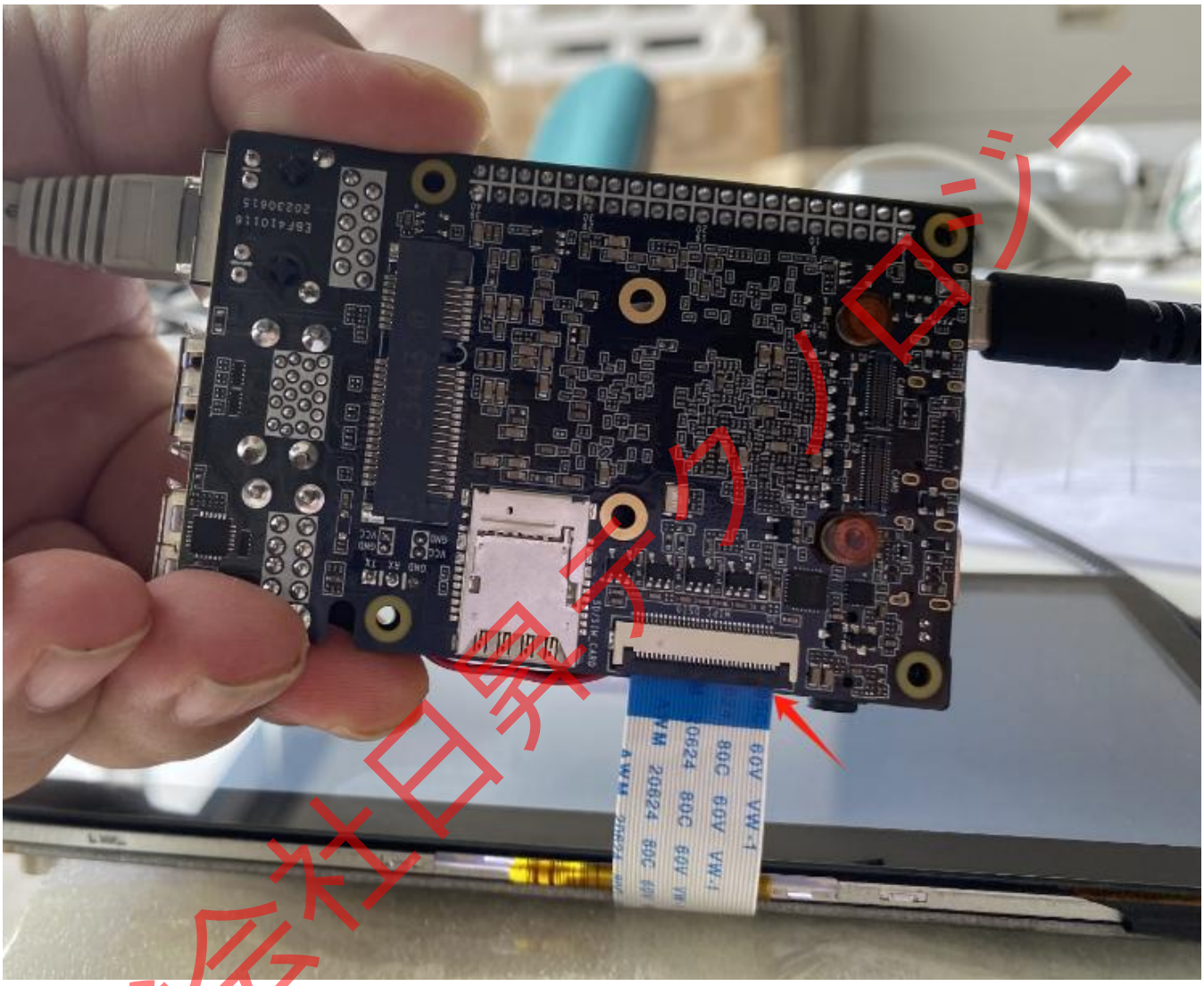
デフォルトでは、rk3568-LubanCat4.dtb デバイスツリーが使用されており、対応するデバイスツリーを選択するためにシンボリックリンクを作成します：

```bash
# boot ディレクトリに移動します
cd /boot
# mipi スクリーンのデバイスツリーに切り替えます
ln -sf dtb/rk3588s-lubancat-4.dtb rk-kernel.dtb
# 再起動して mipi スクリーンを有効にします
sudo reboot
```

詳細は、マニュアル≪Linux 基礎とアプリケーション開発実践ガイド≫「第 27 章 スクリーン」を参照してください。

10.2.2 ハードウェア接続

ハードウェア接続セクションでは、各自の開発環境に応じて内容を調整してください。ここでは、mipi スクリーン（7 インチ）を LubanCat4 ボードの裏面 MIPI-dsi0 に接続し、ボードのシステムはデスクトップを備えています。



10.2.3 Pygame ライブラリのインストール

apt ツールを使用してインストールします。

端末で以下のコマンドを入力して、Pygame ライブラリをインストールします：

```bash
sudo apt -y install python3-pygame
```

既にインストールされている場合は、再インストールする必要はありません。

10.2.4 Pygame ライブラリの使用

ライブラリがインストールされた後、インストールされた Pygame ライブラリを使用してテストコードを作成できます。

表示ドライバは DRM ですが、ドライバは FB デバイスをエミュレートしており、テスト前に Linux Framebuffer を簡単に使用できます。Pygame ライブラリは Linux Framebuffer メカニズムを利用して表示デバイスを調整し、Linux Framebuffer はスクリーン上の各点を線形メモリ空間にマッピングし、プログラムがこのメモリの値を変更することでスクリーン上の特定の点の色を変更できます。以下のコマンドを使用してランダムデータを/dev/fb0 に書き込み、スクリーンを乱れさせることができます：

```bash
cat /dev/urandom > /dev/fb0
```

詳細は、マニュアル≪Linux 基礎とアプリケーション開発実践ガイド≫「第 29 章 画面表示(DRM) の紹介」を参照してください。

10.3 テストコード 1

テストコードは以下の通りです：

サンプルコード io/lcd/lcd_test1.py ファイルの内容

```python
""" Pygame を使用したスクリーン表示テスト """
import os
import sys
import time
import pygame

# システム環境を設定し、マウスを無効にします
os.environ["SDL_NOMOUSE"] = "1"

# 表示デバイスを初期化します
pygame.display.init()

# 表示ウィンドウの範囲をスクリーンサイズに設定し、スクリーンオブジェクトを返します
screen = pygame.display.set_mode((448,418))

# ウィンドウの名前を設定します
pygame.display.set_caption("test")

# pygame image モジュールを使用して画像をロードします
ball = pygame.image.load("test.png")

# 画像をウィンドウにビットブリットします
screen.blit(ball, (0, 0))

# ウィンドウを更新します
pygame.display.flip()

time.sleep(5)
sys.exit()
```

このコードは、以下の図を mipi ディスプレイのウィンドウに表示します：

10.3.1 実験手順

テストコードとテスト画像を同じディレクトリにアップロードします：

```bash
# 端末で以下のコマンドを入力し、ハードウェアアクセスには sudo 権限が必要です：
sudo python3 lcd_test1.py
```

このコマンドを実行すると、端末に送信されたデータが表示されます。

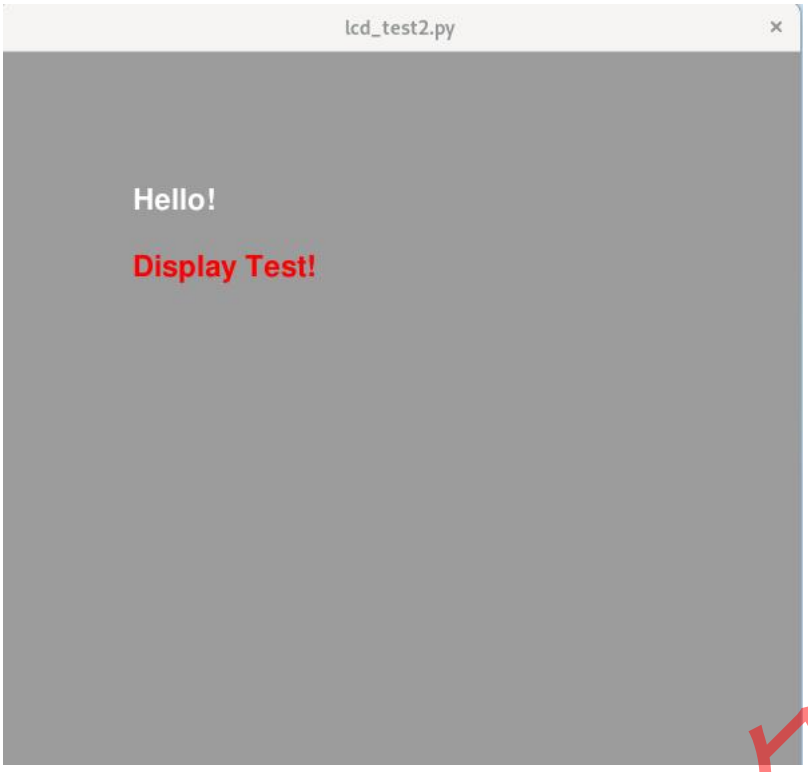
10.4 テストコード 2

サンプルコード io/lcd/lcd_test2.py ファイルの内容

```
""" Pygame を使用したスクリーンテスト """
import os
import sys
import time
import pygame

class PyScope:
    """ スクリーンテストを行う PyScope クラスを定義します """

    screen = None

    def __init__(self):
        """ PyScope クラスの初期化メソッド。framebuffer を使用して pygame が使用するイメージバッファを構築します """

        # システム環境をマウスなしモードに設定します
        # os.environ["SDL_NOMOUSE"] = "1"
        pygame.display.init()

        # pygame が使用するウィンドウサイズを作成し設定します
        self.screen = pygame.display.set_mode((600,600))

        # ウィンドウのタイトルを設定します
```

```
        pygame.display.set_caption('lcd_test2.py')

        # 背景色を灰色に設定します
        self.screen.fill((156,156,156))

        # フォントライブラリを初期化します
        pygame.font.init()

        # デフォルトフォントを使用して英文字を表示します
        font = pygame.font.Font(None, 32)
        text = font.render("Hello!", True, (255, 255, 255))
        self.screen.blit(text, (100, 100))

        # デフォルトフォントを使用して文字を表示します
        text = font.render("Display Test!", True, (255, 0, 0))
        self.screen.blit(text, (100, 150))

        # スクリーン内容を更新します
        pygame.display.flip()

    def __del__(self):
        """ pygame ライブラリを終了するときに呼び出されるメソッド。リソース解放操作をここで追加できます """
        pygame.display.quit()

    def test(self):
        while True:
            # イベントをループで取得し、イベントをリスンします
            for event in pygame.event.get():
                # ユーザーが閉じるボタンをクリックしたかどうかを判断します
                if event.type == pygame.QUIT:
                    # すべてのモジュールをアンロードします
                    pygame.quit()
                    # プログラムを終了します
                    sys.exit()
                # スクリーン内容を更新します
                # pygame.display.flip()

# テストインスタンスを作成し、テストを開始します
scope = PyScope()
# scope クラスのテストメソッドを呼び出します
scope.test()
```


コードの機能はコメントで簡単に説明されています。`__init__`メソッドではウィンドウ表示を初期化し、最終的にはこのコードで 2 行の英文字がウィンドウに表示されます。



10.4.1 実験手順

テストコードを開発ボードの任意のディレクトリにアップロードします：

```bash
# 端末で以下のコマンドを入力します：
sudo python3 lcd_test2.py
```

このコマンドを実行すると、システムデスクトップにウィンドウが表示され、マウスをクリックまたはプログラムを閉じることでウィンドウを閉じることができます。

10.5 参考

この章では、display モジュールの基本関数の使用方法を詳しく紹介しませんでした。Pygame display モジュールの詳細については、Pygame display のドキュメントを参照してください。

Python Pygame ライブラリの使用に関しては、Pygame チュートリアルを参照してください。

第 11 章 音声再生 - Pygame

LubanCat-RK シリーズのボードには、LubanCat4 は音声 rk es8388-codec を備えたモデルがあります。本章では、音声コーデックチップを使用して、Python-Pygame ライブラリの music モジュールを使用し、LubanCat ボードでテストコードを作成して音声再生実験を行います。

11.1 Pygame ライブラリの紹介

Pygame は、Python ライブラリの中でゲーム開発に使用されるツールですが、Pygame の用途はそれだけに限りません。Pygame はゲーム開発だけでなく、表示処理やマルチメディアデバイスの処理など、多くの機能を提供しています。

11.2 実験の準備

11.2.1 ボード上のサウンドリソース

LubanCat4 を例にします：

```bash
# 端末で以下のコマンドを入力し、音声デバイスリソースを確認します：
ls /dev/snd/
```

ボード上の音声デバイスリソースの例：

```bash
cat@lubancat:~/python/code/io/lcd$ ls /dev/snd/
by-path  controlC0  controlC1  controlC2  pcmC0D0p  pcmC1D0c  pcmC1D0p  pcmC2D0p  seq  timer
```

- `controlC0`：サウンドカードの制御（チャネル選択、マイク制御など）
- `pcmC1D0c`：録音用の PCM デバイス
- `pcmC1D0p`：再生用の PCM デバイス
- `seq`：シーケンサー
- `timer`：タイマー

Linux の LubanCat 音声システムについては、マニュアル≪Linux 基礎とアプリケーション開発実践ガイド≫「第 26 章 音声」を参照してください。

11.2.2 ハードウェア接続

LubanCat4 ボードにはヘッドフォン＋マイクジャックがあります。ヘッドフォンや他の再生デバイスを LubanCat のジャックに接続します。

11.2.3 Pygame ライブラリのインストール

apt ツールを使用してインストールします。

```bash
# 端末で以下のコマンドを入力し、Pygame ライブラリをインストールします：
sudo apt -y install python3-pygame
```

11.2.4 Pygame ライブラリの使用

ライブラリをインストールした後、インストールされた Pygame ライブラリを使用してテストコードを作成します。

11.3 テストコード

テストコードは以下の通りです：

サンプルコード../io/sound/sound_test.py ファイルの内容

```
""" pygame 音声再生テスト """
import sys
import time
import threading

if len(sys.argv) == 2:
    m_filename = sys.argv[1]
else:
    print(
        """ 使用方法:
        python3 sound_test.py <music_name.mp3>
        example: python3 sound_test.py test.mp3"""
    )
    sys.exit()

import pygame

exitright = 'e'

def get_quitc():
    '終了文字を取得'
    global exitright
    exitright = input("終了するには 'q' を入力してエンターキーを押してください\n")
try:
    # 音声デバイスを初期化
    pygame.mixer.init()
    # 音声デバイスの初期化状況を確認
    if pygame.mixer.get_init() is None:
        raise RuntimeError('音声リソースが正しく初期化されていません。音声デバイスが使用可能か確認してください')
    # 音声ファイルをロード（mp3 ファイルの再生は限定的にサポートされている）
    pygame.mixer.music.load(m_filename)
    # 音量を 50%に設定
    pygame.mixer.music.set_volume(0.2)
    # 音声ファイルを 1 回再生
    pygame.mixer.music.play(0)

    # サブスレッドを開始して、ユーザーの入力を待つ
    quitthread = threading.Thread(target=get_quitc, daemon=True)
    quitthread.start()
    # 音声再生の状態を確認し、再生中であれば終了まで待機
    while pygame.mixer.music.get_busy():
```

```
        if exitright in ('q', 'Q'):
            raise
        time.sleep(1)
except pygame.error as e:
    print("プログラム実行中にエラーが発生しました:", e, "\n 音声リソースが解放されるのを待ちます")
    pygame.mixer.music.stop()
    while pygame.mixer.music.get_busy():
        time.sleep(1)
finally:
    # 初期化された音声デバイスをアンロード
    print("再生を終了します")
    pygame.mixer.quit()
    sys.exit()
```

11.3.1 実験手順

テストコードの前に、サンプルコードのフォルダ io/sound ディレクトリにある各フォーマットのテスト音声ファイルとコードを同じディレクトリに配置し、プログラムを実行します。

```bash
# 端末で以下のコマンドを入力します：
sudo python3 sound_test.py test.mp3
# または
sudo python3 sound_test.py test.wav
# または
sudo python3 sound_test.py test.ogg
```

Pygame ライブラリのリソースがロードされるまで少し時間がかかることがあります。プログラムが正常に動作すると、再生デバイスで音声が聞こえるようになります。再生中に終了するには、端末に表示される指示に従い、'q'を入力してエンターキーを押します。プログラムが異常終了した場合、音声デバイスリソースが占有されることがあり、再度使用するには開発ボードを再起動する必要があります。

```
cat@lubancat:~/python/code/io/sound$ sudo python3 sound_test.py test.mp3
pygame 1.9.6
Hello from the pygame community. https://www.pygame.org/contribute.html
終了するには 'q' を入力してエンターキーを押してください
q
再生を終了します
cat@lubancat:~/python/code/io/sound$ sudo python3 sound_test.py test.wav
pygame 1.9.6
Hello from the pygame community. https://www.pygame.org/contribute.html
終了するには 'q' を入力してエンターキーを押してください
q
再生を終了します
cat@lubancat:~/python/code/io/sound$ sudo python3 sound_test.py test.ogg
pygame 1.9.6
Hello from the pygame community. https://www.pygame.org/contribute.html
終了するには 'q' を入力してエンターキーを押してください
q
再生を終了します
```

11.3.2 参考

Python Pygame ライブラリの使用については、Pygame チュートリアルを参照してください。Pygame music については、Pygame music を参照してください。

第 12 章 カメラの使用 - OpenCV

LubanCat RK シリーズのボードは、一般的なセンサーであるカメラの使用をサポートしています。ボードには 24pin の mipi csi インターフェースがあり、テストに使用されるカメラは IMX415 です。

注：手元にない場合、弊社通販サイトから購入頂ければと思います。

IMX415 カメラモジュール（800M 画素、MIPI インタフェース）（csun.co.jp）

本章では、opencv-python ライブラリを使用してカメラを操作し、簡単な処理を行います。

注意：特に記載がない限り、本書のチュートリアルは Python 3.9.2 バージョン（イメージシステムは Debian11）に基づいています。この章の内容を学ぶためには、カメラハードウェアが必要ですので、各自で準備してください。

12.1 opencv-python ライブラリの紹介

OpenCV は、コンピュータビジョンの分野で広く使用されているライブラリで、マルチ言語対応、クロスプラットフォームで、強力な機能を持ちます。1999 年に Gary Bradsky によってインテルで設立され、2000 年に最初のバージョンがリリースされました。OpenCV は現在、コンピュータビジョンと機械学習に関連する多くのアルゴリズムをサポートしており、ますます拡張されています。OpenCV は、C++、Python、Java などのさまざまなプログラミング言語をサポートし、Windows、Linux、OS X（現在の macOS）、Android、iOS などのさまざまなプラットフォームで使用できます。CUDA および OpenCL に基づく高速な GPU 操作インターフェースも積極的に開発されています。

OpenCV-Python は、Python ベースのライブラリであり、OpenCV の Python アプリケーションインターフェースです。コンピュータビジョンの問題を解決するために設計されており、OpenCV C++ API と Python 言語のベストな特徴を組み合わせています。

12.2 LubanCat ボードにカメラリソースを追加

本節の実験では、LubanCat4 ボードを使用してデモンストレーションを行います。

下図通り、cam0 前の#をコメントアウトします。

電源を入れる前にカメラを mipi csi cam0 インターフェースに接続し、システム起動後：

```bash
# 端末で以下のコマンドを入力し、カメラデバイスリソースを確認します：
ls /dev/video*
```

```
cat@lubancat:~/python/code/io/camera$ ls /dev/video*
/dev/video-camera0  /dev/video-enc0  /dev/video1  /dev/video11  /dev/video13  /dev/video15  /dev/video17  /dev/video19  /dev/video3  /dev/video5  /dev/video7  /dev/video9
/dev/video-dec0     /dev/video0      /dev/video10 /dev/video12  /dev/video14  /dev/video16  /dev/video18  /dev/video2   /dev/video4  /dev/video6  /dev/video8
```

上記はカメラのことをシステムに追加されたことを判明しましたが、CAMO に接続されているカメラはどれのデバイス番号を分かりません。

全てのカメラデバイスをリストしましょう。

$v4l2-ctl --list-devices

```
cat@lubancat:~/python/code/io/camera$ v4l2-ctl --list-devices
rkisp-statistics (platform: rkisp):
        /dev/video18
        /dev/video19

rkcif-mipi-lvds (platform:rkcif):
        /dev/media0

rkcif (platform:rkcif-mipi-lvds):
        /dev/video0
        /dev/video1
        /dev/video2
        /dev/video3
        /dev/video4
        /dev/video5
        /dev/video6
        /dev/video7
        /dev/video8
        /dev/video9
        /dev/video10

rkisp_mainpath (platform:rkisp0-vir0):
        /dev/video11
        /dev/video12
        /dev/video13
        /dev/video14
        /dev/video15
        /dev/video16
        /dev/video17
        /dev/media1
```

カメラごとに、rkisp_mainpath という行があります、一つずつカメラと一致しています、今回一つカメラ接続しかないので、rkisp_mainpath が一つあります、最初/dev/video11 は接続されているカメラのデバイス番号。

ボード上のカメラデバイスリソースの例：

Linux システムでのカメラの使用については、マニュアル≪Linux 基礎とアプリケーション開発実践ガイド≫「第 25 章 カメラ」を参照してください。

この章では、v4l-utils などのツールを使用してカメラデバイスの関連情報を確認する方法を紹介します。

12.3 関連ツールおよびライブラリのインストール

実験で使用するいくつかのツールをインストールします。以下のコマンドを入力してインストールします（既にインストールされている場合は不要です）：

1. 依存ライブラリやツールをインストール：
```bash
# 端末で以下のコマンドを入力します：
sudo apt update
sudo apt -y install python3-pil python3-numpy python3-pip git wget
# 必要なライブラリをインストール
sudo apt install libavcodec-dev libavformat-dev libswscale-dev
sudo apt install libgstreamer-plugins-base1.0-dev libgstreamer1.0-dev
```

2. opencv-python ライブラリをインストール：
推奨インストール方法：
```bash
# apt コマンドを使用して python3-opencv パッケージをインストール：
sudo apt-get install python3-opencv
# または pip を使用して特定の opencv バージョンをインストール：
pip3 install opencv-python==4.7.0.68
```

whl ファイルを取得してインストールすることもできます。対応する Python バージョンとアーキテクチャの whl ファイルをダウンロードします。
```bash
# 例えば、以下の whl ファイルを使用してインストール：
pip3 install opencv_python-4.5.4.60-cp37-cp37m-manylinux_2_17_aarch64.manylinux2014_aarch64.whl
```

ソースコードをダウンロードしてコンパイルすることもできます。詳細は[リンク](https://github.com/opencv/opencv)を参照してください。

注意: Debian 11 システムを使用している場合、opencv_python を使用してカメラを制御するには、高バージョンの opencv_python と pip3 をインストールする必要があります。pip3 を更新して whl をインストールするには root 権限が必要です。以下のコマンドを使用します：
```bash
sudo pip3 install --upgrade pip
```
その後、上記の方法で opencv_python をインストールします。

3. インストールが成功したか確認：
Python インタラクティブシェル（または IPython）を開きます：
```bash
cat@lubancat:~$ python3
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.
```

```
>>> import cv2 as cv
>>> print(cv.__version__)
4.5.1
```

12.4 カメラで写真を撮る

サンプルコード io/camera/camera_photo.py ファイルの内容

```
""" opencv でカメラで写真を撮る """
import os
import cv2

# フレーム幅と高さ
width = 640
height = 480

num = 1

# VideoCapture オブジェクトを作成し、システムのデフォルトカメラを開きます（ビデオや特定のデバイスを開くこともできます）
cap = cv2.VideoCapture(11)

# カメラを開けない場合
if not cap.isOpened():
    raise RuntimeError('カメラを開くことができませんでした。')

# フレーム幅と高さを設定
cap.set(cv2.CAP_PROP_FRAME_WIDTH, width)
cap.set(cv2.CAP_PROP_FRAME_HEIGHT, height)

while(cap.isOpened()):
    # ret はカメラが正常に開けたかを示し、frame は画像配列（一フレーム）です
    ret, frame = cap.read()

    # ウィンドウ表示、名前は LubanCat_Camera_test
    cv2.imshow("LubanCat_Camera_test", frame)

    # 1ms の遅延を設定し、キーボード入力を取得
    val = cv2.waitKey(1) & 0xFF

    # キーボード入力が 's' の場合、画像を保存します
    if val == ord('s'):
        print("=======================================")
        # 最初のパラメータは保存する画像名、2 番目のパラメータは保存する画像
        cv2.imwrite("photo" + str(num) + ".jpg", frame)
        print("width = ", width)
        print("height = ", height)
        print("写真を保存しました: " + 'photo' + str(num) + ".jpg")
        print("=======================================")
        num += 1

    # キーボード入力が 'q' の場合、終了します
    if val == ord('q'):
        break

# カメラを解放
cap.release()
# ウィンドウを閉じる
cv2.destroyAllWindows()
```

デスクトップ付きのイメージシステムを使用し、キーボードとスクリーンを接続してボードシステムでプログラムを実行します。カメラを使用する方法については、上記のコメントを参照してください。

```bash
sudo python3 camera_photo.py
```

このコマンドを実行すると、画面にウィンドウが表示され、キーボードで’s’を押すと、画像が現在のディレクトリに保存されます。’q’を押すとプログラムが終了します。

12.5 カメラでビデオをキャプチャ、録画

ビデオの各フレームは 1 枚の画像を表しており、フレームごとに画像をキャプチャして圧縮保存することでビデオを録画します。

サンプルコード io/camera/camera_video01.py ファイルの内容

```python
""" opencv を生かしてカメラでビデオをキャプチャ、録画"""
import os
import cv2

# フレーム幅、高さ、フレームレート
width = 640
height = 480
fps = 25.0

# VideoCapture オブジェクトを作成し、システムのデフォルトカメラを開きます
video = cv2.VideoCapture(11)

# フレーム幅と高さを設定
video.set(cv2.CAP_PROP_FRAME_WIDTH, width)
video.set(cv2.CAP_PROP_FRAME_HEIGHT, height)

# ビデオファイルの書き込み形式を定義（未圧縮の YUV カラーエンコーディングタイプ）
fourcc = cv2.VideoWriter_fourcc('I', '4', '2', '0')

# 複数の画像を保存してビデオファイルを作成するためのオブジェクトを作成
out = cv2.VideoWriter('output.avi', fourcc, fps, (width, height))

while(video.isOpened()):
    ret, frame = video.read()
    if ret:
        # ウィンドウ表示、名前は LubanCat_Camera_video
        cv2.imshow('LubanCat_Camera_video', frame)

        # キャプチャした画像を保存
        out.write(frame)

        # 1ms の遅延を設定し、キーボード入力が 'q' の場合、終了します
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
    else:
        print("カメラを読み込めませんでした！")
        break

# リソースを解放
video.release()
out.release()
cv2.destroyAllWindows()
```

プログラムを実行すると、ビデオがキャプチャされ、録画され、ビデオファイルが現在のディレクトリに保存されます。プロセスの詳細については、上記のコメントを参照してください。キーボードで'q'を押すと終了します。

ビデオを再生するには、cv2.VideoCapture を使用します。以下を参照してください：

サンプルコード io/camera/camera_video02.py ファイルの内容

```
""" opencv でビデオを再生する"""

import cv2

# VideoCapture オブジェクトを作成し、再生するビデオファイルを選択します
cap = cv2.VideoCapture('output.avi')

while(cap.isOpened()):
    ret, frame = cap.read()

    # ウィンドウ表示、名前は LubanCat_Camera_video
    cv2.imshow('LubanCat_Camera_video', frame)

    # 1ms の遅延を設定し、キーボード入力が 'q' の場合、終了します
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# リソースを解放
cap.release()
cv2.destroyAllWindows()
```

