

# *Deep Learning* *Learning 2019*

**2<sup>nd</sup> season**

UBI (SAGA) M2 我妻正太郎

本勉強会の目標

ニューラルネットワークを使った  
機械学習の実装のやり方を知る

# 本日の予定

- Google Colaboratory のセットアップ
  - Google Colaboratoryの説明
  - GPU, Google Drive との連携
- Colaboratory上で機械学習
  - TensorflowとKerasの説明
  - Tensorflow + Keras での基本的な機械学習のやり方
  - 転移学習とファインチューニングの説明
  - ファインチューニングを用いて実戦的な課題に着手

# 注意事項

- アルゴリズムの細かい説明は省きます。あくまで全体の流れを知ることを念頭におきます
- 今日一発の説明で完全に理解しきる必要はあんまりないと思っています。いざ機械学習を使う、となった時にリファレンスがわりにしてもらえると幸いです
- 本日使うコード、ファイル等はGithub、ドライブ両方に上がっています。好きな方からダウンロードしてください
  - <https://github.com/iplab-ubiquitous/DeepLearningLearning2019>
  - [https://drive.google.com/drive/folders/1mS8uIDSXpzHr15yg7\\_xcZ0jkQh9YH08D?usp=sharing](https://drive.google.com/drive/folders/1mS8uIDSXpzHr15yg7_xcZ0jkQh9YH08D?usp=sharing)

# Google Colaboratory セットアップ

UBI (SAGA) M2 我妻正太郎

# Google Colaboratory (Colab) とは？

- Googleが提供しているクラウド上の機械学習用Jupyter Notebook環境
- 言語はPython
- 機械学習で使う基本的なライブラリがすでにインストールされている
  - Keras, scikit-learn, tensorflow 等
- **無料で強力なGPUが使える**
  - Tesla K80
  - 詳しく知りません. . . 調べてください
  - 右図を見ると我妻が研究で使用している gtx1080より処理が速い？
  - K80 : 628800円~ (Amazon)



# Colabを使うにあたって必要なもの

- Googleアカウント
  - これだけ

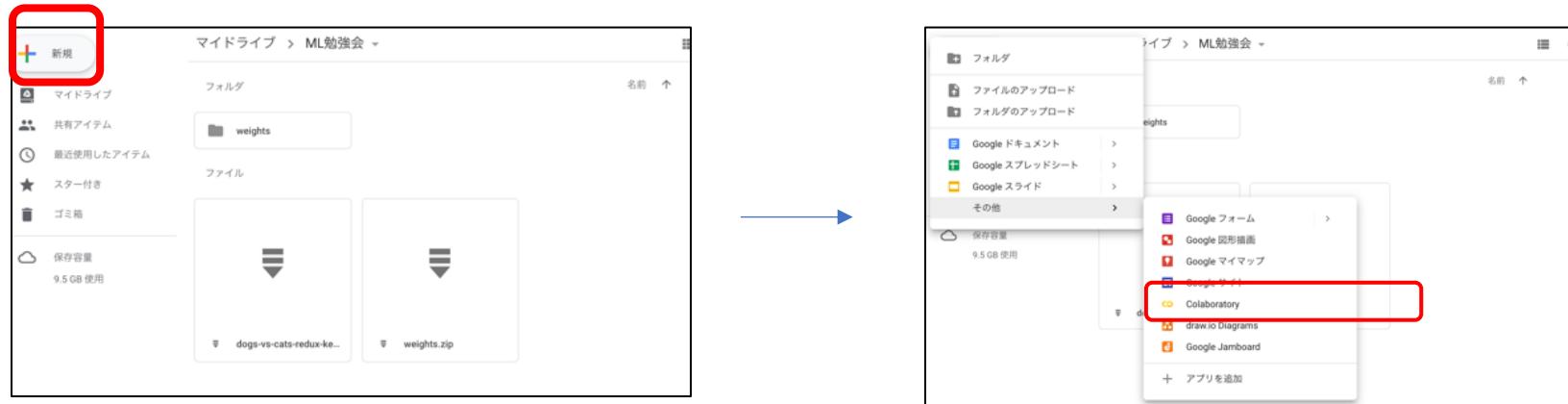
# Colabを使う準備 1/5

- 下記リンクへアクセス
  - <https://colab.research.google.com/>
- 下図の画面が出てくると思うので、下部の「新しいノートブック」からノートブック作成
  - ドライブのホームディレクトリ直下に「google colaboratory」というディレクトリができる、その下に作ったノートブックが格納される



