OCTOPUSでDockerを使ってみた

TensorFlow・PyTorchによる画像分類モデルの実行

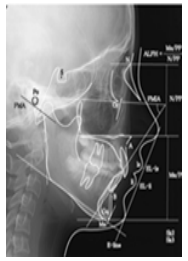
岡山理科大学 情報理工学部
Lee Chonho
2021.12.13

Zoomミーティング
https://zoom.us/j/99993623088?pwd=dGNkcndXRVZROUo5S3JxNUJvdGEyQT09
ミーティングID: 999 9362 3088、パスコード: 550902

様々な応用・アプリケーション

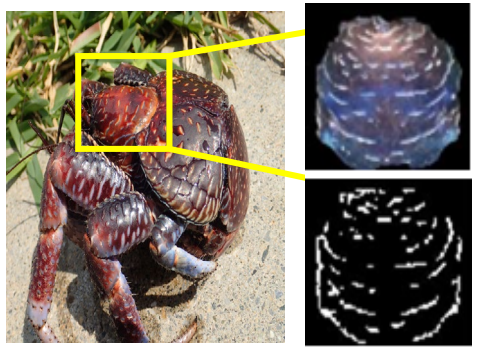
診断
所見自動作成



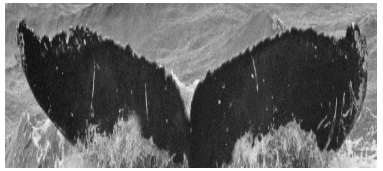


正面観：非対称である。
オトガイの右方への偏位を認める。
鼻尖の左方への偏位を認める。
笑顔表出時には、上顎前歯の歯冠全体と下顎前歯の歯冠の約1/3の露出を認める。
安静時には、上顎前歯の歯冠の約6mmの露出を認める。
側面観：convex type profileである。
口唇閉鎖時および安静時にオトガイの緊張を認める。
E line：上赤唇はE lineより約8.0mm前方に位置している。
下赤唇はE lineより約11.5mm前方に位置している。
鼻唇角：鼻唇角は大きい。Nasolabial angle: 105°
顔面の垂直的バランス：前中顔面の長さに対して、前下顔面の長さは標準的である。
鼻下点から口裂までの長さに対して、口裂からオトガイまでの長さは標準より短い。

