東京都立大学2025講義資料

PyTorch環境構築

セットアップマニュアル

Windows11を中心に説明します．Macでも可
ただし，MacではNVIDIA GPUは不可です
（M1~M4 MacのMPSは利用可能です）
Intel Macは未検証です
Python3.11.9以上が動作するPCが必要です

Anacondaは使いません．pythonのvenvに移行です．
GPUを利用するにはNVIDIA系が必要です．
その際にはCUDA 11.8 + cuDNN 8.9.7_cuda11が必要です．
python 3.11.9
PyTorch 2.7.1
Jupyter Notebook

（作成）
東海国立大学機構　岐阜大学工学部電気電子・情報工学科
東海国立大学機構　岐阜大学人工知能研究推進センター
東海国立大学機構　健康医療ライフデザイン統合研究教育拠点
東海国立大学機構岐阜大学　One Medicine 創薬シーズ開発・育成研究教育拠点
原　武史＋chatGPT

2025年6月11日公開，6月18日修正，6月21日修正．

はじめに

このマニュアルは，PyTorchを利用した深層学習の環境を構築するために，GPU対応のWindowsPCを構築し，その上にPythonの機能：venvで仮想環境を作り，プログラミングの環境を構築するために作成しました．dockerか迷いましたが，MPSが使えないのでvenvにしました．軽いです！

構築は

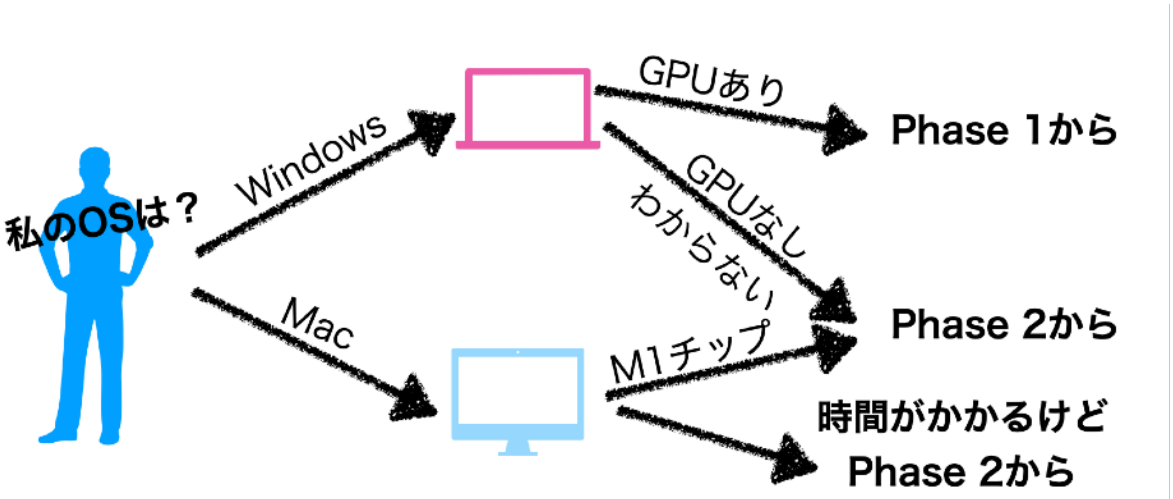
Phase 1　CUDAとcuDNNのインストール
Phase 2　Pythonのインストールと仮想環境構築
Phase 3　PyTorch環境の構築
Phase 4　基本的な動作確認

の４つからなります．

Windowsで，**対応のGPUを搭載している場合には，Phase1から設定**を行なってください．すでにCUDA，cuDNNのインストールされている場合にはバージョンの確認をお願いします．

Windowsを利用していても**GPUがない場合，もしくは，MacOSを利用の場合には，Phase 2から設定**を行なってください．MacOSから操作する場合には，いわゆるNVIDIAの「GPU」は利用できません．

M1~4チップ搭載のMacは，Phase 2から設定が可能です．十分にチェックできていませんが，M1~M4 MacのMetal Performance Shader(MPS)も使えそうです．



目　次

Phase 1

CUDAとcuDNNのインストール

GPU対応のWindowsPCを構築する

CUDA11.8, cuDNN 8.9.7_cuda11を例に説明します
お持ちのGPUの種類を調べましょう
GPUのCompute Capabilityを確認してバージョンを決めてください
（補足を参照）

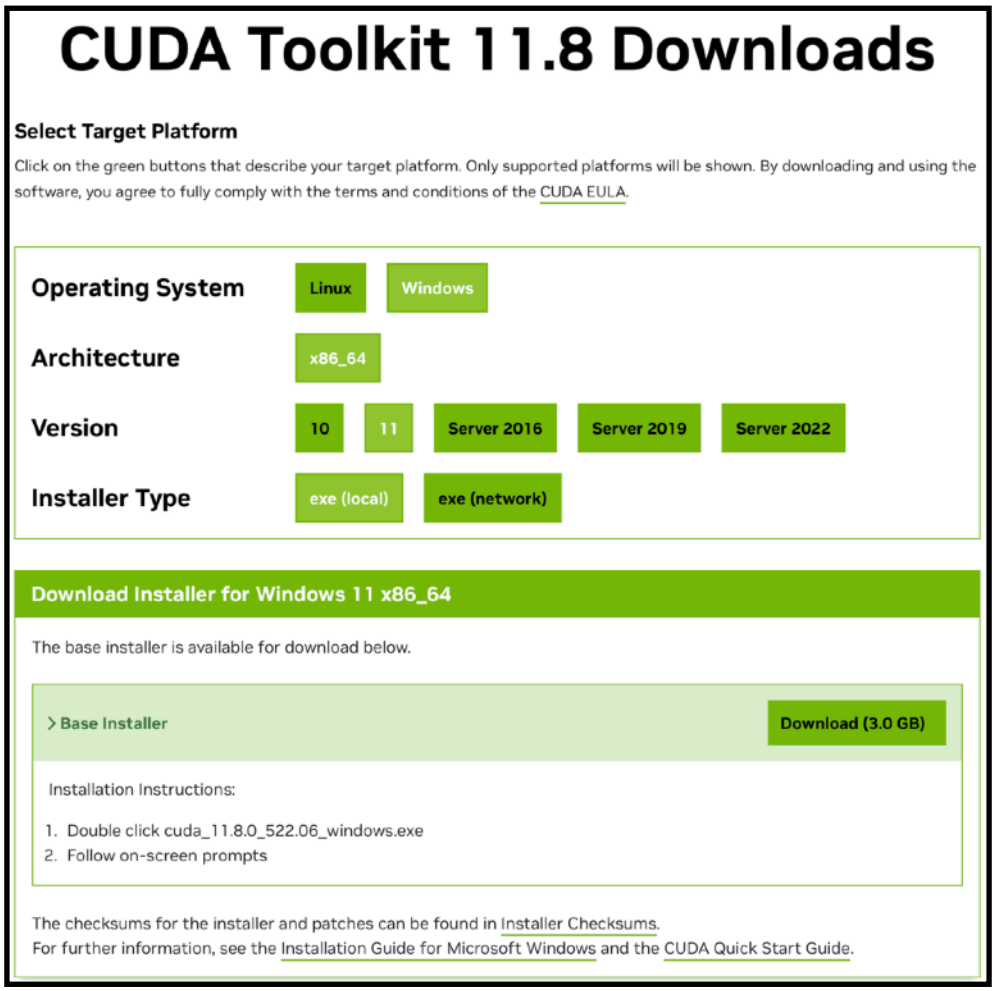
Linux/Windows対応のPyTorchの安定版（2.7.1）は
CUDA 11.8を推奨のようです
(6/1/2025現在)

GPUを利用する場合：利用するGPUの判定が必要です・NVIDIA限定です

Windows11を対象に説明します．Macは利用できません．
Compute Capability（CC）が3.5から対応です．まとめはp.35を参照．
https://developer.nvidia.com/cuda-gpus
NVIDIA RTX2080はCC=7.5です．RTX30x0はCC=8.6です．
1080はCC=6.1です．1650TiはCC=7.5です．
Win10/11用のCUDA11.8とcuDNN 8.6.0が必要です．

GPU Step1: CUDAのダウンロード

以下からダウンロードします．最新がありますが，11.8にします．
https://developer.nvidia.com/cuda-toolkit-archive
CUDA Toolkit 11.8　を探します
Windows, x86_64, 10もしくは11, localです．3.0GBあります．
ダウンロードします．