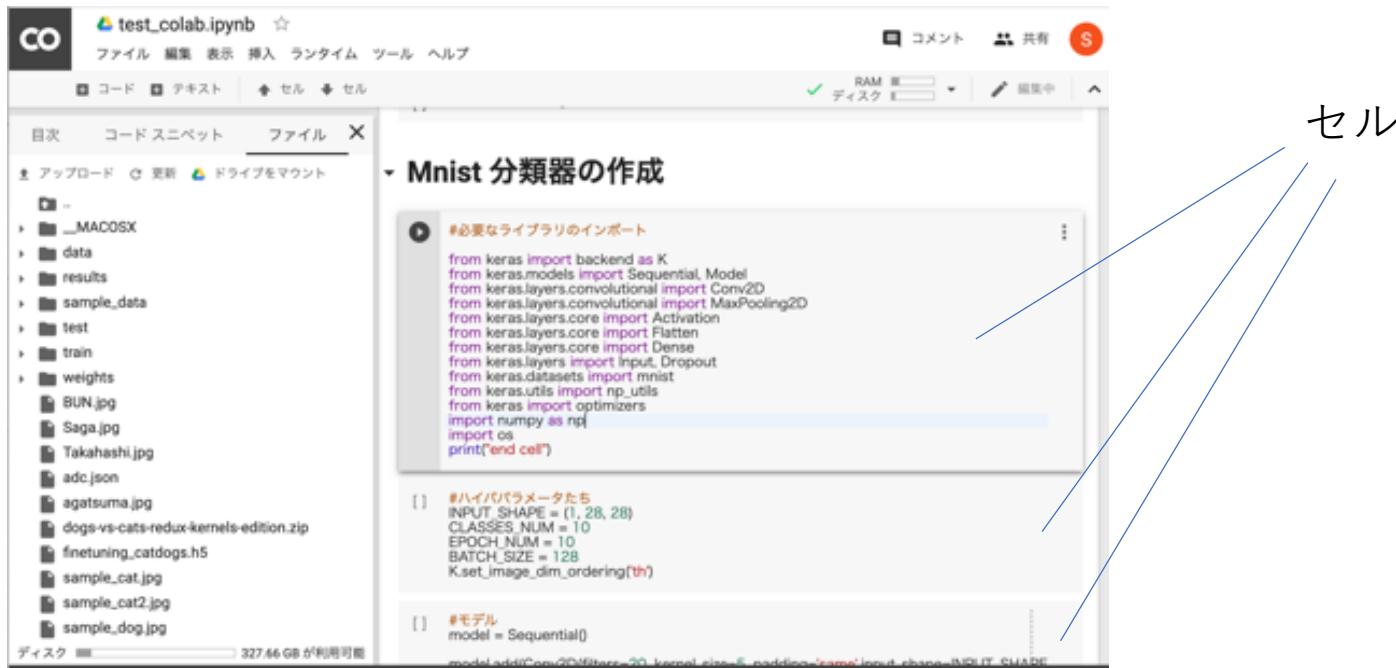
# Colabを使う準備 2/5

- ノートブック作成はドライブ側からも可能



# Colabを使う準備 3/5 Jupyterについて

- コードを分割して実行できる
  - 分割の単位：セル
  - 図左上「+コード」でセルを追加
  - セル左上の 再生ボタン (▶) でセルの中身を実行



# Colabを使う準備 4/5 GPU連携

- 左上「ランタイム」から「ランタイムのタイプを変更」へ
  - 「Python3」と「GPU」と設定



# Colabを使う準備 5/5 GPU確認

- セルを作成、下図のようにコードを入力し実行
  - 実行結果がセル下部に表示される
  - '/device:GPU:0'となればOK
  - これら辺のコードはgitにあがってます



# 学習用データのロード 1/3

- Google ドライブからデータを持ってこれる
- ドライブとノートブックを連携させる
  - Pydriveをクラウド環境に入れる
  - !pip install -U -q Pydrive

```
[62] #googledriveAPIとpythonを連携する PyDriveをcolab上にインストール  
#!をつかうとコマンドが使える  
!pip install -U -q PyDrive  
  
[63] #ログインしているアカウントとgoogleドライブを連携する  
from pydrive.auth import GoogleAuth  
from pydrive.drive import GoogleDrive  
from google.colab import auth  
from oauth2client.client import GoogleCredentials  
  
auth.authenticate_user()  
gauth = GoogleAuth()  
gauth.credentials = GoogleCredentials.get_application_default()  
drive = GoogleDrive(gauth)  
  
[] #ローカルからアップロードするとき用いる  
# from google.colab import files  
# uploaded = files.upload()  
  
[] #犬猫写真データをドライブからもってくる  
id = '1YacZJ9yYDmwew_FvOrHBNowC9nStDR' # 共有リンクで取得した id= より後の部分  
downloaded = drive.CreateFile({'id': id})  
downloaded.GetContentFile('dogs-vs-cats-redux-kernels-edition.zip')
```

# 学習用データのロード 2/3

- 下図赤枠内を実行すると、URLとフォームが出てくる
  - URLに飛ぶとgoogleアカウントログインフォームが出てくるので、ログイン
  - ログイン後、コードが出てくるので、フォームに入力
    - 連携完了



The screenshot shows the Google Colab interface with a red box highlighting the code for logging in to Google Drive. The code uses PyDrive to authenticate with a Google account.

```
#ログインしているアカウントとgoogleドライブを連携する
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials

auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)
```

... Go to this URL in a browser: <https://accounts.google.com/o/oauth2/auth>  
Enter your authorization code:

# 学習用データのロード 3/3

- 赤枠の場所より下のセルを「Mnist分類器の作成」まで全部実行
  - 我妻ドライブから各種データをzipファイルでロード
  - その後解凍
  - 解凍後のデータの一部を抜き出す。抜き出したデータをこの後の学習で用いる

The screenshot shows the Google Colab interface. On the left, there's a sidebar with a file tree containing various files and folders like '\_MACOSX', 'data', 'results', 'sample\_data', 'test', 'train', 'weights', 'BUN.jpg', 'Saga.jpg', 'Takahashi.jpg', 'adc.json', 'agatsuma.jpg', 'dogs-vs-cats-redux-kernels-edition.zip', 'finetuning\_catdogs.h5', and 'sample\_cat.jpg'. The main area has several code cells:

- [62] 

```
#googledriveAPIとpythonを連携する PyDriveをcolab上にインストール
# ! をつかうとコマンドが使える
!pip install -U -q PyDrive
```
- [63] 

```
#ログインしているアカウントとgoogleドライブを連携する
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials

auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)
```
- [ ] 

```
#ローカルからアップロードするとき用いる
# from google.colab import files
# uploaded = files.upload()
```
- [ ] 

```
#犬猫写真データをドライブからもってくる
id = '1YacZJ9yYDmwew_FivQlrHBNowC9nSfDR' # 共有リンクで取得した id= より後の部分
downloaded = drive.CreateFile({'id': id})
downloaded.GetContentFile('dogs-vs-cats-redux-kernels-edition.zip')
```

A red box highlights the last cell containing the code to download the 'dogs-vs-cats-redux-kernels-edition.zip' dataset.