12.6 参考

https://docs.opencv.org/4.6.0/dc/d4d/tutorial_py_table_of_contents_gui.html

第 13 章 Jupyter Notebook

13.1 Jupyter Notebook の紹介

Jupyter Notebook は、Web ベースのインタラクティブな計算用アプリケーションです。Jupyter Notebook は Web ページの形式で開き、各コマンドを記述した後、コード実行後のデータの変化をリアルタイムで確認することができます。プログラム中に説明文書を記述する必要がある場合、同じページで直接記述することができ、タイムリーな説明や解説が容易になります。Python ソフトウェア開発において、Jupyter Notebook は非常に有用なツールです。

Jupyter Notebook アプリケーションは Linux 環境で実行でき、サーバーにデプロイすることもできます。LubanCat ボードの機能は従来の PC に類似しており、Linux 環境で動作しているため、LubanCat ボードにも Jupyter Notebook ソフトウェアをインストールして、Python プログラムの開発に利用することができます。以下に、Jupyter Notebook ソフトウェアのインストール手順を示します。

13.2.1 ソフトウェアのインストール

LubanCat ボードを使用することで、Jupyter Notebook のインストールは非常に簡単で、以下の 2 つのコマンドを実行するだけです。
重要: 以降の操作はすべて一般ユーザーの `cat` で実行します。

```bash
# コマンドを使用
pip3 config set global.index-url https://pypi.org/simple
pip3 install jupyter
# または以下のコマンドを入力してインストール:
sudo apt -y install ipython3
sudo apt -y install jupyter
```

`ipython3` は Python のインタラクティブシェルで、Jupyter Notebook で Python アプリケーションを開発する際に使用します。
`jupyter` はインストールする Jupyter Notebook アプリケーションです。

13.2.2 Jupyter の設定

Jupyter Notebook のファイル保存場所（作業ディレクトリ）を設定するためには、まず Jupyter Notebook の設定ファイルを生成し、その設定ファイルを変更します。Jupyter Notebook を起動すると、アプリケーションは特定の設定に従って起動します。以下のコマンドを入力します:

```bash
# Jupyter Notebook の設定ファイルを生成
jupyter notebook --generate-config
```

```

以下のようなメッセージが表示され、設定ファイルはログインユーザーごとに異なります：
```bash
Writing default config to: /home/cat/.jupyter/jupyter_notebook_config.py
```

このメッセージから、デフォルトの設定ファイルが現在ログインしているユーザー（`cat`）のディレクトリに生成されたことがわかります。そのディレクトリに移動して、対応する設定ファイルを開き、編集します。

```bash
# 設定ファイルを変更する前に、Jupyter Notebook の作業ディレクトリを作成
mkdir /home/cat/jupyter
# vim エディタを使用して設定ファイルを開く
vim /home/cat/.jupyter/jupyter_notebook_config.py
```

設定ファイルを開いたら（`a`キーを押して編集モードに入る）以下の設定を行います：
```python
c.NotebookApp.allow_remote_access = True  # リモートアクセスを許可
c.NotebookApp.ip = '*'  # すべての IP からのアクセスをリスン
c.NotebookApp.notebook_dir = '/home/cat/jupyter'  # jupyter の作業ディレクトリ、存在しない場合は作成する必要がある
c.NotebookApp.token = ''  # トークンキーなしでアクセス（セキュリティが低いが便利）
```

デフォルト設定では、すべてのオプションが`#`でコメントアウトされています。必要なオプションのコメントを解除し、上記の設定内容に従って設定を変更します。または、直接上記の 4 行を追加して保存します。

重要：`c.NotebookApp.notebook_dir`オプションで指定されたディレクトリは Jupyter Notebook 起動時の作業ディレクトリであり、そのディレクトリは存在する必要があります。存在しない場合は先に作成してください。

13.2.3 安全なログイン設定

設定ファイルの例で、トークンログインオプションを無効にしましたが、これは利便性のためであり、安全性は低いです。任意のユーザーが Jupyter Notebook サービスにログインでき、ファイルの安全性が保障されません。安全な運用環境が必要な場合は、以下の手順に従ってください。必要ない場合はこのセクションをスキップしてください：

```bash
# 端末で以下のコマンドを入力：
```
```

```
jupyter server password
```

`Enter password:` が表示されたら、任意のパスワードを入力し、生成されたキーをコピーして保存します。

```python
[JupyterPasswordApp] Wrote hashed password to /home/cat/.jupyter/jupyter_server_config.json
```

生成された HASH パスワードは上記ファイルに保存されています。

```bash
# vim エディタを使用して設定ファイルを開く
vim /home/cat/.jupyter/jupyter_notebook_config.py
# 次にトークン設定オプションをコメントアウト
# c.NotebookApp.token = ''
# パスワードログインを追加し、生成されたパスワードを入力
c.NotebookApp.password = 'sha1:274e5a04060e:e65098703b7e03836b01c261b6ca6d959049a3e9'
```

これでソフトウェアのインストールが完了しました。次に、Jupyter Notebook を起動して使用します。

13.3 Jupyter Notebook 使用前の注意事項

重要: Jupyter Notebook で外部デバイスを操作する際は、本セクションの内容を必ず参照してください。

Python で一部の外部デバイスを操作する際には root ユーザー権限が必要です。同様に、Jupyter Notebook で外部デバイスを操作する際にも root 権限が必要です。解決方法として、root ユーザーでログインし、root ユーザーとして Jupyter Notebook を起動します。

注意点として、Jupyter Notebook の起動時には環境設定がチェックされ、ユーザーごとに異なる設定ファイルが存在します！したがって、root ユーザーとして Jupyter Notebook を起動する前に、root ユーザーで設定手順を再度行う必要があります。すなわち、root ユーザーで Jupyter Notebook を起動する前に、root ユーザーでログインし、`jupyter notebook --generate-config` コマンドを使用して設定ファイルを生成し、Jupyter Notebook 環境を設定します。

13.4 Jupyter Notebook の使用

13.4.1 ソフトウェアの起動

以下のコマンドを入力して、ソフトウェアを起動します。

```bash
# 端末で以下のコマンドを入力して Jupyter Notebook を起動:
# root ユーザーの使用を許可し、起動時にブラウザを開かない
jupyter notebook --no-browser --allow-root
```

ソフトウェアが起動すると、以下のようなメッセージが表示されます：

表示されたメッセージに従って、ブラウザに対応する URL を入力することで、Jupyter Notebook の機能を使用できます。LubanCat4 ボードを例にすると、ボードには 2 つのネットワークポートがあり、ネットワークケーブルを接続するだけで、PC とボードが同じネットワークセグメントにあります。

```
cat@lubancat:~/python/code$ ifconfig
eth0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.11.241  netmask 255.255.254.0  broadcast 192.168.11.255
        inet6 fe80::1aa8:283c:fb91:e9fb  prefixlen 64  scopeid 0x20<link>
        inet6 2400:4051:da0:a500:e253:3a0d:9da2:1414  prefixlen 64  scopeid 0x0<global>
        ether ce:54:18:ce:97:a1  txqueuelen 1000  (イーサネット)
        RX packets 96112  bytes 90213121 (86.0 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 52859  bytes 15416418 (14.7 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
        device interrupt 87

lo: flags=73<UP,LOOPBACK,RUNNING>  mtu 65536
        inet 127.0.0.1  netmask 255.0.0.0
        inet6 ::1  prefixlen 128  scopeid 0x10<host>
        loop  txqueuelen 1000  (ローカルループバック)
        RX packets 27168  bytes 6596446 (6.2 MiB)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 27168  bytes 6596446 (6.2 MiB)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0

usb0: flags=4099<UP,BROADCAST,MULTICAST>  mtu 1500
        ether 8e:7b:eb:38:45:7c  txqueuelen 1000  (イーサネット)
        RX packets 0  bytes 0 (0.0 B)
        RX errors 0  dropped 0  overruns 0  frame 0
        TX packets 0  bytes 0 (0.0 B)
        TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

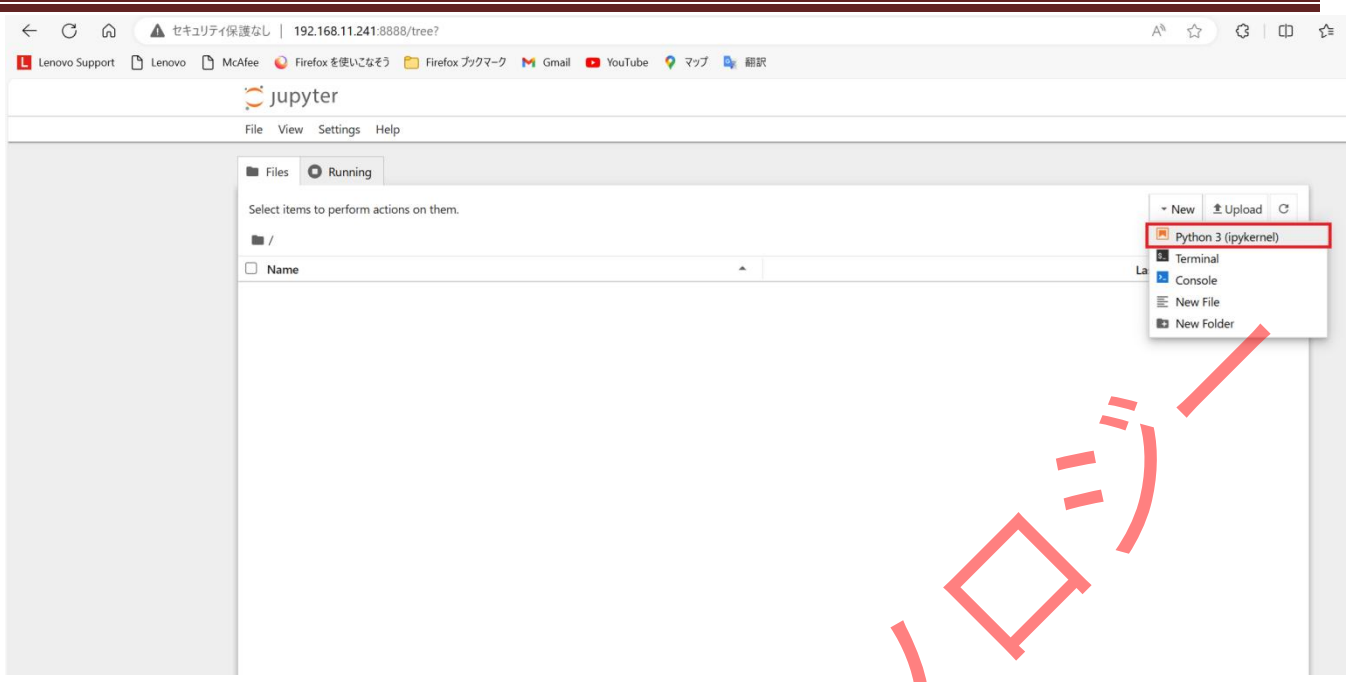
PC のブラウザに以下の URL を入力します（IP アドレスは実際のボードに合わせて変更）：

```bash
http://192.168.11.241:8888/
```

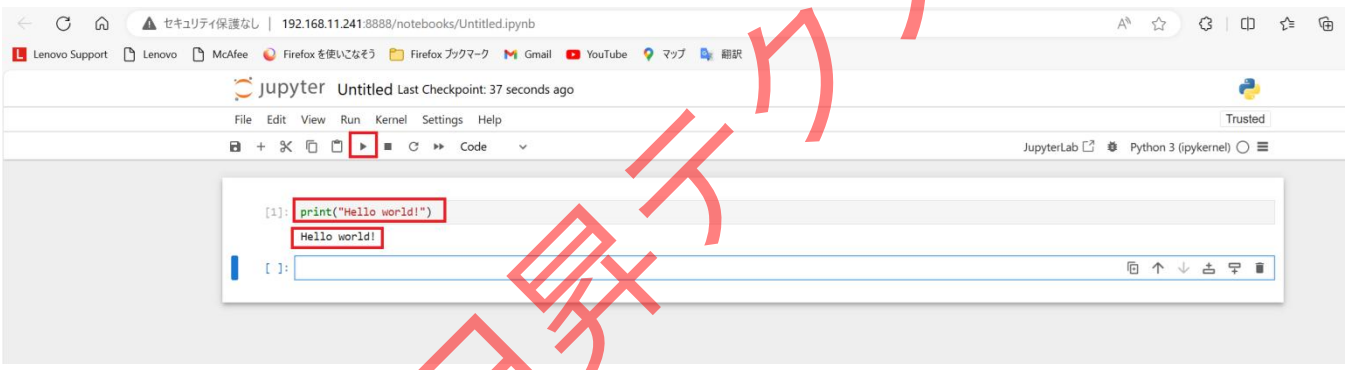
この URL で Jupyter Notebook にアクセスできます。設定したパスワードを入力し、ログインすると起動画面が表示されます：

```
cat@lubancat:~/python/code/io/camera$ jupyter notebook --no-browser --allow-root
[I 2024-05-31 22:36:58.288 ServerApp] jupyter_lsp | extension was successfully linked.
[I 2024-05-31 22:36:58.298 ServerApp] jupyter_server_terminals | extension was successfully linked.
[I 2024-05-31 22:36:58.304 ServerApp] jupyterlab | extension was successfully linked.
[W 2024-05-31 22:36:58.307 JupyterNotebookApp] 'allow_remote_access' has moved from NotebookApp to ServerApp. This config will be passed to ServerApp. Be sure to update your config before our next
 release.
[W 2024-05-31 22:36:58.307 JupyterNotebookApp] 'ip' has moved from NotebookApp to ServerApp. This config will be passed to ServerApp. Be sure to update your config before our next release.
[W 2024-05-31 22:36:58.307 JupyterNotebookApp] 'notebook_dir' has moved from NotebookApp to ServerApp. This config will be passed to ServerApp. Be sure to update your config before our next releas
e.
[W 2024-05-31 22:36:58.307 JupyterNotebookApp] 'password' has moved from NotebookApp to ServerApp. This config will be passed to ServerApp. Be sure to update your config before our next release.
[W 2024-05-31 22:36:58.311 ServerApp] notebook_dir is deprecated, use root_dir
[W 2024-05-31 22:36:58.311 ServerApp] ServerApp.password config is deprecated in 2.0. Use PasswordIdentityProvider.hashed_password.
[I 2024-05-31 22:36:58.311 ServerApp] notebook | extension was successfully linked.
[I 2024-05-31 22:36:58.313 ServerApp] Writing Jupyter server cookie secret to /home/cat/.local/share/jupyter/runtime/jupyter_cookie_secret
[I 2024-05-31 22:36:58.740 ServerApp] notebook_shim | extension was successfully linked.
[W 2024-05-31 22:36:58.784 ServerApp] WARNING: The Jupyter server is listening on all IP addresses and not using encryption. This is not recommended.
[I 2024-05-31 22:36:58.785 ServerApp] notebook_shim | extension was successfully loaded.
[I 2024-05-31 22:36:58.789 ServerApp] jupyter_lsp | extension was successfully loaded.
[I 2024-05-31 22:36:58.790 ServerApp] jupyter_server_terminals | extension was successfully loaded.
[I 2024-05-31 22:36:58.792 LabApp] JupyterLab extension loaded from /home/cat/.local/lib/python3.9/site-packages/jupyterlab
[I 2024-05-31 22:36:58.792 LabApp] JupyterLab application directory is /home/cat/.local/share/jupyter/lab
[I 2024-05-31 22:36:58.793 LabApp] Extension Manager is 'pypi'.
[I 2024-05-31 22:36:58.808 ServerApp] jupyterlab | extension was successfully loaded.
[I 2024-05-31 22:36:58.812 ServerApp] notebook | extension was successfully loaded.
[I 2024-05-31 22:36:58.814 ServerApp] Serving notebooks from local directory: /home/cat/jupyter
[I 2024-05-31 22:36:58.814 ServerApp] Jupyter Server 2.14.1 is running at:
[I 2024-05-31 22:36:58.814 ServerApp] http://localhost:8888/tree
[I 2024-05-31 22:36:58.814 ServerApp]     http://127.0.0.1:8888/tree
[I 2024-05-31 22:36:58.814 ServerApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
```

起動後、Jupyter Notebook を試してみて、簡単なコードを書いてテストを行います：



図一



図二

13.4.2 コードのアップロード

他の人が書いたコードを使用することもできます。対応するファイルをダウンロードして jupyter にアップロードするだけで使用できます。

13.5 参考リンク

https://docs.jupyter.org/en/latest/](https://docs.jupyter.org/en/latest/

第 14 章 PIL ライブラリ

14.1 PIL ライブラリの紹介

PIL（Python Image Library）は、Python の強力な画像処理ライブラリです。このライブラリを使用すると、画像に対して非常に多くの処理を行うことができます。LubanCat ボードに Python-PIL ライブラリをインストールし、jupyter を使っていくつかのテストコードを書いてこのライブラリを利用することができます。

14.2 PIL ライブラリのインストール

apt ツールを使用してインストールします。

```bash
# 端末で以下のコマンドを入力して PIL ライブラリをインストール：
sudo apt -y install python3-pil
```

インストールが完了するまで待ちます。

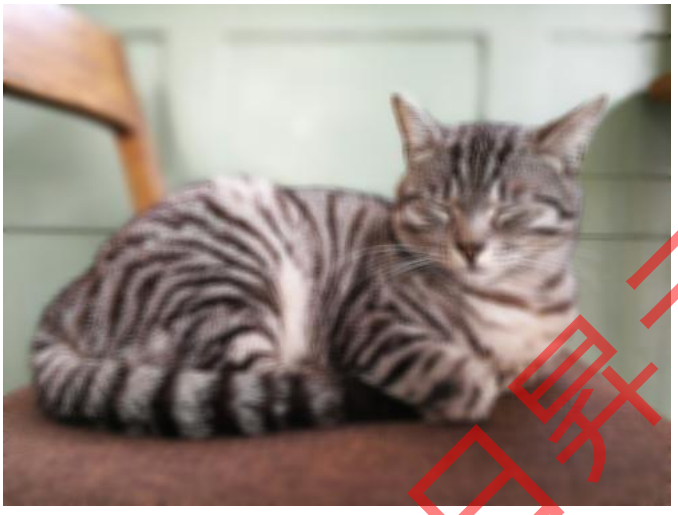
14.3 PIL ライブラリの使用

ライブラリをインストールした後、前述の jupyter を使ってテストコードを書きます。テストコードは以下の通りです：

効果は以下の通りです：

原画像：

コードの実行後：



対応するコードディレクトリ `libdemo¥PIL¥PIL.ipynb` ファイルを jupyter にアップロードして使用します。

Python 画像処理ライブラリの使用方法については、各自で探索してください。Python3 以降のバージョンでは、画像処理ライブラリは pillow であり、PIL を基にして互換性を持たせたバージョンです。

第 15 章 Numpy ライブラリ

15.1 Numpy ライブラリの紹介

NumPy（Numerical Python）は、Python の強力な数値計算拡張ライブラリで、多次元配列や行列演算をサポートしています。また、配列演算のための多くの数学関数ライブラリも提供しています。科学計算、人工知能、画像処理などの多くの Python アプリケーション分野で Numpy が使用されています。LubanCat ボードに Python-NumPy ライブラリをインストールし、jupyter を使っていくつかのテストコードを書いてこのライブラリを利用することができます。

15.2 NumPy ライブラリのインストール

apt ツールを使用してインストールします。ここでは Python3 バージョンの Numpy ライブラリをインストールします。Python2 環境の NumPy ライブラリが必要な場合は、以下のコマンドから`3`を削除してください。

```bash
# 端末で以下のコマンドを入力して numpy ライブラリをインストールし、インストールが完了するまで待ちます：
sudo apt -y install python3-numpy
```

15.3 NumPy ライブラリの使用

ライブラリをインストールした後、前述の jupyter を使ってテストコードを書きます（jupyter のインストールについては前述の Jupyter Notebook 章を参照）。テストコードおよび効果は以下の通りです：

```
import numpy as np
```

```
test_array = np.array([1,2,3]) # 一次元配列
print(test_array)
```

```
[1 2 3]
```

```
test_2dimension = np.array(([1,2,3],[4,5,6])) # 二次元配列
print(test_2dimension)
print("データ型", type(test_2dimension))           # 配列のデータ型を表示する
print("配列要素のデータ型：", test_2dimension.dtype) # 配列要素のデータ型を表示する
print("配列のサイズ：", test_2dimension.size)      # 配列のサイズ、すなわち配列要素の総数を表示する
print("配列の形状：", test_2dimension.shape)         # 配列の形状を表示する
print("配列の次元数", test_2dimension.ndim)      # 配列の次元数を表示する
print("各要素のバイト数", test_2dimension.itemsize)  # 各要素のバイト数を表示する
```

```
[[1 2 3]
 [4 5 6]]
データ型 <class 'numpy.ndarray'>
配列要素のデータ型： int64
配列のサイズ： 6
配列の形状： (2, 3)
配列の次元数 2
各要素のバイト数 8
```

```
test_3dimension = np.array((([1,2,3],[4,5,6]),([1,2,3],[4,5,6]))) # 三次元配列
print(test_3dimension)
print("配列のサイズ：", test_3dimension.size)      # 配列のサイズ、すなわち配列要素の総数を表示する
print("配列の形状：", test_3dimension.shape)         # 配列の形状を表示する
print("配列の次元数", test_3dimension.ndim)      # 配列の次元数を表示する
print("各要素のバイト数", test_3dimension.itemsize)  # 各要素のバイト数を表示する
```

```
[[[1 2 3]
  [4 5 6]]

 [[1 2 3]
  [4 5 6]]]
配列のサイズ： 12
配列の形状： (2, 2, 3)
配列の次元数 3
各要素のバイト数 8
```

対応するコードディレクトリ `libdemo¥Numpy¥Numpy.ipynb` ファイルを jupyter にアップロードして使用します。

Python Numpy ライブラリの使用については、NumPy チュートリアルを参照してください。

第 16 章 Pandas ライブラリ

16.1 Pandas ライブラリの紹介

Pandas は NumPy に基づいたツールで、データ分析の分野でよく使用されます。Pandas はさまざまな雑多なデータや表形式のデータを処理するためのツールを提供しています。このライブラリを使用すると、大規模なデータセットの操作に必要な多くのツールを利用できます。LubanCat ボードに Python-Pandas ライブラリをインストールし、jupyter を使っていくつかのテストコードを書いてこのライブラリを利用することができます。

16.2 Pandas ライブラリのインストール

apt ツールを使用してインストールします。

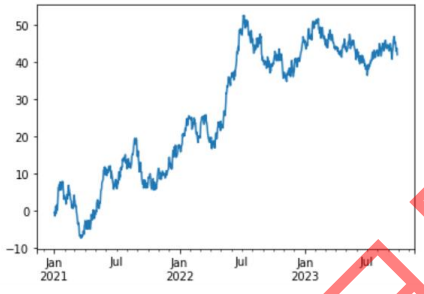
```bash
# 端末で以下のコマンドを入力して Pandas ライブラリをインストール：
sudo apt -y install python3-pandas
```

インストールが完了するまで待ちます。

16.3 Pandas ライブラリの使用

ライブラリをインストールした後、前述の jupyter を使ってテストコードを書きます。テストコードおよび効果は以下の通りです：

```
[2]: import numpy as np
     import pandas as pd

[8]: data_set = pd.Series(np.random.randn(1000),index=pd.date_range("1/1/2021",periods=1000))
     print(data_set)

2021-01-01   -0.616727
2021-01-02    0.174395
2021-01-03   -1.046284
2021-01-04    1.408171
2021-01-05    1.163650
2021-01-06   -1.192502
2021-01-07   -0.730532
2021-01-08    2.255443
2021-01-09   -0.332336
2021-01-10   -1.009804
2021-01-11    2.235770
2021-01-12    1.951027
2021-01-13    1.758318
2021-01-14   -0.059735
2021-01-15    0.945016
2021-01-16   -0.801716
2021-01-17   -0.685644
2021-01-18    2.255487

[10]: data_set.plot()

[10]: <matplotlib.axes._subplots.AxesSubplot at 0x6dfa3cd0>
```



対応するコードディレクトリ `libdemo¥Pandas¥Pandas.ipynb` ファイルを jupyter にアップロードして使用します。

Python Pandas ライブラリの使用については、Pandas チュートリアルを参照してください。

第 17 章 Web ライブラリ - Flask

17.1 Flask ライブラリの紹介

Flask は、Python で Web アプリケーションを構築する際に最も一般的な Web フレームワークの一つです。Django と同様ですが、Flask は非常に軽量で、基本的な Web アプリケーションを簡単に構築できます。

私たちは LubanCat ボードに Python-flask ライブラリをインストールし、いくつかのテストコードを書くことでこのライブラリを利用できます。ボード上に簡単な Web ページをデプロイしましょう。

17.2 Flask ライブラリのインストール

apt ツールを使用してインストールします。

```
# 以下のコマンドをターミナルに入力して、Flask ライブラリをインストールします：
sudo apt -y install python3-flask
# または
pip3 install flask
```

インストールが完了したら、Flask を使用できます。

17.3 Flask ライブラリの使用

対応するライブラリをインストールした後、Flask を利用して簡単な Web ページをデプロイできます。

17.3.1 プロジェクトディレクトリの新規作成

まず、プロジェクト管理のために新しいディレクトリを作成し、ファイル構成を簡潔にします。

```
# Web プロジェクトを格納するためのディレクトリを作成し、そのディレクトリに移動します。
mkdir webapp && cd webapp
# webapp のディレクトリを作成します。
mkdir -p app/static
mkdir app/templates
mkdir tmp
```

17.3.2 コードファイルの追加

17.3.2.1 __init__.py

webapp_flask パッケージの初期化スクリプトを作成し、先ほど作成した app ディレクトリに配置しま

す。
```
# app ディレクトリに __init__.py を新規作成します。
vim __init__.py
```
内容は以下の通りです：
```
__init__.py
from flask import Flask

app = Flask(__name__)
from app import view
```

17.3.2.2 views.py

Flask では、ビューは Python 関数として記述されます。各ビュー関数は 1 つまたは複数の URL にマッピングされます。最初のビュー関数 views.py を作成し、app ディレクトリに配置します。
```
# app ディレクトリに view.py を新規作成します。
vim view.py
```
内容は以下の通りです：
```
view.py
from app import app

@app.route('/')
@app.route('/index')
def index():
return "Hello, Lubancat!"
```

17.3.2.3 run.py

最後に、アプリケーションの Web サーバーを起動するためのスクリプト run.py を作成し、webapp ディレクトリ（プロジェクトの最上位ディレクトリ）に配置します。
```
# webapp ディレクトリに run.py を新規作成します。
nano run.py
```
内容は以下の通りです：
```
run.py
```

```
#!usr/bin/python3
from app import app
# デバッグモード、ホストアドレス"192.168.11.241" でアクセス可能
app.run(debug=True, host="192.168.11.241")
```

run.py を作成した後、実行権限を付与します：

```
# 権限を追加します。
chmod a+x run.py
```

最終的なファイルディレクトリ構成は以下の通りです。

17.3.3 webapp の実行

上記の操作を経て、最も簡単な webapp が完成しました。次にアプリケーションを起動し、ブラウザで確認します。

```
# アプリケーションを実行します。
./run.py
```

テストコードと結果は以下の通りです：

アプリケーションは正常に起動し、ターミナルにはサービス情報が表示されています。情報には、web がホスト 192.168.11.241 のポート番号 5000 にマッピングされたと記載されていますので、http://192.168.11.241:5000/ でデプロイした Web ページにアクセスできます。

みなさんは、libdemo¥Flask¥webapp_flask ディレクトリのコードを LubanCat ボードにアップロードしてテストできます。

17.3.4 参考文献

Flask 公式ドキュメント

第 18 章 Web ライブラリ - Django

18.1 Django ライブラリの紹介

Django は Python プログラミング言語で動作するオープンソースのモデル-ビュー-コントローラー（MVC）スタイルの Web アプリケーションフレームワークで、Web 開発の主流フレームワークの一つです。LubanCat ボードに Python-Django ライブラリをインストールし、簡単な操作でこのライブラリを使用できます。ボード上に簡単な Web ページをデプロイしましょう。

18.2 Django ライブラリのインストール

apt ツールを使用してインストールします。