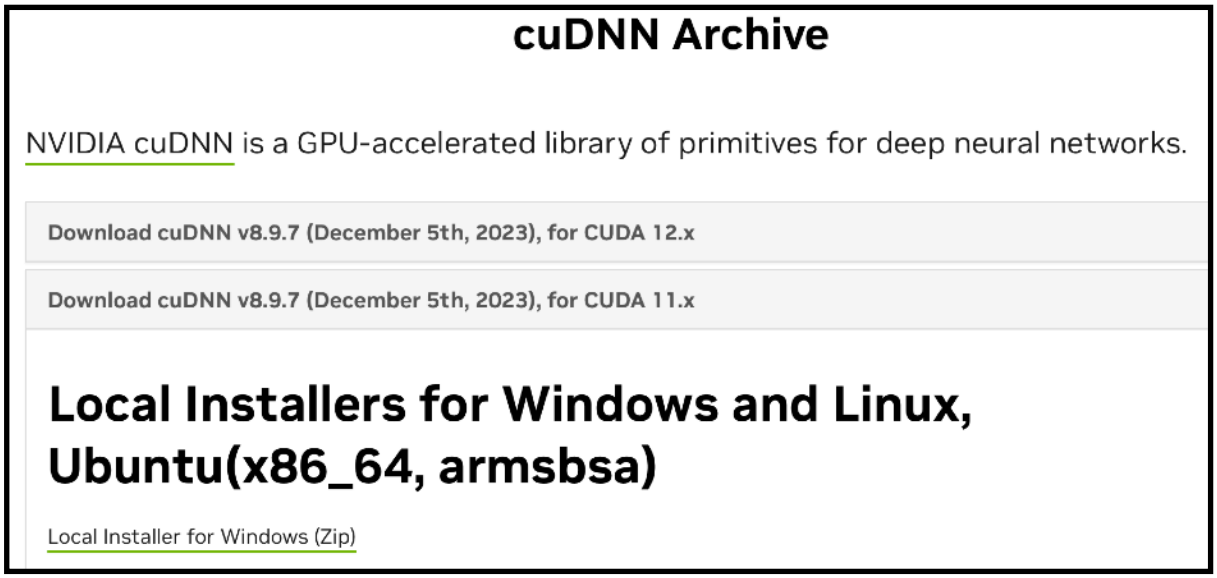


GPU Step2: CUDAのインストール

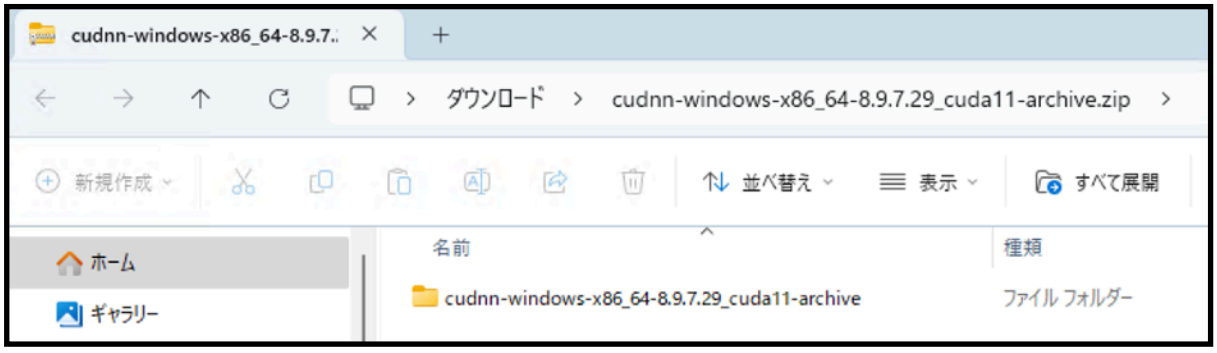
ダウンロードした実行ファイルを実行し，指示通りインストールします.
実行環境にもよりますが，ファイルが大きいためダブルクリックのあと
実行されるまでに無反応の時間が長いです．複数起動しがちです.
ご注意ください.
インストール作業自体には１５分程度必要です.

GPU Step3: cuDNNのダウンロード

以下のページからダウンロードします.
ダウンロードには，NVIDIAのアカウントが必要です.
ログイン要求などされて，認証されるとダウンロードできます.



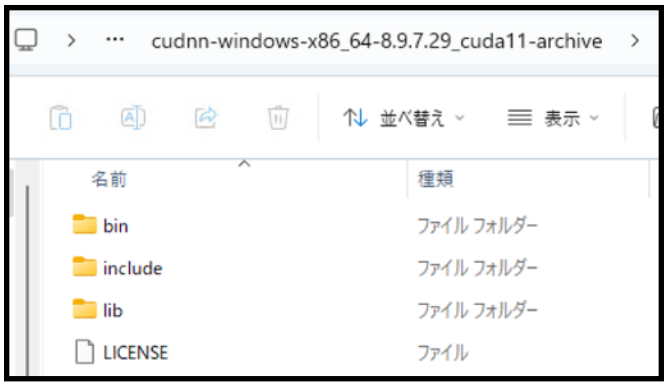
https://developer.nvidia.com/rdp/cudnn-archive
Download cuDNN v8.9.7 for CUDA 11.xを探します.
その中に「Local Installer for Windows (Zip)」があります.



ダウンロードしたら，ZIPファイルを展開します.

GPU Step4: cuDNNのインストール

展開すると以下のようなファイル構成が見えます.



「bin」「include」「lib」の３つのフォルダーが見えます.

GPU Step5: cuDNNのインストール：インストール先の確認

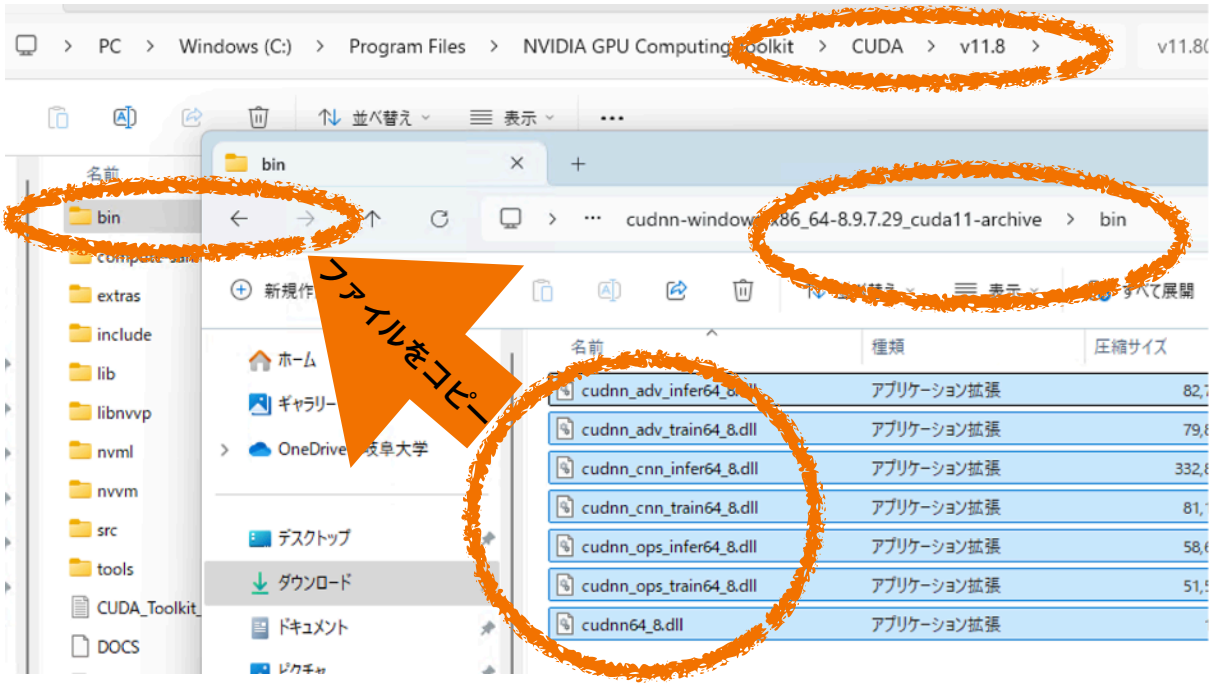
CドライブのProgram Filesの中を見ます.
その中に「NVIDIA GPU Computing Tookkit」フォルダがあります.
その中に「CUDA」フォルダがあります.
その中に「v11.8」フォルダーがあります.
これを**「インストール先フォルダー」**とします.
インストール先フォルダーの中身は，以下のようになります.

これが見えない場合にはCUDAのインストールに失敗しています.

次のステップで，先にダウンロードしたcuDNNのファイルコピーします.
コピーには管理者権限が必要になる場合があります.

GPU Step6: cuDNNのインストール：「bin」の中身をコピー

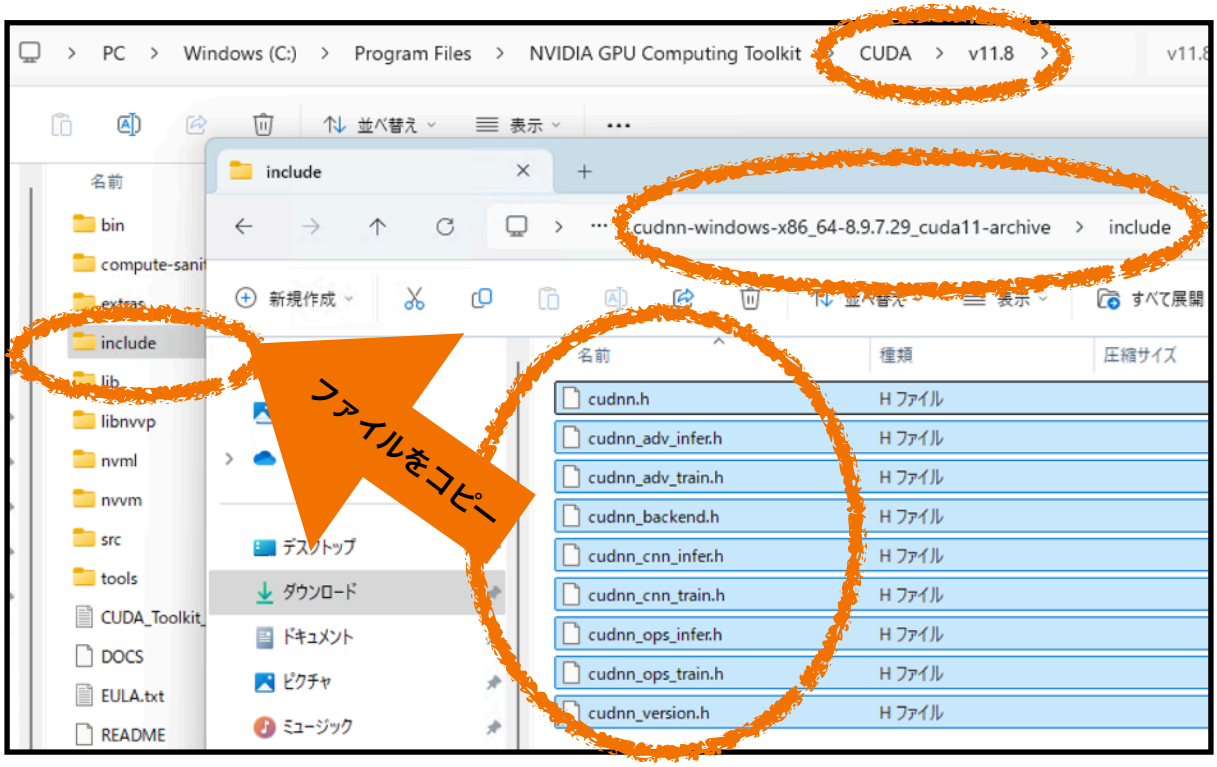
ダウンロードしたファイルの「bin」の中にある
「cudnn64_8.dll」などすべてのファイルを
インストール先フォルダーの「bin」の中にコピーします.