ここから下

# コードしたデータのリセットについて

- Google Colaboratoryの制限によりデータが消える
  - 90分でセッション切断（実行中のセルが止まる、インポートしたライブラリがリセット）
  - 12時間でインスタンスリセット（ノートブック上のデータが消える、環境リセット）

# Tensorflow + Kerasで 画像分類

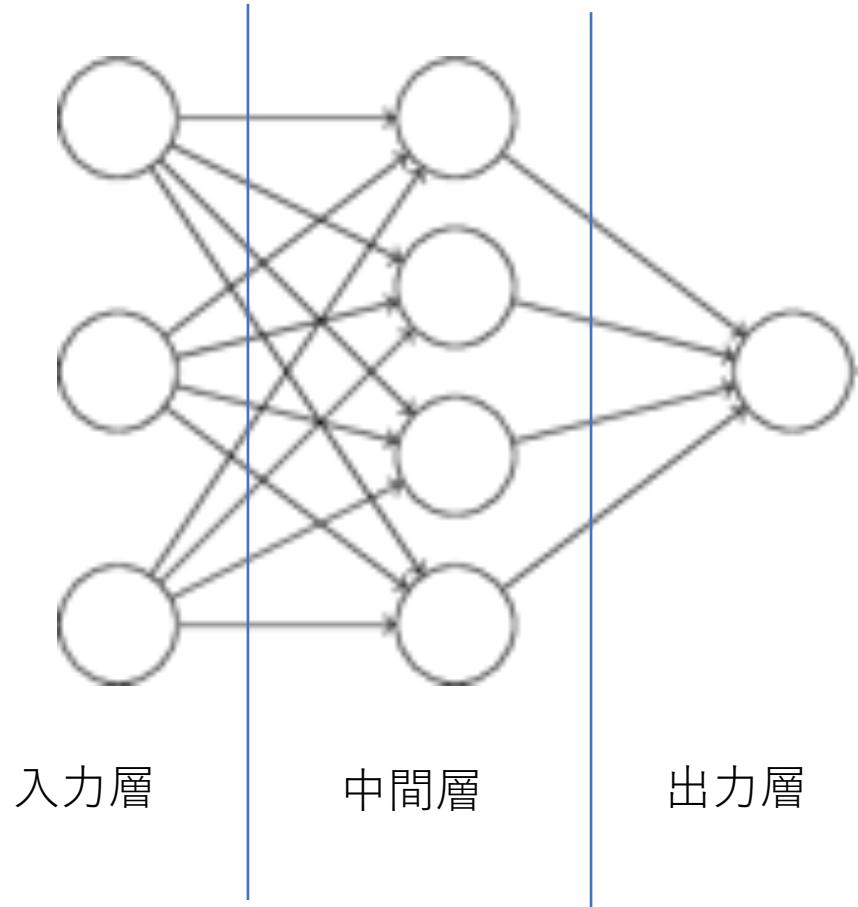
UBI (SAGA) M2 我妻正太郎

# Tensorflow? Keras?

- Tensorflow
  - Google作成の深層学習用機械学習フレームワーク
  - インストールが簡単
  - GPU連携も比較的簡単
  - 参考文献が豊富
- Keras
  - Google社員作成の深層学習用機械学習用フレームワーク
  - Tensorflow等をバックエンドとして用いることで動く
  - コーディングがめちゃくちゃ簡単
  - 処理が遅い
  - 拡張性に乏しい
    - 辎み入ったニューラルネットワークを作る際には不便

# ニューラルネットワークの復習

- 3種類の層
  - 入力層... データを入力
  - 中間層... データを推測
    - 複数存在する
    - 重みとバイアスを持つ
  - 出力層... 結果表示
- 出力層の値を損失関数に入力
- 損失関数の値をもとに各層の重みを更新、各層がより良い値を伝搬できるようになる



# Kerasの機械学習の基本的な流れ

- Sequentialモデルを用いた機械学習
  - 一番基本的なKerasのモデル
  - 下図Testにテストデータの分類結果が格納される

```
model = Sequential()
```

```
model.add(各種 ニューラルネットワーク層)
```

```
model.compile (損失関数, 重み更新手法, 評価関数等)
```

```
model.fit(訓練データ, ラベル, エポック等)
```

```
Test = model.predict(テストデータ)
```

# KerasでMNISTを分類 1/

- MNIST (Mixed National Institute of Standards and Technology database)
  - 手書き数字データセット
  - 0 ~ 9までの白黒手書き数字画像 (図参照)
  - 訓練用 60000枚, テスト用10000枚
  - 縦横 28 pixelの画像, データの形は (28, 28, 1)
- MNISTの分類モデルを作る
  - Convolutional Neural Network (CNN) を使う
- CNNについて
  - 中間層で畳み込み演算するニューラルネット
  - 畳み込み演算層と, データを縮小しつつ特徴抽出するプーリング層の2種類の層を持つ構成が多い



