令和４年度卒業研究報告書

GPUを用いたDeep learningの性能比較，検証

情報技術科　石川　巧弥

指導教員 　菅野　研一

目次

[第 1 章 はじめに 3](#_Toc129006453)

[第 2 章 使用技術 4](#_Toc129006454)

[2.1 コンテナ 5](#_Toc129006456)

[2.1.1 メリット 7](#_Toc129006457)

[2.1.2 デメリット 7](#_Toc129006458)

[2.2 Docker 8](#_Toc129006459)

[2.2.1 Dockerイメージ 8](#_Toc129006460)

[2.2.2 nvidia-docker 8](#_Toc129006461)

[2.3 GPU 9](#_Toc129006462)

[2.4 ライブラリ 10](#_Toc129006463)

[2.4.1 TensorFlow 10](#_Toc129006464)

[2.4.2 Time 10](#_Toc129006465)

[2.4.3 Torch 10](#_Toc129006466)

[2.5 ハードウェア 11](#_Toc129006467)

[第 3 章 環境構築 12](#_Toc129006468)

[3.1 初期設定 12](#_Toc129006469)

[3.1.1 NGCサイトにサインインする 12](#_Toc129006470)

[3.1.2 API　KEYを生成する 12](#_Toc129006471)

[3.1.3 NGCのDockerへログインする 13](#_Toc129006472)

[3.1.4 利用したいDockerイメージを取得する 14](#_Toc129006473)

[3.2 コンテナ起動（TensorFlow） 15](#_Toc129006474)

[3.3 コンテナにローカルのボリュームをマウントする 16](#_Toc129006475)

[3.4 X Window Systemの導入 17](#_Toc129006476)

[3.5 カメラ接続 19](#_Toc129006477)

[3.6 共有メモリ量設定 20](#_Toc129006478)

[3.7 コンテナコミット方法 21](#_Toc129006479)

[第 4 章 処理時間計測 22](#_Toc129006480)

[4.1 計測用サンプルデータ 22](#_Toc129006481)

[4.1.1 手書き数字の画像認識 22](#_Toc129006482)

[4.1.2 物体検知用キャラクター学習 23](#_Toc129006483)

[4.2 計測方法・計測場所 24](#_Toc129006484)

[4.2.1 手書き数字の画像分類 25](#_Toc129006485)

[4.2.2 物体検知用キャラクター学習 26](#_Toc129006486)

[4.3 計測実行 28](#_Toc129006487)

[4.3.1 手書き数字の画像分類 29](#_Toc129006488)

[4.3.2 物体検知用キャラクター学習 31](#_Toc129006489)

[第 5 章 結果・考察 34](#_Toc129006490)

[5.1 計測結果 34](#_Toc129006491)

[5.1.1 手書き数字の画像分類 34](#_Toc129006492)

[5.1.2 物体検知用キャラクター学習 36](#_Toc129006493)

[5.2 考察 37](#_Toc129006494)

[第 6 章 おわりに 38](#_Toc129006495)

[参考文献 39](#_Toc129006496)

# はじめに

今年度，学校に高性能なGPUを搭載している，AI専用PCが導入された．導入されたばかりのため，このPCの利用例はまだ存在しない．そこで，このPCの環境を整え，性能を検証することが，今後の産技短での専用PCの利用やAI開発の足掛かりになると考え，この研究に取り組むことにした．

研究の目的は，専用PCに搭載されているGPUの処理速度を検証し，CPUや他のGPUと比較する．処理させるサンプルデータはDeep Learningを用いたプログラムの学習部分を利用する．サンプルデータは複数用意し，使用するGPUもGoogle Colaboratoryを加えて処理速度を検証していく．

# 使用技術

## 開発環境とDeep learning専用PCの情報を表2-1と表2-2に示す．

|  |  |
| --- | --- |
| OS | Ubuntu 18.04.5 LTS |
| 使用言語 | Python 3.8 |
| 使用ライブラリ | TensorFlow |
| コンテナエンジン | Docker |

表 2-1　開発環境

|  |  |
| --- | --- |
| GPU | GeForce RTX3090 |
| CPU | Intel Xeon W-2223@3.6Ghz |
| RAM | 32GB |
| SSD | 1×2TB |

表 1-2　Deep learning専用PCの情報

## コンテナ

コンテナとはアプリケーションと実行環境を１つにまとめ上げる技術である．アプリケーションの動作に必要なホストOSの基本環境（カーネル）をDockerのようなコンテナエンジンを通して，コンテナ同士が共有できるようにすることで，CPUやメモリなどのハードウェアのリソースと切り離し仮想的な環境を作り出す．[1]

仮想マシンとコンテナの構成の違いとしては，環境内で使用しているOSにある．仮想マシンはハイパーバイザというソフトウェアで立ち上げられるハードウェアの環境にゲストOSやアプリケーションのインストールを行い，ホストOSとは別の実行環境を構築している．(図2-1)

一方，コンテナはコンテナエンジンを通して共有されたホストOSのリソースに，ゲストOS不要でアプリケーションを追加することができる．つまり，ホストOSで実行環境を構築している．（図2-2）