```
# 以下のコマンドをターミナルに入力して、Django ライブラリをインストールします：
sudo apt -y install python3-django
```

インストールが完了したら、Django を使用できます。

18.3 Django ライブラリの使用

該当のライブラリをインストールした後、Django を利用して簡単な Web ページをデプロイできます。

18.3.1 プロジェクトディレクトリの新規作成

まず、プロジェクト管理のために新しいディレクトリを作成し、ファイル構成を簡潔にします。

```
# Web プロジェクトを格納するためのディレクトリを作成し、そのディレクトリに移動します。
mkdir webapp
cd webapp
```

18.3.2 コードファイルと設定の追加

Django の管理ツール django-admin を使用して、新しいプロジェクトテンプレートをすばやく作成できます。

```
# 以下のコマンドをターミナルに入力します：
django-admin startproject Lubancat
# コマンドが実行完了すると、現在のディレクトリにプロジェクトフォルダ Lubancat が自動的に生成されます。そのフォルダに移動します。
cd ./Lubancat/
```

Django の管理ツールはすでにプロジェクトテンプレートを生成しています。ディレクトリには以下の内容が表示されます：

次に、外部ホストから LubanCat ボードにアクセスできるようにプロジェクトの設定ファイルを変更します。プロジェクトフォルダ Lubancat の Lubancat ディレクトリに移動し、settings.py ファイルを編集します。

```
# ターミナルに以下のコマンドを入力します：
cd ./Lubancat/
# 設定ファイルを編集します
vim settings.py
# ALLOWED_HOSTS 設定を見つけ、以下のように変更し、保存して終了します：
ALLOWED_HOSTS = ['*']
```

18.3.3 webapp の起動

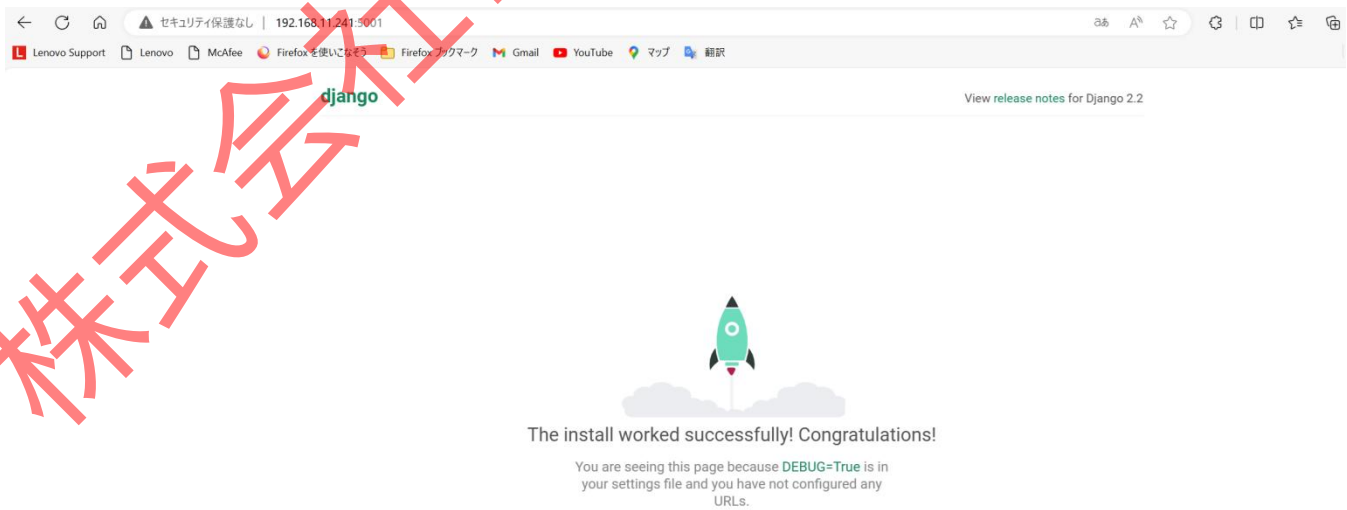
プロジェクトフォルダ Lubancat に戻り、manage.py ファイルを実行してサーバーを起動します：

```
# ターミナルに以下のコマンドを入力します。IP アドレスはボードの具体的な IP に応じて変更します：
sudo python3 manage.py runserver 192.168.11.241:5001
```

少し待つと、サーバーの起動情報が正常に表示されます。

結果は以下の通りです：

webapp は正常に起動し、ターミナルにはサービス情報が表示されています。情報には、web がホスト 192.168.103.115 のポート番号 5001 にマッピングされたと記載されていますので、http://192.168.103.115:5001/ でデプロイした Web ページにアクセスできます。



18.3.4 Django バージョンの注意点

注：LubanCat ボードシステムで apt 方法を使用してインストールされた Django はバージョン 2.2.28 です。特定のバージョンの Django をインストールする

必要がある場合は、pip を使用してください。以下を参照してください：

```
# ターミナルに以下のコマンドを入力して、指定バージョンの Django をインストールします：
pip3 install Django==3.1.7
```

インストールが完了したら、管理ツールのディレクトリを環境変数に追加するか、管理ツールを環境変数に配置します。

```
# ターミナルに以下のコマンドを入力して、管理ツールを環境変数に追加します：
sudo cp /home/cat/.local/bin/django-admin /usr/bin/
```

Python Django ライブラリの使用に関する詳細は、Django 公式ドキュメントを参照してください。

第 19 章 Modbus プロトコル - pymodbus ライブラリ

前章で、LubanCat ボードのさまざまな外部接続インターフェイスを紹介しましたが、その中には産業分野で広く利用されているインターフェイスも含まれています。それでは、LubanCat ボードはこれらのインターフェイスを基に、さらに強力な機能を拡張することができるのでしょうか?

この章の実験では、産業分野で非常に一般的な産業バスプロトコルである Modbus 通信プロトコルを紹介します。例として Modbus-RTU を使用します。

19.1 Modbus プロトコルの実験

LubanCat ボードシステムには豊富なオープンソースソフトウェアのサポートがあります。この章で紹介する Modbus 通信プロトコルにも対応するソフトウェアをインストールして使用できます。

この章の実験では、Python 言語と Python ライブラリ python3-pymodbus を使用して、Modbus プロトコルで定義された RTU マスターステーションとスレーブステーション間の簡単な通信を行います。

まず、python3-pymodbus ライブラリについて簡単に説明します。

19.2 python3-pymodbus ライブラリ

pymodbus は BSD オープンソースライセンスに基づいた Modbus プロトコルの Python ライブラリです。このライブラリは非常に強力で、Modbus プロトコルで定義されたすべての機能を実装しており、同期および非同期（asyncio、tornado、twisted）での通信をサポートしています。これにより、優れたパフォーマンスを持ちながら、Python 開発者が Modbus プロトコルアプリケーションを構築する際に追加の機能拡張を行うための多くの可能性を提供します。

pymodbus は Ethernet とシリアルインターフェースを使用して Modbus プロトコルの物理層をサポートします（pymodbus のシリアルインターフェースの実装は pyserial ライブラリに依存しています）。シリアルインターフェースを使用しない場合、python3-pymodbus ライブラリは他のサードパーティライブラリ

には依存しないため、このライブラリは非常に軽量です。

LubanCat ボードに python3-pymodbus ライブラリをインストールし、公式のサンプルコードを使用してこのライブラリを利用し、この章の実験を行います。

19.3 実験の準備

19.3.1 ボードの uart リソースの追加

重要: この章の実験では、1 つの uart シリアルポート（スレーブ）と PC 側（マスター）を使用して Modbus のハードウェア接続を実現します。このため、uart リソースの追加を示します。pymodbus ライブラリは Modbus プロトコルのフルスタック実装を提供しているため、Ethernet 通信もサポートされます。実際の状況に応じて、この章の内容を参考にしてください。

ボード上の一部の GPIO は有効になっていない場合があります。デバイスツリーノードやデバイスツリープラグインのロードをコメントアウトすることで有効にできます。

基板の 40 ピン GPIO 図：（参考マニュアル：《AI エッジ基板 CSAIEG358803 ハードウェア仕様書》）

LubanCat4 ピン図

拡張機能3	拡張機能2	拡張機能1	共通機能	No.	GPIO	物理ピン		GPIO	No.	共通機能	拡張機能1	拡張機能2	拡張機能3
					3.3V	1	2	5V					
			I2C5_SDA_M3	47	GPIO1_B7	3	4	5V					
			I2C5_SCL_M3	46	GPIO1_B6	5	6	GND					
	I2C2_SDA_M4	UART6_RX_M1		32	GPIO1_A0	7	8	GPIO4_A3	131	UART0_TX_M2			
					GND	9	10	GPIO4_A4	132	UART0_RX_M2			
	I2C2_SCL_M4	UART6_TX_M1		33	GPIO1_A1	11	12	GPIO1_D6	62	PWM14_M2	I2C8_SCL_M2		
	PDM1_SDI0_M1	PWM3_IR_M3		39	GPIO1_A7	13	14	GND					
	PDM1_SDI1_M1			40	GPIO1_B0	15	16	GPIO3_C1	113			UART7_RX_M1	SPI1_CLK_M1
					3.3V	17	18	GPIO3_D2	122				UART9_RTSN_M2
	PDM1_SDI3_M1	UART4_RX_M2	SPI0_MOSI_M2	42	GPIO1_B2	19	20	GND					
	PDM1_SDI2_M1		SPI0_MISO_M2	41	GPIO1_B1	21	22	GPIO3_D4	124		UART9_RX_M2		
	PDM1_CLK1_M1	UART4_TX_M2	SPI0_CLK_M2	43	GPIO1_B3	23	24	GPIO1_B4	44	SPI0_CS0_M2	UART7_RX_M2	PDM1_CLK0_M1	
					GND	25	26	GPIO1_B5	45	SPI0_CS1_M2	UART7_TX_M2		
			I2C6_SDA_M3	136	GPIO4_B0	27	28	GPIO4_B1	137	I2C6_SCL_M3			
				102	GPIO3_A6	29	30	GND					
SPI1_MOSI_M1		I2C3_SCL_M1		111	GPIO3_B7	31	32	GPIO1_D7	63	PWM15_IR_M3	I2C8_SDA_M2		
UART9_CTSN_M2			PWM10_M2	123	GPIO3_D3	33	34	GND					
		UART9_TX_M2	PWM11_IR_M3	125	GPIO3_D5	35	36	GPIO4_A0	128		SPI0_MISO_M1		UART9_RTSN_M1
SPI1_MISO_M1	UART7_TX_M1	I2C3_SDA_M1		112	GPIO3_C0	37	38	GPIO4_A1	129		SPI0_MOSI_M1		UART9_CTSN_M1
					GND	39	40	GPIO4_A2	130		SPI0_CLK_M1		

この章の実験では、LubanCat4 ボードを例に使用します。40 ピンのピンアウト図に基づいて、UART3 を使用することができます。以下は UART0 のデバイスツリープラグインを有効にする手順です：

```
# 使用するボードによって設定ファイルと pwm が異なるため、LubanCat4 を例に説明します。設定ファイルは uEnvLubanCat4.txt です。
# 以下のコマンドを使用します：
sudo nano /boot/uEnv/uEnvLubanCat4.txt
# 編集モードに入り、uart0 の前のコメントを外し、保存してファイルを終了し、システムを再起動します。
```

```
GNU nano 5.4                                    /boot/uEnv/uEnvLubanCat4.txt

enable_uboot_overlays=1
#overlay_start

#dtoverlay=/dtb/overlay/rk3588-lubancat-i2c2-m4-overlay.dtbo
dtoverlay=/dtb/overlay/rk3588-lubancat-i2c3-m1-overlay.dtbo
dtoverlay=/dtb/overlay/rk3588-lubancat-i2c5-m3-overlay.dtbo
#dtoverlay=/dtb/overlay/rk3588-lubancat-i2c6-m3-overlay.dtbo
#dtoverlay=/dtb/overlay/rk3588-lubancat-i2c8-m2-overlay.dtbo
dtoverlay=/dtb/overlay/rk3588-lubancat-pwm3-ir-m3-overlay.dtbo
dtoverlay=/dtb/overlay/rk3588-lubancat-pwm10-m2-overlay.dtbo
dtoverlay=/dtb/overlay/rk3588-lubancat-pwm11-ir-m3-overlay.dtbo
dtoverlay=/dtb/overlay/rk3588-lubancat-pwm14-m2-overlay.dtbo
dtoverlay=/dtb/overlay/rk3588-lubancat-pwm15-ir-m3-overlay.dtbo
dtoverlay=/dtb/overlay/rk3588-lubancat-spi0-m2-overlay.dtbo
dtoverlay=/dtb/overlay/rk3588-lubancat-uart0-m2-overlay.dtbo
#dtoverlay=/dtb/overlay/rk3588-lubancat-uart4-m2-overlay.dtbo
#dtoverlay=/dtb/overlay/rk3588-lubancat-uart6-m1-overlay.dtbo
#dtoverlay=/dtb/overlay/rk3588-lubancat-uart7-m1-overlay.dtbo
#dtoverlay=/dtb/overlay/rk3588-lubancat-uart7-m2-overlay.dtbo
#dtoverlay=/dtb/overlay/rk3588-lubancat-uart9-m2-overlay.dtbo
#dtoverlay=/dtb/overlay/rk3588-lubancat-can0-m0-overlay.dtbo
#dtoverlay=/dtb/overlay/rk3588-lubancat-can2-m0-overlay.dtbo
```

```
他の LubanCat RK シリーズボードを使用する場合も、同様の操作を行います。
```

```
# ターミナルに以下のコマンドを入力して、UART リソースを確認できます：
ls /dev/ttyS*
```

```
cat@lubancat:~/python/code$ ls /dev/ttyS*
/dev/ttyS0
cat@lubancat:~/python/code$
```

ボード上の UART リソースの例です。参考までに：
その中の /dev/ttyS0 がボード上の uart0 リソースに対応しています。

19.3.2 ハードウェア接続

ハードウェア接続のセクションでは、開発環境などの状況に応じてこの章の内容を参考にして調整してください。この章では、LubanCat4 の uart0 を使用してデータ転送を行います。接続は次のようになります（UART 通信の章と同じです）：

USB to TTL でコンピュータに接続します。

重要：pymodbus ライブラリのシリアルデバイスのサポートは pyserial ライブラリに基づいており、デフォルトでは uart をサポートしています。この章では uart を使用して実験を行います。pymodbus を使用して rs485 の物理接続方式を使用する場合は、pymodbus ライブラリのソースコードを修正する必要があります。

19.3.3 pymodbus ライブラリのインストール

```
# ターミナルに以下のコマンドを入力します：
sudo pip3 install -U pymodbus
# またはリポジトリソースを直接クローンして、setuptools ツールを使用してインストールします。前の Python インストール章を参考にしてください。
git clone https://github.com/riptideio/pymodbus.git
```

19.4 テストコード

pymodbus ライブラリをインストールしたら、pymodbus ライブラリを使用してプログラムを書くことができます。ここでは公式のサンプルコードを直接修正し、簡単なテストを行います。このデモでは、pymodbus の asyncio 非同期ライブラリに基づく Modbus プロトコル RTU 通信とシリアル非同期マスターを使用します。コードでは、ハードウェア構成のセクションで有効にしたシリアルポート '/dev/ttyS0' を外部通信ポートとして設定し、ボーレートを 115200 に設定します。

19.4.1 RTU スレーブ slave_server.py

コードは以下の通りです：

ソースコード slave_server.py

```
#!/usr/bin/env python3
"""Pymodbus asynchronous Server Example.

An example of a multi threaded asynchronous server.

usage: slave_server.py [-h] [--comm {serial}]
                       [--framer {rtu}]
                       [--log {critical,error,warning,info,debug}]
                       [--port PORT] [--store {sequential}]
                       [--slaves SLAVES]


Command line options for examples

options:
  -h, --help            show this help message and exit
  --comm {serial}
                        "serial"
  --framer {rtu}
                        "rtu"
  --log {critical,error,warning,info,debug}
                        "critical", "error", "warning", "info" or "debug"
  --port PORT           the port to use
  --store {sequential}
                        "sequential"
  --slaves SLAVES       number of slaves to respond to

The corresponding client can be started as:
    python3 client_sync.py
"""
import asyncio
import logging
```

```
import argparse
import os


from pymodbus import pymodbus_apply_logging_config
from pymodbus.transaction import (
    ModbusRtuFramer,
)
from pymodbus.datastore import (
    ModbusSequentialDataBlock,
    ModbusServerContext,
    ModbusSlaveContext,
    ModbusSparseDataBlock,
)
from pymodbus.device import ModbusDeviceIdentification


from pymodbus.server import (
    StartAsyncSerialServer,
)
#from pymodbus.version import version
from pymodbus import __version__ as version



_logger = logging.getLogger()

def get_commandline(server=False, description=None, extras=None):
    """Read and validate command line arguments"""
    parser = argparse.ArgumentParser(description=description)
    parser.add_argument(
        "--comm",
        choices=["serial"],
        help="set communication",
        default="serial",
        type=str,
    )
    parser.add_argument(
        "--framer",
        choices=["rtu"],
        help="set framer, default depends on --comm",
        type=str,
    )
    parser.add_argument(
        "--log",
        choices=["critical", "error", "warning", "info", "debug"],
        help="set log level, default is info",
        default="info",
        type=str,
    )
    parser.add_argument(
        "--port",
        help="set port",
        type=str,
    )
    if server:
        parser.add_argument(
            "--store",
            choices=["sequential"],
            help="set type of datastore",
            default="sequential",
            type=str,
        )
        parser.add_argument(
            "--slaves",
            help="set number of slaves, default is 0 (any)",
            default=0,
            type=int,
            nargs="+",
        )
    if extras:
        for extra in extras:
            parser.add_argument(extra[0], **extra[1])

    args = parser.parse_args()
```

```
    # set defaults
    comm_defaults = {
        "serial": ["rtu", "/dev/ptyp0"],
    }
    framers = {
        "rtu": ModbusRtuFramer,
    }
    pymodbus_apply_logging_config()
    _logger.setLevel(args.log.upper())
    args.framer = framers[args.framer or comm_defaults[args.comm][0]]
    args.port = args.port or comm_defaults[args.comm][1]
    if args.comm != "serial" and args.port:
        args.port = int(args.port)
    return args


def setup_server(args):
    """サーバー設定を行います。"""
    # データストアは初期化されたアドレスにのみ応答します
    # もし 0x00 から 0xFF のアドレスに DataBlock を初期化した場合、0x100 へのリクエストは無効なアドレス例外を返します。
    # 多くのデバイスがこのような動作を示すためです(すべてではありませんが)。
    _logger.info("### データストアを作成します")
    # co コイルレジスタを定義します。開始アドレスは 0、長さは 10、内容は 5 個の True と 5 個の False です。
    co_block = ModbusSequentialDataBlock(0, [True]*5 + [False]*5)
    # di 離散入力レジスタを定義します。開始アドレスは 0、長さは 10、内容は 5 個の True と 5 個の False です。
    di_block = ModbusSequentialDataBlock(0, [True]*5 + [False]*5)
    # ir 入力レジスタを定義します。開始アドレスは 0、長さは 10、内容は 0~9 の増加する数値リストです。
    ir_block = ModbusSequentialDataBlock(0, [0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
    # hr 保持レジスタを定義します。開始アドレスは 0、長さは 10、内容は 9~0 の減少する数値リストです。
    hr_block = ModbusSequentialDataBlock(0, [9, 8, 7, 6, 5, 4, 3, 2, 1, 0])

    if args.slaves:
        # サーバーは異なるユニット ID に対して異なるスレーブコンテキストを返すコンテキストを使用します。
        # デフォルトでは、すべてのユニット ID に対して同じコンテキストを返します(ブロードキャストモード)。
        # ただし、single フラグを False に設定し、ユニット ID からコンテキストへのマッピングの辞書を提供することで上書きできます。
        # スレーブコンテキストは zero_mode で初期化することもできます。これにより、アドレス(0-7)へのリクエストがアドレス(0-7)にマップされます。
        # デフォルトは False で、仕様書の第 4.4 節に基づいているため、アドレス(0-7)が(1-8)にマップされます。
        context = {
            0x01: ModbusSlaveContext(
                di=di_block,
                co=co_block,
                hr=hr_block,
                ir=ir_block,
                zero_mode=True
            ),
            0x02: ModbusSlaveContext(
                di=di_block,
                co=co_block,
                hr=hr_block,
                ir=ir_block,
            ),
            0x03: ModbusSlaveContext(
                di=di_block,
                co=co_block,
                hr=hr_block,
                ir=ir_block,
            ),
        }
        single = False
    else:
        context = ModbusSlaveContext(
            di=di_block, co=co_block, hr=hr_block, ir=ir_block, unit=1
        )
        single = True

    # データストレージを構築します
    args.context = ModbusServerContext(slaves=context, single=single)
    return args

async def run_async_server(args):
    """サーバーを実行します。"""
```

```
    txt = f"### 非同期サーバーを起動します。ポート {args.port} - {args.comm} でリッスン中"
    _logger.info(txt)
    # socat -d -d PTY,link=/tmp/ptyp0,raw,echo=0,ispeed=9600
    # PTY,link=/tmp/ttyp0,raw,echo=0,ospeed=9600
    server = await StartAsyncSerialServer(
        context=args.context, # データストレージ
        #identity=args.identity, # サーバー識別
        timeout=1, # リクエストが完了するまでの待ち時間
        port=args.port, # シリアルポート
        # custom_functions=[], # カスタムハンドリングを許可
        framer=args.framer, # 使用するフレーマー戦略
        # handler=None, # 各セッションのハンドラー
        stopbits=1, # 使用するストップビットの数
        bytesize=8, # シリアルメッセージのバイトサイズ
        # parity="N", # 使用するパリティの種類
        baudrate=115200, # シリアルデバイスのボーレート
        # handle_local_echo=False, # USB-to-RS485 アダプタのローカルエコーを処理
        # ignore_missing_slaves=True, # 存在しないスレーブへのリクエストを無視
        # broadcast_enable=False, # unit_id 0 をブロードキャストアドレスとして扱う
        # strict=True, # 厳密なタイミングを使用、Modbus RTU の t1.5 を使用
        # defer_start=False, # サーバーを定義するだけでアクティブにはしない
    )
    return server


if __name__ == "__main__":
    cmd_args = get_commandline(
        server=True,
        description="非同期サーバーを実行します。",
    )
    run_args = setup_server(cmd_args)
    asyncio.run(run_async_server(run_args), debug=True)
```

pymodbus を使用して Modbus スレーブデバイスの機能をシミュレートするには、Modbus プロトコルで定義されたスレーブコンテキスト環境を構築する必要があります。この環境は、Modbus スレーブの各機能（例えば、さまざまなレジスタ）を実装する必要があります。最終的に、このコンテキストはサーバーによって管理され、ホストからスレーブへのデータの保存と読み取りを実現します。pymodbus はさらにデータベースインターフェイスも実装しています。興味のある方は pymodbus のソースコードおよびサンプルをご覧ください。

19.4.2 実験手順

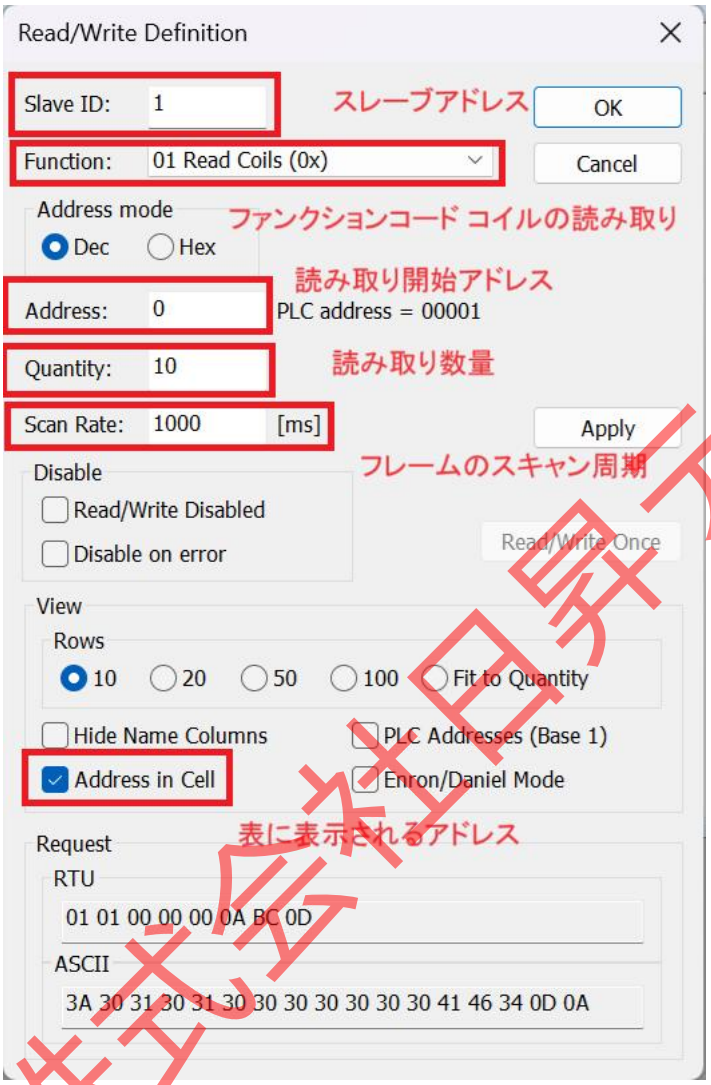
1. スレーブ端で、テストコードを LubanCat 2 ボードシステムにアップロードし、システムターミナルにログインしてスレーブサービスを起動します：