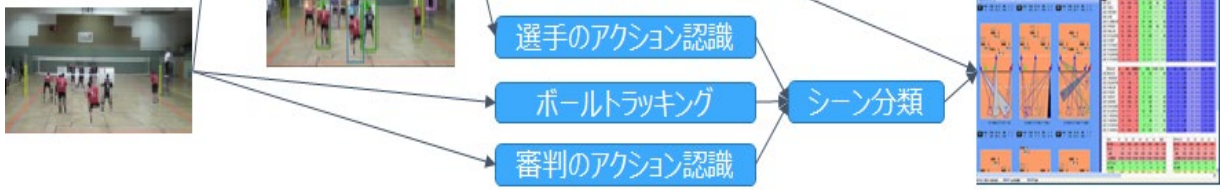
ヤシガニ



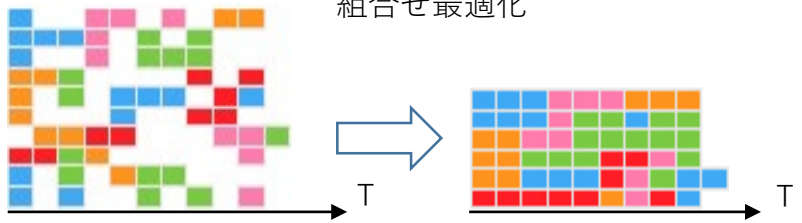
クジラ



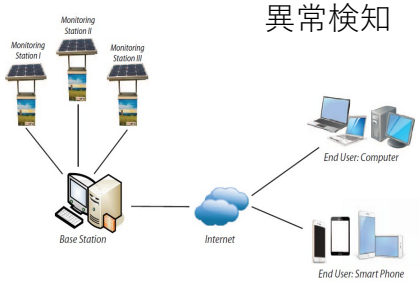

スコアブック自動作成



組合せ最適化



異常検知



Dockerを利用するメリット

- TensorFlowのバージョン
  - Python
  - cuDNN
  - CUDA
    - GPUドライバのバージョン

- 上記バージョン互換性を考慮して事前に作成したDockerコンテナを利用
  - 動くことが保証されている
  - 環境構築の時間を節約

Version	Python version	Compiler	Build tools	cuDNN	CUDA
tensorflow-2.3.0	3.5-3.8	GCC 7.3.1	Bazel 3.1.0	7.6	10.1
tensorflow-2.2.0	3.5-3.8	GCC 7.3.1	Bazel 2.0.0	7.6	10.1
tensorflow-2.1.0	2.7, 3.5-3.7	GCC 7.3.1	Bazel 0.27.1	7.6	10.1
tensorflow-2.0.0	2.7, 3.3-3.7	GCC 7.3.1	Bazel 0.26.1	7.4	10.0
tensorflow_gpu-1.15.0	2.7, 3.3-3.7	GCC 7.3.1	Bazel 0.26.1	7.4	10.0
tensorflow_gpu-1.14.0	2.7, 3.3-3.7	GCC 4.8	Bazel 0.24.1	7.4	10.0
tensorflow_gpu-1.13.1	2.7, 3.3-3.7	GCC 4.8	Bazel 0.19.2	7.4	10.0
tensorflow_gpu-1.12.0	2.7, 3.3-3.6	GCC 4.8	Bazel 0.15.0	7	9
tensorflow_gpu-1.11.0	2.7, 3.3-3.6	GCC 4.8	Bazel 0.15.0	7	9
tensorflow_gpu-1.10.0	2.7, 3.3-3.6	GCC 4.8	Bazel 0.15.0	7	9
tensorflow_gpu-1.9.0	2.7, 3.3-3.6	GCC 4.8	Bazel 0.11.0	7	9
tensorflow_gpu-1.8.0	2.7, 3.3-3.6	GCC 4.8	Bazel 0.10.0	7	9
tensorflow_gpu-1.7.0	2.7, 3.3-3.6	GCC 4.8	Bazel 0.9.0	7	9

CUDA Toolkit	Linux x86_64 Driver Version
CUDA 11.0 (11.0.171)	>= 450.36.06
CUDA 10.2 (10.2.89)	>= 440.33
CUDA 10.1 (10.1.105)	>= 418.39
CUDA 10.0 (10.0.130)	>= 410.48
CUDA 9.2 (9.2.88)	>= 396.26
CUDA 9.1 (9.1.85)	>= 390.46
CUDA 9.0 (9.0.76)	>= 384.81

CUDA 11.4	>=450.80.02
CUDA 11.3	>=450.80.02
CUDA 11.2	>=450.80.02
CUDA 11.1 (11.1.0)	>=450.80.02
CUDA 11.0 (11.0.3)	>=450.36.06*

Dockerを利用するメリット

- PyTorchのバージョン
  - Python
  - CUDA
  - torchvision
  - torchaudio

```
# CUDA 11.1
pip install torch==1.8.0+cu111 torchvision==0.9.0+cu111 torchaudio==0.8.0

# CUDA 10.2
pip install torch==1.8.0 torchvision==0.9.0 torchaudio==0.8.0
```

```
# CUDA 11.0
pip install torch==1.7.1+cu110 torchvision==0.8.2+cu110 torchaudio==0.7.2

# CUDA 10.2
pip install torch==1.7.1 torchvision==0.8.2 torchaudio==0.7.2

# CUDA 10.1
pip install torch==1.7.1+cu101 torchvision==0.8.2+cu101 torchaudio==0.7.2

# CUDA 9.2
pip install torch==1.7.1+cu92 torchvision==0.8.2+cu92 torchaudio==0.7.2
```

現在OCTOPUSで利用可能なDockerコンテナの確認

```
$ qstat --template -l

[Container Template]
==========================================================================
=
Template       L  Image          CPU  Memory  GPU  Custom          Comment
-------------   -  -------------- ---- --------- ---- ------------------ ------------------
tensorflow-2.7    - oct-tensorflow-2.7-gpu   10     16GB    4 (none)      TensorFlow 2.7
tensorflow-1.14  - oct-tensorflow-1.14-gpu  10    16GB    4 (none)      TensorFlow 1.14
pytorch-1.4       - oct-pytorch-1.4           10     16GB    4 (none)      PyTorch 1.4
pytorch-0.4       - oct-pytorch-0.4:py36      10     16GB    4 (none)      PyTorch 0.4
```