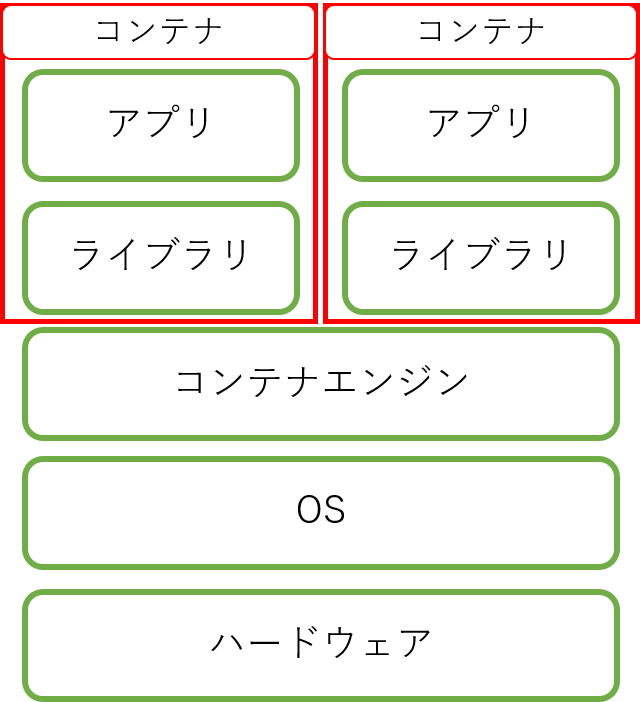


図2-2　コンテナ環境

図 2-1　仮想マシン環境

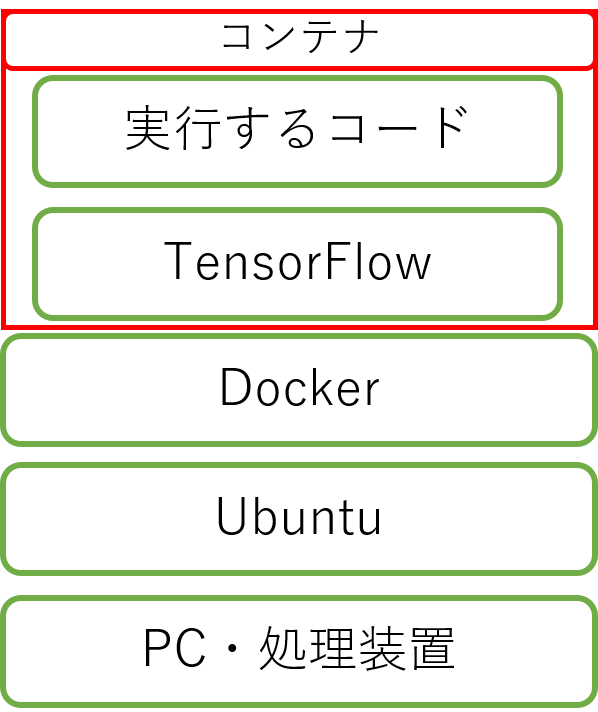
本研究環境を図2-2に当てはめると以下のようなイメージになる．（図2-3）

図 2-3　本研究のコンテナ環境

### メリット

#### コンテナ化されたアプリケーションはOSとの依存関係がないため，移行性が高く，あらゆるプラットフォームやクラウドで一貫性のある均一な動作が可能になる．

#### コンテナ内にOSが入っていないため，機能が軽量となり，起動時間が短縮される．

#### コンテナは独立しているため，アプリケーションをコンテナ化すると，アプリケーションが分離される．そのため，一方に障害が発生しても，他のコンテナには影響しない．[2]

### デメリット

#### コンテナを管理，運用するのに複雑化しやすい．

#### コンテナ自体はホストOSに依存するため，同一環境上で異なるOSを動かすことができない．

#### （本研究環境ではホストOSはUbuntuなので，それ以外のOSは動かせない．）

#### コンテナ化はセキュリティ面でリスクが高くなる．ホストOSに障害が起きた際には，生成したコンテナすべてに影響が出るので，セキュリティレベルを高くする必要がある．[2]

## Docker

DockerとはDocker社が開発している，コンテナ型の仮想環境を作成，配布，実行するためのプラットフォームのことである．環境やアプリケーションをDockerイメージとして保存し，DockerエンジンによりDockerコンテナとして配備，実行できる．[3]

### Dockerイメージ

Dockerイメージとは，Dockerコンテナの動作環境となるテンプレートファイルである．Dockerコンテナを実行するためには，Dockerイメージが必要となる．今回の環境ではNVIDIAのNGCサイトからDockerイメージを取得している．

### nvidia-docker

今回の環境ではnvidia-dockerを利用している．nvidia-dockerとは，NVIDIAのGPUとドライバがインストールされたマシン上で稼働するように開発された，Dockerコンテナプラグインである．コンテナ内にCUDAやcuDNNなどライブラリがインストールされており，機能拡張でコンテナ内からGPUを使用することが可能になる．[4]

## GPU

GPUとは「Graphics Processing Unit」の略でリアルタイム画像処理に特化した演算装置あるいはプロセッサのことである．並列処理能力に優れており，複数のタスクを一度にこなすことができる．CPUとの違いとしては処理をする対象と，スピードに違いがある．CPUは汎用的な処理を行うことが目的であるが，GPUは高速な画像処理を行うことが目的である．また，GPUは単純な処理をするのが得意であるため，機械学習のトレーニング期間の短縮が可能である．[5]

今回使用したGPUはNVIDIAのGeForce RTX3090である．CUDAコア数は10752，メモリサイズは24GBある．また，AI処理に特化したTensorコアと，よりリアルなグラフィックスを表示するレイトレーシングコアを搭載している．(図2-４)[ 6]

今回の研究ではGoogle ColaboratoryのGPUも用いている．Google Colaboratoryは無料版を利用しており，GPUはNVIDIA Tesla T4を用いている．



図 2-4　GeForce RTX3090の画像

## ライブラリ

### TensorFlow

TensorFlowとは，Googleが開発しオープンソースで公開している，機械学習に用いるためのソフトウェアライブラリである．機械学習や数値解析だけでなく，Deep Learningにも対応している．本研究ではDeep Learningを扱うため，環境のベースとなるライブラリとして利用している．[7]

### Time

timeは時間を扱うためのライブラリで，システム時間の取得，時刻のフォーマット変換などが可能である．本研究では現在のシステムの時間を取得するtime.perf\_counter関数を利用している．[8]

### Torch

Torchは，「機械学習ライブラリ」「科学計算フレームワーク」である．GPUを活用する機械学習アルゴリズムを幅広くサポートしている．本研究では，処理時間計測の際にGPUとCPUの処理を同期させるtorch.cuda.synchronize を利用している．[9]