# KerasでMNISTを分類 2/

- test\_colab.ipynb 内 「Mnist分類器の作成」 参照
  - <https://github.com/iplab-ubiquitous/DeepLearningLearning2019>
  - [https://drive.google.com/open?id=1mS8uIDSXpzHr15yg7\\_xcZ0jkQh9YHo8D](https://drive.google.com/open?id=1mS8uIDSXpzHr15yg7_xcZ0jkQh9YHo8D)
  - Codes内
- 以下を実行、必要なライブラリをインポート
  - Mnist 分類器の作成

```
[ ] #必要なライブラリのインポート
from keras import backend as K
from keras.models import Sequential, Model
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.layers.core import Activation
from keras.layers.core import Flatten
from keras.layers.core import Dense
from keras.layers import Input, Dropout
from keras.datasets import mnist
from keras.utils import np_utils
from keras import optimizers
import numpy as np
import os
print("end cell")
```

```
[ ] #ハイパラメータたち
INPUT_SHAPE = (1, 28, 28)
CLASSES_NUM = 10
EPOCH_NUM = 10
BATCH_SIZE = 128
K.set_image_dim_ordering('th')
```

# KerasでMNISTを分類 3/

- モデルの作成
  - Conv2D : 置み込み層, Activation: 活性化関数, MaxPooling2D: プーリング層
  - Dense : 全結合層
- modelに層を一つ一つ add していく

```
[ ] #モデル
model = Sequential()

model.add(Conv2D(filters=20, kernel_size=5, padding='same', input_shape=INPUT_SHAPE))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

model.add( Conv2D(filters=50, kernel_size=5, padding='same'))
model.add(Activation('relu'))
model.add( MaxPooling2D(pool_size=(2, 2), strides=(2, 2)))

model.add(Flatten())
model.add(Dense(500))
model.add(Activation('relu'))

model.add(Dense(units=CLASSES_NUM))
model.add(Activation('softmax'))
```

# KerasでMNISTを分類 4/

- MNISTのロード
- ラベルをone hot vector化
  - 正解クラスに対応した場所だけ1, 他は0のベクトル
- モデルのcompileとfit
- 試しに訓練データから1つ抜いて predict
  - predictの戻り値の配列で, 一番大きな値を持つインデックスを分類結果とする
  - [0.1, 0.1, 0.1, 0.5] なら 分類結果は 3

```
[ ] #mnistのロードと画素の正規化
(x_train, y_train), (x_test, y_test) = mnist.load_data()

x_train = x_train.astype('float32')
x_test = x_test.astype('float32')
x_train /= 255
x_test /= 255

[ ] #cnnに入れられるよう軸を増やす
x_train = x_train[:, np.newaxis, :, :]
x_test = x_test[:, np.newaxis, :, :]

#ラベルをone-hot vector化
y_train = np_utils.to_categorical(
    y=y_train, num_classes=CLASSES_NUM)
y_test = np_utils.to_categorical(
    y=y_test, num_classes=CLASSES_NUM)

[ ] #モデルのコンパイルと学習
model.compile(
    loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

#verbose...データをどこまでログに出すか
#validation_split 検証データの割合
history = model.fit(
    x=x_train, y=y_train, batch_size=BATCH_SIZE,
    epochs=EPOCH_NUM, verbose=2, validation_split=0.2)

Test = x_train[0]
Test = Test[:, np.newaxis, :, :]

test = model.predict(Test)

print("Input: {}".format(np.argmax(y_train[0])))
print("Predict: {}".format(np.argmax(test)))
```

# KerasでMNISTを分類 5/

- 実行結果

- 検証データ正当率 98%
- テストで入れたデータの分類にも成功

```
↳ Train on 48000 samples, validate on 12000 samples
Epoch 1/10
- 4s - loss: 0.1858 - acc: 0.9445 - val_loss: 0.0579 - val_acc: 0.9828
Epoch 2/10
- 3s - loss: 0.0490 - acc: 0.9842 - val_loss: 0.0424 - val_acc: 0.9869
Epoch 3/10
- 3s - loss: 0.0323 - acc: 0.9895 - val_loss: 0.0374 - val_acc: 0.9889
Epoch 4/10
- 3s - loss: 0.0228 - acc: 0.9926 - val_loss: 0.0334 - val_acc: 0.9899
Epoch 5/10
- 3s - loss: 0.0175 - acc: 0.9945 - val_loss: 0.0344 - val_acc: 0.9897
Epoch 6/10
- 3s - loss: 0.0133 - acc: 0.9956 - val_loss: 0.0337 - val_acc: 0.9912
Epoch 7/10
- 3s - loss: 0.0098 - acc: 0.9968 - val_loss: 0.0416 - val_acc: 0.9897
Epoch 8/10
- 3s - loss: 0.0084 - acc: 0.9970 - val_loss: 0.0410 - val_acc: 0.9899
Epoch 9/10
- 3s - loss: 0.0078 - acc: 0.9975 - val_loss: 0.0397 - val_acc: 0.9902
Epoch 10/10
- 3s - loss: 0.0090 - acc: 0.9968 - val_loss: 0.0527 - val_acc: 0.9893
```

Input: 5

Predict: 5

# より実戦的な課題

- Dogs vs. Cats Redux: Kernels Edition
  - 犬と猫の写真を分類する (例: 下図)
  - Kaggleの課題の一つ, 犬猫写真データセットが公開されている
  - <https://www.kaggle.com/c/dogs-vs-cats-redux-kernels-edition>
- 先ほどのMNIST分類の考え方を応用して分類モデルを作る