事前準備：ハンズオン用サンプルのコピー

作業ディレクトリに移動

```
$ cd /octfs/work/kosyuXXX/[ユーザID]
```

XXXはユーザID末尾の
3桁の数字

working directory (wd)
と仮定します。

事前準備：ハンズオン用サンプルのコピー

作業ディレクトリに移動

```
$ cd /octfs/work/kosyuXXX/[ユーザID]
```

XXXはユーザID末尾の
3桁の数字

working directory (wd)
と仮定します。

WDにサンプルデータをコピー（ピリオドも忘れずに）

```
$ cp -r /octfs/apl/kosyu/20211213_docker .
```

```
[wd]/20211213_docker
        ├── tensorflow-sample
        │   ├── datasets/
        │   ├── job-mlp.sh
        │   ├── job-cnn.sh
        │   ├── job-resnet.sh
        │   ├── tf-mlp.py
        │   ├── tf-cnn.py
        │   └── tf-resnet-predict.py
        └── pytorch-sample
            ├── datasets/
            ├── datasets.py
            ├── job-cnn.sh
            └── pt-cnn.py
```

ジョブスクリプトについて

- ソースコード・スクリプト・モジュールを計算機で実行するには、ジョブ要求用のシェルスクリプトファイル（ジョブスクリプトファイル）を作成して、スケジューラに投入します。

http://www.hpc.cmc.osaka-u.ac.jp/system/manual/octopus-use/jobscript/

ジョブスクリプトについて

- ソースコード・スクリプト・モジュールを計算機で実行するには、ジョブ要求用のシェルスクリプトファイル（ジョブスクリプトファイル）を作成して、スケジューラに投入します。

http://www.hpc.cmc.osaka-u.ac.jp/system/manual/octopus-use/jobscript/

ジョブスクリプトの例（job.sh）

```
#!/bin/bash
#PBS -q OCTOPUS
#PBS -l elapstim_req=1:00:00,cpunum_job=24
#PBS -M user@hpc.cmc.osaka-u.ac.jp
#PBS -m b
cd $PBS_O_WORKDIR
./a.out > result.txt
```

大きくは2段階

・利用する計算機のリソースや環境の指定

・計算機に実行させる処理の記述

ジョブスクリプトについて

- ソースコード・スクリプト・モジュールを計算機で実行するには、ジョブ要求用のシェルスクリプトファイル（ジョブスクリプトファイル）を作成して、スケジューラに投入します。

http://www.hpc.cmc.osaka-u.ac.jp/system/manual/octopus-use/jobscript/

ジョブスクリプトの例（job.sh）

```
#!/bin/bash
#PBS -q OCTOPUS
#PBS -l elapstim_req=1:00:00,cpunum_job=24
#PBS -M user@hpc.cmc.osaka-u.ac.jp
#PBS -m b
cd $PBS_O_WORKDIR
./a.out > result.txt
```