## ハードウェア

CPU

今回使用したのはIntel Xeon W-2223@3.6Ghzである．このCPUはコア数が４，スレッド数が８，メモリも最大で１TBまで使用することができる．

RAM

今回の環境のメモリ量は32GBである．

SSD

今回の環境では1TBのメモリを2枚利用している．

# 環境構築

## 初期設定

### NGCサイトにサインインする

NGCのサイトにアクセスする

URL　<https://catalog.ngc.nvidia.com/>

右上のSign inメニューを選択するとメールアドレスを入れるウィンドウが表示されるのでサインインする．

### API　KEYを生成する

Sign inに成功したら，右上のユーザー名からメニューを開き，Setupを選択する．

API の説明画面が出たら右上のGenerate API Keyボタンを押す．

確認のダイアログが表示されるので，Confirmボタンを押す．

API KEYが生成されるのでコピーして保存しておく．

### NGCのDockerへログインする

端末を開き，以下のコマンドを実行する．

sudo docker login nvcr.io

Usernameには　$oauthtokenと入力し，Passwordには先ほど生成したAPI KEYを入力する．

成功すると以下の画面が出る．(図3-1)



図 3-1　ログイン成功画面

### 利用したいDockerイメージを取得する

画面左上のCATALOGメニューを選択し，CONTAINERS，MODELS等から利用したい種類のDockerイメージを選択する．（図3-2）

そして，Overviewタグに記載されている指示に従い，Dockerイメージを取得して利用する．

今回はTensorFlowを選択して，指示に従いコンテナを起動した．



図 3-2　Dockerイメージ選択画面

## コンテナ起動（TensorFlow）

以下のコマンドを実行してTensorFlowを起動する．[10]

コンテナを起動するとrootユーザーに切り替わる．（図3-3）

sudo docker run –gpus all –it –rm nvcr.io/nvidia/tensorflow:22.09-tf2-py3

コマンド説明

--gpus all GPUと連携する．

-it ホストの入力をコンテナの標準出力と繋げる

　　　　　コンテナの標準出力とホストの出力を繋げる．

--rm コンテナ終了時に自動で削除する．

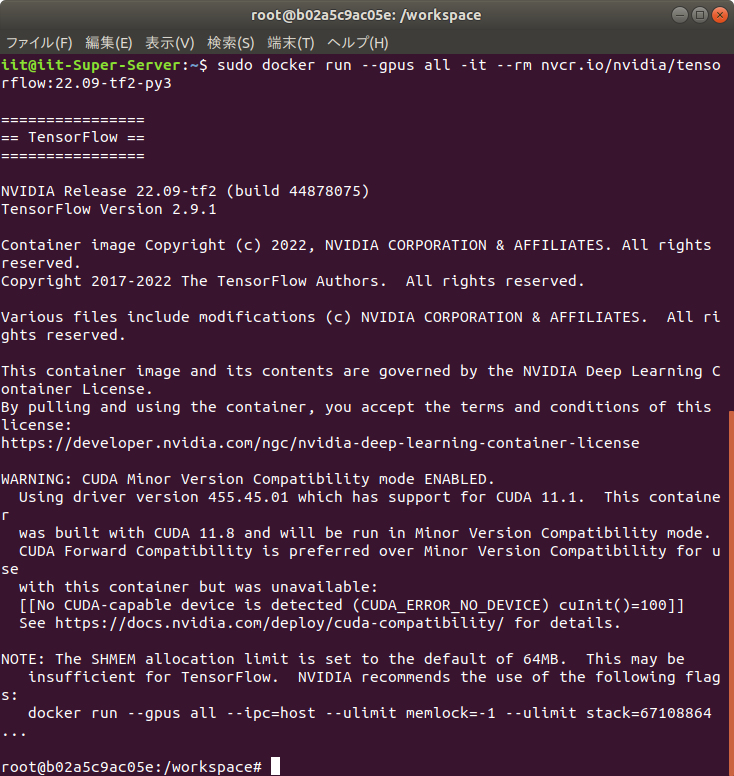


図 3-3　コンテナ起動画面

## コンテナにローカルのボリュームをマウントする

以下のようにコマンドを追加すると，ローカルのフォルダーをコンテナのフォルダーとして繋げることができる．[11]

-v /ホストの任意のパス:/コンテナの任意のパス

コマンド説明

-v ホストのディレクトリを任意の名前でコンテナにマウントすることができる．

実際に実行して，指定したパスでマウントできている様子を図3-4~図3-6に示す．

図 3-6　コンテナ内にマウントしたファイル

図 3-5　マウントを実行するコード

図 3-4 マウントするファイル

## X Window Systemの導入

最初の環境では，コンテナ内からウィンドウを表示することができていなかった．そこでX Window Systemを用いることでウィンドウの表示を可能にした．

X Window SystemとはUNIX系OSで用いられるウィンドウシステムである．ソフトウェアを実行するコンピュータと入出力するコンピュータを分離することができる．本環境ではデスクトップをホスト，ソフトウェアを実行する側をコンテナ内部として繋げている．[12]

