STATE FARM DISTRACTED DRIVER DETECTION

1

PRIVATE SCORE : 0.154

総合順位：14位/1438チーム(※)

※ 期間が終わった後の参加のため、あくまで参考

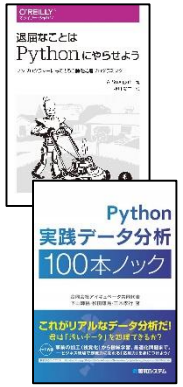
2020/10/7 中向

1. 背景

勉強してきたことを実践するために、Kaggleに挑戦してみました。
期間が終わっているコンペなのであくまで参考ですが、 14位/1438チームまで精度を上げることができたので、内容共有します。

勉強内容

2019/10 業務効率化のためPython勉強開始



2020/2 機械学習の勉強開始

2020/8 Deep Learningの勉強開始

2020/9 実践のため、Kaggleに挑戦

kaggle 統計家、データ分析家が分析手法を競い合う場所

2. テーマ

せっかくなので、運転関係のテーマに挑戦することにしました。

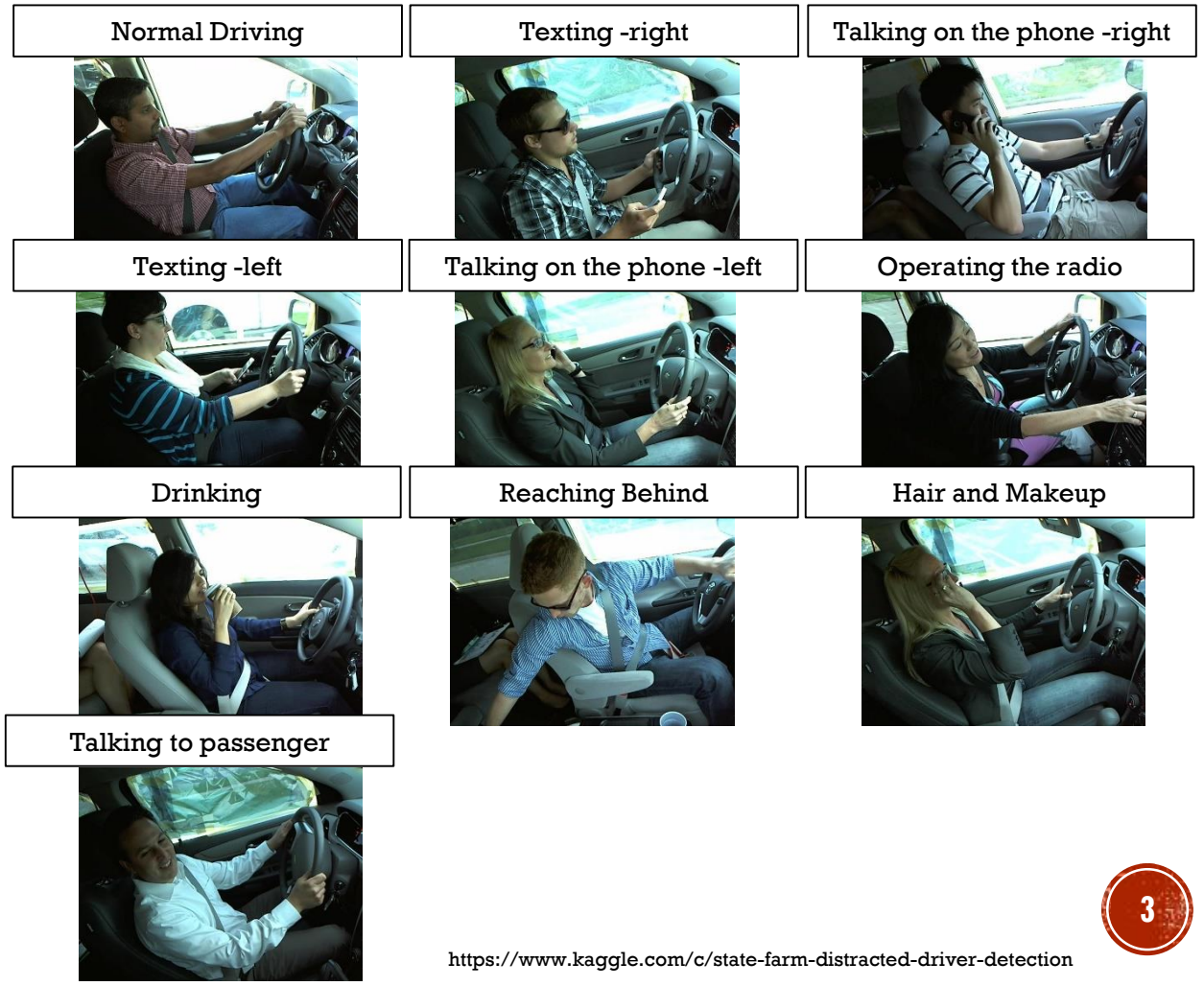
テーマ

運転手の状況を画像から識別する
10クラス分類問題に挑戦。

評価方法

Log lossで評価
(小さいほど正解率が高い)

$$logloss = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{M} y_{ij}\log(p_{ij}),$$



https://www.kaggle.com/c/state-farm-distracted-driver-detection

3.データの分割

情報のリークが発生すると…
Validationは精度よく判定できるのに、
Testを精度よく判定できなくなってしまう