```
# ターミナルに以下のコマンドを入力します：
sudo python3 slave_server.py --comm serial --framer rtu --port /dev/ttyS0 --store sequential --log debug --slaves 1
#--comm 指定 serial 通信方式、--framer rtu フレーム指定、--port シリアルポート指定、--log デバッグ情報指定
# sudo python slave_server.py -h コマンドでヘルプを参照できます。
# コマンドを実行後、ターミナルには以下のように表示されます：
cat@lubancat:~/python/code/io/modbus$ sudo python3 slave_server.py --comm serial --framer rtu
```

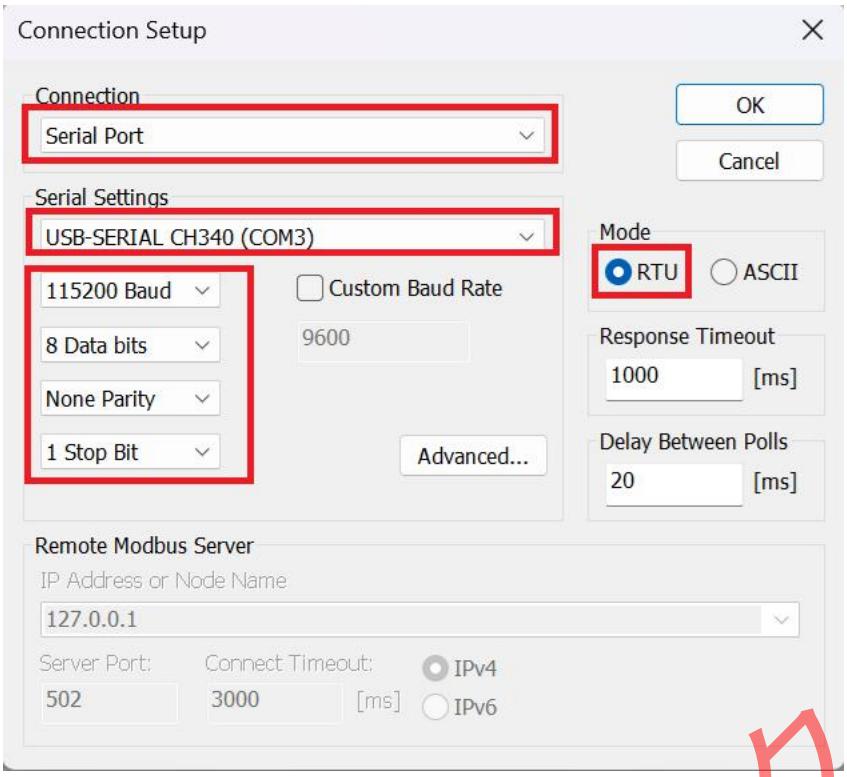
```
--port /dev/ttyS0 --store sequential --log debug --slaves 1
  2024-06-03 00:02:12,695 DEBUG logging:103 Awaiting connections server_listener
  2024-06-03 00:02:12,696 INFO  logging:97 Server listening.
  2024-06-03 00:02:12,696 DEBUG logging:103 Connected to server
```

2. マスター端では、前述のハードウェア接続に従い、PC で Modbus Poll ソフトウェア（30 日間試用）を使用します。ダウンロードリンク：https://modbustools.com/download.html インストール後、ソフトウェアを開きます：

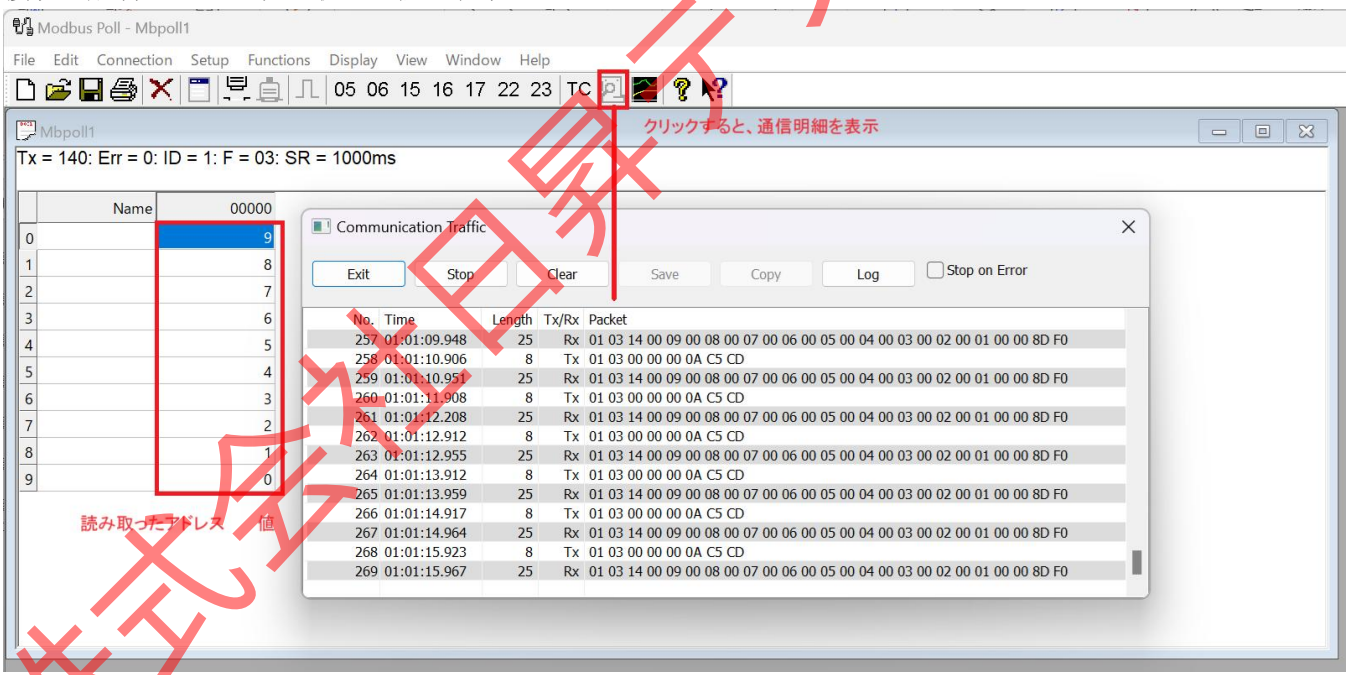
空白部分を右クリックし、「Read/write Definition」を選択し、スレーブアドレスや読み書き設定を行います。ここではアドレスを 1 に設定し、アドレス 0 から 10 個のコイルを読み取ります。



設定が完了したら、メインウィンドウの「connection」をクリックし、シリアルポートを選択します。ここでは COM3 です。具体的な設定は以下の通りです：



接続が成功すると、読み取った値が表示されます：

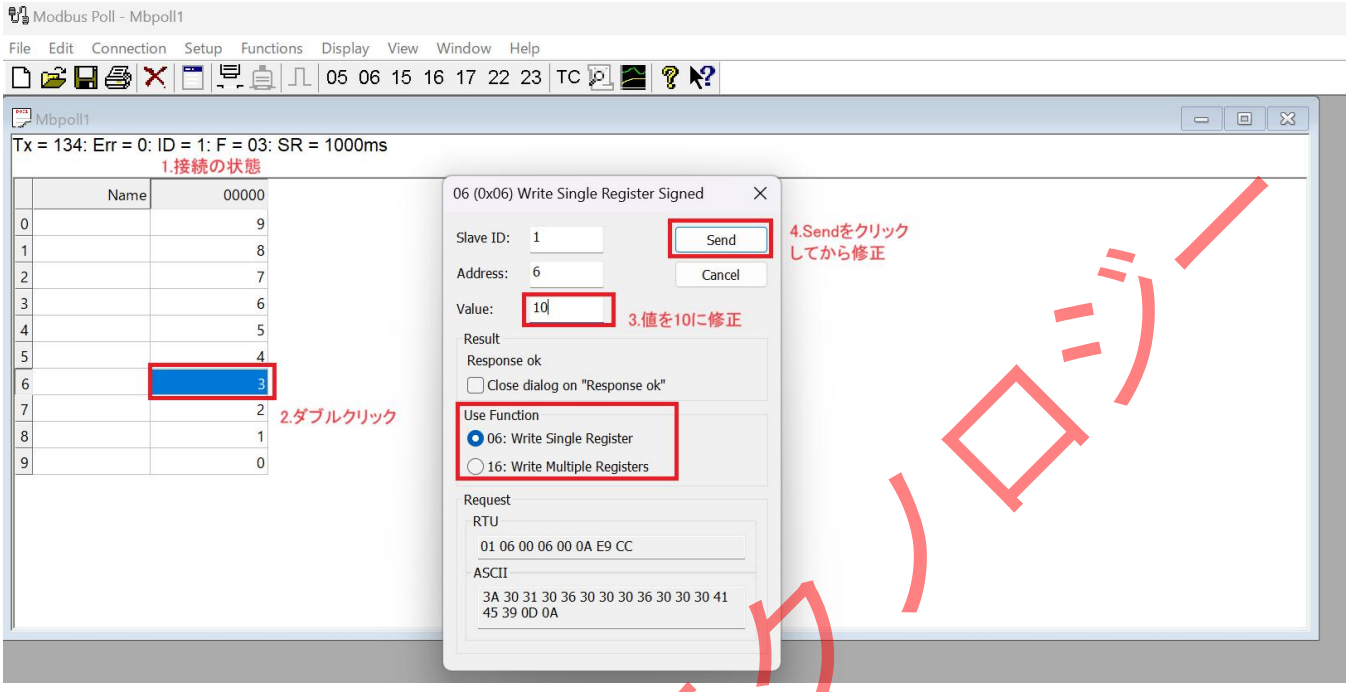


この間、ボードのターミナルに以下の情報が表示されます（実行時に --log debug を指定）：

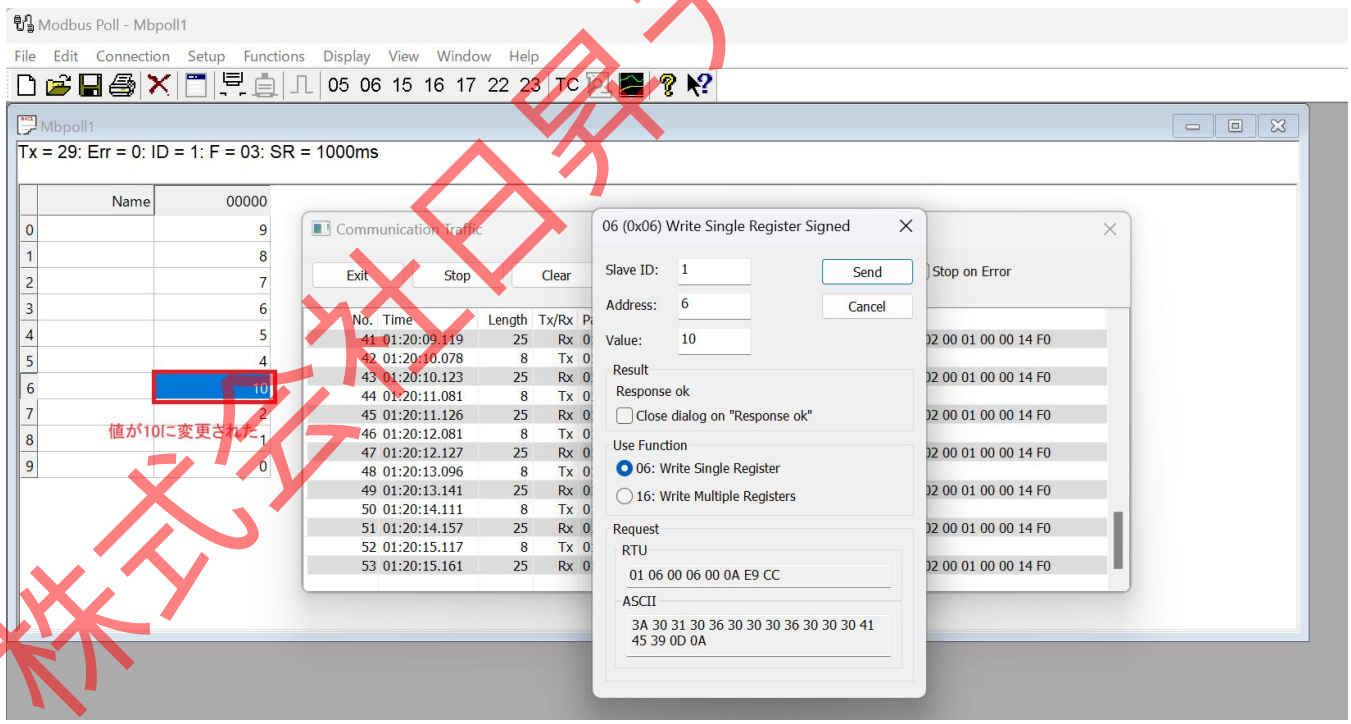
```
問題  出力  デバッグ コンソール  ターミナル  ポート 1
2024-06-03 01:07:49,138 DEBUG logging:103 Handling data: 0x1 0x3 0x0 0x0 0x0 0xa 0xc5 0xcd
2024-06-03 01:07:49,139 DEBUG logging:103 Processing: 0x1 0x3 0x0 0x0 0x0 0xa 0xc5 0xcd
2024-06-03 01:07:49,140 DEBUG logging:103 Getting Frame - 0x3 0x0 0x0 0x0 0xa
2024-06-03 01:07:49,141 DEBUG logging:103 Factory Request[ReadHoldingRegistersRequest': 3]
2024-06-03 01:07:49,141 DEBUG logging:103 Frame advanced, resetting header!!
2024-06-03 01:07:49,141 DEBUG logging:103 validate: fc-[3] address-0: count-10
2024-06-03 01:07:49,141 DEBUG logging:103 getValues: fc-[3] address-0: count-10
2024-06-03 01:07:49,142 DEBUG logging:103 send: 0x1 0x3 0x14 0x0 0x9 0x0 0x8 0x0 0x7 0x0 0x6 0x0 0x5 0x0 0x4 0x0 0x3 0x0 0x2 0x0 0x1 0x0 0x0 0x8d 0xf0
2024-06-03 01:07:50,136 DEBUG logging:103 recv: 0x1 0x3 0x0 0x0 0x0 0xa 0xc5 0xcd old_data:  addr=None
2024-06-03 01:07:50,138 DEBUG logging:103 Handling data: 0x1 0x3 0x0 0x0 0x0 0xa 0xc5 0xcd
2024-06-03 01:07:50,138 DEBUG logging:103 Processing: 0x1 0x3 0x0 0x0 0x0 0xa 0xc5 0xcd
2024-06-03 01:07:50,139 DEBUG logging:103 Getting Frame - 0x3 0x0 0x0 0x0 0xa
2024-06-03 01:07:50,139 DEBUG logging:103 Factory Request[ReadHoldingRegistersRequest': 3]
```

接続状態のまま、他のレジスタの読み取りや書き込みも行うことができます。空白部分を右クリックし、「Read/write Definition」に入り、値を再設定します：

保持レジスタの値を変更します：



送信をクリックすると、読み取られたレジスタの値が表示されます：



より詳細なレジスタの読み書き調整などの操作については、Modbus Poll ソフトウェアの使用ガイドを参照してください。テスト中、マスターがスレーブに送信したコマンド、スレーブからの応答情報および各テスト結果などを確認できます。

19.4.3 現象の簡単な解析

Modbus-RTU マスターとスレーブ間の通信中に発生するリクエストと応答の一部を示す図を簡単に説明します。複数のコイルを読み取るテストを例にします（画像をクリックすると拡大されます）：

図では、PC の上位機器端で modbus Poll を使用して単一または複数のレジスタと対応するパラメータを読み書きします。ボードスレーブがコマンドを受け取ると、対応する応答を行い、デバッグを有効にすると応答のログ情報が表示されます。

pymodbus ライブラリには、ユーザー定義のリクエストおよび応答、つまり Modbus プロトコルで定義されたユーザー定義機能コードの関連内容もサポートされています。また、pymodbus ライブラリはデータの永続化などのサーバー機能も実装しており、ユーザーは一般的なデータベースを使用してスレーブレジスタの内容を保存できます。

先ほど、pymodbus ライブラリが Python 言語で書かれた Modbus プロトコルのフルスタック実装であることに触れましたが、提供したサンプルコードではこのライブラリの一部機能のみを示しました。さらに多くの機能をテストしたい場合は、pymodbus の GitHub リポジトリ pymodbus を参照し、関連する情報を学習してください。

19.4.4 参考文献

公式サンプルコード、pymodbus-examples
公式パッケージおよびドキュメントの説明、pymodbus-doc

第 20 章 GUI ライブラリ - PyQt5

本章では、PyQt5 を簡単に紹介します。まず PyQt を使用して簡単なデスクトップアプリケーションを作成し、次に LubanCat ボードで Qt Creator を使用して簡単な UI 設計を行います。以下の簡単な例は学習用の参考として使用します。詳細な学習は参考資料を参照してください。

- プラットフォーム：LubanCat RK シリーズボード
- システム：Debian11（デスクトップ付き）
- Python バージョン：Python 3.9
- Qt バージョン：システムデフォルト 5.11.3
- PyQt バージョン：PyQt5

20.1 PyQt5 ライブラリ紹介

PyQt は Qt アプリケーションフレームワークと Python の組み合わせであり、Python 2.x と Python 3.x の両方のバージョンをサポートしています。本チュートリアルでは Python 3.9 バージョンと PyQt5 バージョンを使用します。

PyQt5 は一連の Python モジュールで構成されており、620 以上のクラスと 6000 の関数とメソッドが含まれています。Unix、Windows、Mac OS などの主要なオペレーティングシステムで動作し、Python 言語の開発の利便性のおかげで、コードの記述が C++ よりも簡単です。しかし、Python 言語の実行効率は C++ よりも遅いです。

PyQt5 には GPL 版と商用版のライセンスが提供されています。自由開発者は無料の GPL ライセンスを使用できますが、PyQt を商用アプリケーションで使用する場合は商用ライセンスを購入する必要があります。

20.2 PyQt5 ライブラリのインストール

20.2.1 Qt ライブラリのインストール

apt ツールを使用して PyQt5 をインストールします（または pip を使用してインストールします）：

apt ツールを使用してインストール：
```bash
sudo apt-get -y install python3-pyqt5
```

また、pip を使用してインストール：
```bash
```

```
pip3 install pyqt5
```

Qt ライブラリをインストール：

```bash
sudo apt-get install qt5-default
```

インストール成功後、確認：

```bash
cat@lubancat:~$ python3
Python 3.7.3 (default, Oct 31 2022, 14:04:00)
[GCC 8.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import PyQt5
>>>
```

次に、LubanCat ボードで PyQt5 の実行をテストします。

重要：デスクトップシステムで使用する際、実験前にデスクトップ画面を接続していることを確認してください。HDMI または MIPI 画面のどちらでも構いません。

20.3 PyQt5 ライブラリの基本使用

`pyqt5_demo.py` という名前の PyQt5 アプリケーションを通じて、PyQt5 デスクトップアプリケーションの作成方法を簡単に紹介します。このアプリケーションは、表示、タッチ、シグナル、スロット関数、及び PyQt5 の一部のコンポーネント（QWidget、QLCDNumber、QSlider、QVBoxLayout など）の基本的な使用方法をテストするためのものです。

20.3.1 必要なモジュールのインポート

```python
# sys モジュールをインポート。外部からプログラムにパラメータを渡したり、プログラムを終了したり、システムプラットフォームやシステムパスを取得したりする操作を含む。
import sys

# QtWidgets モジュールのサブモジュールをインポート。QWidget クラス（ユーザーインターフェースの基本クラス）、QLCDNumber（LCD スタイルの数字を表示）、QSlider（スライダコンポーネント）、
# QVBoxLayout（コンポーネントの垂直配置など）、QApplication（GUI アプリケーションのコントロールフローとメイン設定の管理など）。
from PyQt5.QtWidgets import QWidget, QLCDNumber, QSlider, QVBoxLayout, QApplication

# PyQt5.QtCore.Qt をインポート。基本的な関数とクラスを含む。
```

```
from PyQt5.QtCore import Qt

# 手間を省きたい場合は `*` を使用してすべてのモジュールをインポートすることもできます（メモリを少し多く使用します）。
from PyQt5.Qt import *
```

一般的に使用されるモジュールには QtWidgets、QtCore、QtGui などがあります。QtWidgets はシステムスタイルに適合するインターフェースを構築するための完全な UI 要素コンポーネントセットを含み、QtGui モジュールはフォント、グラフィックス、アイコン、カラーなどの基本的なグラフィック機能のクラスをカバーし、QtCore は時間、ファイル、ディレクトリ、データ型、テキストストリーム、スレッドプロセスなどの非 GUI インターフェースのコア機能をカバーします。

20.3.2 メインウィンドウとなるクラスの作成

```python
# QWidget に基づいて WinForm という名前のクラスを作成
class WinForm(QWidget):
    # クラスの初期化
    def __init__(self):
        # 親クラスの init を継承
        super().__init__()
        # 自身のウィンドウを初期化
        self.initUI()

    def initUI(self):
        # まずスライダと LCD 部品を作成
        lcd = QLCDNumber(self)
        slider = QSlider(Qt.Horizontal, self)
        # スライダの最大値とデフォルト値、長さと幅などを設定し、デフォルトの LCD スタイルで数値を表示
        slider.setMaximum(1000)
        slider.setValue(666)
        slider.setMinimumWidth(200)
        slider.setFixedHeight(60)
        lcd.display(666)
        # QVBoxLayout でレイアウトを設定し、垂直に分布
        vBox = QVBoxLayout()
        vBox.addWidget(lcd)
        vBox.addWidget(slider)
        self.setLayout(vBox)
        # valueChanged() は QSlider のシグナル関数で、スライダの値が変わるとシグナルを発し、それを connect で受信部品（ここでは LCD）に接続。
```

```
        slider.valueChanged.connect(lcd.display)

        style = "QSlider::groove:horizontal {border:1px solid #999999; height:10px;
background-color:#666666;margin:2px 0;}" ¥
                "QSlider::handle:horizontal {background-color:#ff0000; border:1px solid
#797979; width:50px; margin:-20px; border-radius:25px;}"
        # スタイルを設定
        slider.setStyleSheet(style)
        #self.setGeometry(0,0,800,480)
        # ウィンドウのタイトルを設定
        self.setWindowTitle("スライダをドラッグして数字を表示")
```

20.3.3 アプリケーションの作成

```python
# name は現在のモジュール名であり、モジュールが直接実行されると main になります。以下のコードブロックが実行されます。
if __name__ == '__main__':
    # 各アプリケーションには QApplication インスタンスが必要です。sys.argv はこのモジュールを実行する際に渡されるコマンドライン引数で、空にすることもできます。
    app = QApplication(sys.argv)
    # コンポーネントインスタンス
    form = WinForm()
    # サイズを設定してコンポーネントを表示
    form.resize(800, 480)
    form.show()
    # プログラムを実行し、メッセージイベントループに入り、exit 関数で安全に終了
    sys.exit(app.exec_())
```

QApplication インスタンスを作成し、プログラムを実行し、form というウィンドウを作成してメインウィンドウとして表示します。

20.3.4 全体のプログラム

`pyqt5_demo.py` の全体のコードは以下の通りです：

```
""" PyQt5 ライブラリの基本使用、LCD 数字表示"""
import sys
from PyQt5.QtWidgets import QWidget,QLCDNumber,QSlider,QVBoxLayout,QApplication
from PyQt5.QtCore import Qt

class WinForm(QWidget):
    def __init__(self):
        super().__init__()
        self.initUI()

    def initUI(self):
        #1 まずスライダと LCD 部品を作成
        lcd = QLCDNumber(self)
        slider = QSlider(Qt.Horizontal, self)

        slider.setMaximum(1000)
        slider.setValue(666)
        slider.setMinimumWidth(200)
        slider.setFixedHeight(60)
        lcd.display(666)

        #2 QVBoxLayout でレイアウトを設定
        vBox = QVBoxLayout()
        vBox.addWidget(lcd)
        vBox.addWidget(slider)

        self.setLayout(vBox)
        #3 valueChanged() は QSlider のシグナル関数で、スライダの値が変わるとシグナルを発し、それを connect で受信部品(ここでは LCD)に接続。
        slider.valueChanged.connect(lcd.display)


        style = "QSlider::groove:horizontal {border:1px solid #999999;height:10px;" \
                "background-color:#666666;margin:2px 0;}" \
                "QSlider::handle:horizontal {background-color:#ff0000;border:1px solid #797979;" \
                "width:50px;margin:-20px;border-radius:25px;}" \

        slider.setStyleSheet(style)

        #self.setGeometry(0,0,800,480)
        self.setWindowTitle("スライダをドラッグして数字を表示")

if __name__ == '__main__':
    app = QApplication(sys.argv)
    form = WinForm()
    form.resize(800, 480)
    form.show()
    sys.exit(app.exec_())
```

ターミナルで `pyqt5_demo.py` プログラムを直接実行：

```bash
sudo python3 pyqt5_demo.py
```

表示効果：

配布されたコードディレクトリ `libdemo/PyQt5/pyqt5_demo.py` を開発ボードにアップロードしてテストできます。

20.4 Qt Designer の使用

通常、自分のインターフェースアプリケーションを Python プログラムで作成しますが、アプリケーションが大きくなったり、インターフェースが複雑になったりすると、プログラムで全てのウィジェットを定義するのは面倒になる可能性があります。このような場合、PyQt プログラムの UI インターフェースを作成するために特化した Qt Designer を使用することができます。以下では、Qt Designer を使用して簡単なログイン UI インターフェースを作成する方法を紹介します。

apt で Qt Creator ツールをインストール：