まずは，どのホストを認証するか許可をする設定をホストで行う必要がある．

そこで以下のコマンドをホストユーザーで実行する．

xhost +local:root

xhostコマンドで確認すると図のように接続されているのが確認できる．（図3-7）



図 3-7　 X Window System接続確認

その後以下のコマンドを追加することでX Window Systemを利用することができる．

-e DISPLAY=$DISPLAY -v /tmp/.X11-unix/:/tmp/.X11-unix mino/debian-xterm

コマンド説明

-e 実行中のコンテナで環境変数を指定できる．今回はホストのDISPLAY変数をコンテナのDISPLAY変数に入れている．

これはX Window Systemを用いてウィンドウを表示している様子である．（図3-8）

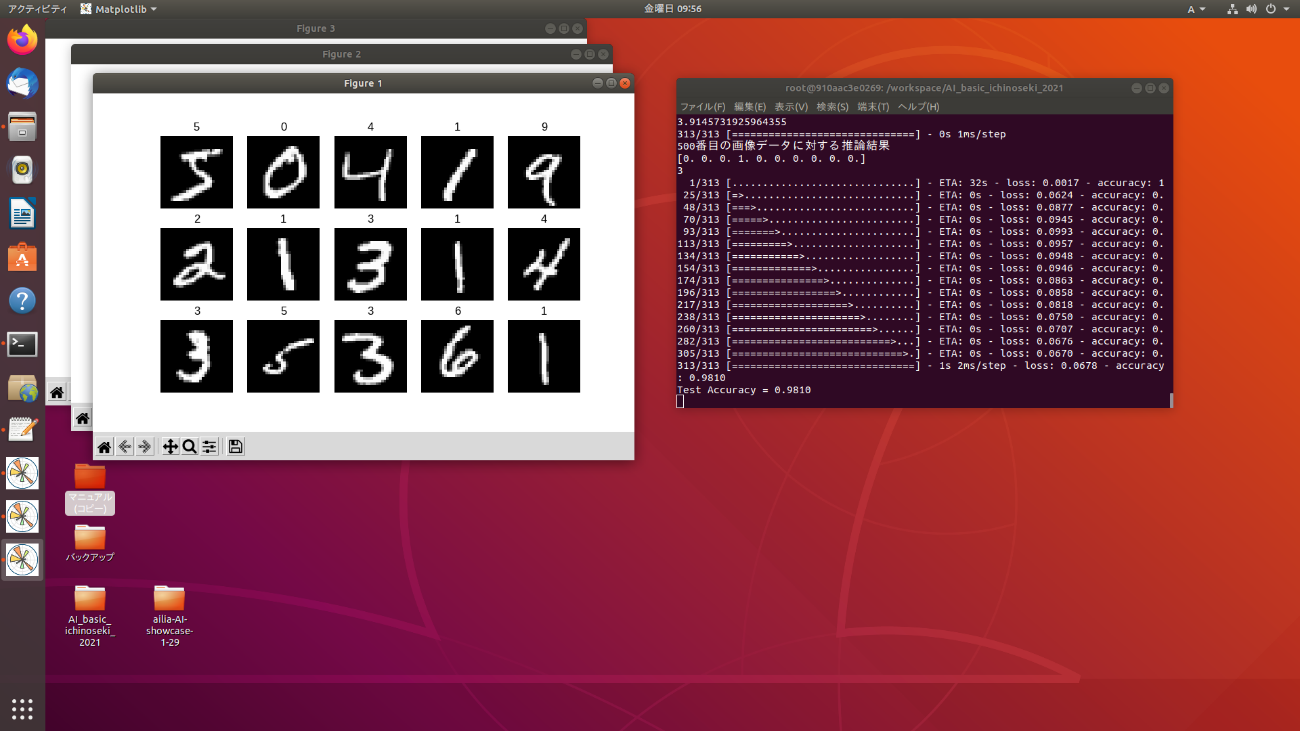


図 3-8　window表示画面

X Window Systemの導入方法は以上だが，pythonのプログラム上でウィンドウ表示を行う記述をしなければ表示されることはないので注意が必要である．（plt.show()を入れる）

## カメラ接続

接続したUSBカメラをコンテナ内で使用するためには，以下のコマンドを追加する[13]．

今回は接続したカメラがvideo0に接続されているのでvideo0を参照する．

--device=/dev/video0:/dev/video0

コマンド説明

--device　デバイスに許可を与える

今回はデモアプリでローカルのカメラを使用したいと思い，接続を行った．（図3-9）



図 3-9　デモアプリによるカメラ利用

## 共有メモリ量設定

今回コンテナを起動する際に以下のエラーが出た．

RuntimeError: DataLoader worker (pid 1067) is killed by signal: Bus error. It is possible that dataloader's workers are out of shared memory. Please try to raise your shared memory limit.

df –hコマンドで確認するとshm（POSIX共有メモリ）が64Mしか割当たっていなく，コンテナを起動できない．

そこで，以下のコマンドを追加してshmの容量を指定した．

--shm-size=2G

コマンドを実行したことで，shmの容量が2Gとなり，コンテナを起動することができた．（図3-10）

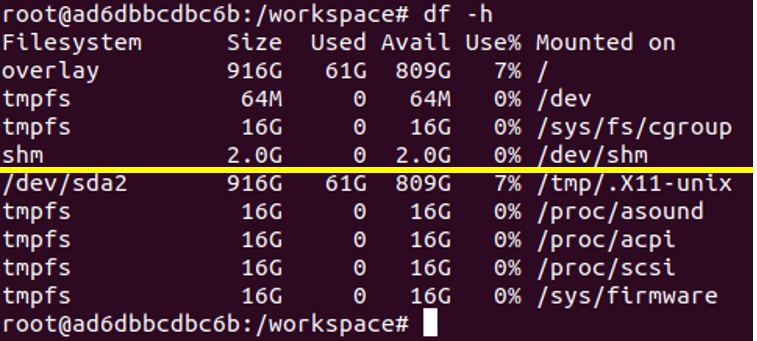


図 3-10　shmの容量

## コンテナコミット方法

コンテナに変更を加えたものから，新たにDockerイメージを作成することができる．これにより変更を加えたコンテナ環境を保存している．[14]

以下のコマンドで起動しているコンテナ一覧を表示し，CONTAINER IDを確認する．（図3-11）

sudo docker ps

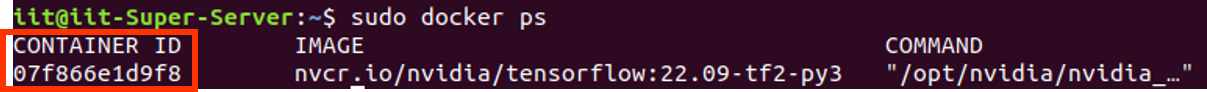


図 3-11　CONTAINER ID確認

そして，下記のコマンドでコンテナをコミットする．

sudo docker commit　 CONTAINER ID 　コミット後のコンテナ名

コミットしたコンテナを使いたい場合は実行コマンドの最後にコンテナの名前を入れることで実行できる．今回はTensorFlowの環境をtensorflow-plusという名前でコミットして利用した．また，yolov5のパッケージをインストールした，コンテナをtensorflow-tiikawaという名前でコミットして利用した．