まだまだつづくよ～

GPU Step7: cuDNNのインストール：「include」の中身をコピー

ダウンロードしたファイルの「include」の中にある
「cudnn.h」などを
インストール先フォルダーの「include」の中にコピーします.

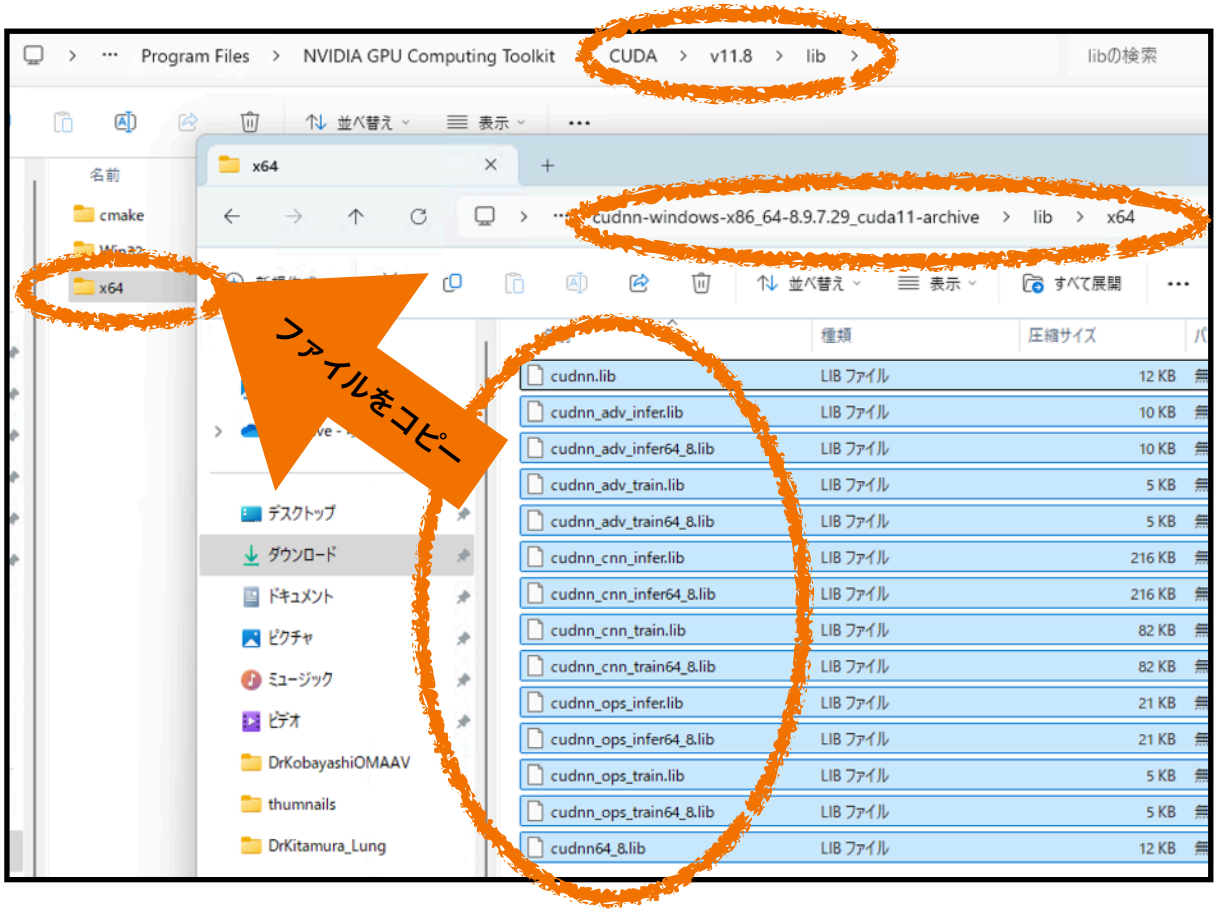


まだまだつづくよ～

GPU Step8: cuDNNのインストール：「lib」の中身をコピー

「lib」のコピーは少し異なります.

ダウンロードしたファイルの「lib」の中のファイルをコピーします
ただし，コピー先は，CUDA/v11.8/lib　の中の「x64」フォルダーです
「lib」ではなくその中の「x64」フォルダーがコピー先です
注意してコピーします.



GPU Step9: 念のため再起動します

ライブラリ系が増えたので念のため再起動します.

以上でCUDAとcuDNNのインストールは終了です.

Phase 2

Pythonのインストールと仮想環境の構築

Pythonをインストールして
仮想環境を作る

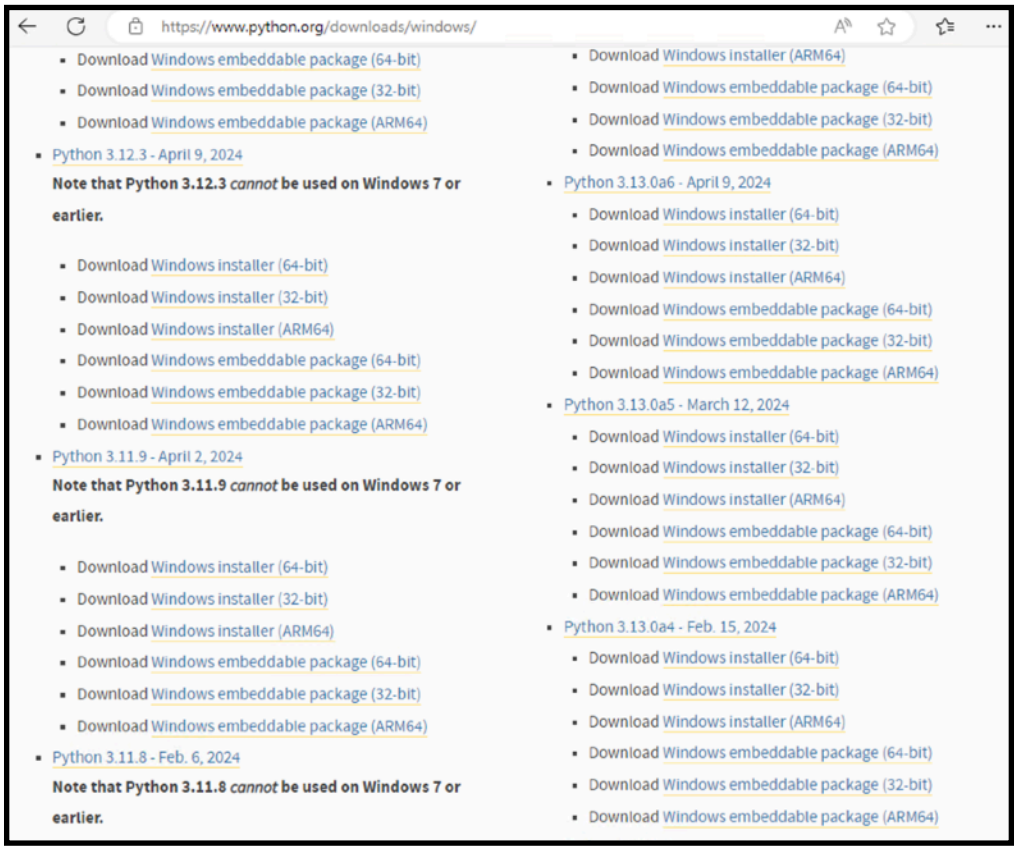
「Pythonのインストール」と
「venv」の設定の２段階です

ライブラリの都合上Python 3.11.9を利用します

PyTorch 2.3.1はPython3.8以上が必要です
少なくともPython3.8以降が動作しないPCは諦めてください

PATHの概念が重要です
カレントディレクトリを意識できてますか？

Step1：ウェブサイトからダウンロードしてインストールします.
Pythonは研究，個人，商用において無償で利用できます．以下からダウンロードしてください.
（Mac）　https://www.python.org/downloads/macos/
（Win）　https://www.python.org/downloads/windows/　から64-bitを.
非常に見にくいですが，頑張って3.11.9を探してください.



Step 2：リンクをクリックしすると，ダウンロードが始まります.
ダウンロードしたファイルはダウンロードフォルダーに保存されます.
Mac/Winいずれでもダウンロードしたファイルを展開してください.
そうすると，インストーラが見つかります.