```bash
sudo apt install qtcreator
```

Qt Designer などのツールをインストール：

```bash
sudo apt install qttools5-dev-tools
```

重要：LubanCat で Qt Creator を使用して複雑なインターフェースを設計すると、フリーズしたりクラッシュしたりすることがあります。これはメモリ不足が原因である可能性があるため、仮想マシンや PC で UI を設計したり、PyQt5 プログラムを作成したりすることをお勧めします。ボード上で直接実行することができます。

インストール後、デスクトップメニューで Qt Creator を検索するか、Qt Designer を直接開いてアプリケーションを起動します。

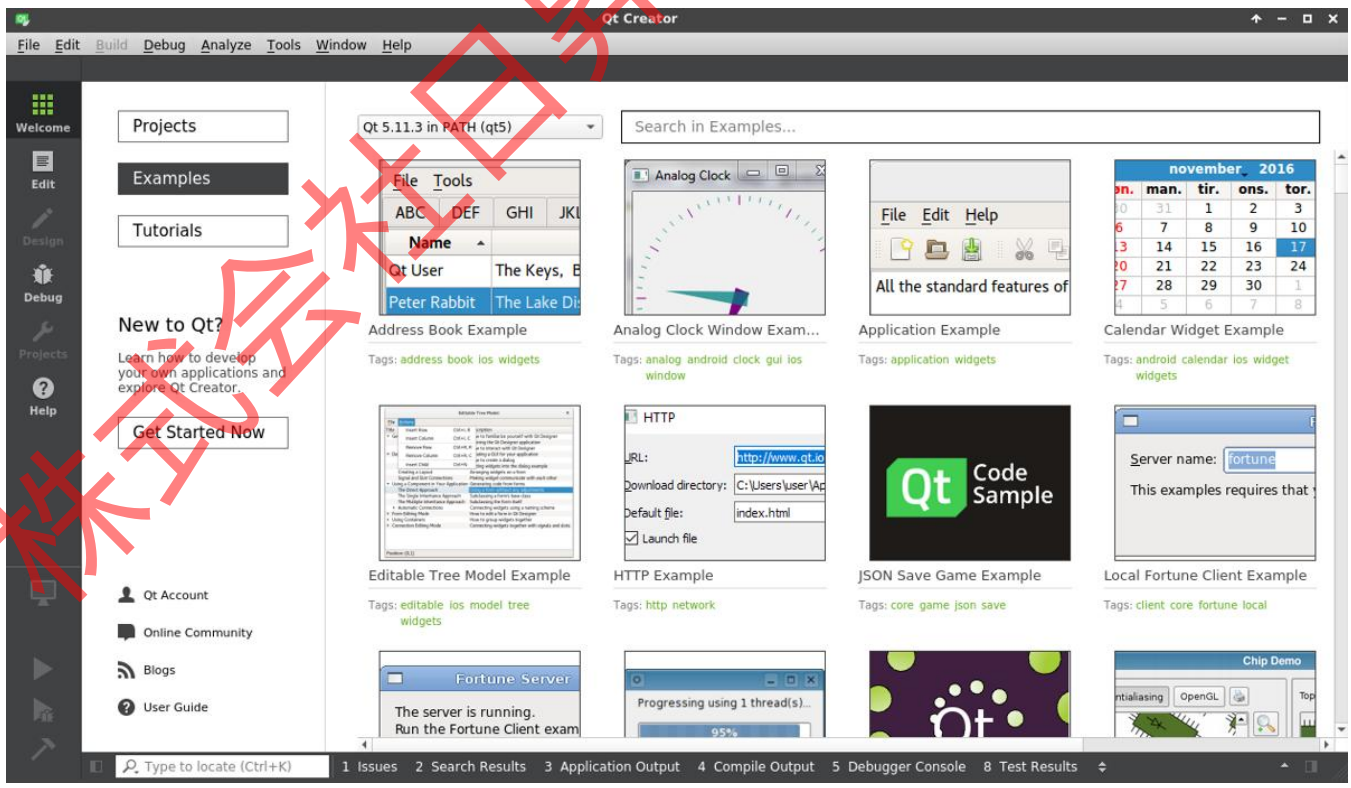
20.4.1 インターフェース UI ファイルの作成

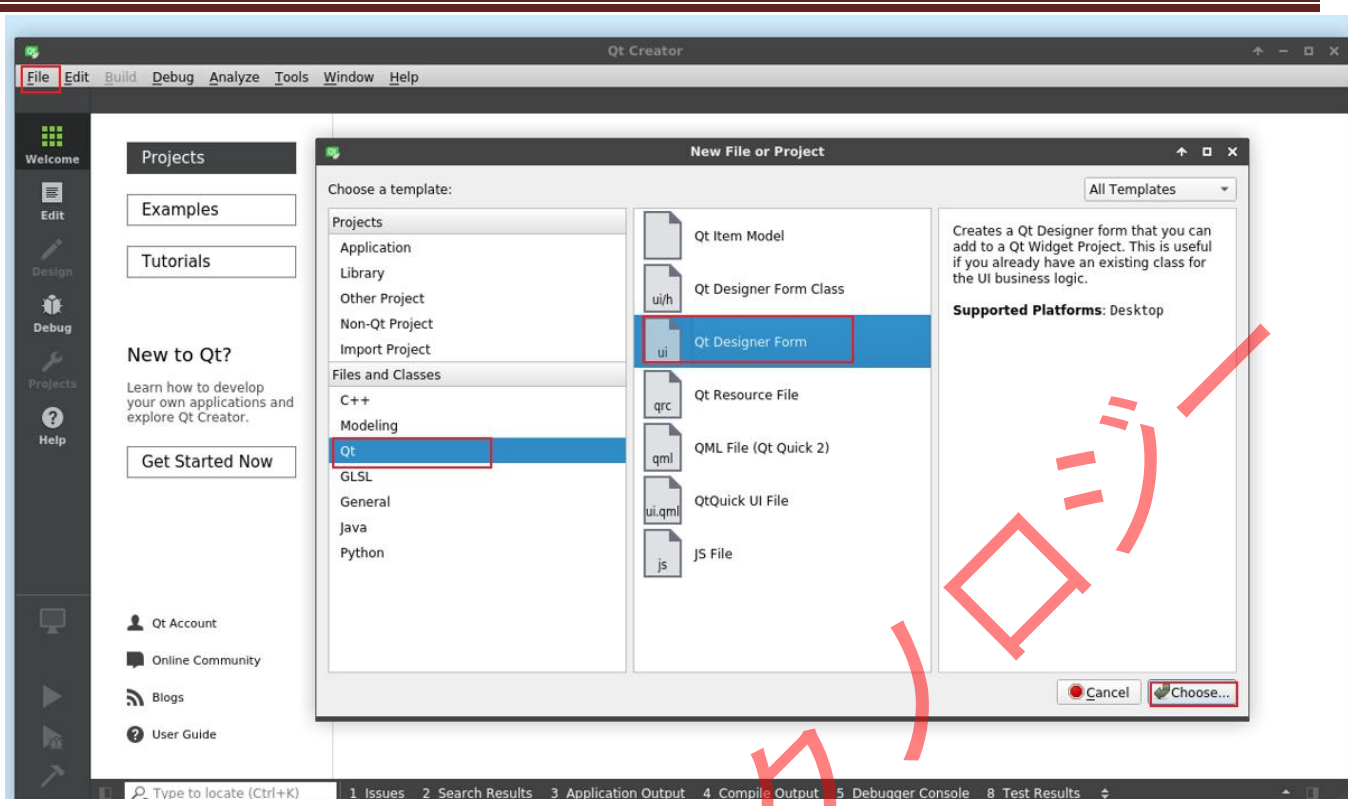
Qt Creator を開き、`File` をクリックし、`New File or Project` を選択します。`Files and Classes` セクションで `QT` をクリックし、`QT Designer Form` を選択します。次に、作成する `.ui` ファイルの名前とパスを設定します：

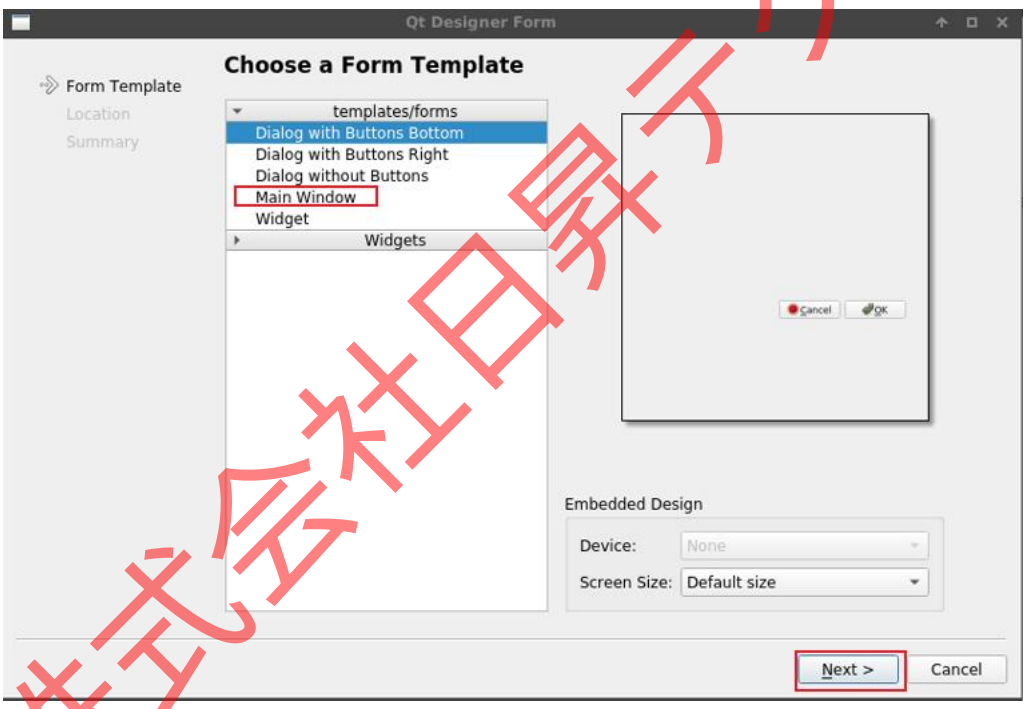
続いて `Main Window` またはその他を選択し、最後にファイルパスとファイル名を設定します：

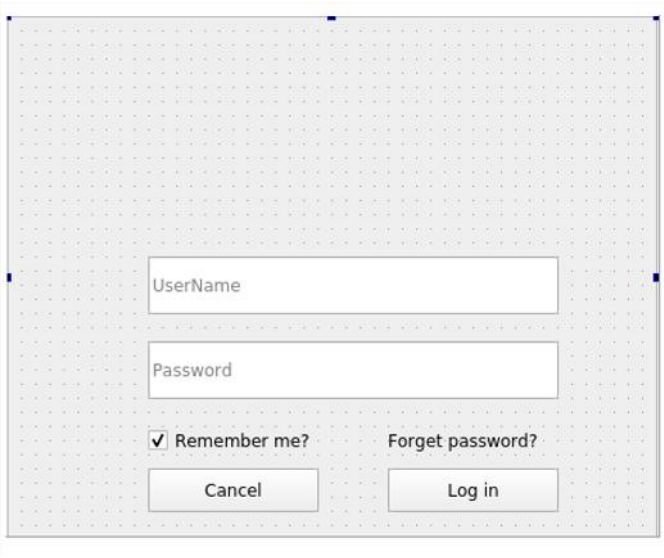
20.4.2 インターフェースの構築

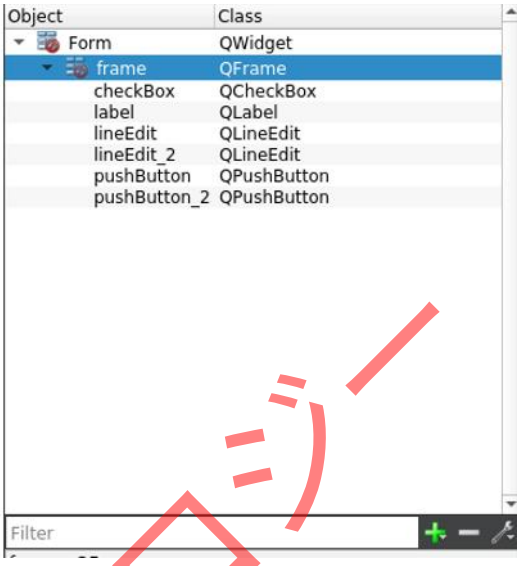
UI インターフェースで、コンテナ `Frame` を画布に追加し、サイズを調整します。簡単なログインインターフェースを作成するには、アカウントとパスワードフィールドが必要です。`Input Widgets：Line Edit` を使用し、ログインとキャンセルボタンには `Buttons：PushButton` を使用します。パスワードを忘れた場合のフィールドには `Display Widgets：TextLabel` を使用し、パスワードを記憶するには `Buttons：CheckBox` を使用します。これらのウィジェットをウィンドウにドラッグし、名前とレイアウトを設定します：





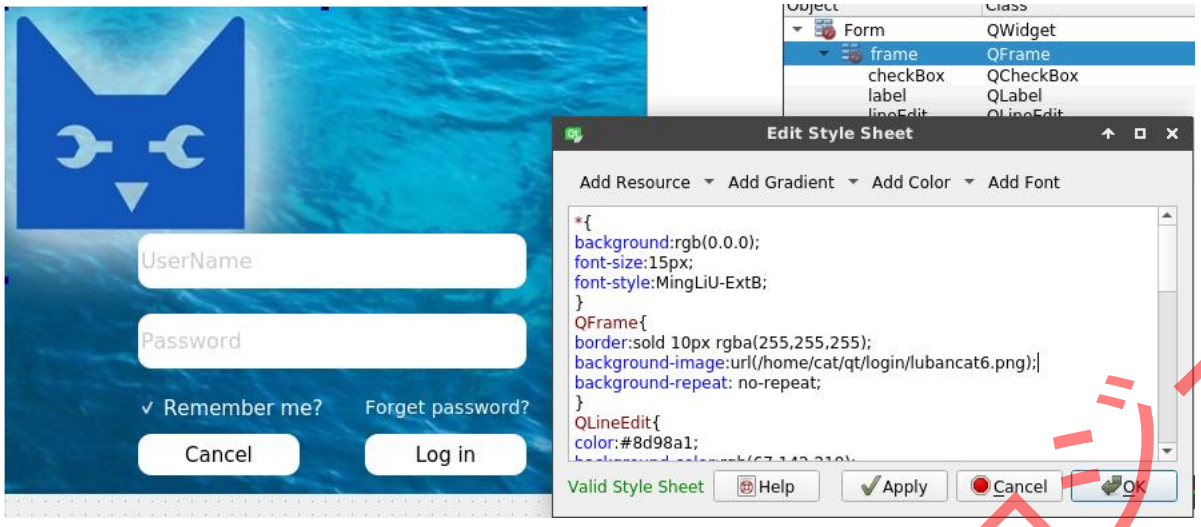






スタイルを設定するには、コンテナ全体を右クリックして `Edit Style Sheet` を選択し、以下のコードを入力します。色、スタイル、背景画像のパスを好みに応じて変更できます。以下は簡単な設定ですが、見た目は良いです：

```
*{
background:rgb(0, 0, 0);
font-size:15px;
font-style:MingLiU-ExtB;
}
QFrame{
border:sold 10px rgba(255,255,255);
background-image:url(/home/cat/qt/login/lubancat6.png);
}
QLineEdit{
color:#8d98a1;
background-color:rgb(0, 0, 0);
font-size:16px;
border-style:outset;
border-radius:10px;
font-style:MingLiU-ExtB;
}
QPushButton{
background:#ced1d8;
border-style:outset;
border-radius:10px;
font-style:MingLiU-ExtB;
}
QPushButton:pressed{
background-color:#405361;
border-style:inset;
font-style:MingLiU-ExtB;
}
QCheckBox{
background:rgba(85,170,255,0);
color:white;
font-style:MingLiU-ExtB;
}
QLabel{
background:rgba(85,170,255,0);
color:white;
font-style:MingLiU-ExtB;
font-size:14px;
}
```



ヒント：UI の設計や Qt Designer の使用については、Qt 関連のチュートリアルを参照してください。

20.4.3 UI ファイルの呼び出し

Qt Designer で設計した `.ui` ファイルは、`uic` モジュールを使用して直接インポートすることができます。または、`.ui` ファイルを Python モジュールに変換して直接インポートして使用することもできます。

1. `uic` モジュールを使用して `.ui` ファイルを直接インポート：

```python
import sys
from PyQt5.QtWidgets import QApplication, QMainWindow
from PyQt5 import uic

class LoginWindow(QMainWindow):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        # uic.loadUi を使用して現在のディレクトリ内の .ui ファイルをインポート
        uic.loadUi("login.ui", self)
        self.setWindowTitle("Lubancat login")

if __name__ == '__main__':
    app = QApplication(sys.argv)
    window = LoginWindow()
    window.show()
    sys.exit(app.exec_())
```

2. Python ファイルに変換してインポート：

`pyuic5` コマンドを使用して `.ui` ファイルを Python ファイルに変換：

apt で `pyqt5-dev-tools` ツールをインストール：
```bash
sudo apt-get install pyqt5-dev-tools
```

コマンドを使用して `.ui` ファイルを Python ファイルに変換：
```bash
pyuic5 -o login_ui.py login.ui
```

Python ファイルに変換した後、直接インポートして使用：

```python
import sys
from PyQt5.QtWidgets import QApplication, QMainWindow
# 変換された Ui_Form クラスをインポート
from login_ui import Ui_Form

class LoginWindow(QMainWindow):
    def __init__(self, parent=None):
        super().__init__(parent)
        # UI インスタンスを作成
        self.ui = Ui_Form()
        # setupUi() を実行して UI を表示
        self.ui.setupUi(self)
        self.setWindowTitle("Lubancat login")

if __name__ == "__main__":
    app = QApplication(sys.argv)
    win = LoginWindow()
    win.show()
    sys.exit(app.exec())
```

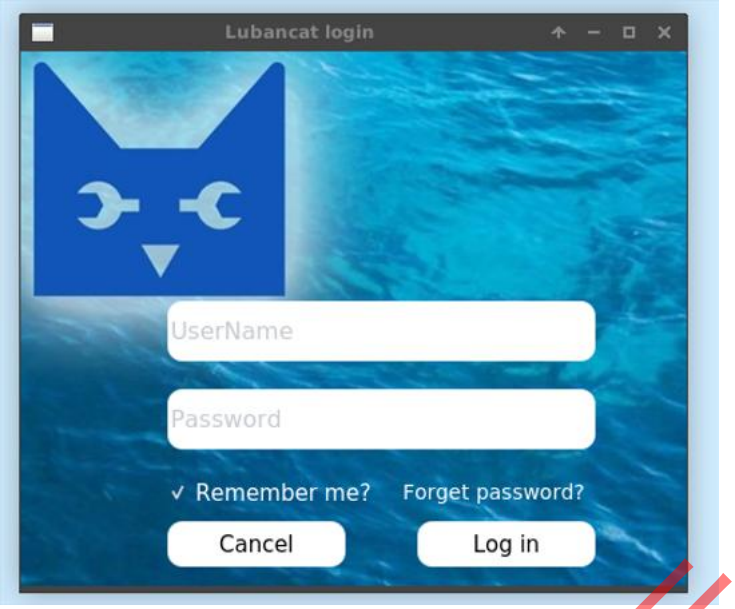
プログラムを実行するには、ターミナルで以下のコマンドを使用：
```bash
sudo python3 login.py
```

または、Qt Creator で Python 実行環境を追加します。画面の `Tool -> Options...` をクリックし、ポップアップウィンドウで `Environment` を選択し、以下の画像のように操作し、最後に `Apply` をク

リックして適用します。

実行時に対応する Python ファイルを開き、`Tool -> External -> RunPython -> RunPy` をクリックして実行します。

表示効果：



プログラムのソースコードは配布された例 `libdemo/PyQt5` ディレクトリのファイルを参照してください。最終的にアプリケーションをパッケージ化して配布する場合は、参考リンクのドキュメントを参照してください。

20.5 参考

PyQt5 に関する詳細なチュートリアルは以下を参照してください：

https://www.riverbankcomputing.com/static/Docs/PyQt5/introduction.html
https://www.pythonguis.com/latest/

第 21 章 人工知能

本モジュールでは、人工知能（AI）とディープラーニングをテーマにして、Python を基盤とし、LubanCat RK シリーズボード上で人工知能開発を行います。様々なディープラーニングフレームワークの LubanCat へのデプロイに焦点を当て、フレームワークモデルの訓練や最適化についても簡単に説明します。より詳細で体系的な学習には、各ディープラーニングの書籍や RKNN ドキュメントを参照してください。

21.1 概要

1. 人工知能（Artificial Intelligence、AI）は、学者や機関によって様々な定義が存在しますが、一般的な定義としては「人工知能とは、人間の知能をシミュレート、拡張、そして拡大するための理論、方法、技術および応用システムを扱う新しい技術科学」です。具体的な実現方法として、人工知能には多くの方法と分野がありますが、機械学習やディープラーニングが現在比較的効果的な実現方法とされています。

2. 機械学習（Machine Learning、ML）は人工知能の重要な分野の一つであり、データや経験を通じてコンピュータが自動的に性能を改善する方法を研究します。人間の学習能力を模倣し、サンプルデータから学習して経験（モデル）を取得し、それを用いて予測を行います。機械学習の一般的なアルゴリズムには、教師あり学習、教師なし学習、半教師あり学習、および強化学習などがあります。

3. ディープラーニング（Deep Learning、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、近年著しい進歩を遂げ、多くの伝統的な機械学習アルゴリズムを置き換えました。簡単に言えば、ディープラーニングは人工神経ネットワークに基づいた機械学習アルゴリズムです。ディープラーニングは、機械ビジョン、音声認識、自然言語処理などの分野で広く応用されており、特に感知型の問題解決に適しています。

4. ディープラーニングフレームワークは、ディープラーニングモデルを構築、訓練、およびデプロイするためのソフトウェアライブラリであり、高レベルの抽象化、事前定義された層とオプティマイザー、および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記述をより簡単かつ効率的にします。利用可能な多くのディープラーニングフレームワークがありますが、以下のものが代表的です：

- 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で、データフローグラフに基づいた計算方式を持ち、様々なプラットフォームと言語をサポートしています。
- PyTorch：Facebook がオープンソース化した最新のディープラーニングフレームワークの一つで、Torch ライブラリに基づき、Python 言語で書かれており、動的グラフと自動微分をサポートしています。
- 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、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、および豊富なツールコンポーネントを一体化しています。

これらのフレームワークに加えて、Caffe や Keras など他にも多くの深層学習フレームワークが存在します。

21.2 人工知能と LubanCat RK シリーズボード

LubanCat RK シリーズボードは、Rockchip RK3588 プロセッサを使用しており、RK3588 の NPU モジュールを搭載しています。これは神経ネットワークの処理ユニットであり、人工知能分野の神経ネットワークアルゴリズムを加速するために使用され、最大 6TOPS の性能を持ち、整数 8 と整数 16 の畳み込み演算をサポートしています。また、TensorFlow、TF-lite、Pytorch、Caffe、ONNX などの深層学習フレームワークをサポートしています。

21.3 参考リンク

- 機械学習 - Wikipedia
- ディープラーニング - Wikipedia
- https://github.com/PaddlePaddle/Paddle
- TensorFlow

第 22 章 NPU の使用

NPU は神経ネットワークの処理ユニットとして特化しています。機械ビジョンや自然言語処理などの人工知能分野のアルゴリズムを加速することを目的としています。人工知能の応用範囲が拡大する中で、現在では顔追跡、ジェスチャーおよびボディトラッキング、画像分類、ビデオ監視、音声認識（ASR）、高度運転支援システム（ADAS）など、様々な機能を提供しています。

LubanCat4 シリーズボードは、Rockchip RK3588 プロセッサを使用しており、RK3588 には NPU モジュールが内蔵されており、最大 6TOPS の性能を持ち、整数 8 と整数 16 の畳み込み演算をサポートしています。また、TensorFlow、TF-lite、Pytorch、Caffe、ONNX などの深層学習フレームワークをサポートしています。

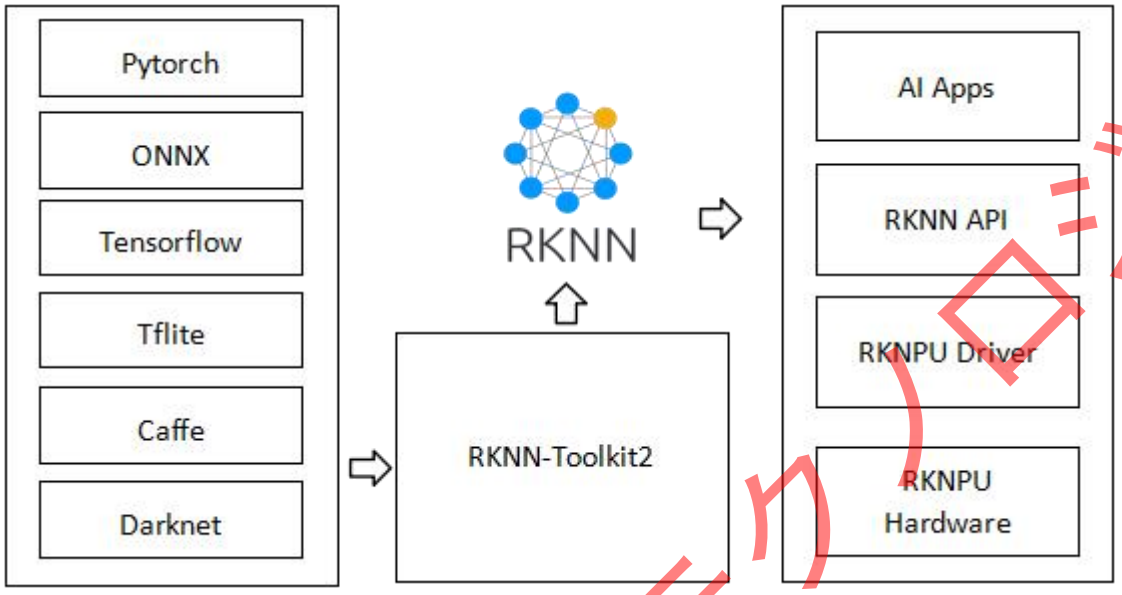
Rockchip 公式は rknpu2 と rknn-Toolkit2 ツールを提供しており、SDK は RKNPU を搭載したチッププラットフォームにプログラミングインターフェースを提供します。Rockchip NPU プラットフォームは RKNN モデルを使用し、RKNN モデルは rknn-Toolkit2 ソフトウェア開発キットを使用して変換およびデプロイされ、ボード上で rknn-Toolkit-Lite2 を使用して実行されます。

本章では、PC（Ubuntu システム）上で RKNN-Toolkit2 を使用してモデルの変換、推論、性能評価を行い、ボード上で RKNN Toolkit Lite2 を使用してデプロイする方法を簡単に紹介します。

重要：本章のチュートリアルテスト環境は、LubanCat RK シリーズボード、Debian11 イメージシステム（Python 3.9.2、OpenCV 4.5.1）、PC 環境は Ubuntu20.04（Python 3.8.10）です。チュートリアル執筆時の RKNN-Toolkit2 はバージョン 1.4.0、ボードの NPU ドライバはバージョン 0.7.2、rknnrt はバージョン 1.4.0、adb ツールはバージョン 1.0.40 です。

22.1 rknn-Toolkit2

RKNN-Toolkit2 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略化します。このツールを使用することで、次の機能を簡単に実現できます：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、モデル暗号化機能。



RKNN-Toolkit2 の現在のバージョンは Ubuntu18.04/Ubuntu20.04/Ubuntu22.04 システムに適しており、RKNN-Toolkit2 プロジェクトファイルは公式 GitHub アドレスから直接取得してください。

22.1.1 rknn-Toolkit2 のインストール

環境インストール（PC Ubuntu20.04）：

```bash
# Python ツールをインストール。Ubuntu20.04 にはデフォルトで Python3.8.10 がインストールされています
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をインストール
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：

```bash
# ディレクトリを作成。テストに使用する Ubuntu20.04 には異なるバージョンのパッケージがインストールされている可能性があるため、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
```

```
python3 -m venv .toolkit2_env
# 環境をアクティブにします
source .toolkit2_env/bin/activate
# ソースコードを取得するか、RKNN-Toolkit2 をこのディレクトリにコピーします
git clone https://github.com/airockchip/rknn-toolkit2.git
*2024/6/7 時点の最新バージョンは 2.0.0beta です。
# pip3 を使用してパッケージをインストールする際に速度が遅い場合は、ソースを設定します
pip3 config set global.index-url https://pypi.org/simple
# 依存ライブラリをインストールします。
cd rknn-toolkit2/rknn-toolkit2
*path:project-Toolkit2/rknn-toolkit2/rknn-toolkit2
packages/requirements_cp38-2.0.0b0.txt に基づいて
pip3 install numpy
pip3 install -r packages/requirements_cp38-2.0.0b0.txt
# rknn_toolkit2 をインストール
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します：

```bash
(.toolkit2_env) llh@-:~/project-Toolkit2$ python
Python 3.8.10 (default, Jun 22 2022, 20:18:18)
[GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from rknn.api import RKNN
>>>
```

22.1.2 モデル変換とモデル推論

RKNN-Toolkit2 ファイルの example ディレクトリには様々な機能のサンプルが含まれています。`../examples/onnx/yolov5` ディレクトリに入り、このデモでは PC 上で onnx モデルを RKNN モデルに変換し、エクスポート、推論、NPU プラットフォームにデプロイして結果を取得するプロセスを示します。

```bash
# examples/onnx/yolov5 ディレクトリに切り替えます
cd examples/onnx/yolov5
# モデル変換とモデル推論
python3 test.py
```

test.py コード説明： (main 関数のみ)

```
if __name__ == '__main__':
```

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

# モデル変換パラメータを設定します。ここでプラットフォームを指定できます。target_platform='rk3588'、デフォルトは rk3566 です。
# 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')

# onnx モデルをインポートし、model で onnx モデルのパスを指定します。
print('--> Loading model')
ret = rknn.load_onnx(model=ONNX_MODEL)
if ret != 0:
    print('Load model failed!')
    exit(ret)
print('done')

# RKNN モデルを構築します。ここで do_quantization を true に設定して量子化を有効にし、dataset で量子化補正に使用するデータセットを指定します。
print('--> Building model')
ret = rknn.build(do_quantization=QUANTIZE_ON, dataset=DATASET)
if ret != 0:
    print('Build model failed!')
    exit(ret)
print('done')

# RKNN モデルをエクスポートし、export_path でエクスポートするモデルのパスを指定します。ここではデフォルトで RKNN_MODEL に設定しています。
print('--> Export rknn model')
ret = rknn.export_rknn(RKNN_MODEL)
if ret != 0:
    print('Export rknn model failed!')
    exit(ret)
print('done')

# init_runtime インターフェースを呼び出してランタイム環境を初期化します。パラメータがなければ、デフォルトでは PC 上でシミュレーションされます。
print('--> Init runtime environment')
ret = rknn.init_runtime()
#ret = rknn.init_runtime('rk3588')
if ret != 0:
    print('Init runtime environment failed!')
    exit(ret)
print('done')

# モデル推論に使用する出力を設定します。
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))

# 推論を実行し、ターゲットを設定していない場合はデフォルトでシミュレーターを使用します。その後、出力データを後処理して結果を保存します。
print('--> Running model')
outputs = rknn.inference(inputs=[img])
np.save('./onnx_yolov5_0.npy', outputs[0])
np.save('./onnx_yolov5_1.npy', outputs[1])
np.save('./onnx_yolov5_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([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)

    img_1 = cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
    if boxes is not None:
        draw(img_1, boxes, scores, classes)
        cv2.imwrite('result.jpg', img_1)
        print('Save results to result.jpg!')

    rknn.release()
```

実行してエクスポートが成功すると、現在のディレクトリに `yolov5s_relu.onnx` ファイルが生成されます。このファイルは次の Toolkit Lite2 のセクションで使用されます。

上記の `yolov5` サンプルの実行は同時にモデル推論も行い、現在のディレクトリに結果画像（result.jpg）を出力します。

22.1.3 性能とメモリ評価

RKNN-Toolkit2 は、シミュレーション推論テストだけでなく、簡単なボード接続デバッグも可能です。LubanCat の一部のボードの USB インターフェースは直接 USB ADB デバッグをサポートしていませんが、ネットワーク ADB 接続を使用してデバッグすることができます。