コード実行例

iit@iit-Super-Server:~$ sudo docker commit 421d292c222b tensorflow-plus

# 処理時間計測

## 計測用サンプルデータ

処理時間計測では2種類のサンプルデータを用いた．

### 手書き数字の画像認識

データセットとして「MNIST」を使用した．「MNIST」とは「０」～「９」の手書き数字の画像データセットである．(図4-1)

主に画像認識を目的としたディープラーニング/機械学習の初心者向けチュートリアルでよく使われる．データがきれいに整形されているので，高い精度が出やすいといわれている．

データセット全体で6万枚の学習用データと1万枚のテストデータがある．各画像にはラベル（正解を示す教師データ）がついている．[15]

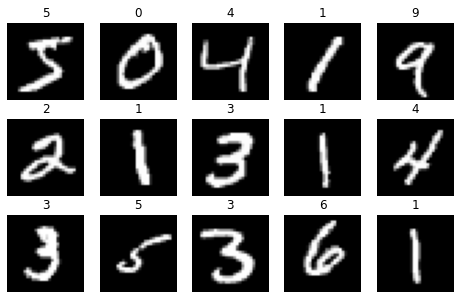


図 4-1　MNISTのデータセット

### 物体検知用キャラクター学習

他の卒業研究[16]でyolov5を用いたキャラクターの物体検知を行っていた．そのデータセットを使わせていただき処理時間の計測を行った．画像から学習させたいキャラクターの座標を，xmlファイルで記述し学習させる．学習用データとして29枚の画像が用意されている．(図4-2)



図 4-2キャラクター学習用データ

## 計測方法・計測場所

計測するのはサンプルが学習を行う部分である．その際，学習開始位置と学習終了位置に変数としてstartとendを作成し，それぞれtime.perf\_counter関数を代入することで，現在の時間を取得する．そして，開始位置startと終了位置endの差を処理時間とする．

終了時間（end） – 開始時間（start） =　処理時間

また，GPUを利用する際にはtorch.cuda.synchronizeをtime.perf\_counter関数の前に入れて処理を同期させる．

使用関数説明

time.perf\_couter()

パフォーマンスカウンターの値（小数点以下がミリ秒）を返す．クロックは短期間の計測が行えるよう，可能な限り高い分解能を持つ（今回は小数点第16位まで表示された）．

torch.cuda.synchronize()

CUDAデバイス上のすべてのストリームのすべてのカーネルが完了するまで待機する．つまり，CPUとGPUを同期することで，処理終了時間を一致させることができる．CUDAを利用しているGPUでしか利用することができない．

### 手書き数字の画像分類

計測する位置はmodel.fit()の中である．（図4-3）



図 4-3　手書き数字の学習計測位置

コードの解説（model.fit）

trainXとtrainYは学習用データを保持する変数である．

batch\_sizeは学習用データをわけた際の一つのサイズを指定している．

epochsはデータ全体を何回学習させるかを指定している．

validation\_dataは検証用データを保持する変数である．

shuffleは読み込ませるデータをランダムにすることができる．

EarlyStoppingはある程度学習の精度が良くなると，自動で学習を終了する．

val\_lossが0に近づけば，学習をもとに正解を導けていることになる．

verboseは処理の過程を詳細に表示することができる．

### 物体検知用キャラクター学習

計測する位置はyolov5のファイルにあるtrain.py内の学習部分で行う．（図4-4，図4-5）

変数としてprocessing\_timeを用意しておくことで，epochが一回終了するタイミングの時間を記録している．



図 4-4　キャラクター学習の学習開始位置



図 4-5　キャラクター学習の学習終了位置

## 計測実行

Google ColaboratoryのGPUを使う場合は，設定を行いGPUと繋げる必要がある．

①Google Colaboratoryの画面の右上にある接続メニューを開き，「リソースを表示」を選択する．

②「ランタイムのタイプを変更」を選択する．（図4-6）

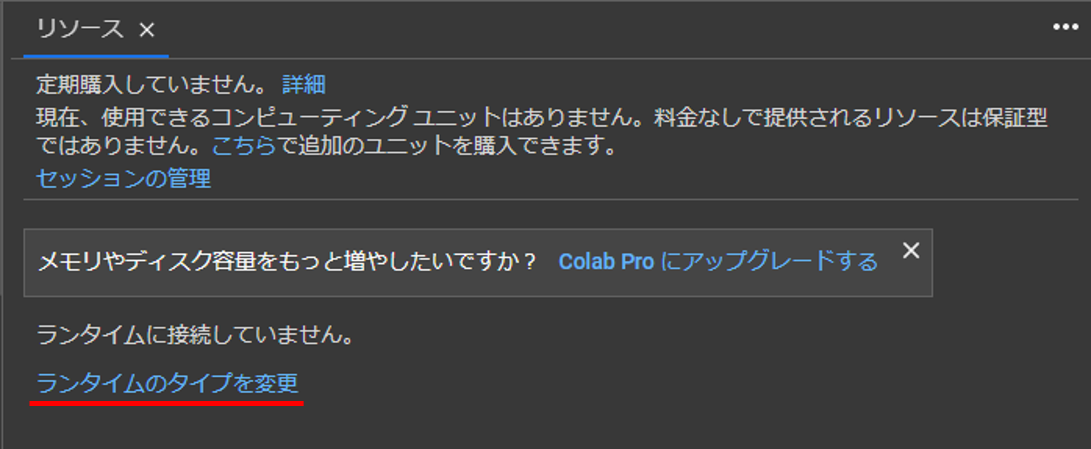


図 4-6　リソース表示画面

③「ハードウェアアクセラレータ」からGPUを選択することで設定完了である．(図4-7)

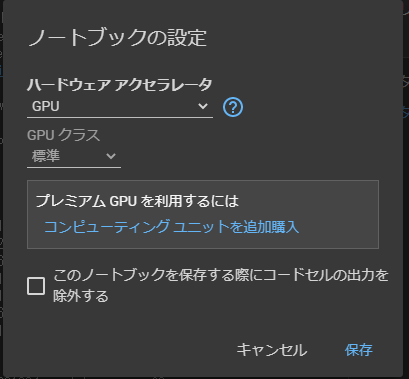


図　4-7　ランタイムのタイプ変更画面

なお，今回は繋げていないが，ローカルの処理装置と繋げることも可能である．

### 手書き数字の画像分類

GPUを用いて処理する場合は以下のコードでコンテナを作成する．

sudo docker run --gpus all -e DISPLAY=$DISPLAY -v /tmp/.X11-unix/:/tmp/.X11-unix -v /home/iit/Document/AI\_basic\_ichinoseki\_2021:/workspace/AI\_basic\_ichinoseki\_2021 -it --rm tensorflow-plus

