



STATE FARM DISTRACTED DRIVER DETECTION



PRIVATE SCORE : 0.154

総合順位 : 14位 / 1438チーム
社内順位 : 3位 / 16チーム

1. テーマ

kaggle

統計家、データ分析家が
分析手法を競い合う場所

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)

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

Normal Driving



Texting -right



Talking on the phone -right



Texting -left



Talking on the phone -left



Operating the radio



Drinking



Reaching Behind



Hair and Makeup



Talking to passenger



2. チーム PQI ; PROCESS QUALITY INNOVATION

名前	中向 (リーダー)	瀧 (期待の星)	岸川 (沈黙の反逆児)	山村 (人志 松本)
写真				
所属	データ分析推進Grp	データ分析推進Grp	データ分析企画Grp	データ分析推進Grp
AI歴	AI歴1年 Python講座 講師 機械学習講座 講師	AI歴1年 Python講座 講師 機械学習講座 講師 機械学習道場 師範代	AI歴2年 事務系AI講座 担当 機械学習道場 師範代	AI歴2年 機械学習道場 師範 Deep Learners代表

3. データの分割

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

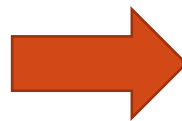


Test Data

ラベルの付いていない画像 79,726枚



最初ランダムに8:2に分割したら
情報のリークが発生
⇒人で分割することに



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

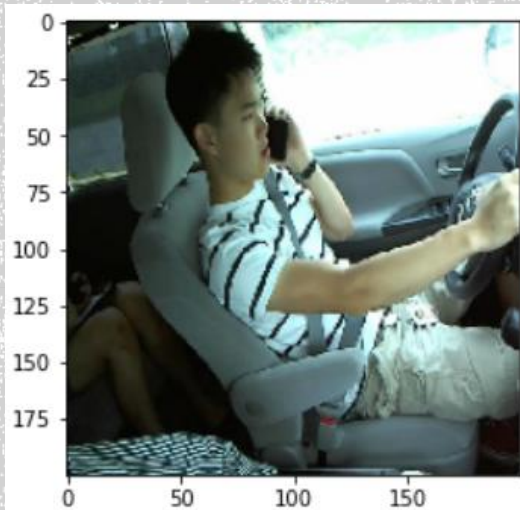
過学習対策としてデータ拡張を実施

#データ拡張を実施

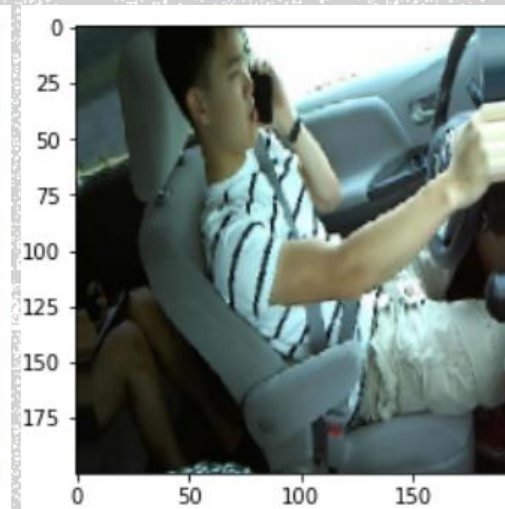
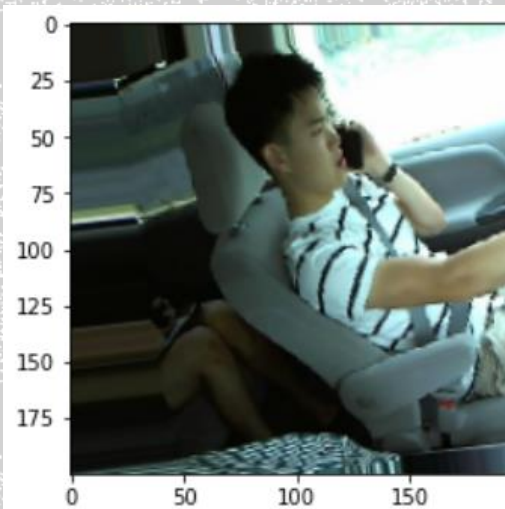
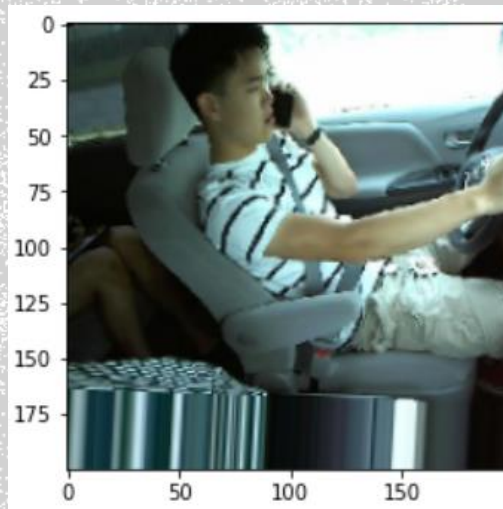
```
train_datagen = ImageDataGenerator(rescale=1./255,  
                                   rotation_range=5*2,  
                                   width_shift_range=0.1*2,  
                                   height_shift_range=0.1*2,  
                                   shear_range=0.1*2,  
                                   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
拡大率
左右反転可否

元画像



データ拡張



5.モデルの作成

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

Flatten

Dense(64,dropout=0.5)

Dense(64,dropout=0.5)

Softmax(10)

いろいろ試した結果、
VGG16を
ファインチューニング
することに決定

6. モデルの学習

モデルは2段階に分けて学習
(少しずつ解凍したほうがいいらしい)

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

Flatten
Dense
Dense
Softmax

学習②

学習①

学習①



学習②



7. GRAD CAMによる結果の確認

結果を提出したら...
Private Score : 0.45

Deep Learningがどこを見て判断したかを可視化

改善の余地がないか確認