ノード群のジョブクラス

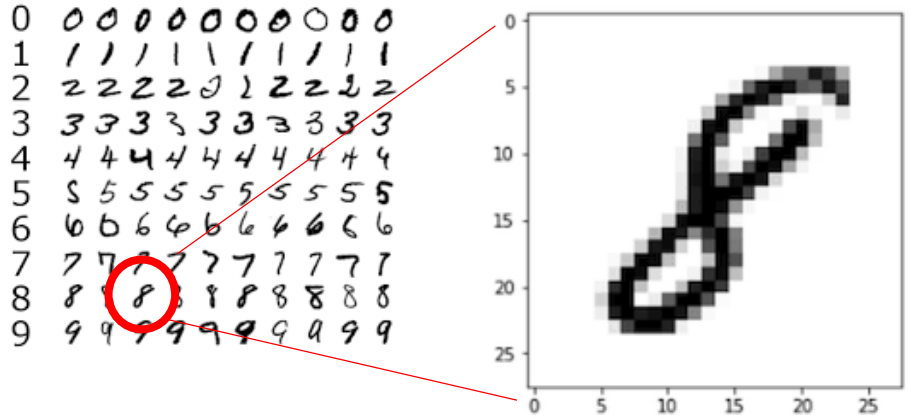
ジョブが実行される経過時間, ノードあたりの使用CPUコア数

メールアドレス指定
メールオプション

S1 TensorFlow：MLPサンプルの概要

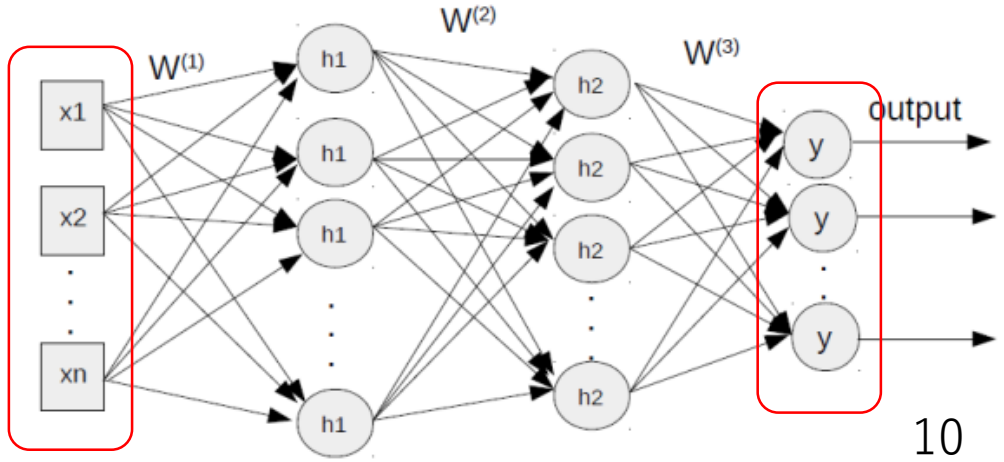
```
$ cd 20211213_docker/tensorflow-sample
```

Dataset：MNIST – 手書き文字画像



Task：画像に書かれた数字の分類

Model：Multi-Layer Perceptron（MLP）



28x28＝726

S1 TensorFlow：学習・推論コードの概要

tf-mlp.py

① データセット読み込み
② モデルの構築
③ モデルのコンパイル
④ 学習開始
⑤ モデルの評価
⑥ モデルの保存と読み込み
⑦ テストデータを用いて分類

```
num_classes = 10


model = tf.keras.models.Sequential()
model.add( tf.keras.layers.Flatten(
             input_shape=(28, 28)) )
model.add( tf.keras.layers.Dense(
             128, activation='relu') )
model.add( tf.keras.layers.Dropout(0.2) )
model.add( tf.keras.layers.Dense(
             num_classes, activation='softmax') )
```

S1 TensorFlow：学習・推論コードの概要

tf-mlp.py

① データセット読み込み
② モデルの構築
③ モデルのコンパイル
④ 学習開始
⑤ モデルの評価
⑥ モデルの保存と読み込み
⑦ テストデータを用いて分類

```
model.compile(
    optimizer='Adam',
    loss='sparse_categorical_crossentropy',
    metrics=['accuracy'])
```

ラベル（y_train, y_test）をone-hot-vectorとして扱う場合は、“categorical_crossentropy”

S1 TensorFlow：学習・推論コードの概要

tf-mlp.py

① データセット読み込み
② モデルの構築
③ モデルのコンパイル
④ 学習開始
⑤ モデルの評価
⑥ モデルの保存と読み込み
⑦ テストデータを用いて分類

旧バージョンで使われていた保存方法

※tf-1.14, tf-2.7用のコンテナを用意してるので
意図的に旧バージョン用のサンプルにします。

```
# 保存
model_filename = 'mymodel_mlp.h5'
model.save(model_filename)

# 読み込み
model = tf.keras.models.load_model(model_filename)
```

S1 TensorFlow：ジョブスクリプト

ジョブスクリプト

```
$ more job-mlp.sh

#!/bin/sh
#PBS -q ODT
#PBS -y 373
#PBS -l elapstim_req=0:05:00
#PBS --template=tensorflow-1.14
#PBS -v TF_CPP_MIN_LOG_LEVEL=2
cd $PBS_O_WORKDIR
python3 tf-mlp.py
```

ノード群のジョブクラス
Docker使用「ODT」

ハンズオン用に
今日だけ追加　–y 373

コンテナ指定

※tf-2.7でも動くが
Warningが出る

TFのログ、メッセージ出力設定

S1 TensorFlow：サンプル実行

```
$ qsub job-mlp.sh
Request 127803.oct submitted to queue: ODT
```