ボード接続デバッグでは、ボード側で adbd と rknn_server サービスを起動する必要があります。PC 側の adb サーバーは adbd に接続し、これによりクライアントとデバイス間で通信が可能になります。rknn_server はボード上で実行されるバックグラウンドプロキシサービスで、PC から USB 経由で送信された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、結果を PC 側に返します。

```bash
# ファイルをボードにコピー（scp、sftp、nfs などの方法を使用できます。ファイルは付属のサンプルまたは RKNPU2 の runtime¥RK356X¥Linux¥rknn_server ディレクトリから取得できます）
cd ../rknpu2/runtime/Linux/rknn_server/aarch64/usr/bin
*path:project-Toolkit2/rknn-toolkit2/rknpu2/runtime/Linux/rknn_server/aarch64/usr/bin
基板の IP が 192.168.11.241、必要のファイルが/home/cat に転送します。
scp restart_rknn.sh rknn_server start_rknn.sh cat@192.168.11.241:/home/cat

基板側実施：

# 実行権限を追加
cd /usr/bin
実施する前に、ファイルが存在するかを確認（出荷時にすでに入れた場合もあります）
ls -ls rknn_server restart_rknn.sh start_rknn.sh
chmod +x rknn_server restart_rknn.sh start_rknn.sh
# rknn_server を起動
restart_rknn.sh
```

```
# 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 systemctl status adbd.service
また
sudo /usr/bin/adbd &
```

ボードと PC が同じローカルネットワーク内に接続されていることを確認し、本チュートリアルの付属プログラムを使用してテストします：

```bash
# PC 端に adb をインストール
sudo apt install adb
# adb サーバーを起動
adb start-server
# ボードに接続。ボードの実際の IP に基づいて、デフォルトは 5555 ポート
adb connect 192.168.11.241
# 接続されたデバイスを確認し、ランタイムの device_id を確認
adb devices
cd examples/onnx/yolov5
# プログラムを実行し、ボード接続デバッグ
```

*github からダウンロードした test.py はデフォルトで rk3566 をテストしますので、rk3588 に修正します。

```
nano test.py
rknn.config(mean_values=[[0, 0, 0]], std_values=[[255, 255, 255]], target_platform='rk3588')
ret = rknn.init_runtime(target='rk3588', device_id='192.168.11.241:5555', perf_debug=True, eval_mem=True)
python test_eval.py
```

test_eval.py コード説明： (main 関数のみ)

```python
if __name__ == '__main__':
    # RKNN を作成します
    # テスト中に問題が発生した場合は、verbose=True を設定してデバッグ情報を確認してください。
    #rknn = RKNN(verbose=True)
    rknn = RKNN()

    # RKNN モデルをインポートし、path パラメータで rknn モデルのパスを指定します
    print('--> Loading model')
    ret = rknn.load_rknn(path=RKNN_MODEL)
    if ret != 0:
        print('Load model failed!')
        exit(ret)
    print('done')
```

```
    # ランタイム環境を初期化し、接続するボードの NPU プラットフォームを指定します。device_id は前述の adb で接続したボードのデバイス ID を指定します。
    # perf_debug は性能評価を行う際にデバッグモードを有効にし、eval_mem はメモリ評価モードに入ります
    print('--> Init runtime environment')
    ret = rknn.init_runtime(target='rk3588', device_id='192.168.11.241:5555', perf_debug=True, eval_mem=True)
    if ret != 0:
        print('Init runtime environment failed!')
        exit(ret)
    print('done')

    # モデルの性能を評価し、デフォルトでは is_print が true になっており、メモリ使用状況を出力します
    print('--> eval_perf')
    rknn.eval_perf()
    print('done')

    # デバッグとして、モデルの性能を評価します。デフォルトでは is_print が true になっており、メモリ使用状況を出力します
    print('--> eval_memory')
    rknn.eval_memory()
    print('done')


    rknn.release()
```

*実行エラー：ホスト PC 側が基板側のバージョンを一致していない。

```
adb server version (41) doesn't match this client (40); killing...
* daemon started successfully
error: no devices/emulators found
E init_runtime: Connect to Device Failure (-1), Please check the USB connection!
E init_runtime: Catch exception when init runtime!
E init_runtime: Traceback (most recent call last):
E init_runtime:   File "rknn/api/rknn_base.py", line 2492, in
rknn.api.rknn_base.RKNNBase.init_runtime
E init_runtime:   File "rknn/api/rknn_runtime.py", line 216, in
rknn.api.rknn_runtime.RKNNRuntime.__init__
E init_runtime:   File "rknn/api/rknn_platform.py", line 352, in
rknn.api.rknn_platform.start_ntp_or_adb
E init_runtime: Exception: Init runtime environment failed!
W If you can't handle this error, please try updating to the latest version of the toolkit2
and runtime from:
    https://console.zbox.filez.com/l/IO0fc3 (Pwd: rknn)  Path: RKNPU2_SDK / 1.X.X / develop /
    If the error still exists in the latest version, please collect the corresponding error
logs and the model,
    convert script, and input data that can reproduce the problem, and then submit an issue on:
    https://redmine.rock-chips.com (Please consult our sales or FAE for the redmine account)
Init runtime environment failed!
```

1.0.40 バージョンの adb をダウンロードしましょう。（Ubuntu20.04 上に apt で 1.0.39 をインストールされます。）

wget https://dl.google.com/android/repository/platform-tools_r28.0.1-linux.zip

簡単なテスト結果：

```bash
(.toolkit2_env) csun@CSUN-PC-0013:~/project-Toolkit2/rknn-toolkit2/rknn-
```

```
toolkit2/examples/onnx/yolov5$ python3 test_eval.py
  I rknn-toolkit2 version: 2.0.0b0+9bab5682
  --> Loading model
  done
  --> Init runtime environment
  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 Flag perf_debug has been set, it will affect the performance of inference!
  I Flag eval_mem has been set, it will affect the performance of inference!
  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: 1.6.0 (535b468 build@2023-12-11T17:05:07)
  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=output, 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=0.003860
  D RKNNAPI:   index=1, name=283, 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=285, 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=0.003915
  done
  --> eval_perf
  CPU Current Frequency List:
      - 1800000
      - 2256000
      - 2256000
  NPU Current Frequency List:
      - 1000000000
  DDR Current Frequency List:
```

- 1560000000

Warning: The performance result is just for debugging, may worse than actual performance!

Network Layer Information Table

ID	OpType	DataType	Target	InputShape	OutputShape	Cycles(DDR/NPU/Total)	Time(us)	MacUsage(%)	WorkLoad(0/1/2)	RW(KB)	FullName
1	InputOperator	UINT8	CPU	¥	(1,3,640,640)	0/0/0	8	¥	0.0%/0.0%/0.0%	0	InputOperator:images
2	Conv	UINT8	NPU	(1,3,640,640),(12,3,2,2),(12)	(1,12,320,320)	164162/409600/409600	732	1.97/0.00/0.00	100.0%/0.0%/0.0%	1201	Conv:Conv_0
3	ConvRelu	INT8	NPU	(1,12,320,320),(32,12,3,3),(32)	(1,32,320,320)	281598/921600/921600	1064	32.48/0.00/0.00	100.0%/0.0%/0.0%	1604	Conv:Conv_1
4	ConvRelu	INT8	NPU	(1,32,320,320),(64,32,3,3),(64)	(1,64,160,160)	282403/460800/460800	733	62.86/0.00/0.00	100.0%/0.0%/0.0%	3218	Conv:Conv_3
5	ConvRelu	INT8	NPU	(1,64,160,160),(32,64,1,1),(32)	(1,32,160,160)	140792/102400/140792	276	18.55/0.00/0.00	100.0%/0.0%/0.0%	1602	Conv:Conv_5
6	ConvRelu	INT8	NPU	(1,64,160,160),(32,64,1,1),(32)	(1,32,160,160)	140792/102400/140792	277	18.48/0.00/0.00	100.0%/0.0%/0.0%	1602	Conv:Conv_12
7	ConvRelu	INT8	NPU	(1,32,160,160),(32,32,1,1),(32)	(1,32,160,160)	93847/102400/102400	212	12.08/0.00/0.00	100.0%/0.0%/0.0%	801	Conv:Conv_7
8	ConvReluAdd	INT8	NPU	(1,32,160,160),(32,32,3,3),(32),...	(1,32,160,160)	141202/230400/230400	315	73.14/0.00/0.00	100.0%/0.0%/0.0%	1609	Conv:Conv_9
9	Concat	INT8	NPU	(1,32,160,160),(1,32,160,160)	(1,64,160,160)	187546/0/187546	309	¥	100.0%/0.0%/0.0%	1600	Concat:Concat_14
10	ConvRelu	INT8	NPU	(1,64,160,160),(64,64,1,1),(64)	(1,64,160,160)						

187810/204800/204800 405 25.28/0.00/0.00 100.0%/0.0%/0.0% 1604
Conv:Conv_15
11 ConvRelu INT8 NPU (1,64,160,160),(128,64,3,3),(128) (1,128,80,80)
144938/460800/460800 602 76.54/0.00/0.00 100.0%/0.0%/0.0% 1673
Conv:Conv_17
12 ConvRelu INT8 NPU (1,128,80,80),(64,128,1,1),(64) (1,64,80,80)
70828/51200/70828 150 34.13/0.00/0.00 100.0%/0.0%/0.0% 808
Conv:Conv_19
13 ConvRelu INT8 NPU (1,128,80,80),(64,128,1,1),(64) (1,64,80,80)
70828/51200/70828 150 34.13/0.00/0.00 100.0%/0.0%/0.0% 808
Conv:Conv_31
14 ConvRelu INT8 NPU (1,64,80,80),(64,64,1,1),(64) (1,64,80,80)
47151/51200/51200 116 22.07/0.00/0.00 100.0%/0.0%/0.0% 404
Conv:Conv_21
15 ConvReluAdd INT8 NPU (1,64,80,80),(64,64,3,3),(64),... (1,64,80,80)
72469/230400/230400 275 83.78/0.00/0.00 100.0%/0.0%/0.0% 836
Conv:Conv_23
16 ConvRelu INT8 NPU (1,64,80,80),(64,64,1,1),(64) (1,64,80,80)
47151/51200/51200 116 22.07/0.00/0.00 100.0%/0.0%/0.0% 404
Conv:Conv_26
17 ConvReluAdd INT8 NPU (1,64,80,80),(64,64,3,3),(64),... (1,64,80,80)
72469/230400/230400 276 83.48/0.00/0.00 100.0%/0.0%/0.0% 836
Conv:Conv_28
18 Concat INT8 NPU (1,64,80,80),(1,64,80,80) (1,128,80,80)
93773/0/93773 165 ¥ 100.0%/0.0%/0.0% 800
Concat:Concat_33
19 ConvRelu INT8 NPU (1,128,80,80),(128,128,1,1),(128) (1,128,80,80)
94770/102400/102400 214 47.85/0.00/0.00 100.0%/0.0%/0.0% 817
Conv:Conv_34
20 ConvRelu INT8 NPU (1,128,80,80),(256,128,3,3),(256) (1,256,40,40)
87327/460800/460800 575 80.14/0.00/0.00 100.0%/0.0%/0.0% 1090
Conv:Conv_36
21 ConvRelu INT8 NPU (1,256,40,40),(128,256,1,1),(128) (1,128,40,40)
37099/51200/51200 105 48.76/0.00/0.00 100.0%/0.0%/0.0% 433
Conv:Conv_38
22 ConvRelu INT8 NPU (1,256,40,40),(128,256,1,1),(128) (1,128,40,40)
37099/51200/51200 105 48.76/0.00/0.00 100.0%/0.0%/0.0% 433
Conv:Conv_55
23 ConvRelu INT8 NPU (1,128,40,40),(128,128,1,1),(128) (1,128,40,40)
24440/25600/25600 69 37.10/0.00/0.00 100.0%/0.0%/0.0% 217
Conv:Conv_40
24 ConvReluAdd INT8 NPU (1,128,40,40),(128,128,3,3),(128),... (1,128,40,40)
43664/230400/230400 264 87.27/0.00/0.00 100.0%/0.0%/0.0% 545
Conv:Conv_42
25 ConvRelu INT8 NPU (1,128,40,40),(128,128,1,1),(128) (1,128,40,40)

24440/25600/25600 70 36.57/0.00/0.00 100.0%/0.0%/0.0% 217
Conv:Conv_45
26 ConvReluAdd INT8 NPU (1,128,40,40),(128,128,3,3),(128),... (1,128,40,40)
43664/230400/230400 262 87.94/0.00/0.00 100.0%/0.0%/0.0% 545
Conv:Conv_47
27 ConvRelu INT8 NPU (1,128,40,40),(128,128,1,1),(128) (1,128,40,40)
24440/25600/25600 69 37.10/0.00/0.00 100.0%/0.0%/0.0% 217
Conv:Conv_50
28 ConvReluAdd INT8 NPU (1,128,40,40),(128,128,3,3),(128),... (1,128,40,40)
43664/230400/230400 263 87.60/0.00/0.00 100.0%/0.0%/0.0% 545
Conv:Conv_52
29 Concat INT8 NPU (1,128,40,40),(1,128,40,40) (1,256,40,40)
46887/0/46887 95 ¥ 100.0%/0.0%/0.0% 400
Concat:Concat_57
30 ConvRelu INT8 NPU (1,256,40,40),(256,256,1,1),(256) (1,256,40,40)
50755/102400/102400 157 65.22/0.00/0.00 100.0%/0.0%/0.0% 466
Conv:Conv_58
31 ConvRelu INT8 NPU (1,256,40,40),(512,256,3,3),(512) (1,512,20,20)
102916/460800/460800 557 82.73/0.00/0.00 100.0%/0.0%/0.0% 1556
Conv:Conv_60
32 ConvRelu INT8 NPU (1,512,20,20),(256,512,1,1),(256) (1,256,20,20)
25202/51200/51200 90 56.89/0.00/0.00 100.0%/0.0%/0.0% 330
Conv:Conv_62
33 ConvRelu INT8 NPU (1,512,20,20),(256,512,1,1),(256) (1,256,20,20)
25202/51200/51200 90 56.89/0.00/0.00 100.0%/0.0%/0.0% 330
Conv:Conv_69
34 ConvRelu INT8 NPU (1,256,20,20),(256,256,1,1),(256) (1,256,20,20)
15590/25600/25600 59 43.39/0.00/0.00 100.0%/0.0%/0.0% 166
Conv:Conv_64
35 ConvReluAdd INT8 NPU (1,256,20,20),(256,256,3,3),(256),... (1,256,20,20)
51458/230400/230400 264 87.27/0.00/0.00 100.0%/0.0%/0.0% 778
Conv:Conv_66
36 Concat INT8 NPU (1,256,20,20),(1,256,20,20) (1,512,20,20)
23444/0/23444 58 ¥ 100.0%/0.0%/0.0% 200
Concat:Concat_71
37 ConvRelu INT8 NPU (1,512,20,20),(512,512,1,1),(512) (1,512,20,20)
38682/102400/102400 142 72.11/0.00/0.00 100.0%/0.0%/0.0% 460
Conv:Conv_72
38 ConvRelu INT8 NPU (1,512,20,20),(256,512,1,1),(256) (1,256,20,20)
25202/51200/51200 90 56.89/0.00/0.00 100.0%/0.0%/0.0% 330
Conv:Conv_74
39 MaxPool INT8 NPU (1,256,20,20) (1,256,20,20)
11722/0/11722 48 ¥ 100.0%/0.0%/0.0% 100
MaxPool:MaxPool_76
40 MaxPool INT8 NPU (1,256,20,20) (1,256,20,20)

11722/0/11722	48	¥		100.0%/0.0%/0.0%	100	
MaxPool:MaxPool_77						
41	MaxPool	INT8	NPU	(1,256,20,20)		(1,256,20,20)
11722/0/11722	47	¥		100.0%/0.0%/0.0%	100	
MaxPool:MaxPool_78						
42	MaxPool	INT8	NPU	(1,256,20,20)		(1,256,20,20)
11722/0/11722	48	¥		100.0%/0.0%/0.0%	100	
MaxPool:MaxPool_79						
43	MaxPool	INT8	NPU	(1,256,20,20)		(1,256,20,20)
11722/0/11722	49	¥		100.0%/0.0%/0.0%	100	
MaxPool:MaxPool_80						
44	MaxPool	INT8	NPU	(1,256,20,20)		(1,256,20,20)
11722/0/11722	48	¥		100.0%/0.0%/0.0%	100	
MaxPool:MaxPool_81						
45	Concat	INT8	NPU	(1,256,20,20),(1,256,20,20),...		(1,1024,20,20)
46887/0/46887	96	¥		100.0%/0.0%/0.0%	400	
Concat:Concat_82						
46	ConvRelu	INT8	NPU	(1,1024,20,20),(512,1024,1,1),(512)		(1,512,20,20)
65407/204800/204800	268	76.42/0.00/0.00		100.0%/0.0%/0.0%	916	
Conv:Conv_83						
47	ConvRelu	INT8	NPU	(1,512,20,20),(256,512,1,1),(256)		(1,256,20,20)
25202/51200/51200	91	56.26/0.00/0.00		100.0%/0.0%/0.0%	330	
Conv:Conv_85						
48	Resize	INT8	NPU	(1,256,20,20),(0),(4)		(1,256,40,40)
29305/0/29305	74	¥		100.0%/0.0%/0.0%	100	
Resize:Resize_88						
49	Concat	INT8	NPU	(1,256,40,40),(1,256,40,40)		(1,512,40,40)
93773/0/93773	168	¥		100.0%/0.0%/0.0%	800	
Concat:Concat_89						
50	ConvRelu	INT8	NPU	(1,512,40,40),(128,512,1,1),(128)		(1,128,40,40)
62418/102400/102400	194	52.78/0.00/0.00		100.0%/0.0%/0.0%	865	
Conv:Conv_90						
51	ConvRelu	INT8	NPU	(1,512,40,40),(128,512,1,1),(128)		(1,128,40,40)
62418/102400/102400	194	52.78/0.00/0.00		100.0%/0.0%/0.0%	865	
Conv:Conv_96						
52	ConvRelu	INT8	NPU	(1,128,40,40),(128,128,1,1),(128)		(1,128,40,40)
24440/25600/25600	70	36.57/0.00/0.00		100.0%/0.0%/0.0%	217	
Conv:Conv_92						
53	ConvRelu	INT8	NPU	(1,128,40,40),(128,128,3,3),(128)		(1,128,40,40)
31942/230400/230400	262	87.94/0.00/0.00		100.0%/0.0%/0.0%	345	
Conv:Conv_94						
54	Concat	INT8	NPU	(1,128,40,40),(1,128,40,40)		(1,256,40,40)
46887/0/46887	95	¥		100.0%/0.0%/0.0%	400	
Concat:Concat_98						
55	ConvRelu	INT8	NPU	(1,256,40,40),(256,256,1,1),(256)		(1,256,40,40)

50755/102400/102400	158		64.81/0.00/0.00	100.0%/0.0%/0.0%	466	
Conv:Conv_99						
56	ConvRelu	INT8	NPU	(1,256,40,40),(128,256,1,1),(128)	(1,128,40,40)	
37099/51200/51200	105		48.76/0.00/0.00	100.0%/0.0%/0.0%	433	
Conv:Conv_101						
57	Resize	INT8	NPU	(1,128,40,40),(0),(4)	(1,128,80,80)	
58609/0/58609	117		¥	100.0%/0.0%/0.0%	200	
Resize:Resize_104						
58	Concat	INT8	NPU	(1,128,80,80),(1,128,80,80)	(1,256,80,80)	
187546/0/187546	310		¥	100.0%/0.0%/0.0%	1600	
Concat:Concat_105						
59	ConvRelu	INT8	NPU	(1,256,80,80),(64,256,1,1),(64)	(1,64,80,80)	
118184/102400/118184	256		40.00/0.00/0.00	100.0%/0.0%/0.0%	1616	
Conv:Conv_106						
60	ConvRelu	INT8	NPU	(1,256,80,80),(64,256,1,1),(64)	(1,64,80,80)	
118184/102400/118184	256		40.00/0.00/0.00	100.0%/0.0%/0.0%	1616	
Conv:Conv_112						
61	ConvRelu	INT8	NPU	(1,64,80,80),(64,64,1,1),(64)	(1,64,80,80)	
47151/51200/51200	118		21.69/0.00/0.00	100.0%/0.0%/0.0%	404	
Conv:Conv_108						
62	ConvRelu	INT8	NPU	(1,64,80,80),(64,64,3,3),(64)	(1,64,80,80)	
49026/230400/230400	275		83.78/0.00/0.00	100.0%/0.0%/0.0%	436	
Conv:Conv_110						
63	Concat	INT8	NPU	(1,64,80,80),(1,64,80,80)	(1,128,80,80)	
93773/0/93773	174		¥	100.0%/0.0%/0.0%	800	
Concat:Concat_114						
64	ConvRelu	INT8	NPU	(1,128,80,80),(128,128,1,1),(128)	(1,128,80,80)	
94770/102400/102400	213		48.08/0.00/0.00	100.0%/0.0%/0.0%	817	
Conv:Conv_115						
65	ConvRelu	INT8	NPU	(1,128,80,80),(128,128,3,3),(128)	(1,128,40,40)	
67107/230400/230400	325		70.89/0.00/0.00	100.0%/0.0%/0.0%	945	
Conv:Conv_117						
66	Concat	INT8	NPU	(1,128,40,40),(1,128,40,40)	(1,256,40,40)	
46887/0/46887	100		¥	100.0%/0.0%/0.0%	400	
Concat:Concat_119						
67	ConvSigmoid	INT8	NPU	(1,128,80,80),(255,128,1,1),(255)	(1,255,80,80)	
142653/204800/204800	901		22.64/0.00/0.00	100.0%/0.0%/0.0%	834	
Conv:Conv_145						
68	OutputOperator	INT8	CPU	(1,255,80,80)	¥	
0/0/0	98		¥	0.0%/0.0%/0.0%	1600	
OutputOperator:output						
69	ConvRelu	INT8	NPU	(1,256,40,40),(128,256,1,1),(128)	(1,128,40,40)	
37099/51200/51200	108		47.41/0.00/0.00	100.0%/0.0%/0.0%	433	
Conv:Conv_120						
70	ConvRelu	INT8	NPU	(1,256,40,40),(128,256,1,1),(128)	(1,128,40,40)	

37099/51200/51200	105		48.76/0.00/0.00	100.0%/0.0%/0.0%	433	
Conv:Conv_126						
71	ConvRelu	INT8	NPU	(1,128,40,40),(128,128,1,1),(128)	(1,128,40,40)	
24440/25600/25600	69		37.10/0.00/0.00	100.0%/0.0%/0.0%	217	
Conv:Conv_122						
72	ConvRelu	INT8	NPU	(1,128,40,40),(128,128,3,3),(128)	(1,128,40,40)	
31942/230400/230400	261		88.28/0.00/0.00	100.0%/0.0%/0.0%	345	
Conv:Conv_124						
73	Concat	INT8	NPU	(1,128,40,40),(1,128,40,40)	(1,256,40,40)	
46887/0/46887	94		¥	100.0%/0.0%/0.0%	400	
Concat:Concat_128						
74	ConvRelu	INT8	NPU	(1,256,40,40),(256,256,1,1),(256)	(1,256,40,40)	
50755/102400/102400	157		65.22/0.00/0.00	100.0%/0.0%/0.0%	466	
Conv:Conv_129						
75	ConvRelu	INT8	NPU	(1,256,40,40),(256,256,3,3),(256)	(1,256,20,20)	
63180/230400/230400	312		73.85/0.00/0.00	100.0%/0.0%/0.0%	978	
Conv:Conv_131						
76	Concat	INT8	NPU	(1,256,20,20),(1,256,20,20)	(1,512,20,20)	
23444/0/23444	61		¥	100.0%/0.0%/0.0%	200	
Concat:Concat_133						
77	ConvSigmoid	INT8	NPU	(1,256,40,40),(255,256,1,1),(255)	(1,255,40,40)	
50755/102400/102400	259		39.38/0.00/0.00	100.0%/0.0%/0.0%	466	
Conv:Conv_147						
78	OutputOperator	INT8	CPU	(1,255,40,40)	¥	
0/0/0	30		¥	0.0%/0.0%/0.0%	400	
OutputOperator:283						
79	ConvRelu	INT8	NPU	(1,512,20,20),(256,512,1,1),(256)	(1,256,20,20)	
25202/51200/51200	92		55.65/0.00/0.00	100.0%/0.0%/0.0%	330	
Conv:Conv_134						
80	ConvRelu	INT8	NPU	(1,512,20,20),(256,512,1,1),(256)	(1,256,20,20)	
25202/51200/51200	90		56.89/0.00/0.00	100.0%/0.0%/0.0%	330	
Conv:Conv_140						
81	ConvRelu	INT8	NPU	(1,256,20,20),(256,256,1,1),(256)	(1,256,20,20)	
15590/25600/25600	57		44.91/0.00/0.00	100.0%/0.0%/0.0%	166	
Conv:Conv_136						
82	ConvRelu	INT8	NPU	(1,256,20,20),(256,256,3,3),(256)	(1,256,20,20)	
45598/230400/230400	263		87.60/0.00/0.00	100.0%/0.0%/0.0%	678	
Conv:Conv_138						
83	Concat	INT8	NPU	(1,256,20,20),(1,256,20,20)	(1,512,20,20)	
23444/0/23444	59		¥	100.0%/0.0%/0.0%	200	
Concat:Concat_142						
84	ConvRelu	INT8	NPU	(1,512,20,20),(512,512,1,1),(512)	(1,512,20,20)	
38682/102400/102400	142		72.11/0.00/0.00	100.0%/0.0%/0.0%	460	
Conv:Conv_143						
85	ConvSigmoid	INT8	NPU	(1,512,20,20),(255,512,1,1),(255)	(1,255,20,20)	

25202/51200/51200 92 55.43/0.00/0.00 100.0%/0.0%/0.0% 330
Conv:Conv_149
86 OutputOperator INT8 CPU (1,255,20,20) ¥
0/0/0 11 ¥ 0.0%/0.0%/0.0% 100
OutputOperator:285

Total Operator Elapsed Per Frame Time(us): 17247
Total Memory Read/Write Per Frame Size(KB): 56429.8

Operator Time Consuming Ranking Table

OpType	CallNumber	CPUTime(us)	GPUTime(us)	NPUTime(us)	TotalTime(us)	TimeRatio(%)
ConvRelu	50	0	0	10934	10934	63.40%
ConvReluAdd	7	0	0	1919	1919	11.13%
Concat	13	0	0	1784	1784	10.34%
ConvSigmoid	3	0	0	1252	1252	7.26%
Conv	1	0	0	732	732	4.24%
MaxPool	6	0	0	288	288	1.67%
Resize	2	0	0	191	191	1.11%
OutputOperator	3	139	0	0	139	0.81%
InputOperator	1	8	0	0	8	0.05%

---

done
--> eval_memory
==========

Memory Profile Info Dump
==========

NPU model memory detail(bytes):
Weight Memory: 6.96 MiB
Internal Tensor Memory: 7.42 MiB
Other Memory: 928.12 KiB
Total Memory: 15.29 MiB

```
INFO: When evaluating memory usage, we need consider
the size of model, current model size is: 7.96 MiB
=============================================================

done
```

上記のテストでは量子化が有効になっており、eval_perf を呼び出してモデル性能を評価し、テストモデルの全体の所要時間および各レイヤーの所要時間を取得しています。これが簡単な接続デバッグの一例です。

さらに多くの rknn-toolkit2 の機能テスト例については、https://github.com/airockchip/rknn-toolkit2/tree/master/rknn-toolkit2/examples/functions を参照してください。

22.2 RKNN Toolkit Lite2

RKNN Toolkit Lite2 は Rockchip NPU プラットフォームに Python プログラミングインターフェースを提供し、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。

22.2.1 ボード上での RKNN Toolkit Lite2 のインストール

Toolkit-lite2 は現在 Debian10/11（aarch64）システムに適しています。

RKNN Toolkit Lite2 を入手するには、公式の GitHub から直接取得するか、付属のサンプルから取得し、ボードにファイルを転送します。Toolkit Lite2 のディレクトリ構造は以下の通りです：

環境インストール（LubanCat4、Debian11 を例とします）：

```bash
sudo apt update
# その他の Python ツールをインストール
sudo apt-get install python3-dev python3-pip gcc
# 関連する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
sudo apt-get install -y python3-numpy
sudo apt -y install python3-setuptools
```

Toolkit Lite2 ツールのインストール：

```bash
# rknn_toolkit_lite2/packages ディレクトリに移動し、Debian11 ARM64 with python3.9.2 の whl フ
```