```
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]

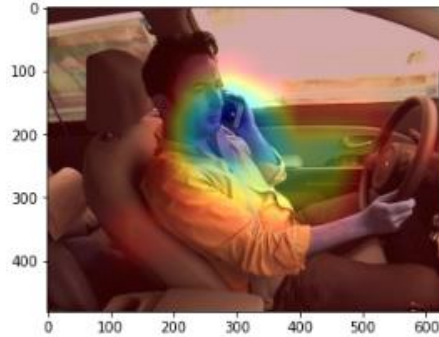
    heatmap=np.mean(conv_layer_output_value,axis=-1)

    #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]))

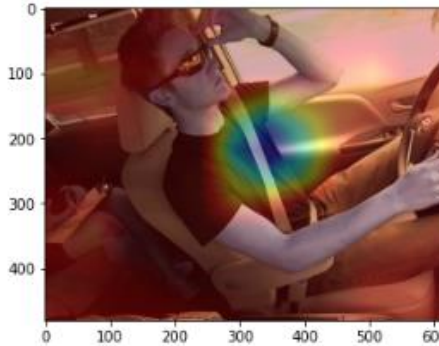
    heatmap=np.uint8(255*heatmap)
    heatmap=cv2.applyColorMap(heatmap,cv2.COLORMAP_JET)

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



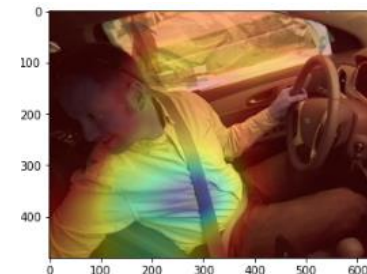
予測 : Talking on the phone –left
結果 : **正解**

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.対策の立案

- (対策1) 疑似ラベルを活用して、学習データを増やす
- (対策2) 姿勢を検出する
- (対策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

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 $0.1 \times \text{epoch}$ 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)

10. 姿勢の検出(POSENET)