ジョブをスケジューラーへ投入

http://www.hpc.cmc.osaka-u.ac.jp/system/manual/scheduler/

S1 TensorFlow：サンプル実行

```
$ qsub job-mlp.sh
Request 127803.oct submitted to queue: ODT
```

ジョブをスケジューラーへ投入

リクエストの状態を表示

```
$ qstat
RequestID       ReqName  UserName Queue     Pri STT S   Memory      CPU   Elapse R H M Jobs
--------------- -------- -------- -------- ---- --- - -------- -------- -------- - - - ----
127796.oct      job-mlp. w6a010   ODT          0 QUE -    0.00B     0.00        0 Y Y Y    1
```

リクエストのスケジュール状態を表示

```
$ sstat
RequestID       ReqName  UserName Queue                  Pri STT PlannedStartTime
--------------- -------- -------- -------- ---------------- --- ------------------
127796.oct      job-mlp. w6a010   ODT        0.5002/  0.5002 ASG 2021-12-08 15:33:59
```

http://www.hpc.cmc.osaka-u.ac.jp/system/manual/scheduler/

S1 TensorFlow：実行結果の確認

```
$ sstat
Request does not exist on JobManipulator
```

またはジョブスクリプトに
#PBS –m e
を追記しておいて、終了時にメールを受け取る

```
$ ls –l
      job-mlp.sh.o[リクエストID]
      job-mlp.sh.e[リクエストID]
```

o：標準出力とe:エラー出力が生成されます。
#PBS –o [ファイル名]
#PBS –e [ファイル名]
でファイル名を指定もできます。

今回のサンプルでは

mymodel_mlp.h5 ファイルが生成されており、学習済みモデルの情報が保存されます。

実際には、モデル学習以外の前処理「データセットの準備」、そして、後処理「⑥学習済みモデルを読み込んで、⑦テストする」を分けて、それぞれのジョブスクリプトを別途用意して、計算機を利用することをお勧めします。

S2

TensorFlow：CNNサンプルの概要

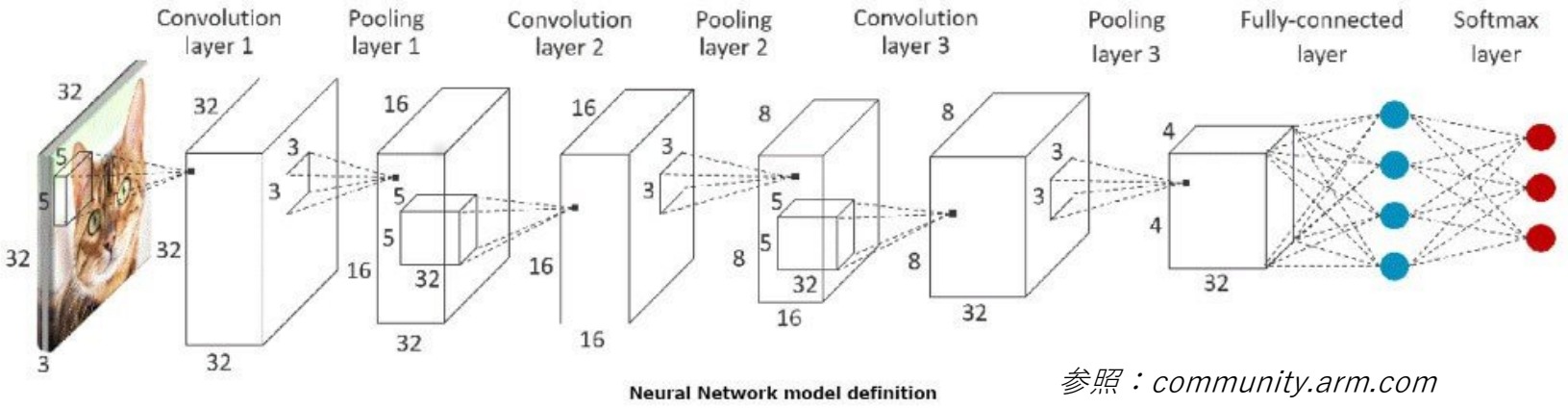
```
$ cd 20211213_docker/tensorflow-sample
```