Step 3：インストーラーを起動します.

Win　以下の画面に注意してインストールします.
Add python.exe to PATH　にチェックをします.
その後，Install Now をクリックします.

Mac　そのまま設定を変更せずインストールします.

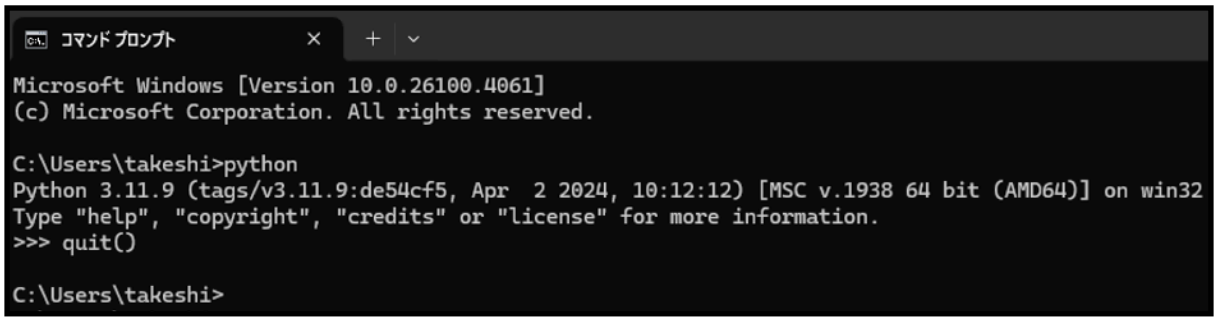
Step 4：その後，様々な画面が表示されます.
実際のインストール作業が開始されます．しばらく時間がかかります.

Step 5：インストールが終了すると最後の案内が表示され完了できます.

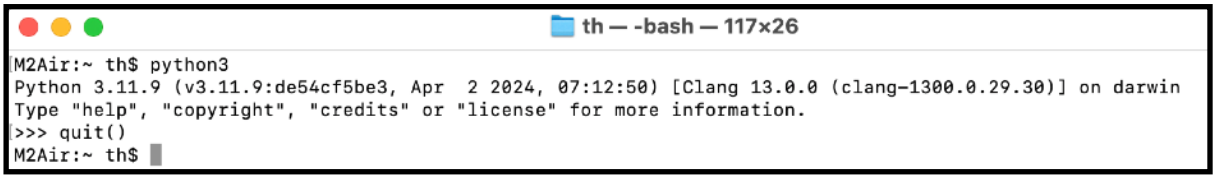
これで，第１段階「Pythonのインストールは終了」です.
次はPythonの機能を利用した「venv」のインストールです.

Step 7：動作確認
Pythonのインストールが完了しても大きく変わりません.
そのため，コマンドラインで実際に確認しましょう.

Win　コマンドライン（cmd）を開いて　python　と入力します.
　　python3　ではありません．python です.



Mac　ターミナルを開いて　python3　と入力します．以前にAnacondaを利用していた場合，プロンプトの先頭に(base)と表示される場合があります．その時は
conda deactivate
と入力してAnacondaを止めてください．そうすると(base)が消えます.



いずれもPython 3.11.9が起動していることが確認できます.
quit()で終了すると，通常のターミナルに戻ります.

Step 8：仮想環境の構築（１）原理の理解
環境名は　testenv　として構築します．この名前は自由に決めてください.
仮想環境は，その環境のファイルがすべて環境名のフォルダーの中に保存されます．したがって，そのフォルダーを消せばいつでも消去できます.

　　リアルPC　　実物が必要．起動などが時間がかかる.
　　仮想環境　　すべてソフトウェアで管理．起動が早い.

ただし，Pythonの仮想環境のみ対応なので，いわゆるVirtual環境ではないです.

Step 9：仮想環境の構築（2）環境名の設定
環境名は「testenv」とします．ターミナル／コマンドウィンドウを開きます．
それぞれ以下のコマンドを投入します．

Win　python -m venv testenv

Mac　python3 -m venv testenv

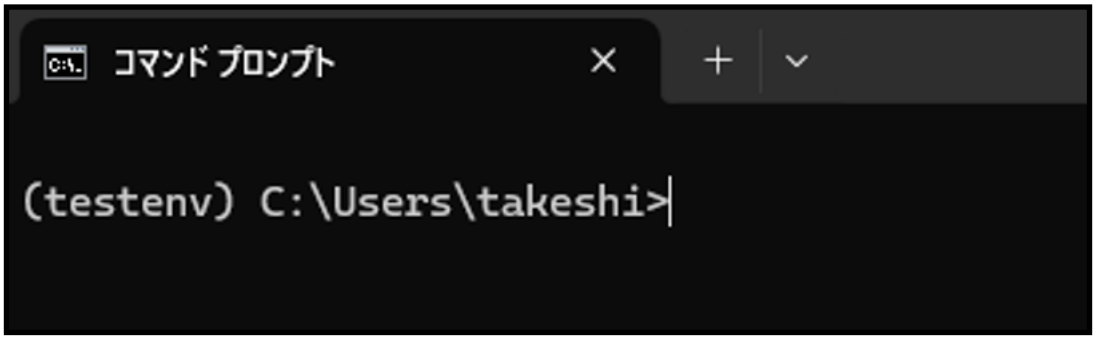
これで，pythonがもつvenvの機能を使って，環境名「testenv」ができました．
この結果，testenvという名前のパソコンが（ただしPython3.11.9にかぎる），クリーンな状態で構築できました．

Step 10：仮想環境の構築（3）仮想環境の起動
構築しただけでは起動していません．
起動にはそれぞれ以下のコマンドを投入します．

Win　testenv\Scripts\activate

Mac　source testenv/bin/activate

activateすると，ターミナルの表示が変わります．
注意してみると，先頭に環境名（testenv）が入ります．
これで仮想環境のPCが動いていることが確認できます．



なお，仮想環境を終了する場合には，
Win　testenv\Scripts\deactivate
Mac　deactivate
です．

以上でPythonのインストールと
venvによる仮想環境の構築は終了です.

Anacondaとの共存の注意

Windowsではほぼ影響はないと思います
PATHに注意が必要だと思います

Macでは，ターミナルを開いた際に

(base) MacBookM4:~ thara2$
と先頭に(base)がついていると
condaが動いていると思われます.
そのときは，そのターミナルで
conda deactivate
と投入してください.

注意（1）

ちょっと古い（2014年頃）のPC上のWindowsだと
Python 3.8以降は動かないようです.
諦めてください！

注意（2）

環境名は別の名前でもよいです.
ただし**英文半角文字**を強くお勧めします.

Phase 3

PyTorch環境の構築

構築した仮想環境に
PyTorchをインストールする

注意

GPU対応と非対応で操作が少し異なります

Step 1：ターミナルを開き，設定した仮想環境をactiveにします.
前のステップでは「testenv」としました.
ターミナルを開いていればそのままでよいです.

まずライブラリーのデータベースをアップデートします.

Win python.exe -m pip install --upgrade pip
Mac pip install --upgrade pip

なんどか繰り返して投入しましょう.

Step 2：ターミナルにコマンドを入力します：色々なライブラリインストール
GPU対応版と非対応版（CPU版），Mac版共通です.
1行ずつターミナルに入力して実行します.

```
pip3 install jupyter ipywidgets matplotlib seaborn

pip3 install pandas openpyxl imageio pillow natsort pydicom

pip3 install tqdm joblib

pip3 install imgaug albumentations timm

pip3 install opencv-python-headless

pip3 install scikit-learn scikit-image statsmodels
pip3 install cython umap-learn
pip3 install numpy==1.26.4
pip3 install ultralytics

pip3 install openai
```

それぞれ以下のライブラリに対応しています.
ブロック1：Jupyter本体と視覚化系
- jupyter: Notebookの本体
- ipywidgets: プログレスバー等をNotebookで使う場合に必要
- matplotlib, seaborn: グラフ描画

ブロック２：データ処理とI/O系
- pandas, openpyxl: Excelファイルを扱う場合に必須
- imageio, pillow: 画像の読み書き

- natsort: 自然順ソート（ファイル名などに便利）
- pydicom: 医用画像（DICOM）の読み書き

ブロック3：進行表示と並列処理

- tqdm: プログレスバー（CLIやNotebook両方で使用可）
- joblib: 並列処理（特に scikit-learn と連携）

ブロック4：画像拡張・データ拡張・学習用

- imgaug, albumentations: データ拡張ライブラリ
- timm: 画像分類用のTransformerモデル（ViTなど）

ブロック5：画像処理／OpenCV（念のため）

- OpenCV: ComputerVisionの画像処理ライブラリ一式

ブロック6：科学計算と統計系

- numpy: 基本の数値計算
- scikit-learn, scikit-image: 機械学習・画像処理
- statsmodels: 統計モデル
- cython: C拡張の高速化（scikit-learn などが使うことあり）

ブロック7：ChatGPT関連

- OpenAIのChatGPTのAPI利用などで必要.

Step 3：PyTorch関連のインストール

前のステップでターミナルを開いていればそのままでもよいです.

GPU対応版と非対応版（CPU版）でコマンドが異なります.

PyTorchは2.7.1をインストールします.

状況に合わせてどれか（1）～（3）のいずれか1行を入力してください.

Windowsの人は（1）か（2）のいずれか.

（1）はPhase1のようにGPUの設定をした人

（2）はCPU環境のみで実行する人.