# 犬猫分類 1/18

- まずは先ほどのMNISTとほぼ同じネットワーク構成で分類モデルを作ってみる
  - 畳み込み層2層 + 全結合層
- 訓練に使える写真数は 犬1000枚, 猫1000枚
  - 検証用で別口に犬400枚, 猫400枚ある
  - 本当はもっとあるけど, 今回勉強のためにあえて使いません. . .
- データ数が心もとないので水増しする
  - Data Augmentation
  - 画像を回転させたり, 反転させたり, サイズを変えたりして水増し

# 犬猫分類 2/18

- 以下「犬猫分類器作成」参照
- 前置き
  - 必要なライブラリのインポート
  - 正答率記録用関数の定義
  - 各種ハイパーパラメータ設定

```
[ ] #必要なライブラリのインポート
```

```
from keras import backend as K
from keras.applications.vgg16 import VGG16
from keras.models import Sequential, Model
from keras.layers.convolutional import Conv2D
from keras.layers.convolutional import MaxPooling2D
from keras.layers.core import Activation
from keras.layers.core import Flatten
from keras.layers.core import Dense
from keras.layers import Input, Dropout
from keras.datasets import mnist
from keras.utils import np_utils
from keras import optimizers
from keras.preprocessing.image import ImageDataGenerator
import numpy as np
import os
print("end cell")
```

```
[ ] #学習時のlossや検証データでの正答率を記録する関数
```

```
def save_history(history, result_file):
    print(history.history.keys())
    loss = history.history['loss']
    acc = history.history['acc']
    val_loss = history.history['val_loss']
    val_acc = history.history['val_acc']
    nb_epoch = len(acc)

    with open(result_file, "w") as fp:
        fp.write("epoch\tloss\tacc\tval_loss\tval_acc\n")
        for i in range(nb_epoch):
            fp.write("%d\t%.4f\t%.4f\t%.4f\t%.4f\n" % (i, loss[i], acc[i], val_loss[i], val_acc[i]))
    print("end cell")
```

```
#画像の大きさ、訓練データのパス、epoch数などを設定
```

```
img_width, img_height = 150, 150
train_data_dir = './data/train'
validation_data_dir = './data/validation'
nb_train_samples = 2000
nb_validation_samples = 800
nb_epochs = 20
result_dir = './results'
K.set_image_dim_ordering('tf')
print("end cell")
```

# 犬猫分類 3/18

- 自作モデル
  - 畳み込み層2つ
  - 全結合層2つ

```
[ ] #モデルの作成
model = Sequential()
model.add(Conv2D(32, 3, 3, input_shape=(150, 150, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(32, 3, 3))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(1))
model.add(Activation('sigmoid'))

model.compile(loss='binary_crossentropy',
              optimizer='adam',
              metrics=['accuracy'])
```

# 犬猫分類 4/18

- データを水増しする
- ImageDataGenerator
  - Kerasの関数
  - データセットをマイナーチェンジしたデータを無限に作成できる生成器を作成する
  - 引数でどうマイナーチェンジするか決める
- ImageDataGenerator(引数)で定義
- ImageDataGenerator.flow(引数)で生成器作成
- .flow\_from\_directory(引数)を使うと訓練データを入れた際、生成データにディレクトリ名に対応したラベルを自動でつけてくれる

```
# 訓練データとバリデーションデータを生成するジェネレータを作成
train_datagen = ImageDataGenerator(
    rescale=1.0 / 255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True)

test_datagen = ImageDataGenerator(rescale=1.0 / 255)

train_generator = train_datagen.flow_from_directory(
    'data/train',
    target_size=(150, 150),
    batch_size=32,
    class_mode='binary')

validation_generator = test_datagen.flow_from_directory(
    'data/validation',
    target_size=(150, 150),
    batch_size=32,
    class_mode='binary')

# 訓練
history = model.fit_generator(
    train_generator,
    samples_per_epoch=2000,
    nb_epoch=nb_epochs,
    validation_data=validation_generator,
    nb_val_samples=800)

# 結果を保存
model.save_weights(os.path.join(result_dir, 'smallcnn.h5'))
save_history(history, os.path.join(result_dir, 'history_smallcnn.txt'))
```

# 犬猫分類 5/18

- fitで訓練
  - fit\_generator(引数)を使うと、データ生成器を引数にとれる
- model.save\_weight(引数)で学習した重みを保存
  - 保存した重みをロードすることで、再学習する必要がなくなる

```
# 訓練データとバリデーションデータを生成するジェネレータを作成
train_datagen = ImageDataGenerator(
    rescale=1.0 / 255,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True)

test_datagen = ImageDataGenerator(rescale=1.0 / 255)

train_generator = train_datagen.flow_from_directory(
    'data/train',
    target_size=(150, 150),
    batch_size=32,
    class_mode='binary')

validation_generator = test_datagen.flow_from_directory(
    'data/validation',
    target_size=(150, 150),
    batch_size=32,
    class_mode='binary')

# 訓練
history = model.fit_generator(
    train_generator,
    samples_per_epoch=2000,
    nb_epoch=nb_epochs,
    validation_data=validation_generator,
    nb_val_samples=800)

# 結果を保存
model.save_weights(os.path.join(result_dir, 'smallcnn.h5'))
save_history(history, os.path.join(result_dir, 'history_smallcnn.txt'))
```