Dataset：Cifar10 （一般物体画像）

Task：画像に写った物体の分類

Model：Convolutional Neural Network

32x32



参照：community.arm.com

S2 TensorFlow：CNNサンプルコードの概要

tf-cnn.py

① データセット読み込み
② モデルの構築
③ モデルのコンパイル
④ 学習開始
⑤ モデルの評価
⑥ モデルの保存

```
model = tf.keras.models.Sequential()
model.add( tf.keras.layers.Conv2D(32, (3, 3),
      input_shape=(32, 32, 3), activation='relu') )
model.add( tf.keras.layers.MaxPooling2D(
                              pool_size=(2,2)) )
model.add( tf.keras.layers.Conv2D(32, (3, 3),
                              activation='relu') )
model.add( tf.keras.layers.MaxPooling2D(
                              pool_size=(2,2)) )
model.add( tf.keras.layers.Flatten() )
model.add( tf.keras.layers.Dense(128,
                              activation='relu') )
model.add( tf.keras.layers.Dropout(0.2) )
model.add( tf.keras.layers.Dense(num_classes,
                              activation='softmax') )
```

S2 TensorFlow：CNNサンプルコードの概要

tf-cnn.py

① データセット読み込み
② モデルの構築
③ モデルのコンパイル
④ 学習開始
⑤ モデルの評価
⑥ モデルの保存

SavedModel形式

※フォルダ名を指定

```
# 保存
model_filename = 'mymodel_cnn'
model.save(model_filename)

# 読み込み
model = tf.keras.models.load_model(model_filename)
```

S2 TensorFlow：CNNサンプル用のジョブスクリプト

```
$ more job-cnn.sh

#!/bin/sh
#PBS -q ODT
#PBS -y 373
#PBS -l elapstim_req=0:05:00
#PBS --template=tensorflow-2.7
#PBS -v TF_CPP_MIN_LOG_LEVEL=2
cd $PBS_O_WORKDIR
python3 tf-cnn.py

$ qsub job-cnn.sh
```