Macな人は（3）

（3）はIntel系MacもしくはM1/M2 Macな人

MacOSは12.3以上であること

MacでもM1／M2（MPS利用はApple Siliconのみ）

Intel版ではCPU版として動作する可能性あり（未検証）

（1）Win+GPU版・Phase1で設定したGPU対応版・Windowsの場合：

```
pip3 install torch torchvision torchaudio --index-url https://download.pytorch.org/whl/cu118
```

（1行で入れます）

（2）Win+CPU版・Phase1でGPUの設定をしていないCPU版・Windows

```
pip3 install torch torchvision torchaudio
```

（3）Mac+MPS(Metal performance shaders対応・MacOS

```
pip3 install torch torchvision torchaudio
/Applications/Python\ 3.11/Install\ Certificates.command
```

（2行あります）

これで仮想環境の構築が完了です．次は，この環境を確認します.

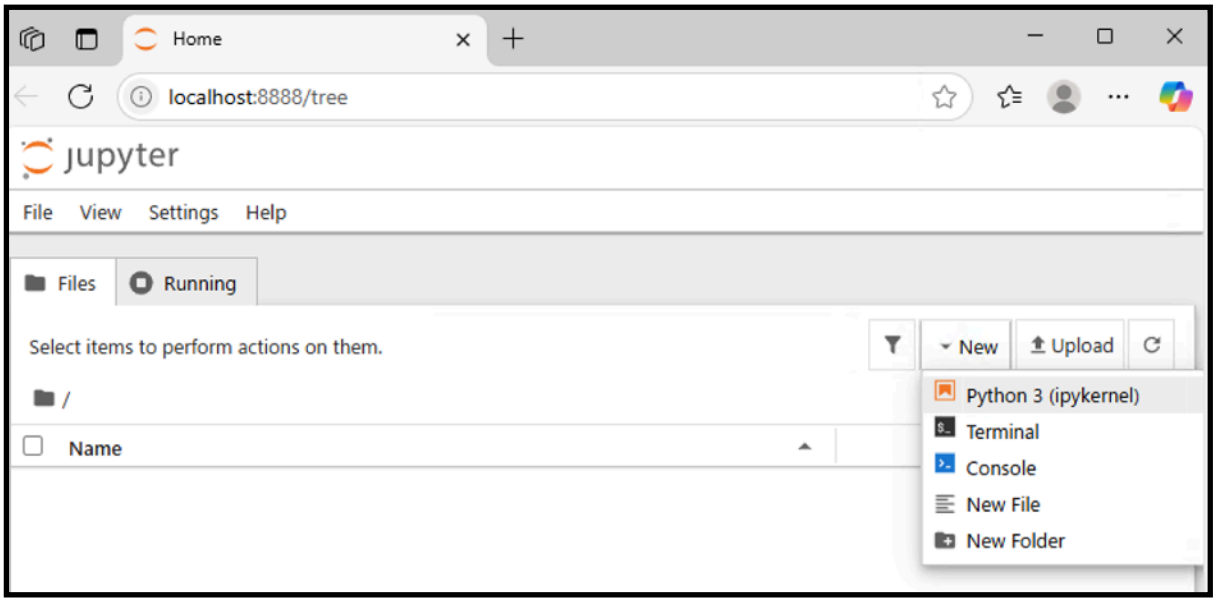
Step 5：環境の確認
インストールしたターミナルでJupyter Notebookが利用できます.
その環境で対話的に確認してみましょう．起動は，ターミナルで

Win/Macとも　　jupyter notebook

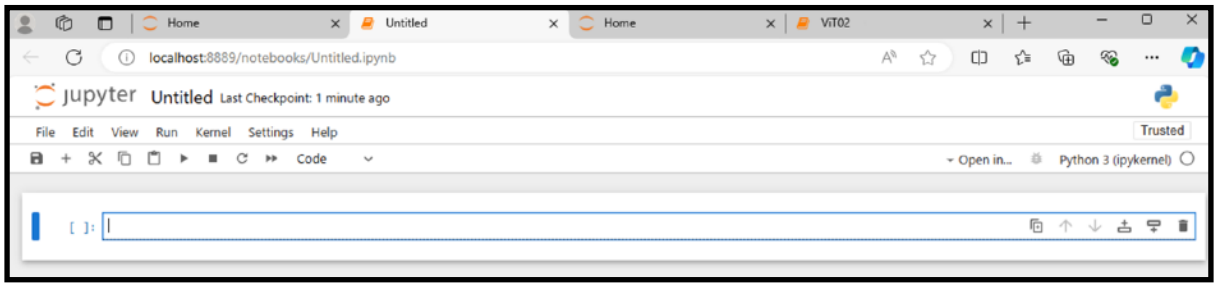
とコマンドを入力します.

Step 6：入力すると「Jupyter Notebook」がブラウザ経由で起動します．Edge（Windows10/11）やSafari（Mac）で利用できます.

Step 7：早速，入力してみましょう．右上の「New」をクリックすると，「Python 3 (ipykernel)」が表示されます．それを選択します.



Step 8：スクリプトを入力する画面が表示されます.



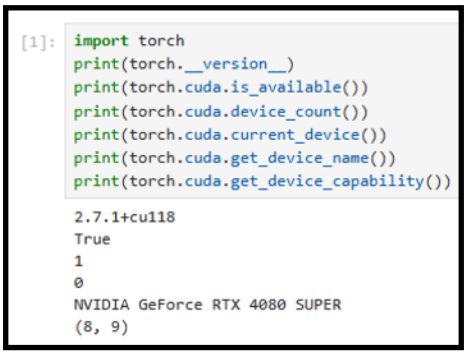
Step 9：緑の枠の中に次のスクリプトを入力してみましょう.

実行するときは，シフトキーを押しながらリターンを入力します.

Windows／GPU版の場合はバージョンが表示されます

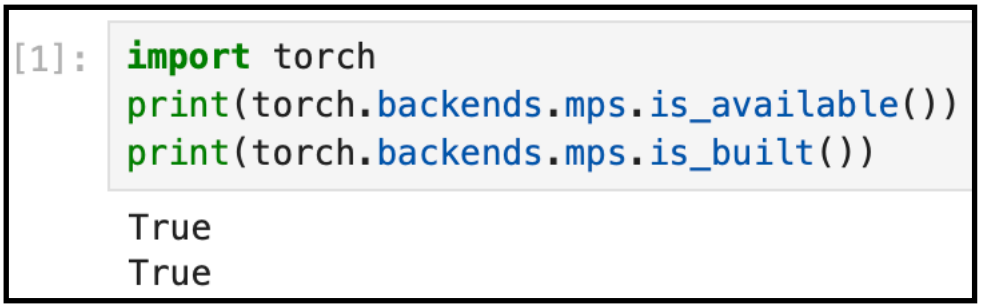
Windows CPU版／Mac版の場合は長文のエラーが出ます

```
import torch
print(torch.__version__)
print(torch.cuda.is_available())
print(torch.cuda.device_count())
print(torch.cuda.current_device())
print(torch.cuda.get_device_name())
print(torch.cuda.get_device_capability())
```



（M1~M4なMacな人の場合は以下．Trueが表示されればMPS利用可能）

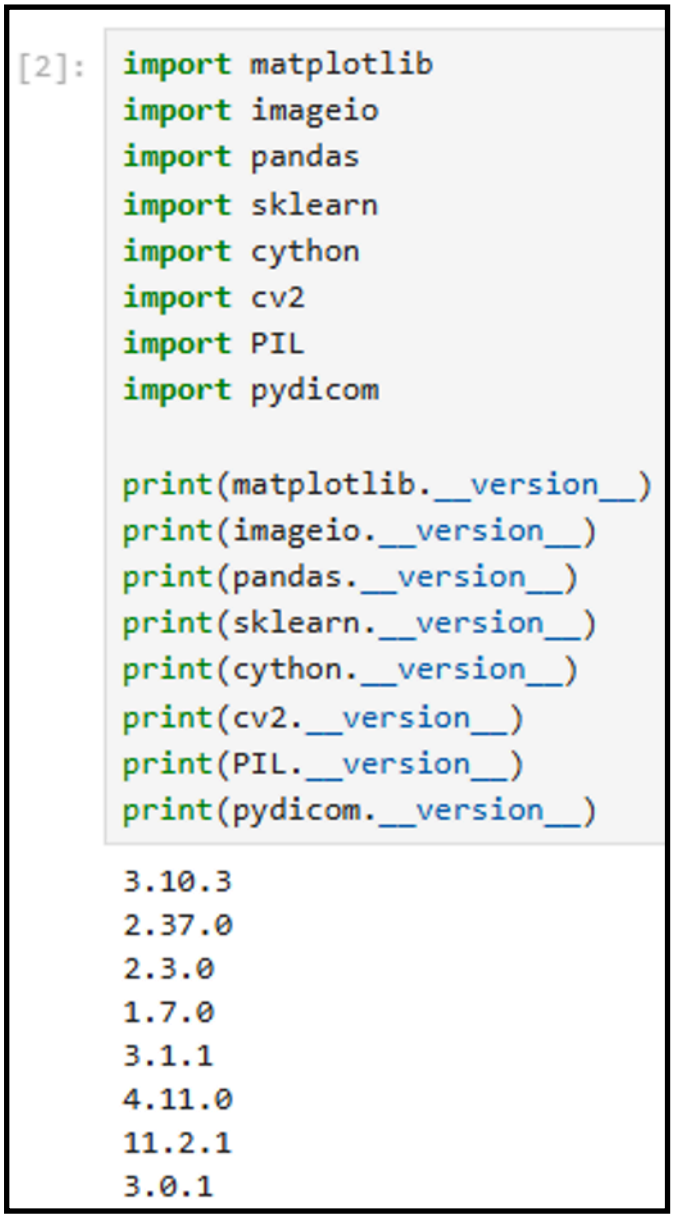
```
import torch
print(torch.backends.mps.is_available())
print(torch.backends.mps.is_built())
```