# 犬猫分類 6/18

- 「predict」以下参照
- 保存した重みをロードし、画像を分類してみる
  - モデルはまた定義する必要がある
  - モデルを設定したあと、
  - model.load\_weights(引数)で学習済みモデルをロード
  - model.compileでモデル定義

```
#モデルの作成
model = Sequential()
model.add(Conv2D(32, 3, 3, input_shape=(150, 150, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(32, 3, 3))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Flatten())
model.add(Dense(64))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(1))
model.add(Activation('sigmoid'))

#学習済みモデルをロード、コンパイル
model.load_weights(os.path.join(result_dir, 'smallcnn.h5'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

# 犬猫分類 7/18

- CNNに入るよう， 画像の形を調整
  - 軸増量
  - 値の正規化
- model.predict(引数)で分類
  - 今回2値分類なので， CNNの出力を1つにし， その値が高いか低いかで判定する
  - 多クラス分類したい場合は， CNNの出力を分類したいクラス数に合わせる必要がある

```
#学習済みモデルをロード， コンパイル
model.load_weights(os.path.join(result_dir, 'smallcnn.h5'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# 画像をパスを参照して読み込み， ついでにサイズを調整する
# cnnに入れるため， 軸を一つ増やし， 4次元テンソルへ変換
#(h,w,rgb) -> (batchsize, h, w, rgb) 今回は軸増やすだけなのでbatchsize = 1となる
img = image.load_img(filename, target_size = (img_height, img_width))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)

#今回のモデルではcnnに入れる際， rgbを正規化する必要がある
x = x/255.0

#画像表示
display_jpeg(Image(filename))

#推測， 2値分類なので， 0.5をボーダーとし， 高いか低いかで判定
pred = model.predict(x)[0]
print(pred)

if pred < 0.5:
    print("cat")
else:
    print("dog")
```

# 犬猫分類 8/18 自作モデルの結果

- 検証データで 70% の正答率(右上)
- あれ、間違ってる... (右下)

epoch	loss	acc	val_loss	val_acc	
0	0.734273	0.487399	0.693058	0.500000	
1	0.693898	0.501524	0.692865	0.507500	
2	0.692359	0.517276	0.690583	0.518750	
3	0.690902	0.522866	0.712995	0.497500	
4	0.694611	0.524390	0.685060	0.606250	
5	0.687836	0.559451	0.702571	0.528750	
6	0.687164	0.582825	0.675029	0.582500	
7	0.674429	0.600102	0.655965	0.636250	
8	0.655397	0.631606	0.637579	0.650000	
9	0.625763	0.664126	0.628863	0.645000	
10	0.628230	0.664126	0.619503	0.643750	
11	0.608627	0.671748	0.605727	0.673750	
12	0.600397	0.681911	0.611981	0.673750	
13	0.567298	0.715955	0.589204	0.681250	
14	0.558264	0.710874	0.583810	0.682500	
15	0.541181	0.737805	0.581200	0.716250	
16	0.536798	0.716972	0.570875	0.713750	
17	0.517351	0.735772	0.566282	0.698750	
18	0.511705	0.745427	0.590084	0.687500	
19	0.479260	0.775407	0.578250	0.707500	

```
↳ smallCNN  
input: sample_cat2.jpg  
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:2  
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py:3
```



[0.92423284]

dog

# 犬猫分類 9/18 改善策の検討

- パラメータチューニング
  - 下に考えうるパラメータを列挙する
  - epoch数, ミニバッチサイズ, 署み込みフィルタサイズ, フィルタ枚数, 署み込み総数, プーリング層数, 全結合総数, 全結合ノード数, Dropoutの有無 etc etc...
  - CNNを使うのが本当に正しいのか？
  - **結論：チューニングはめんどくさい**
- チューニングなしで解決できる策を考える
  - **解決したい課題と関連していて, かつ, うまくいっているモデルを流用する -> 転移学習**

# 転移学習とファインチューニング

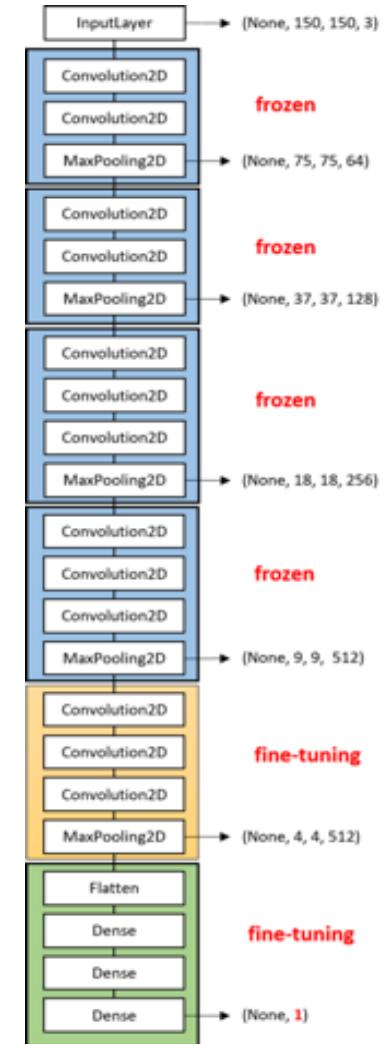
- 転移学習
  - 機械学習モデルを流用して問題解決を行う
  - イメージ：課題Aのために作られたモデルをうまくつかって課題Bの解決を試みる
- ファインチューニング
  - 機械学習モデルを流用したのち、解決したい課題に合わせて、流用したモデルを再学習する
  - イメージ：課題Aのために作られたモデルの重みを課題Bの訓練データを用いて更新、モデルを課題Bに適応させることで課題Bのより効果的な解決を試みる

# 犬猫分類 10/18 VGG16のファインチューニング

- VGG16
  - 畳み込み層とプーリング層合わせて16層のCNN
  - 数々の画像処理コンテストや研究で実績あり
  - 大型画像データセット, CIFAR-10を学習したモデルの重み, 構成がKerasに用意されている
- CIFAR-10を訓練データとして学習したVGG16をファインチューニングする
  - CIFAR-10には動物の画像も含まれており, 動物である犬と猫の特徴を効率的に抽出できる可能性が高い
  - ファインチューニングすることで, 犬と猫に特化させる

# 犬猫分類 11/18 VGG16のファインチューニング

- 具体的な学習の戦略
  - 最後の畳み込み層4つと全結合層のみを学習させる
  - 他の層は学習させず、あくまで特徴抽出器として用いる
- CNNにおいて、浅い層ではデータの大まかな形、深い層ではデータの本質的特徴を学習していることが知られている。深い層だけを学習することで、効率的にモデルを犬と猫に特化させる



# 犬猫分類 12/18

- 「VGG16のfinetuning」 参照
- KerasにあるVGG16の重みと構成を流用
  - VGG16(引数)
- Functional APIを用いてVGGと全結合層を連結
  - Model(input = X, output = Y)とすることで、X~Yのモデルの処理を全て繋げて行ってくれる
- model.layersで各層に干渉可能
  - layer.trainable = False とすると、その層の重みが更新されなくなる