最初ランダムに8:2でValidation/Trainを分割すると、情報のリークが発生。
人単位で分割することで改善しました。

Train Data
ラベルの付いている画像22,424枚
登場人物は26人

Normal Driving

Texting -right

Test Data
ラベルの付いていない画像 79,726枚

Validation Data
2人分のデータ（1,958枚)を使用

Train Data
24人分のデータ（20,466枚)を使用

4.データ拡張

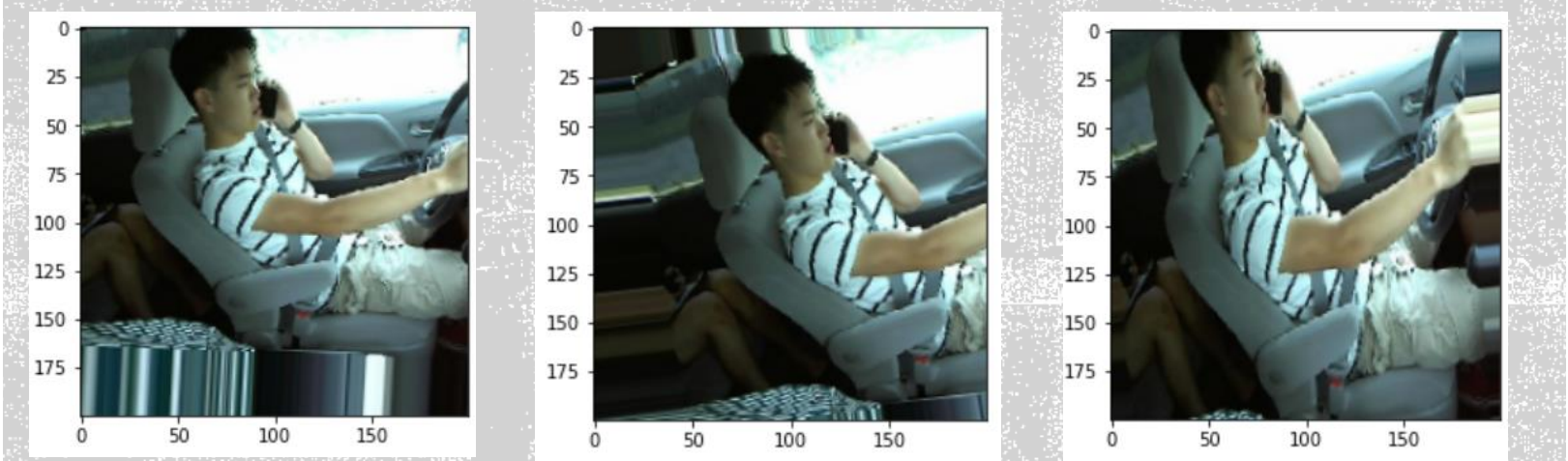
過学習対策として、Trainデータに対してデータ拡張を行いました。

```
#データ拡張を実施
train_datagen = ImageDataGenerator(rescale=1./255,
                                   rotation_range=5*2,
                                   width_shift_range=0.1*2,
                                   height_shift_range=0.1*2,
                                   shear_range=0.1*2,
                                   zoom_range=0.1*2,
                                   horizontal_flip=False,
                                   fill_mode='nearest')
```