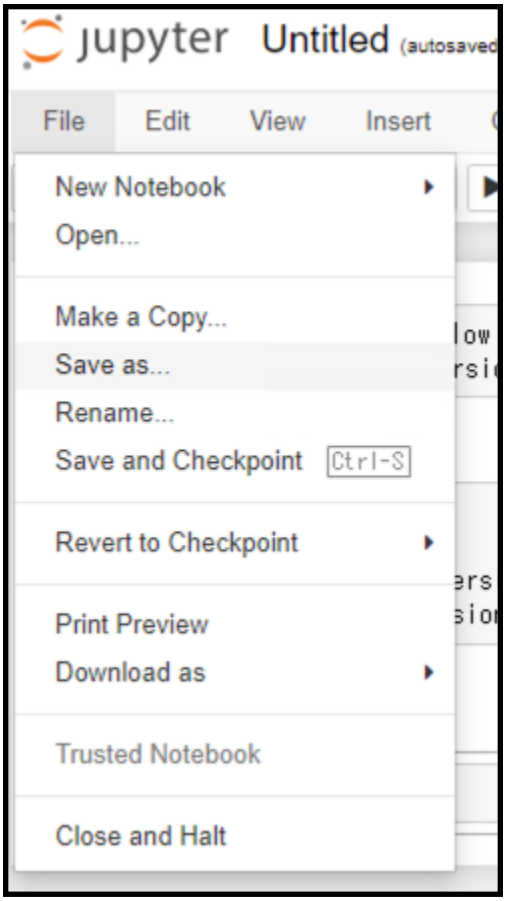
Step 10：バージョンの確認
これで，インストールされたPyTorchのバージョンが表示されればOKです．GPUの種類や，GPUを搭載していない場合には，結果が異なります．表示されなければ，環境構築に問題があります．

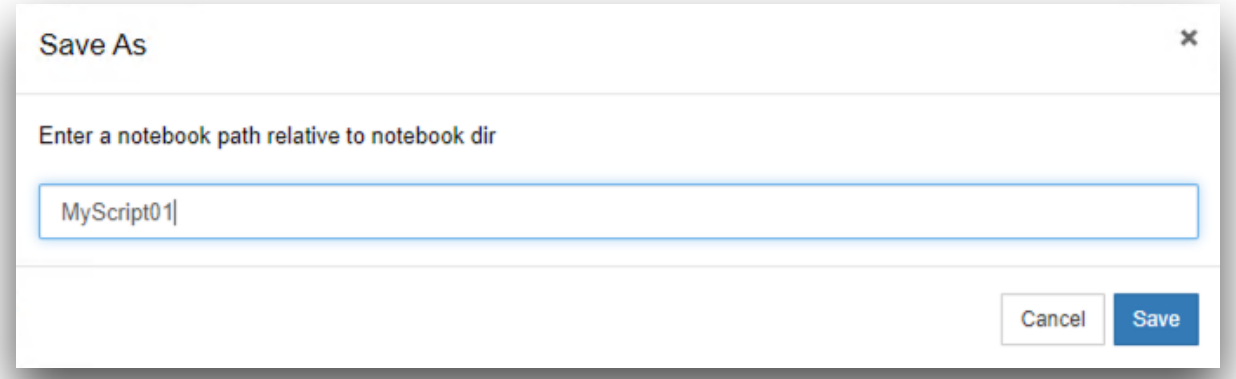
Step 11：OpenCVなどインストールしたライブラリのバージョンも確認しましょう．以下を投入．バージョンらしき数字が表示されれば動作します．
読み込みができない場合には，途中でエラーがでます．
その際には一度再起動を試みてください．
cv2のエラーがなかなかしつこいかもしれません．

```
import matplotlib
import imageio
import pandas
import sklearn
import cython
import cv2
import PIL
import pydicom

print(matplotlib.__version__)
print(imageio.__version__)
print(pandas.__version__)
print(sklearn.__version__)
print(cython.__version__)
print(cv2.__version__)
print(PIL.__version__)
print(pydicom.__version__)
```

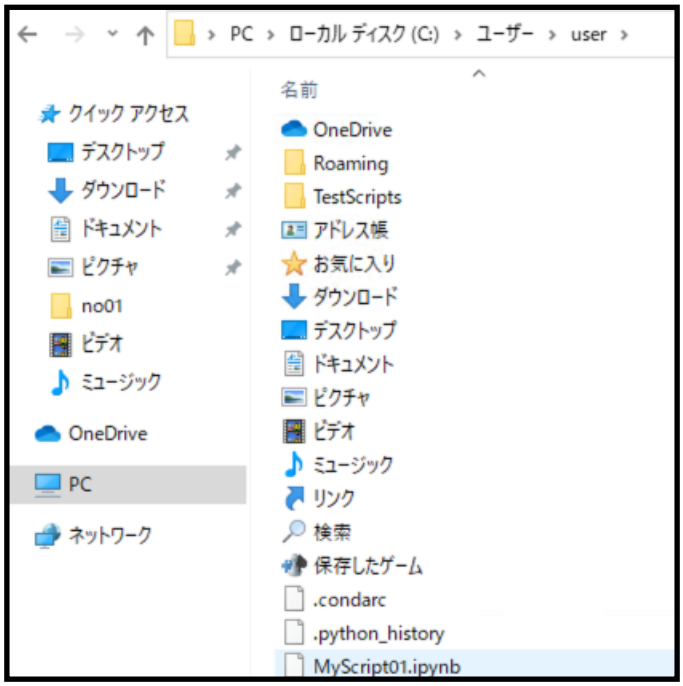


Step 12：作成したスクリプトは名前をつけて保存できます．以下の例は，「Save as... 」から「MyScript01」と名前をつけて保存しています．

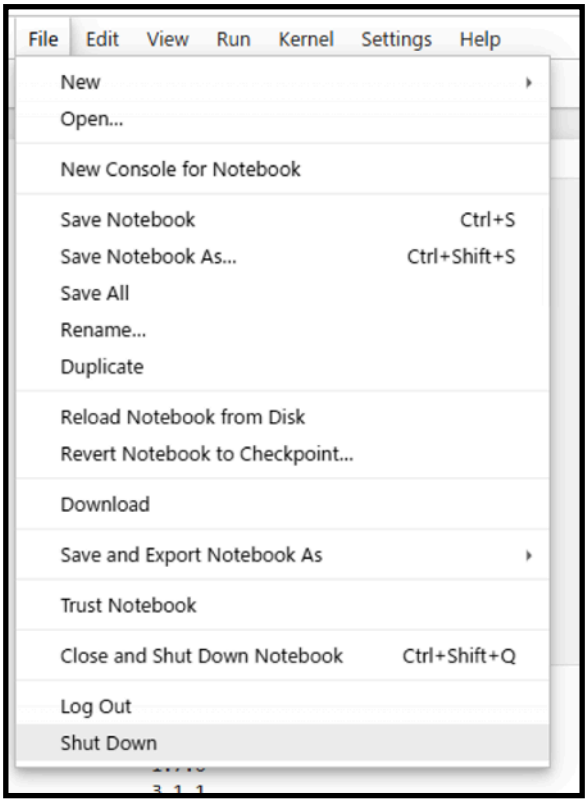




Step 13：保存先は特に指定しないと，仮想環境を構築してJupyter notebookを起動したディレクトリの直下になるようです．以下の例は，ユーザ名「user」でそのホームディレクトリの内容です．一番下に「MyScript01.ipynb」として保存されています．拡張子は「.ipynb」で，自動的に付加されます．あらかじめフォルダーを作っておいたほうが良いと思われます．



Step 14：終了するときは，Close and Haltから行います．複数のスクリプトが実行した状態のままになり，コンピュータのリソースを無駄に使う場合があるので，確実に終了します．実行中のままになっているスクリプトは「Running」と表示されます．意図していないプログラムが動かないよう，うまく管理しましょう．



Phase 4

基本的な動作確認

基本的な動作確認をする少し練習です
GPUが望ましいですがCPUでも可

以下のファイルをダウンロードしてください.
200MBくらいあります.