CPUを用いて処理する場合は以下のコードでコンテナを作成する．

sudo docker run -e DISPLAY=$DISPLAY -v /tmp/.X11-unix/:/tmp/.X11-unix -v /home/iit/Document/AI\_basic\_ichinoseki\_2021:/workspace/AI\_basic\_ichinoseki\_2021 -it --rm tensorflow-plus

ディレクトリを移動する．

cd AI\_basic\_ichinoseki\_2021

プログラムを実行する．（図4-8）

python Chapter05.py

Google Colaboratoryで実行する場合はソースコードを直接書き込み実行する．

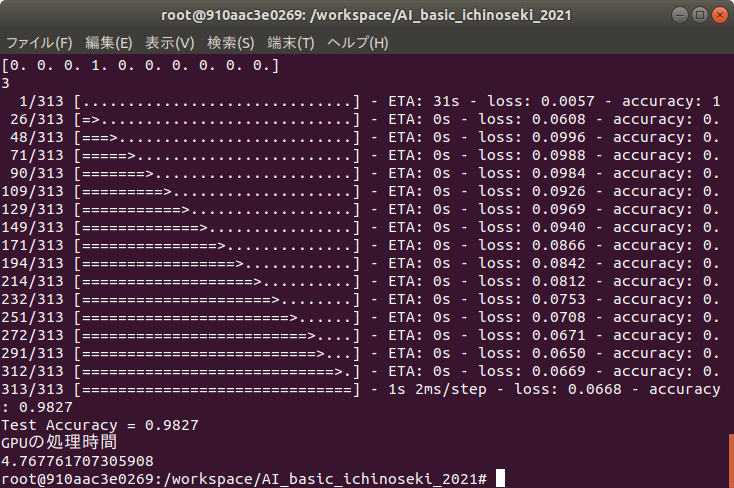


図 4-8　手書き数字学習実行画面（専用PC）

学習データを利用して，テスト用データを推論する．(図4-9)

結果を見ると正しく推論されていることが確認できる．(図4-10)

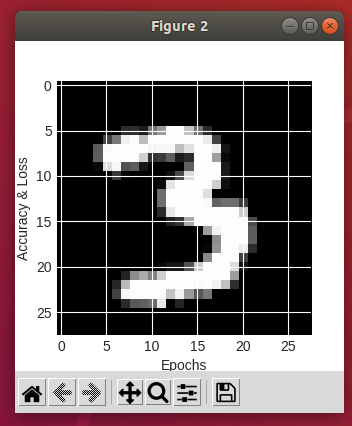


図4-9　推論対象データ



図4-10　推論結果

### 物体検知用キャラクター学習

GPUを用いて処理する場合は以下のコードでコンテナを作成する．

sudo docker run --gpus all -e DISPLAY=$DISPLAY -v /tmp/.X11-unix/:/tmp/.X11-unix -v /home/iit/Document/tiikawa:/workspace/tiikawa --shm-size=2G -it --rm tensorflow-tiikawa

CPUを用いて処理する場合は以下のコードでコンテナを作成する．

sudo docker run -e DISPLAY=$DISPLAY -v /tmp/.X11-unix/:/tmp/.X11-unix -v /home/iit/Document/tiikawacpu:/workspace/tiikawacpu --shm-size=2G -it --rm tensorflow-tiikawa

ディレクトリを移動する．

cd tiikawa/yolov5

プログラムを実行する．Batch-sizeとepoch数は実行する際に指定する．

python train.py --batch 20 --epochs 10 --data '/workspace/tiikawa/yolov5/tiikawa.yaml' --name tiikawa