```
#VGG16モデルをロード、ここは学習せず特徴抽出器として使う
#学習を凍結させる処理は後ろで
input_tensor = Input(shape=(img_height, img_width, 3))
vgg16_model = VGG16(include_top = False,
                     weights='imagenet',
                     input_tensor = input_tensor)

#全結合層構築
top_model = Sequential()
top_model.add(Flatten(input_shape = vgg16_model.output_shape[1:]))
top_model.add(Dense(256, activation='relu'))
top_model.add(Dropout(0.5))
top_model.add(Dense(1, activation='sigmoid'))

#下のようにすれば多クラス分類にも対応できる
# top_model.add(Dense(units=クラス数))
# top_model.add(Activation('softmax'))

#Functional APIを使って vggと全結合層をつなげる
model = Model(input = vgg16_model.input,
               output=top_model(vgg16_model.output))
print('vgg16_model:', vgg16_model)
print('top_model:', top_model)
print('model:', model)
model.summary()

for i in range(len(model.layers)):
    print(i, model.layers[i])

for layer in model.layers[:15]:
    layer.trainable = False

model.summary()

model.compile(loss='binary_crossentropy',
              optimizer = optimizers.SGD(lr=1e-4, momentum = 0.9),
              metrics=['accuracy'])
```

# 犬猫分類 13/18

- predict
  - 自作モデルの時と基本的には変わらない
  - モデルをVGGのものにする

```
#vgg16のロード
input_tensor = Input(shape=(img_height, img_width, channels))
vgg16_model = VGG16(include_top = False, weights = 'imagenet', input_tensor=input_tensor)

#全結合層作成
top_model = Sequential()
top_model.add(Flatten(input_shape=vgg16_model.output_shape[1:]))
top_model.add(Dense(256, activation='relu'))
top_model.add(Dropout(0.5))
top_model.add(Dense(1, activation='sigmoid'))

#vgg16と全結合層を結合
model = Model(input=vgg16_model.input, output=top_model(vgg16_model.output))

#学習済みモデルをロード、コンパイル、学習しないのでoptimizerは適当でも大丈夫
model.load_weights(os.path.join(result_dir, 'finetuning_catdogs.h5'))
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

# 犬猫分類 14/18 ファインチューニングモデルの結果

- 検証データで92%の正答率 (右上)
- 自作モデルでは分類に失敗した画像の分類に成功 (右下)
  - ファインチューニングモデルを用いることで精度向上

epoch	loss	acc	val_loss	val_acc	
0	0.696574	0.593750	0.568169	0.783750	
1	0.551266	0.719512	0.458301	0.832500	
2	0.466228	0.790650	0.384064	0.848750	
3	0.404674	0.828760	0.332787	0.871250	
4	0.359586	0.848069	0.313987	0.868750	
5	0.335172	0.855183	0.274269	0.888750	
6	0.296793	0.871443	0.259639	0.890000	
7	0.272452	0.890752	0.237843	0.897500	
8	0.259609	0.890752	0.224938	0.905000	
9	0.245346	0.902947	0.226028	0.910000	
10	0.233486	0.904472	0.228177	0.903750	
11	0.235288	0.904980	0.208487	0.911250	
12	0.209749	0.913618	0.209557	0.916250	
13	0.200764	0.918191	0.213814	0.912500	
14	0.204149	0.915650	0.193545	0.917500	
15	0.185214	0.930386	0.191613	0.922500	
16	0.186997	0.920224	0.210948	0.907500	
17	0.188235	0.926829	0.197535	0.920000	
18	0.183389	0.929370	0.182651	0.920000	
19	0.177619	0.934451	0.178173	0.926250	

```
↳ VGG16  
input: sample_cat2.jpg  
/usr/local/lib/python3.6/dist-packages/ipykernel_launcher.py
```



[0.10610417]  
cat

# 犬猫分類 15/18 成功例

- Google画像検索より抜粋

▷ VGG16  
input: sample\_cat5.jpg  
/usr/local/lib/python3.6/dist-packages/ipykernel\_launcher.py:37: UserWarning: Update yo...



[0.09025415]  
cat

▷ VGG16  
input: sample\_cat3.jpg  
/usr/local/lib/python3.6/dist-packages/ipykernel\_la...



[0.2929047]  
cat

▷ VGG16  
input: sample\_dog2.jpg  
/usr/local/lib/python3.6/dist-packages/ipyke...



[0.9120609]  
dog

▷ VGG16  
input: sample\_dog3.jpg  
/usr/local/lib/python3.6/dist-packages/ipy...