http://www.fjt.info.gifu-u.ac.jp/dls2025/setup.zip

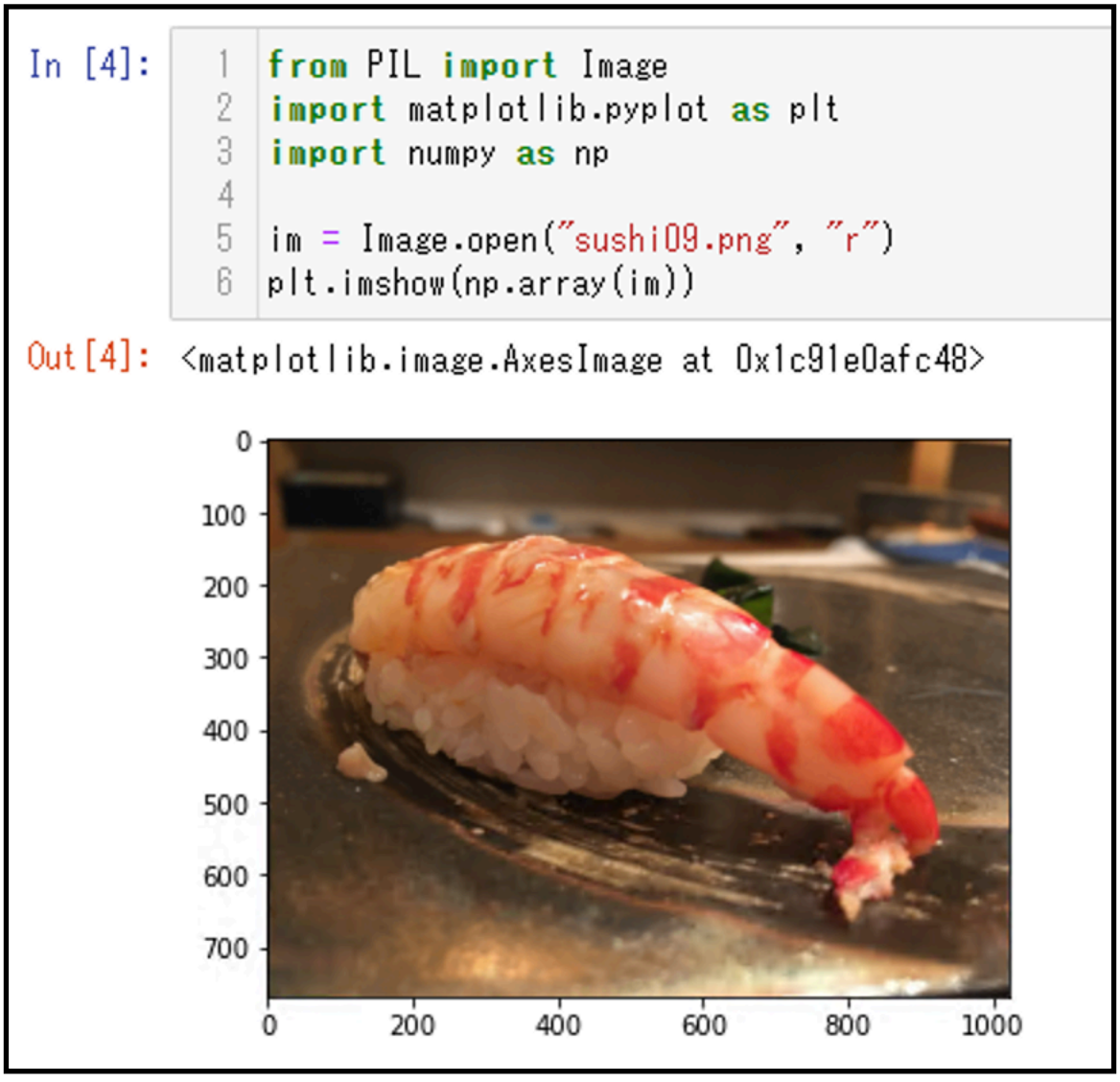
展開するとsetupフォルダーの中に
check01, check02, check03, check04, Emotion
の５つのフォルダーがあります.

Step 1：練習1（check01）

保存済みのスクリプトを開く場合には，Jupyter Notebookで「Open」からファイルを指定します．もしくは，起動した画面からそのフォルダー／スクリプトにたどり着きます．**ダウンロードしたファイルをダブルクリックしても動作しません．**

動作確認用のスクリプトのZIPファイルをウェブページからダウンロードしてください．ZIPファイルを解凍して，作業用フォルダーに置きましょう． 3つのフォルダーがあります．check01の中にある「CheckScript1.ipynb」ファイルを開くと，スクリプトが表示されます．シフトクリックで動作してください．スクリプトの前半はインストールしたライブラリのバージョンが表示されます．**importに失敗していないかエラーをよく確認しましょう．**

このサンプルが最後まで実行できれば，最後には美味しそうな「エビの握り」が表示されます．ネタに困るとお寿司です．

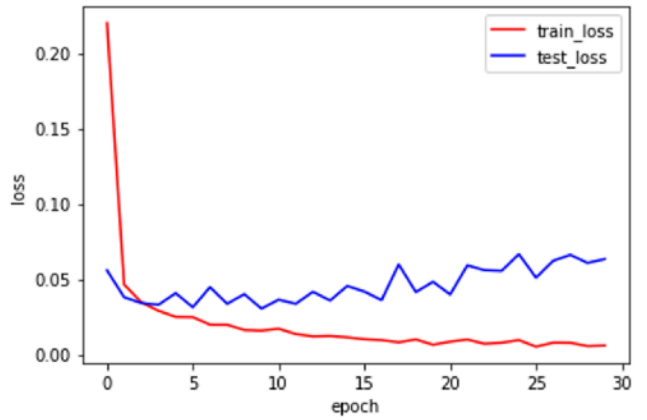


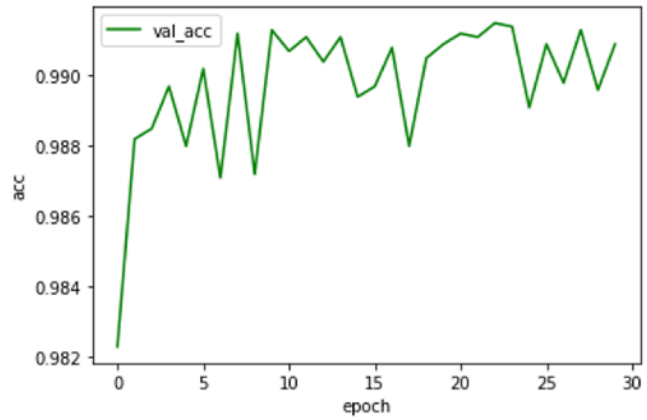
Step 2：練習２（check02）

動作確認用のスクリプトの「CheckScript2.ipynb」ファイルを開くと，スクリプトが表示されます．シフトクリックで動作してください．
手書き文字認識：MNISTをCNNで解釈します．最後には学習曲線と正解率（ほぼ１００%）が表示されます．

```
'''結果の表示'''
plt.plot(range(EPOCHS), loss_list, 'r-', label='train_loss')
plt.plot(range(EPOCHS), val_loss_list, 'b-', label='test_loss')
plt.legend()
plt.xlabel('epoch')
plt.ylabel('loss')

plt.figure()

plt.plot(range(EPOCHS), val_acc_list, 'g-', label='val_acc')
plt.legend()
plt.xlabel('epoch')
plt.ylabel('acc')

print('正解率：',val_acc_list[-1]*100, '%')
```

```
正解率： 99.09 %
```





Step 3：練習３（check03）

動作確認用のスクリプトの「CheckScript3.ipynb」ファイルを開くと，スクリプトが表示されます．シフトクリックで動作してください．ファイルを開くと，スクリプトが表示されます．シフトクリックで動作してください．練習１で使った画像を，事前学習済みの「ResNet101」に入力して，1000分類から該当したカテゴリを出力します．最後には「king_crab」の画像が「121番　king_crab」と近からず遠からずの分類結果が得られます．

```
[16]: print(out.shape)

      torch.Size([1, 1000])

[17]: with open("./imagenet_classes.txt") as f:
          labels = [line.strip() for line in f.readlines()]

[18]: labels

[18]: ['0, tench',
       '1, goldfish',
       '2, great_white_shark',
       '3, tiger_shark',
       '4, hammerhead',
       '5, electric_ray',
       '6, stingray',
       '7, cock',
       '8, hen',
       '9, ostrich',
       '10, brambling',
       '11, goldfinch',
       '12, house_finch',
       '13, junco',
       '14, indigo_bunting',
       '15, robin',
       '16, bulbul',
       '17, jay'

[19]: _, index = torch.max(out, 1)

[20]: labels[index[0]]

[20]: '121, king_crab'
```

Step 4：練習4（check04）

Jupyter　notebookは対話的で親しみやすいですが，繰り返しの実行には不向きです．その場合には，pythonのファイルを直接実行する方法があります．
練習4は，コマンドラインだけで行います．

コマンドラインで，check04ディレクトリに移動してください．
そこで以下のコマンドを投入してください．

Win　python library_check.py

Mac　python3 library_check.py

```
(testenv) C:\Users\takeshi\dls\setup\check04>python library_check.py
Python version
  python         : Python 3.11.9
  python3        : Python

pip version
  pip            : pip 25.1.1 from C:\Users\takeshi\testenv\Lib\site-packages\pip (python 3.11)

  pip3           : pip 25.1.1 from C:\Users\takeshi\testenv\Lib\site-packages\pip (python 3.11)


ステップ1：Jupyter本体と視覚化系
  notebook       : 7.4.3
  ipywidgets     : 8.1.7
  matplotlib     : 3.10.3
  seaborn        : 0.13.2

ステップ2：データ処理とI/O系
  pandas         : 2.3.0
  openpyxl       : 3.1.5
  imageio        : 2.37.0
  pillow         : 11.2.1
  natsort        : 8.4.0
  pydicom        : 3.0.1
```