正規化
回転角度
横方向の移動量
縦方向の移動量
ねじれ量
拡大量
左右反転要否

元画像

データ拡張



5.モデルの作成

VGG16の後ろに、DenseとSoftmaxを付けました。

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	(None, 200, 200, 3)	0
block1_conv1 (Conv2D)	(None, 200, 200, 64)	1792
block1_conv2 (Conv2D)	(None, 200, 200, 64)	36928
block1_pool (MaxPooling2D)	(None, 100, 100, 64)	0
block2_conv1 (Conv2D)	(None, 100, 100, 128)	73856
block2_conv2 (Conv2D)	(None, 100, 100, 128)	147584
block2_pool (MaxPooling2D)	(None, 50, 50, 128)	0
block3_conv1 (Conv2D)	(None, 50, 50, 256)	295168
block3_conv2 (Conv2D)	(None, 50, 50, 256)	590080
block3_conv3 (Conv2D)	(None, 50, 50, 256)	590080
block3_pool (MaxPooling2D)	(None, 25, 25, 256)	0
block4_conv1 (Conv2D)	(None, 25, 25, 512)	1180160
block4_conv2 (Conv2D)	(None, 25, 25, 512)	2359808
block4_conv3 (Conv2D)	(None, 25, 25, 512)	2359808
block4_pool (MaxPooling2D)	(None, 12, 12, 512)	0
block5_conv1 (Conv2D)	(None, 12, 12, 512)	2359808
block5_conv2 (Conv2D)	(None, 12, 12, 512)	2359808
block5_conv3 (Conv2D)	(None, 12, 12, 512)	2359808
block5_pool (MaxPooling2D)	(None, 6, 6, 512)	0
flatten_3 (Flatten)	(None, 18432)	0
dense_1 (Dense)	(None, 64)	1179712
dropout_5 (Dropout)	(None, 64)	0
dense_2 (Dense)	(None, 64)	4160
dropout_6 (Dropout)	(None, 64)	0
dense_3 (Dense)	(None, 10)	650

VGG16

Dense(64,dropout=0.5)

Dense(64,dropout=0.5)

Softmax(10)

モデルについて…

ResNetとかEfficientNetとかいろいろある。
一般的にはEfficientNetが精度が良いらしいが、
今回はVGG16の方が良さそうだったので選択。