-- name　保存するファイルの名前を指定する．

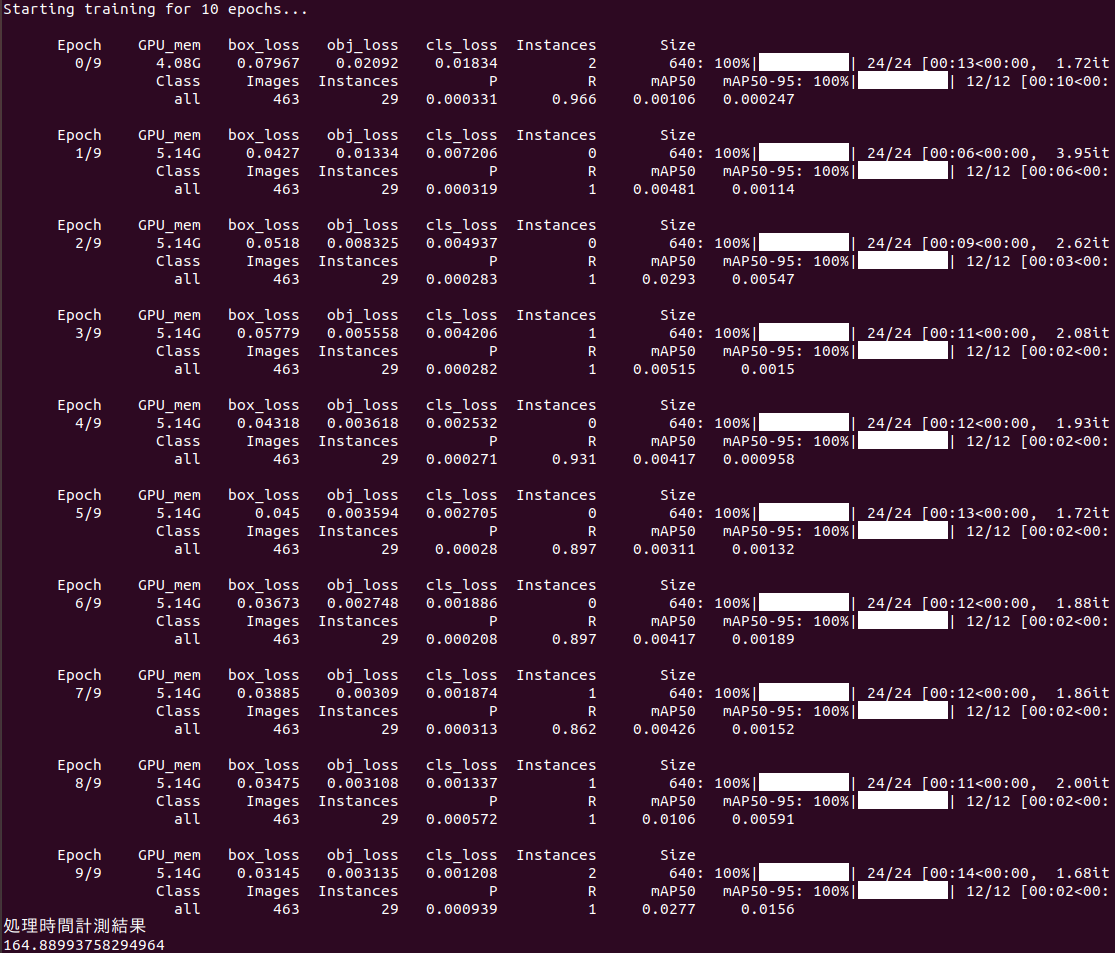


図4-11　キャラクター学習実行画面

Google Colaboratoryを用いるときは，プログラムを実行する前に，データセットが入っているドライブと繋げる必要がある．

以下のコードを実行し，ドライブをマウントする．

from google.colab import drive

drive.mount('/content/drive')

以下のコードを実行し，yolov5のパッケージを一括でインストールする．

!pip install -r /content/drive/MyDrive/tiikawa/yolov5/requirements.txt

ディレクトリを移行し，プログラムを実行する．

%cd /content/drive/MyDrive/tiikawa/yolov5

!python train.py --batch 20 --epochs 10 --data '/content/drive/My Drive/tiikawa/tiikawa.yaml' --name tiikawa

学習データを利用して，物体検知を行う．

専用PCで実行したデータは/workspace/tiikawa/yolov5/runs/trainに保存されている．

Google Colaboratoryで実行したデータは/マイドライブ/tiikawa/yolov5/runs/trainに保存されている．

テスト用データを見るとキャラクターを検知していることが確認できる．

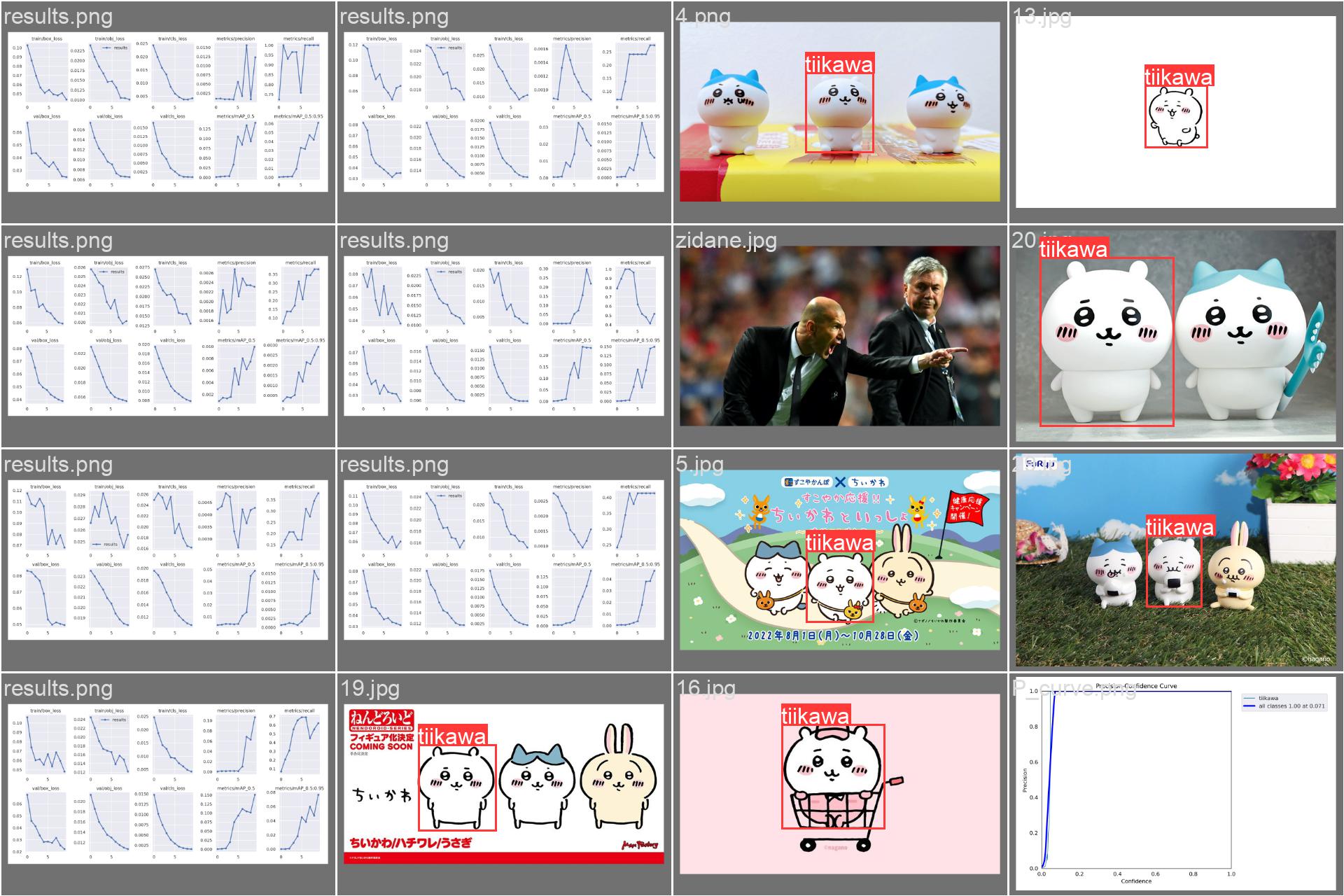


図 4-12　物体検知結果

# 結果・考察

## 計測結果

### 手書き数字の画像分類

3つの処理装置でそれぞれ5回ずつ計測し，平均をとった結果を表5-1に示す．

|  |  |
| --- | --- |
| GPU | 3.88秒 |
| CPU | 13.16秒 |
| Google ColaboratoryのGPU | 4.32秒 |