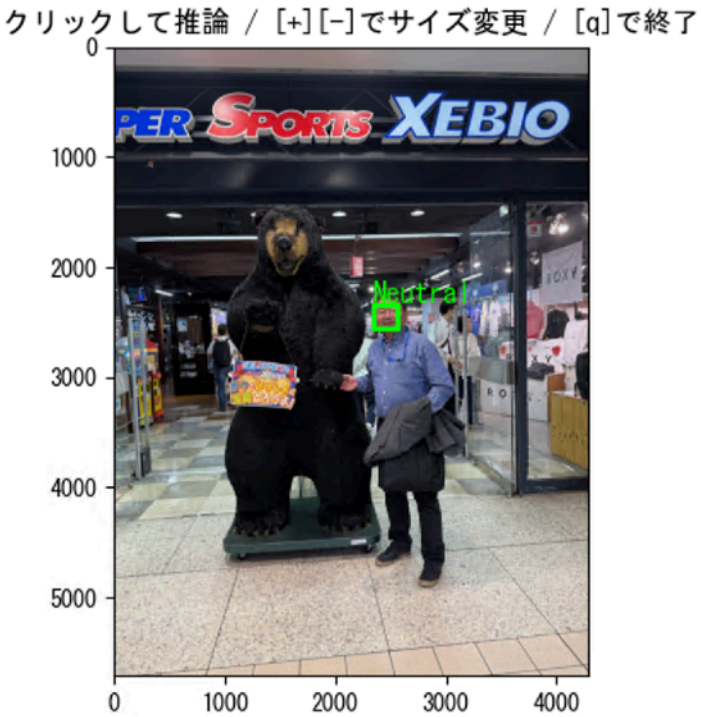
pythonのプログラムが実行されて，ライブラリーのバージョンが表示されるはずです．このように，コマンドラインから直接プログラムを実行できます．

Step 5：練習5（Emotion）

コマンドラインでのプログラムは，並列処理やインタラクティブなプログラムの実行には必須です．ここでは，画像を表示して，顔部分をクリックするとその人の表情から感情（Emotion）を分類するプログラムを動かしてみましょう．
プログラムはEmotionディレクトリにあるEmotion.pyです．
これは，https://github.com/Open-Debin/Emotion-FAN（Frame attention networks for facial expression recognition in videos, Debin Meng, Xiaojiang Peng, Kai Wang, Yu Qiao, https://arxiv.org/abs/1907.00193）で公開されいるモデルを利用しています．

Win　python Emotion.py TakeshiHara/1.jpeg
Mac　python3 Emotion.py TakeshiHara/1.jpeg

こうなると，このプログラム（Emotion.py）の中身が見たくなることでしょう．プログラムの修正は簡単です．さくらエディタなどの無料のテキストエディタで修正すればよいのです．**ただし，UTF-8の文字コード，CR＋LFの改行コードに対応している必要があります．**VS Codeなどの開発統合環境を使いたい人はそれもよしです．（統合環境の使い方は，余分な話になるのでここでは扱いません）



以上でセットアップは終了です

どのGPUはどのバージョンのCUDAが必要か？

導入するGPUのHardware Generationを調べそのCompute Capabilityを調べます．それに応じたCUDAのバージョンを決定します．そして，そのCUDAのバージョンに対応したcuDNNをインストールします．

NVIDIAのアーキテクチャとCompute CapabilityおよびCUDAバージョン一覧 (6/4/2025版)

Hardware Generation	Compute Capability	CUDA 10.1	CUDA 11.0-3	CUDA 11.4	CUDA 11.5	CUDA 11.6	CUDA 11.7	CUDA 11.8	CUDA 12.x
Hopper	9.0	**NO**	**NO**	**NO**	**NO**	**NO**	**NO**	Yes	Yes
Ada Lovelace	8.9	**NO**	**NO**	**NO**	**NO**	**NO**	**NO**	Yes	Yes
Ampere	8.0-8.7	**NO**	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Turing	7.5	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Volta	7.x	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Pascal	6.x	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Maxwell	5.x	Yes	Yes	Yes	Yes	Yes	Yes	Yes	Yes
Kepler	3.5	Yes	Yes	Yes	Yes	Yes	Yes	Yes	**NO**
Kepler	3.0	Yes	Yes	Yes	**NO**	**NO**	**NO**	**NO**	**NO**

代表的なGPUの商品名

Generation	CC	代表的な機種名 (GF: GeForce)
Blackwell	12	RTX PRO 6k/5k/4.5k/4k Blackwell W/Q, RTX5090/80/70/60
Blackwell?	10	GB200, B200
Hopper	9.0	GH200, H100/200/800
Ada Lovelace	8.9	GF 4090-4060, RTX 6000Ada, RTX4000SFF, L4/40
Ampere	8.0-8.7	GF 3090-3050, RTX A6000-A2000, A100/40/30/16/10/2
Turing	7.5	Titan RTX, GF2080-2060, GTX1660, RTX8000, Tesla T4
Volta	7.x	Titan V, Quadro GV100, Tesla V100/V100S
Pascal	6.x	Titan X, GF 1080-1010, Quadro P6000-1000, Tesla P100/40
Maxwell	5.x	GF GTX Titan X/980, Quadro M6000, Tesla M60/40/10/6/4
Kepler	3.5	GF GTX Titan Z/780-710, Quadro K6000, Tesla K40/20x/20

（参考）

https://developer.nvidia.com/cuda-gpus

https://docs.nvidia.com/cuda/cuda-toolkit-release-notes/index.html

Apple Metal Performance Shaders (MPS）の利用

MPSはdockerでは動かない！venvじゃないとダメです.

NVIDIAのGPUは，CUDAとcuDNNを用いて深層学習の計算を高速化しました．Appleが提供するGPU環境であるMetal Performance Shaders (MPS）は，Apple Silicon（いわゆるM1やM2）に搭載されたGPU環境であり，PyTorchを利用した深層学習においても活用できます.

Apple Siliconでは，メインメモリとGPUメモリが統合されるため，NVIDIAのGPUのようにGPU専用の物理メモリの概念がありません．CPUとGPUはメインメモリを共有するため，メインメモリから自動的にGPU用のメモリを割り当てて実行されます．その結果，メインメモリが十分に搭載されていれば，NVIDIAのGPUメモリでは動作しなかったバッチサイズやモデルをMPSでは実行できる可能性があります．PyTorchにおけるMPSの利用方法は比較的簡単で,

・Apple Siliconを搭載のMacであること
・macOS12.3以上であること
・MPS対応のPyTorchをインストールすること

で環境構築ができます．実行時には，CUDAでは，以下のようにCUDAかCPUの判定を行いデバイスを指定していましたが,

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
```

MPSの場合は，以下のようにMPSの利用判定を行います.

```
device = torch.device('mps' if torch.backends.mps.is_available() else 'cpu')
```

deviceにmpsが指定されていれば，M1/M2のMPSがCUDAのように利用できるはずです．deviceの指定の状況は,

```
print(device)
```

で確認できます．CUDA, MPS, CPUの判定を自動で行いたい場合には,

```
device = 'cuda' if torch.cuda.is_available() else 'mps' if torch.backends.mps.is_available() else 'cpu'
```

で指定できます．具体的なスクリプトは，CheckScript1を参考にしてください.

　このようにデバイスにMPSを指定してCheckScript2のモデルの学習（30 epoch）で評価すると次のような結果になりました．ただし，macOS14.5を除き，PyTorch1.13系です（2023年実施の結果．更新なし）．

MPSを利用した場合の計算速度の比較

OS version	種類	M1/M2	メインメモリ	計算時（秒）	Batch size
macOS 14.5	MB Air	M2, 10GPU	24GB	106	10000
macOS 13.1	MB Air	M2, 10GPU	24GB	203	変更なし
macOS 13.1	MB Air	M2, 10GPU	24GB	84	10000
macOS 13.0	Mac Studio	M1, 48GPU	128GB	186	変更なし
macOS 13.0	Mac Studio	M1, 48GPU	128GB	61	10000
macOS 13.0	MB Pro	M1, 32GPU	64GB	227	変更なし
macOS 12.5	MB Air	M1, 8GPU	16GB	394	変更なし
macOS 12.4	MB Pro	M1, 32GPU	64GB	366	変更なし
macOS 12.5	MB Air	M1, 8GPU	16GB	**CPUのみ736**	変更なし
macOS 12.4	MB pro	M1, 32GPU	64GB	**CPUのみ737**	変更なし

CUDAを利用した場合の計算速度の比較

OS version	種類	CPU	GPU	計算時（秒）	Batch size
Windows11	Desktop PC	Core i7	RTX 3060	110	変更なし
Windows11	Desktop PC	Core i7	RTX 3060	72	10000
Windows10	Laptop PC	Core i7	GTX 2070	266	変更なし
Windows10	Laptop PC	Core i7	GTX 1070	287	変更なし
Windows10	Laptop PC	Core i7	−	**CPUのみ912**	変更なし

Windows+CUDA+GeForceの組み合わせは，変更なしのバッチサイズでもかなり早いことがわかります．以上のことから，
・M1/M2のGPUの数が少なくても高速化に効果はあり
・M1/M2のMPSの利用はGTX 2070のGPUと同程度の高速化が期待できる
・バッチサイズの大きさは計算時間に影響がある（新しい知識ではない）

とわかりました．

不明なトラブルメモ

OpenCV系が不安定：
　　インストール順に依存しそう
　　torchvisionとの関連がありそう

TorchvisionがCPU版になってしまう
　　原因不明．再インストール．

Python + venv　よく使うまとめ

Windows系

1.仮想環境「testenv」を作る：
python -m venv testenv

2.仮想環境を有効化する：
testenv\Scripts\activate

3.仮想環境を終了するとき：
testenv\Scripts\deactivate

Mac系

注意：ターミナルで
(base) MacBookM4:~ thara2$
ならばcondaが動いている.

conda deactivate

でanacondaと無関係のターミナルになる.

1.仮想環境「testenv」を作る：
$ python3 -m venv testenv

2.仮想環境を有効化する：
$ source testenv/bin/activate

3.仮想環境を終了するとき：
$ deactivate