ァイルを選択してインストール
ホストPCから基板にファイルを転送
scp rknn_toolkit_lite2-2.0.0b0-cp39-cp39-linux_aarch64.whl cat@192.168.11.241:/home/cat
基板上に実行
pip3 install rknn_toolkit_lite2-2.0.0b0-cp39-cp39-linux_aarch64.whl
```

インストールが成功：
```bash
(.toolkit2_env) llh@-:~/project-Toolkit2$ python
Python 3.8.10 (default, Jun 22 2022, 20:18:18)
[GCC 9.4.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from rknnlite.api.rknn_lite import RKNNLite
>>>
```

22.2.2 ボード上でのデプロイと推論

`librknnrt.so` はボード上のランタイムライブラリであり、実行に必要です。ボードのデフォルトイメージの `/usr/lib` ディレクトリには `librknnrt.so` ライブラリが含まれていますが、バージョンを更新する必要があります。LubanCat4 ボードの場合、`project-Toolkit2/rknn-toolkit2/rknpu2/runtime/Linux/librknn_api/aarch64/` ディレクトリの `librknnrt.so` を選択し、こ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。

rknpu2 プロジェクトファイルは https://github.com/airockchip/rknn-toolkit2/tree/master/rknpu2 から入手できます。

RKNN Toolkit Lite2 のデモを実行し、取得した Toolkit Lite2 ディレクトリの `../examples/inference_with_lite` ディレクトリに入り、以下のコマンドを実行します：
cd lubancat_ai_manual_code/dev_env/rknn_toolkit_lite2/examples/inference_with_lite

```
cat@lubancat:~/lubancat_ai_manual_code/dev_env/rknn_toolkit_lite2/examples/inference_with_lite$ python3 test.py
--> Load RKNN model
done
--> Init runtime environment
I RKNN: [18:52:26.558] RKNN Runtime Information, librknnrt version: 2.0.0b0 (35a6907d79@2024-03-24T10:31:14)
I RKNN: [18:52:26.558] RKNN Driver Information, version: 0.9.2
I RKNN: [18:52:26.558] 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
resnet18
-----TOP 5-----
[812] score:0.999680 class:"space shuttle"
[404] score:0.000249 class:"airliner"
[657] score:0.000013 class:"missile"
[466] score:0.000009 class:"bullet train, bullet"
[833] score:0.000008 class:"submarine, pigboat, sub, U-boat"

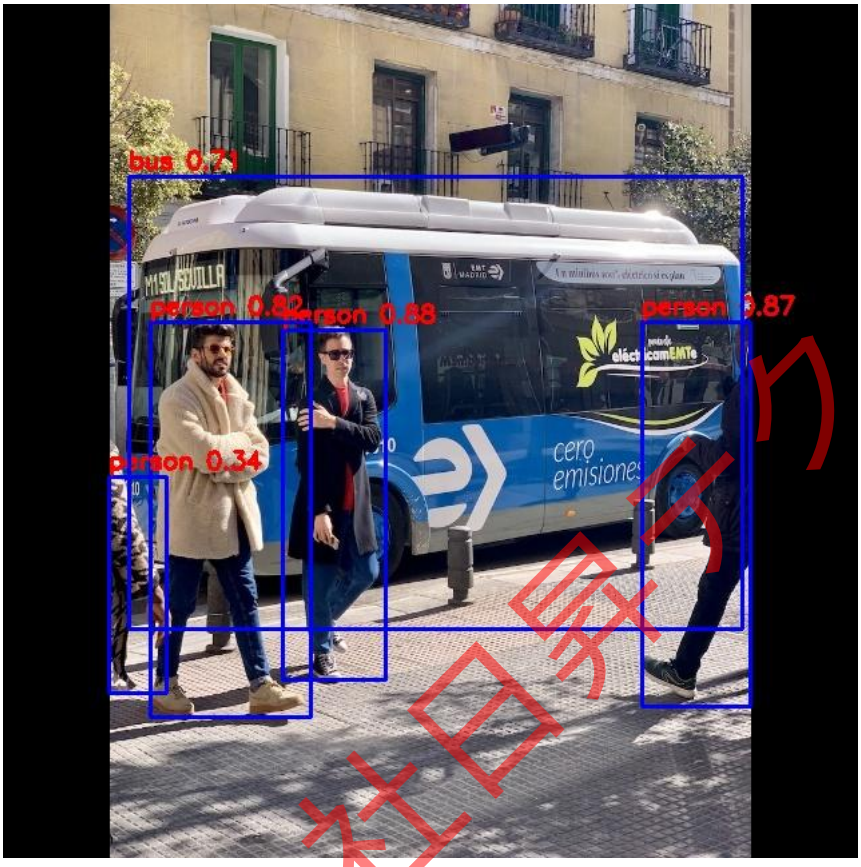
done
```

Toolkit Lite2 ツールの `examples/onnx/yolov5` サンプルを変更し、`test.py` を簡単に変更して RKNN Toolkit Lite2 を使用してデプロイし、前の RKNN-Toolkit2 で変換した `yolov5s.rknn` をインポートします（変更後のソースファイルは付属のサンプルを参照）：
```

正常に実行されると、画像が表示され（または出力画像 `out.jpg` を確認）されます：

```
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: [16:25:52.333] RKNN Runtime Information, librknnrt version: 2.0.0b0 (35a6907d79@2024-03-24T10:31:14)
I RKNN: [16:25:52.333] RKNN Driver Information, version: 0.9.2
I RKNN: [16:25:52.333] 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 typ
e: static_shape
done
--> Running model
Save results to result.jpg!
```

ヒント：デフォルトのサンプル `../yolov5/test.py` は `cv2.imshow()` を使用して画像を表示しません。



22.3 関連する周波数設定（rk3588）

LubanCat ボードの CPU はデフォルトで `interactive` 状態にあり、CPU 使用率と目標負荷に基づいて CPU 周波数を動的に調整します。より高い動作速度を得るか、性能を評価するために、手動で周波数を固定する必要があります。以下を参考にしてください：

1. CPU 周波数設定：

```bash
# 現在の CPU 周波数を確認
cat /sys/devices/system/cpu/cpu0/cpufreq/scaling_cur_freq
# 現在の CPU の調頻戦略を確認
cat /sys/devices/system/cpu/cpufreq/policy0/scaling_governor
# 設定可能な周波数を確認
cat /sys/devices/system/cpu/cpufreq/policy0/scaling_available_frequencies
```

```
# root 権限のユーザーが sysfs の “scaling_setspeed” フィールドを通じて CPU 周波数を設定することを許可
echo userspace > /sys/devices/system/cpu/cpufreq/policy0/scaling_governor
# 固定したい周波数を設定
echo 1800000 > /sys/devices/system/cpu/cpufreq/policy0/scaling_setspeed
```

2. DDR 周波数設定：

```bash
# 現在の DDR 周波数を確認
cat /sys/class/devfreq/dmc/cur_freq
# 現在の DDR の調頻戦略を確認
cat /sys/class/devfreq/dmc/governor
# 設定可能な DDR 周波数を確認
cat /sys/class/devfreq/dmc/available_frequencies
# root 権限のユーザーが sysfs の “scaling_setspeed” フィールドを通じて CPU 周波数を設定することを許可
echo userspace > /sys/class/devfreq/dmc/governor
# 固定したい周波数を設定、ここでは 1560000000
echo 1560000000 > /sys/devices/system/cpu/cpufreq/policy0/scaling_setspeed
```

3. NPU 周波数設定：

```bash
# 利用可能な NPU 周波数を確認
cat /sys/class/devfreq/fdab0000.npu/available_frequencies
echo userspace > /sys/class/devfreq/fdab0000.npu/governor
# 周波数を設定
echo 1000000000 > /sys/kernel/debug/clk/clk_scmi_npu/clk_rate
# 現在の NPU 周波数を確認
cat /sys/class/devfreq/fdab0000.npu/cur_freq
# または以下のコマンドを使用して確認
sudo cat /sys/kernel/debug/clk/clk_summary | grep npu
```

22.4 参考

- RKNPU2 関連ドキュメントとソース：
  https://github.com/airockchip/rknn-toolkit2/tree/master/rknpu2
- RKNN-toolkit2 関連ドキュメントとソース：
  https://github.com/airockchip/rknn-toolkit2

第 23 章 手書き数字認識 - PaddlePaddle

23.1 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。

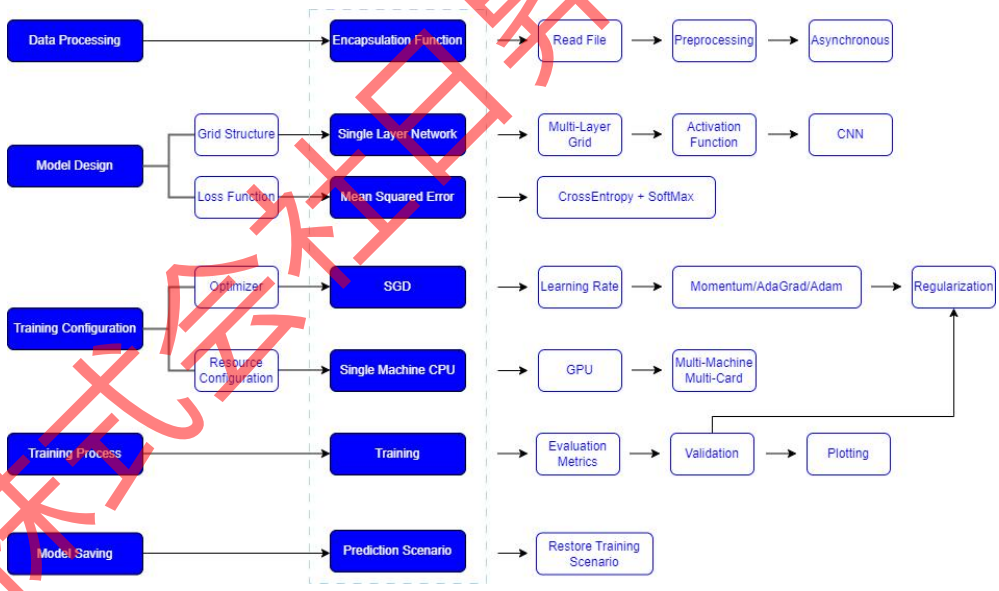
注意: テスト環境は Lubancat ボードで Debian11 を使用し、PC は ubuntu20.04 です。PaddlePaddle は CPU バージョンであり、rknn-Toolkit2 バージョンは 2.0.0 です。

23.2 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。

## 23.3 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

## 23.3.1 訓練データセットとテストデータセットの準備

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。

```python
# 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)
```

23.3.2 モデルネットワークの構築

数字認識の過程では、Paddle に内蔵された LeNet モデルを直接使用し、上層部の API を使用して、1 行のコードで LeNet のネットワーク構築と初期化を完了します。独自にネットワークを構築することもできます。

```python
# モデルネットワークを構築し、ネットワークを初期化します。
lenet = paddle.vision.models.LeNet(num_classes=10)
model = paddle.Model(lenet)
# paddle.summary(lenet,(1, 1, 28, 28))
```

LeNet モデルは 2 つの Conv2D 畳み込み層、2 つの ReLU 活性化層、2 つの MaxPool2D プーリング層、および 3 つの Linear 全結合層で構成されています。上記 paddle.summary(lenet,(1, 1, 28, 28)) のコメントを解除して、ネットワークの構造とパラメーターの統計をターミナルでプリントします：

```python
---------------------------------------------------------------------------
 Layer (type)       Input Shape          Output Shape         Param #
===========================================================================
   Conv2D-1       [[1, 1, 28, 28]]      [1, 6, 28, 28]          60
    ReLU-1        [[1, 6, 28, 28]]      [1, 6, 28, 28]           0
 MaxPool2D-1      [[1, 6, 28, 28]]      [1, 6, 14, 14]           0
   Conv2D-2       [[1, 6, 14, 14]]      [1, 16, 10, 10]       2,416
    ReLU-2        [[1, 16, 10, 10]]     [1, 16, 10, 10]          0
 MaxPool2D-2      [[1, 16, 10, 10]]      [1, 16, 5, 5]           0
   Linear-1           [[1, 400]]            [1, 120]          48,120
   Linear-2           [[1, 120]]            [1, 84]           10,164
   Linear-3           [[1, 84]]             [1, 10]            850
```

```
==========================================================================
Total params: 61,610
Trainable params: 61,610
Non-trainable params: 0
--------------------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.11
Params size (MB): 0.24
Estimated Total Size (MB): 0.35
--------------------------------------------------------------------------

```

23.3.3 モデルの訓練と評価

モデルの訓練には多くの反復（epoch）が含まれます。各反復では訓練データセットを一度全体的に遍歴し、その中から小さなバッチ（mini-batch）のサンプルを取得し、モデルに送信して前方計算を実行し、予測値を取得し、予測値（predict_label）と真の値（true_label）の間の損失関数値（loss）を計算します。次に、逆伝播を行い、設定された最適化アルゴリズム（optimizer）に基づいてモデルのパラメーターを更新します。各反復の損失値の減少傾向を観察することで、モデルの訓練効果を判断できます。

```python
# 最適化器およびその学習率を設定し、ネットワークのパラメーターを最適化器に渡し、損失関数と精度計算方法を設定します。
# optimizer 最適化器は、最適な解を見つける方法です。Loss は損失関数です。metric は評価指標です。
# ここでは Adam 最適化器を使用し、交差エントロピー損失関数 CrossEntropyLoss を使用して分類タスクを評価します。
model.prepare(paddle.optimizer.Adam(parameters=model.parameters()),
              paddle.nn.CrossEntropyLoss(),
              paddle.metric.Accuracy())

# モデルの訓練、epoch は訓練の反復回数、batch_size はバッチのサイズ、verbose=1 は訓練中のログを表示します。
model.fit(train_dataset, epochs=5, batch_size=64, verbose=1)

# モデルの評価、テスト

```

データセットを訓練済みのモデルに入力して評価し、予測値を取得し、予測値と真の値の間の損失関数値（loss）を計算します。

```
# また、評価指標値（metric）を計算してモデルの効果を評価します。
model.evaluate(test_dataset, batch_size=64, verbose=1)
```

23.3.4 モデルの保存と ONNX モデルのエクスポート

訓練済みのモデルはファイルに保存することができ、後で再訓練や推論のデプロイ時にロードして実行することができます。

```python
# モデルの保存
model.save('./output/mnist')

# モデルのロード
model.load('output/mnist')

# テストデータセットから一枚の画像を取り出します。
img, label = test_dataset[0]

# 画像の shape を 1*28*28 から 1*1*28*28 に変更し、バッチの次元を追加して、モデルの入力形式に一致させます。
img_batch = np.expand_dims(img.astype('float32'), axis=0)

# 推論を実行し、結果を表示します。この場合、predict_batch はリストを返し、その中のデータを取り出して予測結果を取得します。
out = model.predict_batch(img_batch)[0]
pred_label = out.argmax()
print('true label: {}, pred label: {}'.format(label[0], pred_label))
```

Paddle モデルを ONNX フォーマットに変換するには、paddle.onnx.export インターフェースを呼び出すだけで、指定されたパスに ONNX モデルが生成されます。また、Paddle モデルをデプロイメントモデル（静的グラフモデル）として保存し、Paddle2ONNX コマンドラインツールを使用して変換することもできます。

```python
# 保存するパスを指定します。ここでは onnx ディレクトリに保存します。
save_path = 'onnx/lenet'

# モデルに対して入力の形状とデータ型を指定します。
x_spec = paddle.static.InputSpec([1, 1, 28, 28], 'float32', 'x')

# ONNX モデルを生成します。
paddle.onnx.export(lenet, save_path, input_spec=[x_spec], opset_version=11)
```

23.3.5 完全なプログラム

```python
```

サンプルソースコード：paddlepaddle/digital_recognition.py

```python
import paddle
import numpy as np
from paddle.vision.transforms import Normalize

transform = Normalize(mean=[127.5], std=[127.5], data_format='CHW')
# データセットをダウンロードし、DataSet を初期化
train_dataset = paddle.vision.datasets.MNIST(mode='train', transform=transform)
test_dataset = paddle.vision.datasets.MNIST(mode='test', transform=transform)

# モデルのネットワークを構築し、ネットワークを初期化
lenet = paddle.vision.models.LeNet(num_classes=10)
model = paddle.Model(lenet)
paddle.summary(lenet, (1, 1, 28, 28))

# モデル訓練の準備、損失関数、オプティマイザ、評価指標を準備
model.prepare(paddle.optimizer.Adam(learning_rate=0.001, parameters=model.parameters()),
              paddle.nn.CrossEntropyLoss(),
              paddle.metric.Accuracy())

# モデル訓練
model.fit(train_dataset, epochs=5, batch_size=64, verbose=1)
# モデル評価
model.evaluate(test_dataset, batch_size=64, verbose=1)

# モデルを保存
model.save('./output/mnist')
# モデルをロード
# model.load('output/mnist')

# テストセットから 1 枚の画像を取得
img, label = test_dataset[0]
# 画像の shape を 1*28*28 から 1*1*28*28 に変換し、バッチ次元を追加してモデル入力フォーマットに一致させる
img_batch = np.expand_dims(img.astype('float32'), axis=0)

# 推論を実行し、結果を出力。この predict_batch はリストを返すため、そのデータを取り出して予測結果を取得
out = model.predict_batch(img_batch)[0]
pred_label = out.argmax()
print('true label: {}, pred label: {}'.format(label[0], pred_label))

# ONNX にエクスポート
save_path = 'onnx/lenet' # 保存先のパス
x_spec = paddle.static.InputSpec([1, 1, 28, 28], 'float32', 'x')
# モデルの入力形状とデータ型を指定。Tensor または InputSpec をサポート。InputSpec は動的形状をサポート。
paddle.onnx.export(lenet, save_path, input_spec=[x_spec], opset_version=12)
```

#実行結果：

The loss value printed in the log is the current step, and the metric is the average value of previous steps.

Epoch 1/5

step 938/938 [==============================] - loss: 0.0536 - acc: 0.9301 - 4ms/step

Epoch 2/5

step 938/938 [==============================] - loss: 0.0376 - acc: 0.9753 - 4ms/step

Epoch 3/5

step 938/938 [==============================] - loss: 0.0019 - acc: 0.9802 - 4ms/step

Epoch 4/5

step 938/938 [==============================] - loss: 0.0014 - acc: 0.9826 - 4ms/step

Epoch 5/5

step 938/938 [==============================] - loss: 0.0086 - acc: 0.9852 - 4ms/step

Eval begin...

step 157/157 [==============================] - loss: 5.6002e-04 - acc: 0.9825 - 4ms/step

Eval samples: 10000

true label: 7, pred label: 7

I0608 07:50:09.890259 649836 program_interpreter.cc:212] New Executor is Running.

2024-06-08 07:50:09 [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...

[Paddle2ONNX] Use opset_version = 12 for ONNX export.

[Paddle2ONNX] PaddlePaddle model is exported as ONNX format now.

2024-06-08 07:50:09 [INFO]	ONNX model saved in onnx/lenet.onnx.

23.3.6 モデルの変換とシミュレーション推論

RKNN-Toolkit2 を使用してモデルの変換とシミュレーション推論を行います。

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

INPUT_SIZE = 28
IMG_PATH = '0.jpg'

# RKNN 実行オブジェクトを作成し、情報を表示
# rknn = RKNN(verbose=True)
rknn = RKNN()

# モデル入力を設定し、NPU に対するデータ入力の前処理を行う
```

```
# mean_values:入力の平均値
# std_values:入力の標準化値
# target_platform: 目標プラットフォーム
# 詳細は RKNN toolkit2 ユーザーガイドを参照
print('--> モデルを設定中')
rknn.config(mean_values=[127.5], std_values=[127.5], target_platform='rk3568')
# rknn.config(target_platform='rk3568')

# ONNX モデルをロード
# inputs はモデル内の入力ノードを指定
# outputs はモデル内の出力ノードを指定
# input_size_list はモデル入力のサイズを指定
print('--> モデルをロード中')
rknn.load_onnx(model='./onnx/lenet.onnx')
print('完了')

print('--> モデルをビルド中')
ret = rknn.build(do_quantization=False)
print('完了')

# RKNN 実行環境を初期化し、テストを実行
print('--> 実行環境を初期化中')
ret = rknn.init_runtime()
#    if ret != 0:
#       print('実行環境の初期化に失敗')
#        exit(ret)
print('完了')

# 画像
img = cv2.imread(IMG_PATH)
img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
img = cv2.resize(img, (28, 28))
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("結果: ", outputs)
print("予測した数字は:", np.argmax(outputs))

# rknn モデルファイルをエクスポート
print('--> rknn モデルをエクスポート中')
rknn.export_rknn('./lenet.rknn')
print('完了')

# rknn を解放
rknn.release()
```

paddlepaddle/rknn_transfer.py

#rknn モデルに変換及び推論実行結果：

```
I rknn-toolkit2 version: 2.0.0b0+9bab5682
--> モデルを設定中
--> モデルをロード中
I It is recommended onnx opset 19, but your onnx model opset is 12!
I Loading : 100%|███████████████████████████████████████████| 14/14 [00:00<00:00, 85974.02it/s]
完了
--> モデルをビルド中
I rknn building ...
I rknn buiding done.
完了
--> 実行環境を初期化中
I Target is None, use simulator!
完了
--> モデルを実行中
I GraphPreparing : 100%|███████████████████████████████████| 11/11 [00:00<00:00, 5613.50it/s]
I SessionPreparing : 100%|██████████████████████████████████| 11/11 [00:00<00:00, 2341.05it/s]
W inference: The dims of input(ndarray) shape (1, 28, 28) is wrong, expect dims is 4! Try expand dims to (1, 1, 28, 28)!
結果:  [array([[ 6.7890625 ,  2.6894531 ,  2.0117188 , -3.6484375 , -2.0664062 ,
         0.32739258,  0.27514648, -2.1621094 , -2.0664062 ,  1.1611328 ]],
      dtype=float32)]
予測した数字は: 0
--> rknn モデルをエクスポート中
完了
```

23.4 ボードでの推論デプロイ

23.4.1 RKNN Toolkit Lite2 と関連ライブラリのインストール

Toolkit Lite2 のインストールについては前述の「NPU の使用」章を参照してください。

```bash
# ボードで関連ライブラリをインストール
sudo apt update
sudo apt -y install python3-pil python3-numpy python3-pip git wget
sudo apt install python3-opencv
```

23.4.2 推論テスト

前述の rknn モデルを使用し、テストプログラムを作成します（配布されているサンプルコードを使用することもできます）。

```python
```

サンプルソースコード：paddlepaddle/rknn_test.py

```python
import numpy as np
import cv2
from rknnlite.api import RKNNLite

IMG_PATH = '0.jpg'
RKNN_MODEL = 'lenet.rknn'

# 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
img = cv2.imread(IMG_PATH)
if img is None:
    print(f"Error: Failed to load image {IMG_PATH}.")
    exit(1)

# グレースケール化とリサイズ
img_gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
img_resize = cv2.resize(img_gray, (28, 28))

# 画像の前処理
im = img_resize.astype(np.float32)
im = np.expand_dims(im, 0)  # バッチ次元を追加
im = np.expand_dims(im, 0)  # チャンネル次元を追加

# 推論
print('--> Running model')
outputs = rknn_lite.inference(inputs=[im])
print("result: ", outputs)
print("今回認識された数字は:", np.argmax(outputs))

rknn_lite.release()
```

実行結果：

```
cat@lubancat:~/python/code/ai/paddlepaddle$ python3 rknn_test.py
--> Load RKNN model
done
--> Init runtime environment
I RKNN: [09:17:36.398] RKNN Runtime Information, librknnrt version: 2.0.0b0 (35a6907d79@2024-03-24T10:31:14)
I RKNN: [09:17:36.398] RKNN Driver Information, version: 0.9.2
I RKNN: [09:17:36.398] 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([[-4.9882812 , -3.0859375 , -2.3046875 ,  1.7929688 ,  3.6523438 ,
        -0.44580078, -7.3789062 ,  2.3085938 ,  4.3125    ,  8.7109375 ]],
      dtype=float32)]
今回認識された数字は: 9
```

23.5 まとめ

以上は LeNet を使用して手書き数字データセット MNIST を分類する簡単な例であり、トレーニングおよびテストのためのインターフェースを呼び出してモデルの構築と予測を迅速に完了します。これは簡単な学習リファレンスとして使用されます。モデルを一歩一歩構築してトレーニングする方法については、[PaddlePaddle 初心者向け実践ガイド] を参照してください。

23.6 参考リンク

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

第 24 章 PaddlePaddle FastDeploy

24.1 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。

24.2 環境設定

24.2.1 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 22 章 NPU の使用》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(≪23.2 PaddlePaddle のインストール≫を参考)
python -m pip3 install paddlepaddle
```

24.2.2 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 22 章 NPU の使用》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
```

24.3 デプロイ推論の例

24.3.1 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)

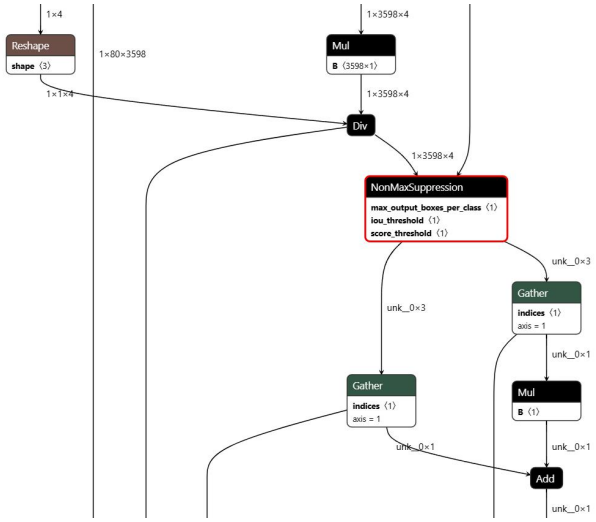
1. PC 側モデル変換

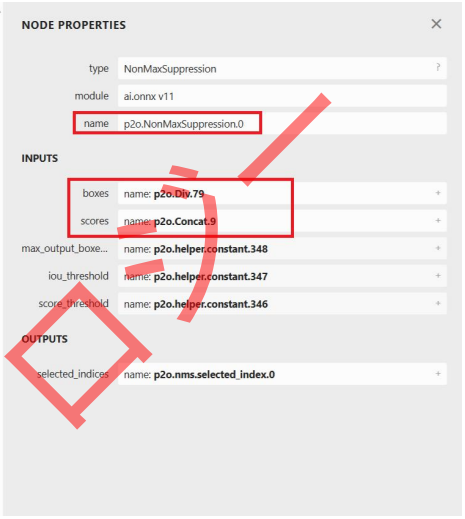
rknn-Toolkit2 は現在 Paddle モデルを直接 RKNN モデルとしてエクスポートすることをサポートしていません。Paddle2ONNX を使用して onnx モデルに変換し、次に rknn モデルをエクスポートする必要があります。

```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
```

24.4 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 25 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。

25.1 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。

25.2 画像分類

ここでは、tf.keras.Sequential モデルを使用して花卉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。

25.2.1 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。

25.2.2 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。

25.2.3 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)
```

25.2.4 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。

25.2.5 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
```

25.3 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.
```

25.4 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。

25.5 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 26 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。