[0.98696786]  
dog

# 犬猫分類 16/18 成功例 2

- ・イラストもいける！

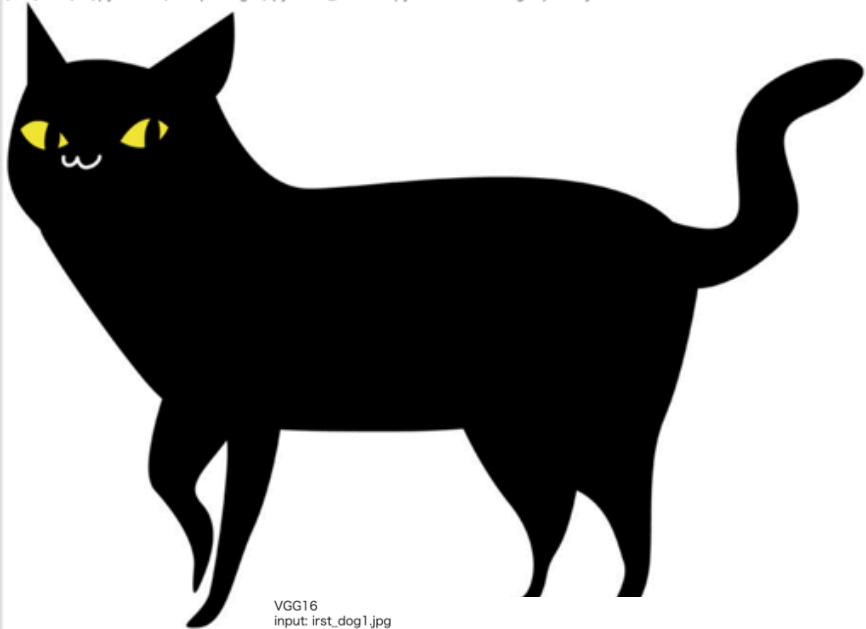
☞ VGG16  
input: irst\_cat3.jpeg  
/usr/local/lib/python3.6/dist-packages/ipykernel/launcher.py:37: UserWarning: Update your 'Model' call to the Keras 2 API: 'Modi



[0.0122739]  
cat



VGG16  
input: irst\_cat2.jpg  
/usr/local/lib/python3.6/dist-packages/ipykernel\_launcher.py:37: UserWarning: Update your 'Model' call to the Keras 2 API: 'Modi



[0.28969386]  
cat

VGG16  
input: irst\_dog1.jpg  
/usr/local/lib/python3.6/dist-packages/ipykernel\_launcher.py:37: UserWarning: U



[0.6310285]  
dog

# 犬猫分類 17/18 失敗例

- 顔の輪郭が関係している?
  - 正確に調べるには CAM (Class Activation Map) を使うと良いらしい
  - <https://qiita.com/daisukelab/items/381099590f22e4f9ab1f>

VGG16  
input: sample\_cat4.jpg  
/usr/local/lib/python3.6/dist-packages/ipykernel\_launcher.py:37: UserWarning: Update you



[0.98120904]  
dog

VGG16  
input: irst\_dog2.jpg  
/usr/local/lib/python3.6/dist-packages/ipykernel\_



[0.06462548]  
cat

# 犬猫分類? 18/18 おまけ

VGG16

input: agatsuma.jpg

/usr/local/lib/python3.6/dist-pa



[0.97795594]

dog

VGG16

input: Saga.jpg

/usr/local/lib/python3.6/dist-pa



[0.88497716]

dog

VGG16

input: BUN.jpg

/usr/local/lib/python3.6/dist-packages



[0.6124716]

dog

dog チーム

# 参考文献

- GithubのReadmeにまとめてあります
- 犬猫分類
  - <http://aidiary.hatenablog.com/entry/20170110/1484057655>
- 小さいデータセットで学習する時
  - <https://qiita.com/daisukelab/items/381099590f22e4f9ab1f>
- Keras+Tensorflowで転移学習を行う
  - <https://qiita.com/yampy/items/6f1f48fee16db7888f07>
- Google Colabで画像やcsvにアクセスするのが一苦労だった話
  - [https://qiita.com/yoshizaki\\_kkgk/items/bf01842d1a80c0f9e56c](https://qiita.com/yoshizaki_kkgk/items/bf01842d1a80c0f9e56c)
- Google Colabでファイルを読み込む方法
  - <https://qiita.com/uni-3/items/201aaa2708260cc790b8>

# まとめ(説明したこと)

- Google Colabotatory上のGPU環境を使用した
- Tensorflow + Kerasの基本的な使い方を試した
  - Sequential モデル
  - Sequential () -> model.add -> model.compile -> model.fit -> model.predict
- ファインチューニングでより実戦的な画像分類に挑戦した
  - 犬猫画像分類
  - VGG16のファインチューニングで精度92%達成
  - イラストも分類できるモデルができた

# おまけ Keras所感

- Sequentialモデルだけでは全体のモデル構築が難しい場合あり
  - ネットワークの結合 (GAN関連等)
  - 別の層の出力を学習に使う構成 (ResNet等)
  - 活性化関数の値の操作する場合 (WaveGAN等)
- Kerasはすでに作ってある層は豊富だが、Kerasにない層を作らなければならなくなつた時、融通が利かない
  - 作ってある層を組み合わせることを念頭においてる
  - 機械学習の構成を売りにする研究するのはKerasでは難しい
  - あくまで、手段として楽に機械学習したい人たち向け
    - 我々
- 実行速度も遅いので、学習に慣れたらTensorflow純に移行した方がいい、僕も頑張って勉強します