Private Score : 0.39



Private Score : 0.31

```
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



新しい画像でモデルを
1から作って学習

11.顔検出(DLIB : SHAPE PREDICTOR 68 FACE LANDMARKS)

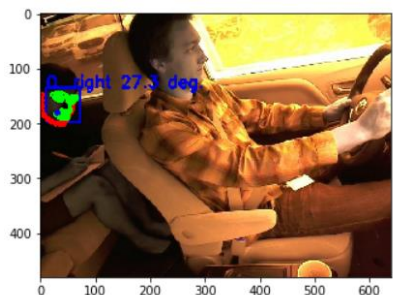
顔の輪郭の重心と顔のパーツの重心の差分から顔の角度を計算



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

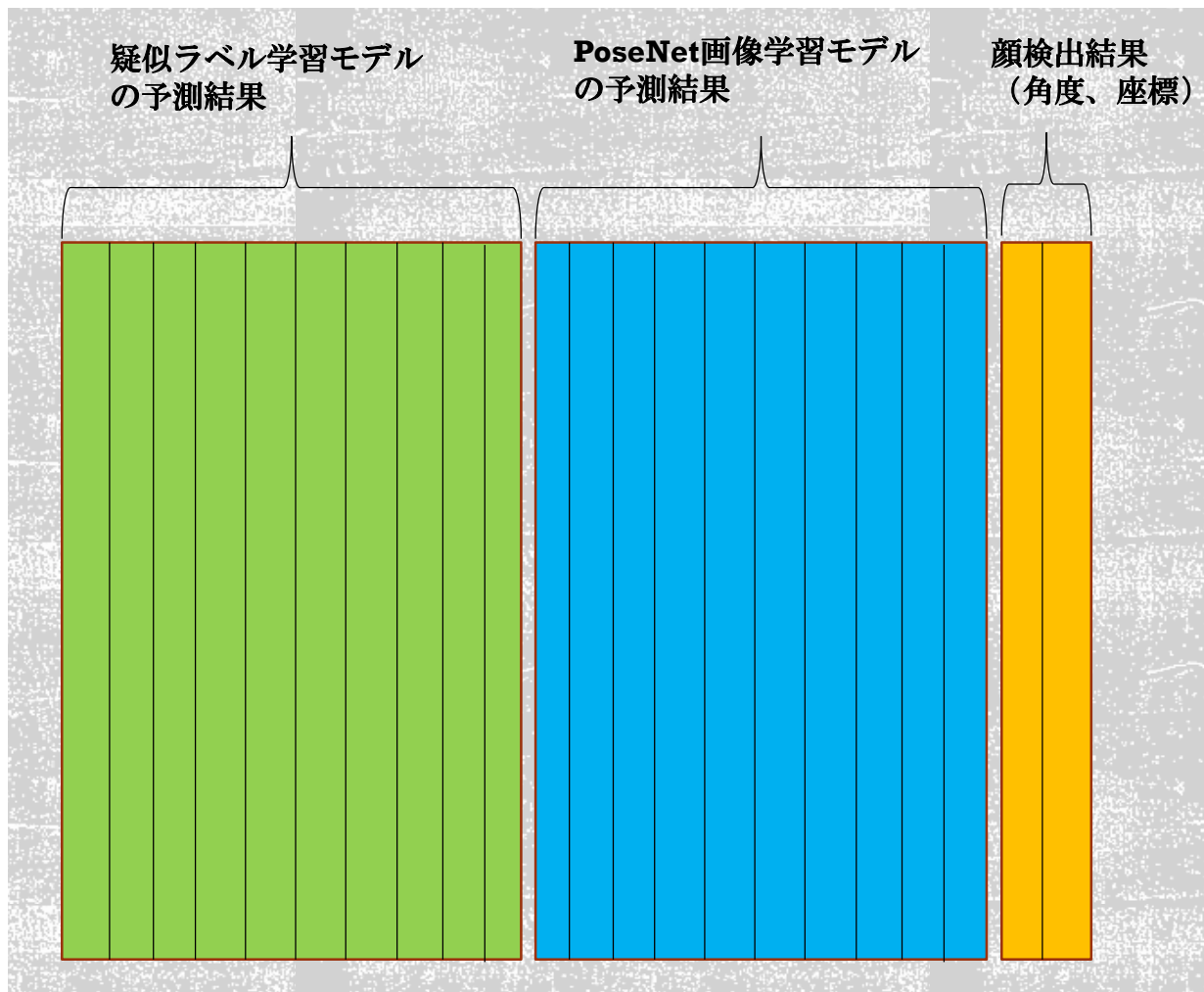
12.スタッキング

Kaggleでよく使われているスタッキングを適用

Private Score : 0.31



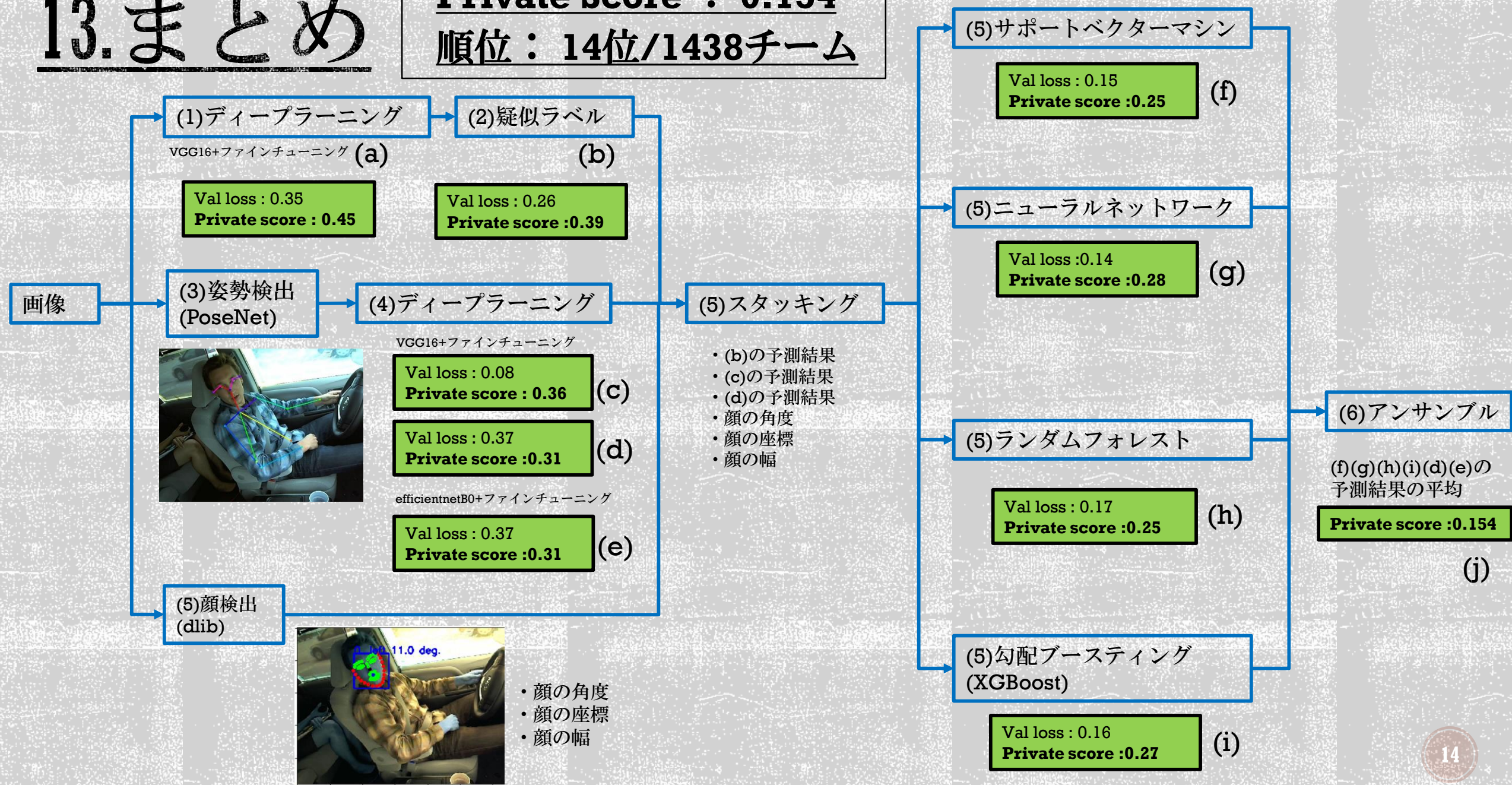
Private Score : 0.25



Tableデータとして
XGBoost等で学習

13.まとめ

Private Score : 0.154
順位 : 14位/1438チーム



14.参考文献