26.1 PyTorch と ResNet18

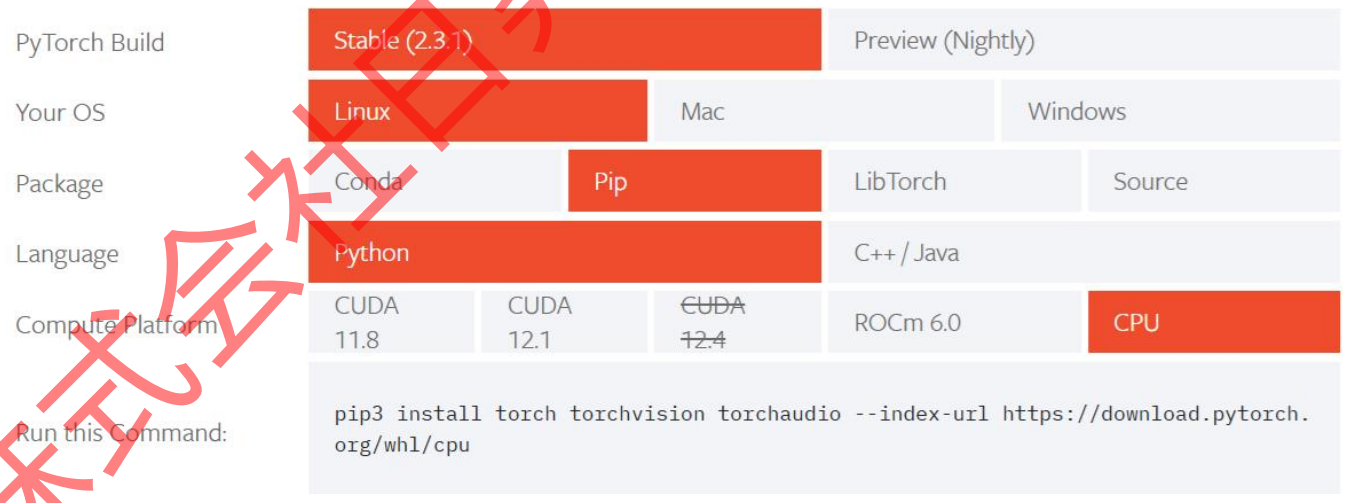
PyTorch は、Facebook 人工知能研究所の Torch7 チームによって開発されたオープンソースのディープラーニングフレームワークです。このフレームワークの基礎は Torch に基づいていますが、実装と使用はすべて Python で行われます。このフレームワークは主に人工知能分野の科学研究と応用開発に使用されます。

26.1.1 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。

26.1.2 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。

26.1.3 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

26.2 ResNet18 の実装

26.2.1 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
])
```

26.2.2 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)
)
```

26.2.3 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 %
```

26.2.4 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'])
```

26.2.5 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()
```

26.2.6 基板上でのデプロイとテスト

26.2.6.1 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.
```

26.2.6.2 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
```

26.3 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 27 章 YOLOv5

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。

27.1 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。

仮想環境を作成（名前は.yolov5_env）、ここでは venv を使用しますが、先に anaconda をインストールして、conda コマンドを使ってもよい

注：本章独自実践の場合、下記のようなコマンドでパッケージをインストールします、そうしないと、基本的に、前の仮装環境.toolkit2_env を使ってください。

足りないパッケージのみをインストール：

```
source ~/project-Toolkit2/.toolkit2_env/bin/activate
pip install seaborn ultralytics thop gitpython
```

バージョンアップ：

```
pip install --upgrade setuptools
#本章独自実践の場合
sudo python3 -m venv .yolov5_env
source .yolov5_env/bin/activate
# 最新の yolov5 リポジトリをクローンします（pytorch 版）
git clone https://github.com/ultralytics/yolov5
cd yolov5

# 依存ライブラリをインストール
pip3 install -r requirements.txt
# Python コマンドラインに入り、インストールされた環境を確認
import torch
import utils
```

```
display = utils.notebook_init()
# テスト環境の表示：
>>> import torch
>>> import utils
>>> display = utils.notebook_init()
Checking setup...
Downloading https://ultralytics.com/assets/Arial.ttf to
/home/csun/.config/Ultralytics/Arial.ttf...
YOLOv5 v7.0-321-g3742ab49 Python-3.8.10 torch-2.1.0+cu121 CUDA:0 (NVIDIA GeForce GTX 1070
Ti, 8106MiB)
Setup complete (12 CPUs, 23.4 GB RAM, 67.6/853.7 GB disk)
```

27.2 YOLOv5 簡単使用

27.2.1 事前トレーニング済みの重みファイルの取得

`yolov5s.pt`、`yolov5m.pt`、`yolov51.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

27.2.2 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。
```

具体的なコマンド例は以下の通りです：

```
python detect.py --source ./data/images --weights yolov5s.pt
```

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：

```
(.toolkit2 env) (base) csun@CSUN-PC-0013:~/linuxshare/work/AI/jupyter/python_lubancat_RK_tutorial_code/ai/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_tx
t=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, n
ame=exp, exist_ok=False, line_thickness=3, hide_labels=False, hide_conf=False, half=False, dnn=False, vid_stride=1
YOLOv5 🚀 v7.0-321-g3742ab49 Python-3.8.10 torch-2.1.0+cu121 CUDA:0 (NVIDIA GeForce GTX 1070 Ti, 8106MiB)

Fusing layers...
YOLOv5s summary: 213 layers, 7225885 parameters, 0 gradients, 16.4 GFLOPs
WARNING ⚠ NMS time limit 0.550s exceeded
image 1/2 /home/csun/linuxshare/work/AI/jupyter/python_lubancat_RK_tutorial_code/ai/yolov5/data/images/bus.jpg: 640x480 4 persons, 1 bus, 30.5ms
image 2/2 /home/csun/linuxshare/work/AI/jupyter/python_lubancat_RK_tutorial_code/ai/yolov5/data/images/zidane.jpg: 384x640 2 persons, 2 ties, 34.3ms
Speed: 2.3ms pre-process, 32.4ms inference, 420.5ms NMS per image at shape (1, 3, 640, 640)
Results saved to runs/detect/exp
```

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。

27.2.3 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
注：以前基本的にインストールされましたので、以下の 2 つパッケージをインストール：
pip install coremltools onnx-simplifier
# 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：

```
(.toolkit2_env) (base) csun@CSUN-PC-0013:~/linuxshare/work/AI/jupyter/python_lubancat_RK_tutorial_code/ai/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    v7.0-321-g3742ab49 Python-3.8.10 torch-2.1.0+cu121 CPU

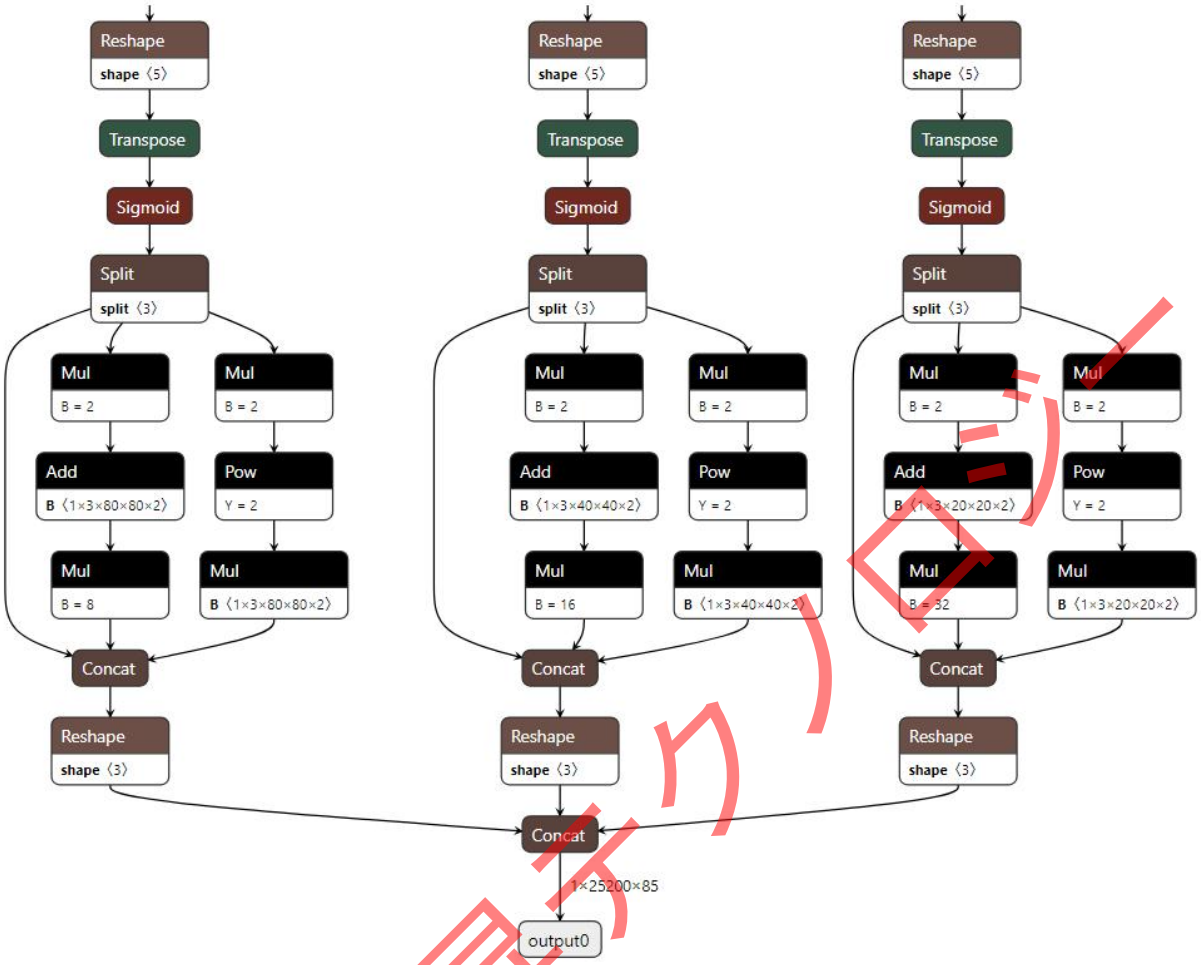
Fusing layers...
YOLOv5s summary: 213 layers, 7225885 parameters, 0 gradients, 16.4 GFLOPs

PyTorch: starting from yolov5s.pt with output shape (1, 25200, 85) (14.1 MB)

ONNX: starting export with onnx 1.14.1...
ONNX: export success ✓ 0.9s, saved as yolov5s.onnx (28.0 MB)

Export complete (1.8s)
Results saved to /home/csun/linuxshare/work/AI/jupyter/python_lubancat_RK_tutorial_code/ai/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

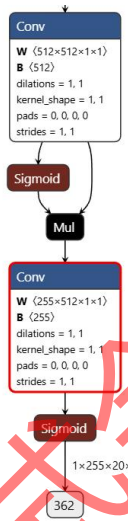
```
# export.py の run()関数を修正

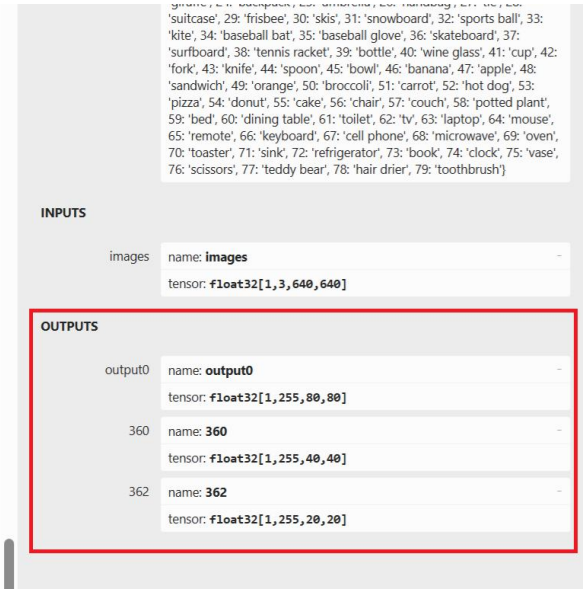
    if half and not coreml:
        im, model = im.half(), model.half()  # to FP16
-   shape = tuple((y[0] if isinstance(y, tuple) else y).shape)  # model output shape
+   shape = tuple((y[0] if (isinstance(y, tuple) or (isinstance(y, list))) else y).shape)  # model output shape
    metadata = {'stride': 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 22 章 NPU の使用》章を参考にしてください。

```
# 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 os
import cv2
import traceback
import numpy as np
from rknn.api import RKNN

ONNX_MODEL = 'yolov5s.onnx'
RKNN_MODEL = 'yolov5s.rknn'
IMG_PATH = './bus.jpg'
DATASET = './dataset.txt'
PYTORCH_MODEL = './yolov5_relu.pt'
IMG_SIZE = 640


if __name__ == '__main__':
    # RKNN の作成
    # テスト中に問題が発生した場合は、verbose=True を設定してデバッグ情報を確認してください。
    #rknn = RKNN(verbose=True)
    rknn = RKNN()

    # 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))

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

    # Load pytorch model
    print('--> Loading model')
    ret = rknn.load_pytorch(model=PYTORCH_MODEL, input_size_list=[[1,3,IMG_SIZE,IMG_SIZE]])
    if ret != 0:
        print('Load model failed!')
        exit(ret)
    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')

    # モデルを構築します。デフォルトで量子化が有効になっています。
    print('--> Building model')
    ret = rknn.build(do_quantization=True, dataset=DATASET)
    #ret = rknn.build(do_quantization=False)
    if ret != 0:
        print('Build model failed!')
        exit(ret)
    print('done')

    # Accuracy analysis
    #print('--> Accuracy analysis')
    #Ret = rknn.accuracy_analysis(inputs=[img])
    #if ret != 0:
    #    print('Accuracy analysis failed!')
    #    exit(ret)
    #print('done')


    # RKNN モデルをエクスポート
    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()
    #ret = rknn.init_runtime(target='rk3588', device_id='192.168.103.115:5555', perf_debug=True)
    if ret != 0:
        print('Init runtime environment failed!')
        exit(ret)
    print('done')

    # デバッグとモデル性能の評価
    #rknn.eval_perf(inputs=[img], is_print=True)

    rknn.release()
```

rknn-Toolkit2 のモデル変換機能を使用：

ソースコード 3：rknn_transfer.py（サンプルを参照）

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、eval_perf インターフェースのコメントを解除すると、性能評価が実施され、簡単なデバッグが可能です。

27.2.4 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：

```
# 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 os
import cv2
import traceback
import numpy as np
from rknn.api import RKNN

ONNX_MODEL = 'yolov5s.onnx'
RKNN_MODEL = 'yolov5s.rknn'
IMG_PATH = './bus.jpg'
DATASET = './dataset.txt'
PYTORCH_MODEL = './yolov5_relu.pt'
IMG_SIZE = 640

if __name__ == '__main__':
    # RKNN の作成
    # テスト中に問題が発生した場合は、verbose=True を設定してデバッグ情報を確認してください。
    #rknn = RKNN(verbose=True)
    rknn = RKNN()

    # 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))

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

# Load pytorch model
print('--> Loading model')
ret = rknn.load_pytorch(model=PYTORCH_MODEL, input_size_list=[[1,3,IMG_SIZE,IMG_SIZE]])
if ret != 0:
    print('Load model failed!')
    exit(ret)
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')

# モデルを構築します。デフォルトで量子化が有効になっています。
print('--> Building model')
ret = rknn.build(do_quantization=True, dataset=DATASET)
#ret = rknn.build(do_quantization=False)
if ret != 0:
    print('Build model failed!')
    exit(ret)
print('done')

# Accuracy analysis
#print('--> Accuracy analysis')
#Ret = rknn.accuracy_analysis(inputs=[img])
#if ret != 0:
#    print('Accuracy analysis failed!')
#    exit(ret)
#print('done')

# RKNN モデルをエクスポート
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()
#ret = rknn.init_runtime(target='rk3588', device_id='192.168.103.115:5555', perf_debug=True)
if ret != 0:
    print('Init runtime environment failed!')
    exit(ret)
print('done')

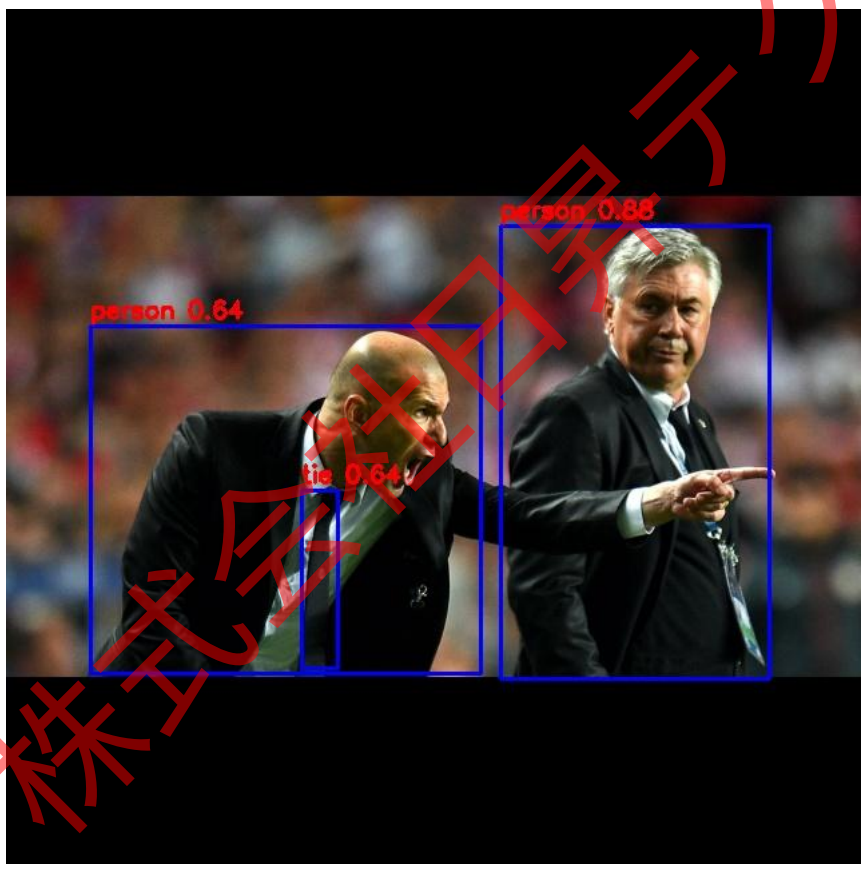
# デバッグとモデル性能の評価
#rknn.eval_perf(inputs=[img], is_print=True)

rknn.release()
```

リスト 4: rknnlite_inference.py（部分を抜粋、対応するサンプルを参照）

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：

```
cat@lubancat:~/python/code/ai/yolov5$ python rknnlite_inference.py
--> Load RKNN model
done
--> Init runtime environment
I RKNN: [13:23:04.306] RKNN Runtime Information, librknnrt version: 2.0.0b0 (35a6907d79@2024-03-24T10:31:14)
I RKNN: [13:23:04.306] RKNN Driver Information, version: 0.9.2
I RKNN: [13:23:04.306] RKNN Model Information, version: 6, toolkit version: 2.0.0b0+9bab5682(compiler version: 2.0.0b0
(35a6907d79@2024-03-24T02:34:11)), target: RKNPUget platform: rk3588, framework name: PyTorch, framework layout: NCHW, model
inference type: static_shape
done
--> Running model
done
class: person, score: 0.881165087223053
box coordinate left,top,right,down: [370, 162, 571, 501]
class: person, score: 0.6419469714164734
box coordinate left,top,right,down: [63, 237, 355, 497]
class: tie, score: 0.6430739760398865
box coordinate left,top,right,down: [221, 360, 248, 493]
```



27.3 airockchip/yolov5 簡単テスト

上記は公式の YOLOv5 v7.0 を使用した簡単なテストです。次に、airockchip/yolov5 を使用します。このリポジトリの yolov5 は rknpu デバイス向けにデプロイ最適化されています。テスト時のバージョンは

v6.2 です。

最新の airockchip/yolov5 リポジトリをクローンし、インストール方法は類似しているので、前述を参考にしてください：

```
# 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
cd yolov5
# 重みファイルを取得するために再トレーニングが必要です：
python3 train.py --data coco128.yaml --weights yolov5s.pt  --img 640
# 再トレーニングを行い、最適化された重みファイルを取得します。ここでは`yolov5s.pt`に基づいています。または--cfg を使用して設定ファイルを指定します
python3 export.py --weights runs/train/exp/weights/best.pt --rknpu rk3588
```

トレーニング後の出力情報：

```
...
300 epochs completed in 0.383 hours.
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...
...
```

トレーニング分析と重みは`runs/train/exp/`ディレクトリに保存されます。これを`yolov5s_relu.pt`とリネームし、ソースコードのルートディレクトリにコピーします。次に torchscript をエクスポートします：

```
# --weights は重みファイルのパスを指定
# --rknpu はプラットフォームを指定（rk_platform は rk1808、rv1109、rv1126、rk3399pro、rk3566、rk3568、rk3588、rv1103、rv1106 をサポート）
# --include は onnx モデルのエクスポートを指定し、指定しない場合はデフォルトで torchscript になります
python3 export.py --weights yolov5s_relu.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。現在のディレクトリに`yolov5s.onnx`ファイルが生成されます。
python export.py --weights yolov5s.pt --include onnx --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 yolov5_relu.pt
```

あと、前の変換 python ファイル等をここコピーしてください。

```
yolov5$ cp ../yolov5-org/bus.jpg ./
yolov5$ cp ../yolov5-org/dataset.txt ./
yolov5$ cp ../yolov5-org/rknn_transfer.py ./
```

27.3.1 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。

(.toolkit2_env) (base) csun@CSUN-PC-0013:~/linuxshare/work/AI/jupyter/python_lubancat_RK_tutorial_code/ai/yolov5$ python rknn_transfer.py

ヒント: テスト時の CPU 周波数は 1.8Ghz、DDR 周波数は 1.56Ghz、NPU 周波数は 1000Mhz です。後のデプロイテストは前述と類似しているので、省略します。

27.4 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 28 章 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

28.1 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
```

28.2 ビデオストリームサーバーとカメラフレームの取得

28.2.1 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
......
```

28.3 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

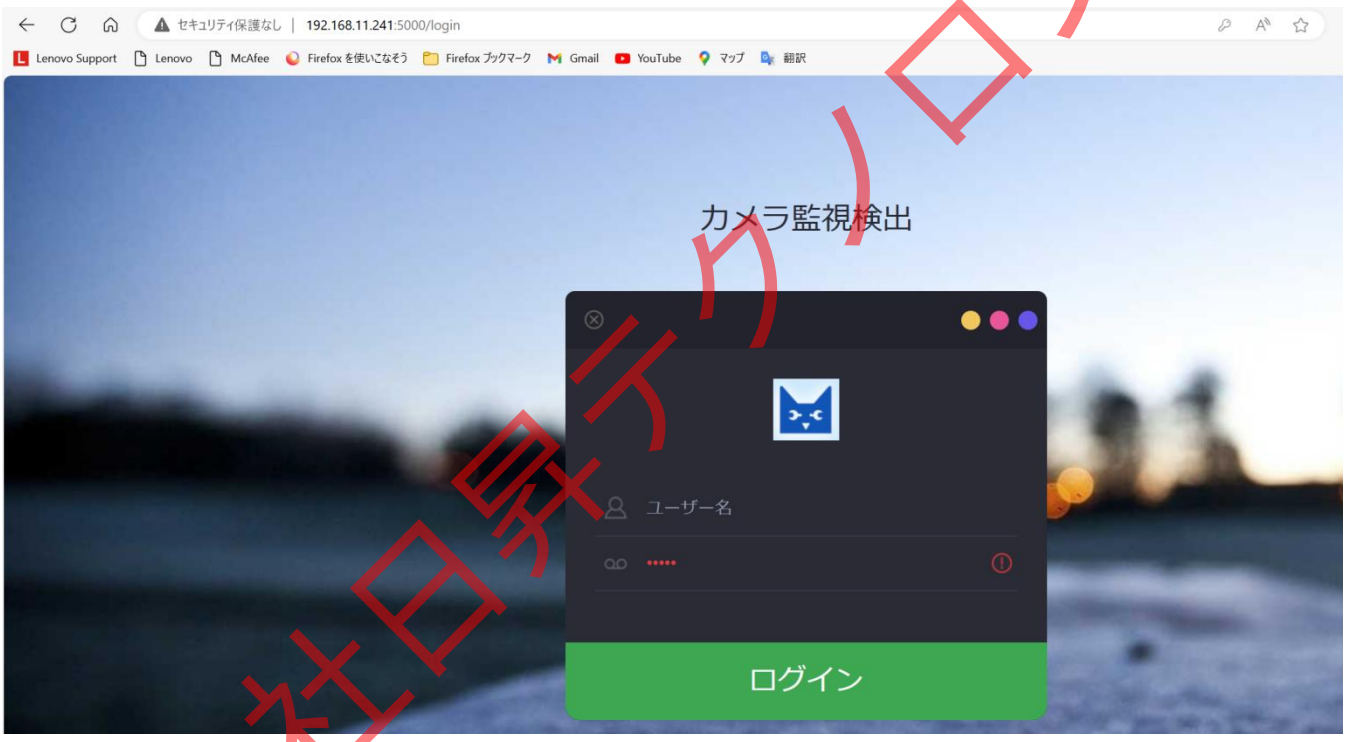
```
self.cap = cv2.VideoCapture(11)
```

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
```

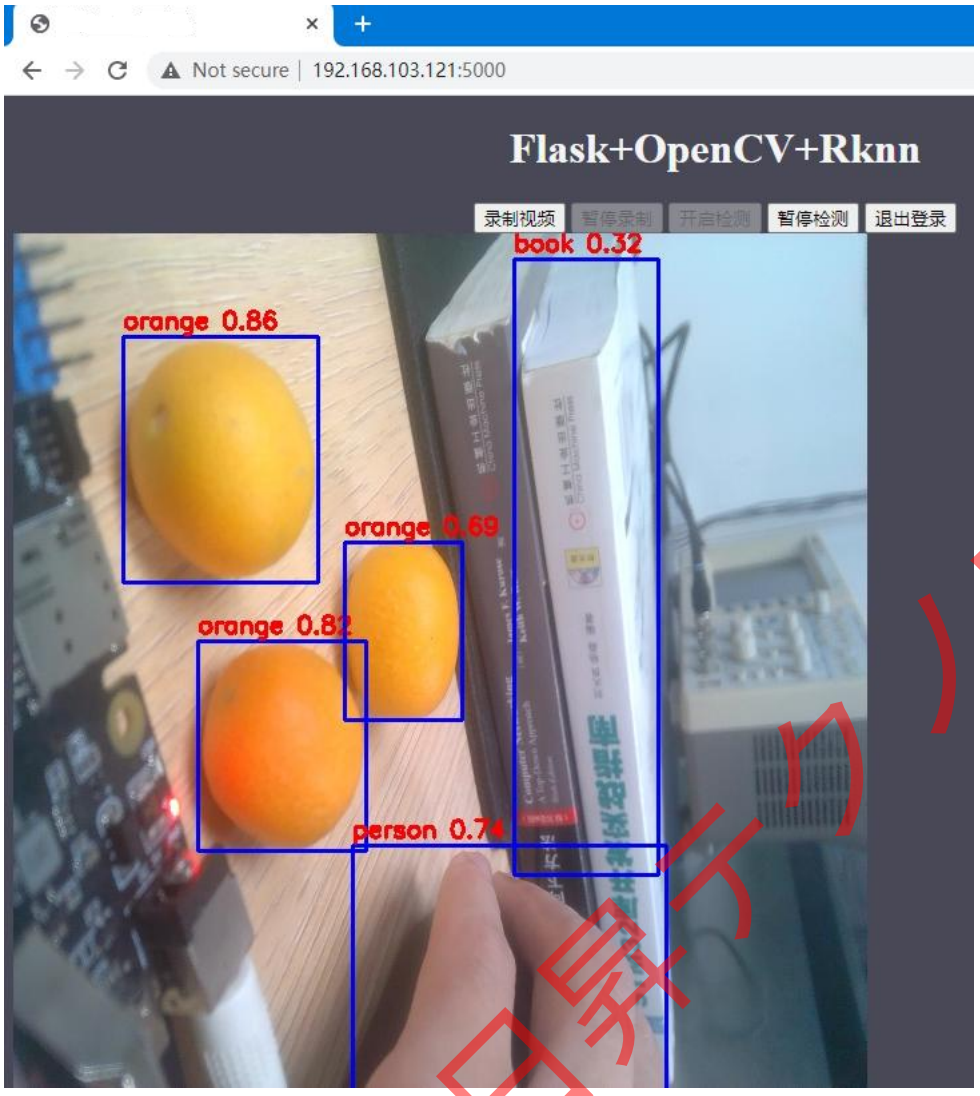
実験の現象を以下の URL でブラウザから確認できます：`http://192.168.11.241:5000`



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。

28.6 参考リンク

- https://github.com/miguelgrinberg/flask-video-streaming
- https://github.com/rockchip-linux/rknn-toolkit2