ノード群のジョブクラス
Docker使用「ODT」

ハンズオン用に
今日だけ追加　–y 373

コンテナ指定

※tf-1.14だとエラー

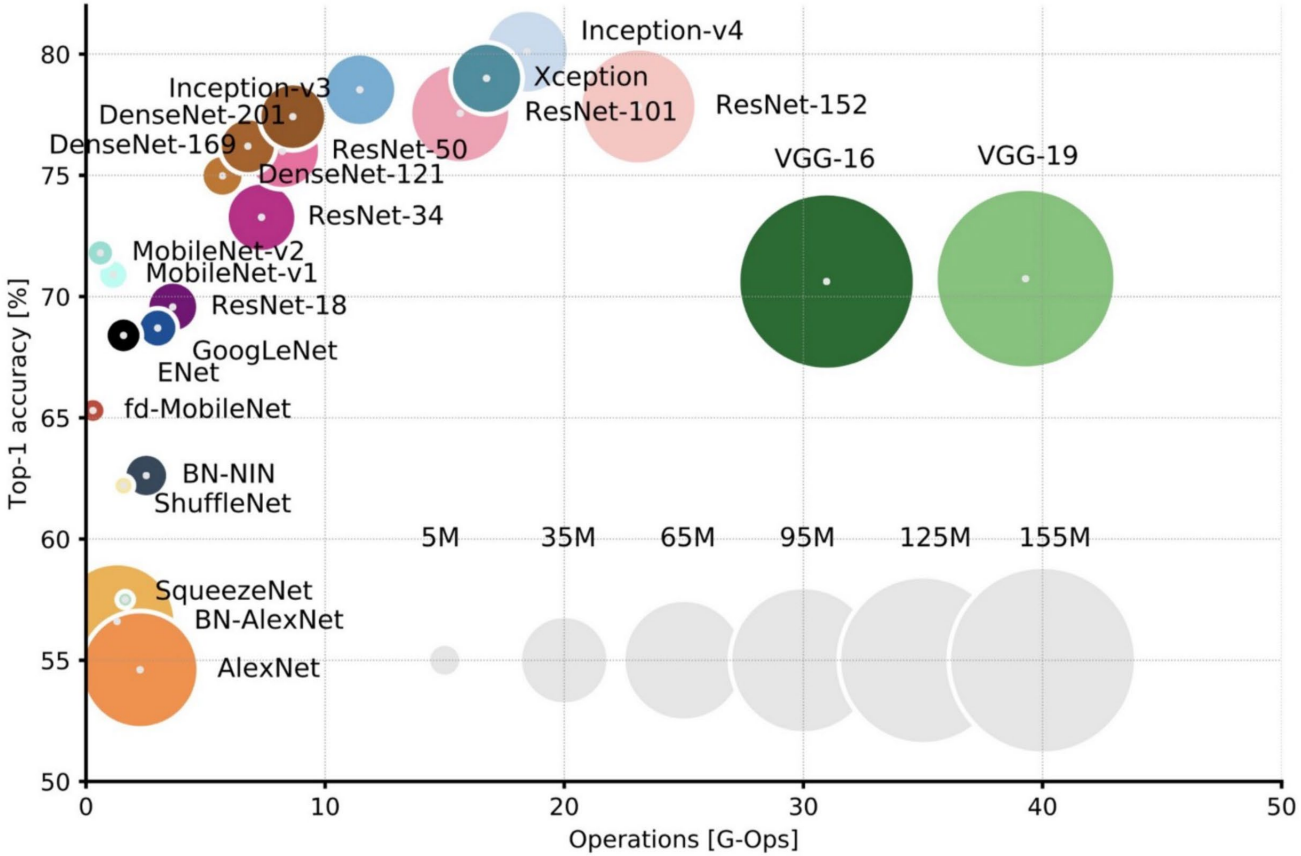
ジョブをスケジューラーへ投入

今回のサンプルでは

mymodel_cnn フォルダが生成され

学習済みモデルの情報が保存されます。

S3 TensorFlow：学習済みモデルを用いて推論



Alfred Canziani, Analysis of deep neural networks (2018)

ResNet50モデルで画像分類

(https://www.tensorflow.org/api_docs/python/tf/keras/applications)

- 事前に**imagenetデータセット**を学習させた「**学習済みモデル**」を配布しています。
  - 1400万枚以上もある大規模な、「カラー写真」の教師ラベル(1000クラス)付き画像データベース

tf-resnet-predict.py

job-resnet.sh

dataset/

resnet50_weights_*.h5

imagenet_class_index.json

test-images/

S3 TensorFlow：学習済みモデルを用いて推論

```
$ mkdir ~/.keras/models
$ cp dataset/*.h5  ~/.keras/models/
```

※自動でダウンロードされますが、ネットがつながっていないので手動で保存します。

S3 TensorFlow：学習済みモデルを用いて推論

```
$ mkdir ~/.keras/models
$ cp dataset/*.h5  ~/.keras/models/
```

※自動でダウンロードされますが、ネットがつながっていないので手動で保存します。

```
$ more tf-resnet-predict.py
```

```
# 読み込み
model = tensorflow.keras.models.ResNet50(weights="imagenet")
preds = model.predict(test_images)
```

```
$ qsub job-resnet.sh
```

今回のサンプルでは

5枚のテスト画像を分類した結果をCSVファイルに書き込んでいます。

`results.csv`が生成されます。

S4 PyTorch：サンプルの概要

```
$ cd 20211213_docker/pytorch-sample
```

Dataset：自前で集めた犬と猫の画像

Task：犬と猫の分類

Model：CNN

256x256

```
datasets
    |--- train
            |--- dogs
            |--- cats
    |--- validation
            |--- dogs
            |--- cats
```

S4 PyTorch：CNNサンプル用のジョブスクリプト

```
$ more job-ptcnn.sh

#!/bin/sh
#PBS -q ODT
#PBS -y 373
#PBS -l elapstim_req=0:05:00
#PBS --template=pytorch-1.4
cd $PBS_O_WORKDIR
python3 pt-cnn.py

$ qsub job-ptcnn.sh
```

ノード群のジョブクラス
Docker使用「ODT」

ハンズオン用に
今日だけ追加　–y 373

コンテナ指定

ジョブをスケジューラーへ投入

今回のサンプルでは
`mymodel.pth` ファイルが生成され
学習済みモデルの情報が保存されます。

アナウンス

- 自前のデータセット・コードをお持ちの方は実行してみてください。
- 質問やコメントのある方は「チャット」、また講習後はメールでお受けします。
- 1週間計算機を無料でご利用いただけます。
- 講習後に計算機を使用するさい、ジョブスクリプトの修正が必要です
  - #PBS -y 373　の行を削除