表 5-1 処理装置別手書き数字の学習処理時間

専用PCのGPUとCPUの処理速度を比較すると，3.4倍GPUが速く処理していることが確認できた．

また，学習枚数に応じた処理時間を比較したグラフを以下に示す．(図5-1)

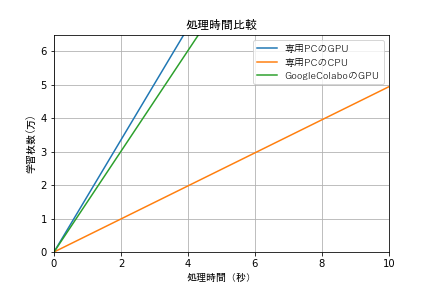


図5-1 処理時間比較グラフ（手書き）

### 物体検知用キャラクター学習

3つの処理装置に同様の学習処理を行わせた結果を表5-2に示す．

|  |  |
| --- | --- |
| GPU | 157.9秒 |
| CPU | 1222秒 |
| Google ColaboratoryのGPU | 330.3秒 |

表5-2　処理装置別キャラクター学習処理時間

専用PCのGPUとCPUの処理速度を比較すると，7.7倍GPUが速く処理していることが確認できた．

また，実行回数に応じた処理時間を比較したグラフを以下に示す．（図5-2）

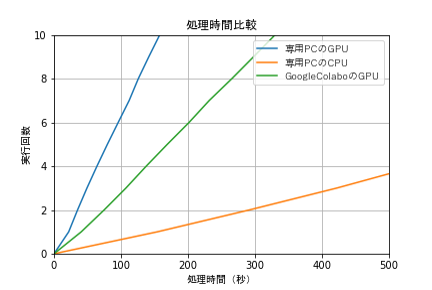


図 5-2　処理時間比較グラフ（キャラクター学習）

## 考察

手書き数字の学習時間と，キャラクター学習の時間を見ると，どちらも専用PCのGPUの処理が一番速いことを確認できた．

また，グラフを見ると手書き数字の学習よりもキャラクター学習の時間のほうがGPUの処理性能に差があった．理由として考えられるのは，手書き数字のデータセットが整いすぎていることだと思われる．学習枚数は手書き数字のほうが多いが，画像のサイズが均一で，ラベルのつけ方が簡単であることから処理に差があまり生まれなかったと推測できる．より高速にDeep Learningを処理するためには，データセットの作り方を工夫する必要があると思われる．

また，複雑な処理であればGPUの性能の差が顕著に出たので，今後大容量で複雑な処理を行う研究や開発に利用されることを期待している．

# おわりに

今回の研究では，コンテナ環境という未知の環境を扱うことになり大変苦戦した．始めは環境のことをよく理解せずに研究を行っていたが，それにより環境を壊してしまうことがあった．しっかりと環境を理解して使う大切さを学ぶことができた．また，研究を通してDeep Learningの仕組みとGPUの重要性を体験することができた．自分でデータセットを作って学習をさせ、システムを作ることはできなかったが，専用PCの可能性を十分に感じたので，これから何らかの機能を作り，活用することができればよいと思った．

# 参考文献

[1] 株式会社ピーエスシー 【話題のITトレンド】コンテナ技術と仮想マシンの違いとは？

<https://psc-smartwork.com/topics/2021/08/162.html>

[2] VERITAS コンテナ化とそのメリットについて

<https://www.veritas.com/ja/jp/information-center/containerization>

[3] Wikipedia Docker

<https://ja.wikipedia.org/wiki/Docker>

[4] DevOps GPUを利用するためのDocker プラグイン「NVIDIA Docker」とは？導入手順は？(前編)

<https://licensecounter.jp/devops-hub/blog/nvidia-1/>

[5] TECH CAMPブログ 【3分解説】CPUとGPUの違いをわかりやすく紹介！用途やコア数も

<https://tech-camp.in/note/technology/42935/>

[6] NVIDIA GEFORCERTX 3090 ファミリ

<https://www.nvidia.com/ja-jp/geforce/graphics-cards/30-series/rtx-3090-3090ti/>

[7] Wikipedia TensorFlow

<https://ja.wikipedia.org/wiki/TensorFlow>

[8] python time --- 時刻データへのアクセスと変換

https://docs.python.org/ja/3.8/library/time.html?highlight=time#module-time

[9] OSS×Cloud オープンソースのAI・人工知能／Torchとは

https://www.ossnews.jp/oss\_info/Torch

[10] NVIDIA NGC CATALOG TensorFlow

<https://catalog.ngc.nvidia.com/orgs/nvidia/containers/tensorflow>

[11] Docker-docs-ja docker run ボリュームのマウント

<https://docs.docker.jp/engine/reference/commandline/run.html>

[12] IT用語辞典e-Words XWindow System[x11]x.Org

<https://e-words.jp/w/X_Window_System.html>

[13] ITmediaエンタープライズ　　第17回 Dockerで植物が育つ様子を自動録画してみよう――その1

<https://www.itmedia.co.jp/enterprise/articles/1603/02/news031_2.html>

[14] Docker-docs-ja docker commit

https://docs.docker.jp/engine/reference/commandline/commit.html

[15] ＠IT MNIST：手書き数字の画像データセット

<https://atmarkit.itmedia.co.jp/ait/articles/2001/22/news012.html>

[16] 遠藤綾李：深層学習を用いたごみ分別支援ツールの作成，令和4年度卒業研究発表会，番号07， 2023