6.モデルの学習

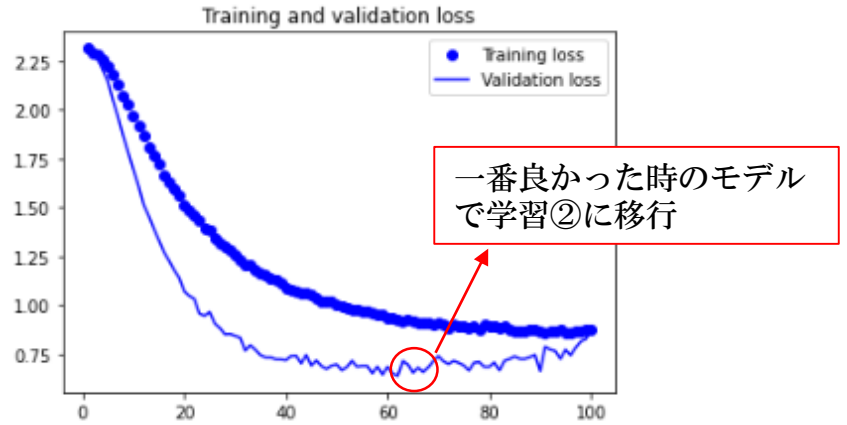
モデルは2段階に分けて学習しました。

Layer (type)	Output Shape	Param #
input_2 (InputLayer)	(None, 200, 200, 3)	0
block1_conv1 (Conv2D)	(None, 200, 200, 64)	1792
block1_conv2 (Conv2D)	(None, 200, 200, 64)	36928
block1_pool (MaxPooling2D)	(None, 100, 100, 64)	0
block2_conv1 (Conv2D)	(None, 100, 100, 128)	73856
block2_conv2 (Conv2D)	(None, 100, 100, 128)	147584
block2_pool (MaxPooling2D)	(None, 50, 50, 128)	0
block3_conv1 (Conv2D)	(None, 50, 50, 256)	295168
block3_conv2 (Conv2D)	(None, 50, 50, 256)	590080
block3_conv3 (Conv2D)	(None, 50, 50, 256)	590080
block3_pool (MaxPooling2D)	(None, 25, 25, 256)	0
block4_conv1 (Conv2D)	(None, 25, 25, 512)	1180160
block4_conv2 (Conv2D)	(None, 25, 25, 512)	2359808
block4_conv3 (Conv2D)	(None, 25, 25, 512)	2359808
block4_pool (MaxPooling2D)	(None, 12, 12, 512)	0
block5_conv1 (Conv2D)	(None, 12, 12, 512)	2359808
block5_conv2 (Conv2D)	(None, 12, 12, 512)	2359808
block5_conv3 (Conv2D)	(None, 12, 12, 512)	2359808
block5_pool (MaxPooling2D)	(None, 6, 6, 512)	0
flatten_3 (Flatten)	(None, 18432)	0
dense_1 (Dense)	(None, 64)	1179712
dropout_5 (Dropout)	(None, 64)	0
dense_2 (Dense)	(None, 64)	4160
dropout_6 (Dropout)	(None, 64)	0
dense_3 (Dense)	(None, 10)	650

②：全て学習

①：Flatten以降のみ学習

学習①

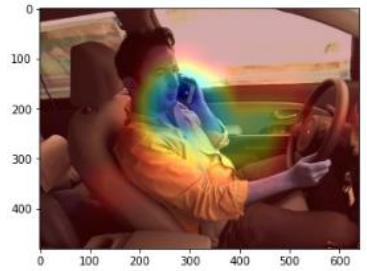


学習②

7.GRAD CAMによる結果の確認

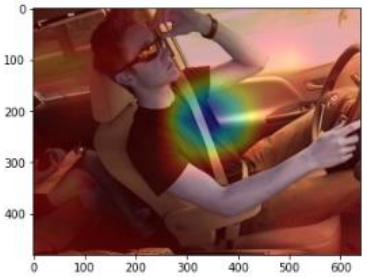
Deep Learningがどこを見て判断したかを可視化しました。
まだ改善の余地があることがわかります。

```
from keras import backend as K
import cv2

def gradcam(x,predicted_class):
    driver_output=model.output[:,predicted_class]

    last_conv_layer=model.get_layer('block5_conv3')

    g = tf.Graph()
    with g.as_default():
        grads = tf.gradients(driver_output,last_conv_layer.output)[0]

    pooled_grads=K.mean(grads,axis=(0,1,2))

    iterate=K.function([model.input],[pooled_grads,last_conv_layer.output[0]])

    pooled_grads_value,conv_layer_output_value=iterate([x])

    for i in range(512):
        conv_layer_output_value[:,:,i]*=pooled_grads_value[i]

    heatmap=np.mean(conv_layer_output_value,axis=-1)

    #GradCamの結果を表示
    heatmap=np.maximum(heatmap,0)
    heatmap/=np.max(heatmap)
#     plt.matshow(heatmap)
#     plt.show()

    #元の画像に重ね合わせる
    img=cv2.imread(img_path)
    heatmap=cv2.resize(heatmap,(img.shape[1],img.shape[0]))

    heatmap=np.uint8(255*heatmap)
    heatmap=cv2.applyColorMap(heatmap,cv2.COLORMAP_JET)

    superimposed_img=heatmap*0.5+img
    plt.imshow(superimposed_img/superimposed_img.max())
```



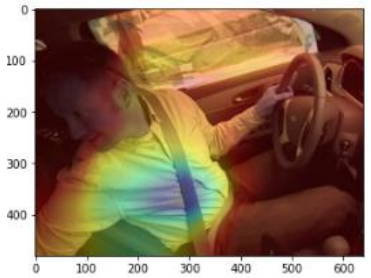
予測：Talking on the phone –left
結果：正解

ただしく携帯電話を見て判断してくれている



予測：Reaching Behind
結果：不正解 (正解は hair and make up)

シートベルトが見えたら、
Reaching Behindと判断してしまっている…。



← 姿勢ではなくベルトで判断している。

8.結果の確認、対策の立案

ここまでやって結果を提出したら、Scoreが0.45でした。
ここから、 Score 0.154 まで精度を向上させていきます。

対策

(1) 疑似ラベルを活用して、学習データを増やす
(2) 姿勢を検出する
(3) 顔の向きを計算する

9.疑似ラベルの追加

Private Score ： 0.45

Private Score ： 0.39

疑似ラベルを追加することで、学習データを増やしました。
これにより、Scoreが0.39まで向上しました。

半教師あり学習

$$\alpha(t) = \begin{cases} 0 & t < T_1 \\ \frac{t-T_1}{T_2-T_1}\alpha_f & T_1 \leq t < T_2 \\ \alpha_f & T_2 \leq t \end{cases}$$

$$\alpha_f = 3, T_1 = 10, T_2 = 70$$

(Step1) 作成したモデルでTest Dataを予測し、
ラベルを付ける
(Step2) Test Dataの重みを0.1×epochに更新して学習
※ただし重みが3となったら、重みは更新しない
(Step3) Step1,2を繰り返す

最初は**Train Data**のみを学習、
徐々に**Test Data**の重みが増えていき、
最終的には**Test Data**の重みが3倍で学習していく
(weight_colという引数で重みを指定できます)

https://qiita.com/koshian2/items/f4a458466b15bb91c7cb

10.姿勢の検出(POSENET)

Private Score ： 0.39

Private Score ： 0.31

姿勢に注目させるために、PoseNetによる姿勢検出を行いました。
これにより、Scoreが0.31まで向上しました。

```
from chainer.serializers import npz
from chainer.links.caffe.caffe_function import CaffeFunction

caffemodel = CaffeFunction('pose_iter_440000.caffemodel')
npz.save_npz('coco_posenet.npz', caffemodel, compression=False)

%matplotlib inline

import os
import sys

import cv2
import matplotlib.pyplot as plt
import matplotlib.patches as mpatches
import numpy as np

# モジュール検索パスの設定
REPO_ROOT = '..'
sys.path.append(REPO_ROOT)

# PoseDetectorクラスのインポート
from pose_detector import PoseDetector

# モデルのロード
arch_name = 'posenet'
image_path = os.path.join(REPO_ROOT, 'data', 'person.png')
weight_path = 'coco_posenet.npz'

model = PoseDetector(arch_name, weight_path)
```

Train Data, Test Data合わせて
102,150枚に対してPoseNetを適用

※1枚10秒ほどかかるので、283時間かかる

こんな画像を作って、VGG16で学習

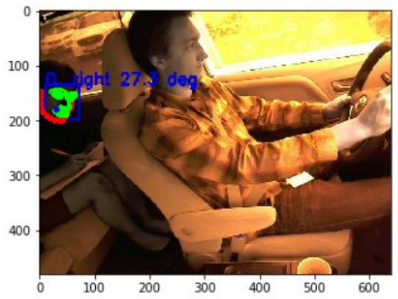
11.顔検出(DLIB)

顔検出を行い、顔の輪郭の重心と顔のパーツの重心の差から顔の角度を計算しました。

Train Data, Test Data合わせて
102,150枚に対して顔角度を計算

※1枚3秒ほどかかるので、85時間かかる

新たな特徴量として顔の角度を抽出



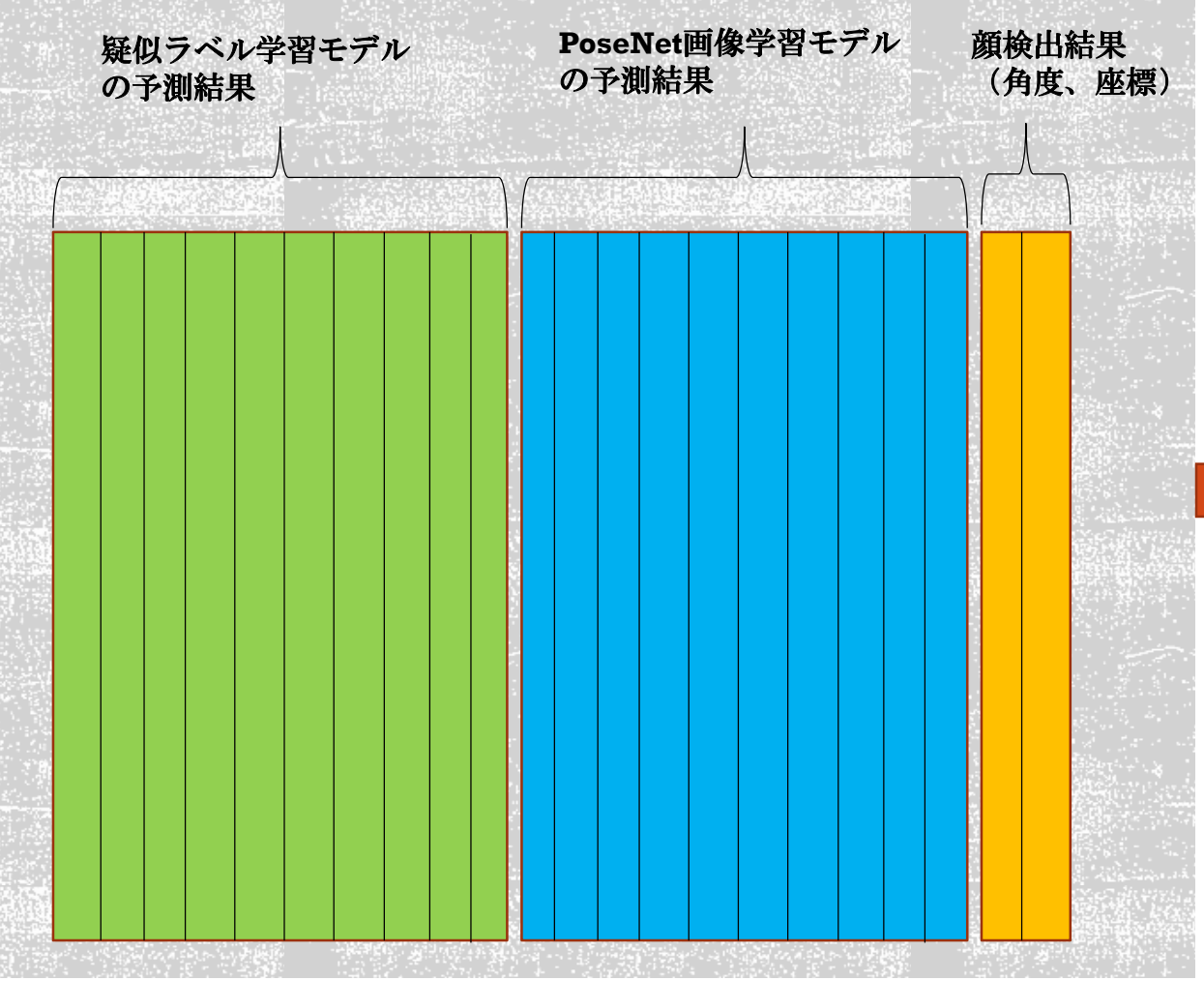
※関係ないものに反応する場合があるので、顔の座標、大きさ　等で異常値をはじく必要がある

12.スタッキング

Private Score ： 0.31

Private Score ： 0.25

Kaggleでは定番のスタッキングを行いました。



Tableデータとして
XGBoost等で学習

13.まとめ

Private Score ： 0.25

Private Score ： 0.154

スタッキングした後に、アンサンブルを加えることで精度が向上しました。

画像

(1)ディープラーニング
VGG16+ファインチューニング (a)
Val loss : 0.35
Private score : 0.45

(2)疑似ラベル (b)
Val loss : 0.26
Private score :0.39

(3)姿勢検出 (PoseNet)

(4)ディープラーニング
VGG16+ファインチューニング
Val loss : 0.08
Private score : 0.36 (c)
Val loss : 0.37
Private score :0.31 (d)
efficientnetB0+ファインチューニング
Val loss : 0.37
Private score :0.31 (e)

(5)顔検出 (dlib)

・顔の角度
・顔の座標
・顔の幅

(5)スタッキング

・(b)の予測結果
・(c)の予測結果
・(d)の予測結果
・顔の角度
・顔の座標
・顔の幅

(5)サポートベクターマシン
Val loss : 0.15
Private score :0.25 (f)

(5)ニューラルネットワーク
Val loss :0.14
Private score :0.28 (g)

(5)ランダムフォレスト
Val loss : 0.17
Private score :0.25 (h)

(5)勾配ブースティング (XGBoost)
Val loss : 0.16
Private score :0.27 (i)

(6)アンサンブル
(f)(g)(h)(i)(d)(e)の予測結果の平均
Private score :0.154 (j)