State Farm Driver Distraction Detection Kaggle Challenge

MOD006567

Year:2020/21
SSID:2021497

Contents

1. Motivation and Introduction 3
2. Literature review 3-5
3. Dataset and Image Preprocessing 5-6
   3.1Dataset 5
   3.2Data augmentation 6
4. Methodology 6-10
   4.1 Model Evaluation 10
5. Results 11
6. Conclusion and Future work 12

References 13

Appendix: GitHub link for project 15

1.Introduction

Driver distraction is accepted globally as the primary reason for accidents observed in national highways and busy streets. It has stood out to be a concern for policing authorities to manage and detect drivers that are behind the cause of such accidents. Since driving requires impeccable motor skills and focus on the road, any distraction leads a vehicle off road or becomes the source of accidents which would cause issues for other vehicle drivers. Distraction of drivers is considered to fall in three categories where mental distraction, visual and manual distraction plays a critical role. The need to avoid distraction by detecting and alerting the concerned driver is the sole intention of this report. Moreover, the need to test the result with machine algorithm has been one of the motivating factors to pursue this report where log loss results helped in generating probabilities and determine the performance of algorithmic approach

Research questions

1. What is the result of a log loss function which can help in detecting distractions?
2. What is the probability of the next action for every image which is considered in the algorithmic simulation?
3. What are the architectures that can be used in performing machine learning algorithms?

Objectives

1.To identify the result of log loss function for detecting distractions

2. To estimate the probability of next action of image collected from state farm data simulation

3. To determine architectures that can be utilized for performing the algorithm

Aim

Our goal is to create a better accuracy model that can detect whether a driver is driving safely or distractedly. Photographs of the driver taken in the taxi serve as input for our model. We employ multiple SoftMax classifiers to predict the sort of distracted behaviour that drivers are participating in after preprocessing these photos to produce input vectors.

2. Literature Review

Some of the papers including Single, Ensemble of CNNs, and semi supervised models are presented here. It is found that CNNs are better solution for this type of classification.

In the State Farm Distracted Driver Detection Kaggle challenge, (Liuming Zhao, 2017) employed several Convolutional Neural Network (CNN) models to identify driving distraction, including basic CNN, VGG-16, and Inception-v4. The graph below summarises each of our models' overall quality as well as their Kaggle scores. The public score of 0.5067 was achieved with an ensemble of VGG-16, KNN, and InceptionV4 as final model.

Table 1. Model performance summary

Models	Validation Accuracy	Validation Loss	Kaggle Score
FC	11.4%	6.1	6.8
Basic CNN	74.38%	1.2081	1.32
VGG-16	85.01%	0.4954	0.64
VGG-16 + Inception-v4 +KNN	88.65%	0.4521	0.50

Table source: Liuming Zhao in his paper "*Research and Assessment of Driving Distraction Classification Methods"* compared the above mentioned models and log loss score in kaggle challenge.

A deep learning model for detecting distractions is presented by Koesdwiady A et al., (2017). The proposed architecture uses VGG-19 and pre-trained convolutional neural networks to extract features. Despite the Kaggle dataset's variances in illumination, camera angle, driver race, and gender. With VGG19, they attained the greatest test accuracy of 95% and an average validation accuracy of 80% per class.

Anthony D et al., (2019) conducted similar research with 21 algorithms, with the Random Forest method surpassing the others. They have used simulator to create a dataset with the help of 78 drivers. Some of the other experimented supervised machine learning algorithms are Naives, SVM with a linear kernel and radial kernel function, KNN, and neural networks.

On the State Farm Distracted Driver Detection Kaggle challenge, Prof. Pramila M. Chawan et al. (2018) tested multiple Convolutional Neural Network (CNN) models for categorization of distracted drivers, including Small CNN, VGG16, VGG19, and Inception. With a final log loss of 0.795, they constructed their best ensemble by averaging the chances provided by VGG-16, VGG-19, and InceptionV3.

CNN-based models are shown to have a high level of accuracy. Semi-supervised learning is another important strategy. Liu et al., (2016), the use of semi-supervised learning is the third primary strategy. Because the datasets obtained in this problem are likely to contain a large number of unlabeled images, using semi-supervised learning increases the amount of data available for schooling and may improve prediction performance. They have suggested a semi-supervised model with the inclusion of unlabeled data outperformed typical supervised methods in terms of detection.

W. Kim et.al., 2017 compared two single convolutional neural networks comparing the computational time for 3 devices (desktop, Laptop, and GPU), models depth and time for prediction, optimizer used, predicted scores required for similar problem. For preprocessing they

have used random flipping, cropping, adjustment in their models. Concluded with MobileNet as their final model for better performance on test data when compared to InceptionResNetV2 model.

The goal of this research is to create a machine learning system that can identify and categorise various distracted driving conditions in automobiles. Deep convolutional neural networks are the primary strategy (CNNs). Using the GPU given by Kaggle, we explored with several CNN architectures and exploited pre-trained networks (learning transfer) using VGG16, ResNet50, and EFficientNetB3. We were able to get a loss score of 0.4540 using Data Preprocessing methods such as Image Processing and Data Augmentation.

3. Dataset and Image Preprocessing

3.1 Dataset

The Kaggle website provided a dataset with a driver doing ten various tasks in the vehicle, such as texting, eating, chatting on the phone, applying cosmetics, reaching behind the wheel, and so on. The data collection includes 28 photos of drivers engaging in various distractions, which are divided into two categories: train and test. There are 79725 test shots and 22423 train images in all.

These ten classes are used to categorise all of the training photos in the train folder. These category names are encoded to numerical values from 0 to 9 using label encoding technique.

Each picture is 640 x 480 pixels in size. Each picture is preprocessed using cv2 after being imported into a dataframe. The input picture is scaled and flattened to a vector depending on the model. The input size for the EfficientNetB3 model, is 224x 224. All of the class names have been given to the photographs in the appropriate categories.

After splitting the train data set, There are now 17939 pictures in the train dataset and 4485 in the validation dataset, since the dataset is framed around 28 drivers.

3.2 Data Augmentation

We added extra variance to the supplied data by augmenting it. We spin and translate the infrared pictures at random within certain parameters. This means that each frame in a sequence is rotated and read in the same way. Movement between frames will appear when there is no genuine movement if the frames are not enhanced in the same manner. To increase the temporal dimension of the sequences, frames at certain time steps are duplicated or discarded, delaying or speeding up

the portions of the series. Data augmentation is applied both on train and validation data but not on test data.

The following settings for data augmentation on train data were selected:

```
train_gen = ImageDataGenerator(
    width_shift_range=0.3,
    height_shift_range=0.3,
    shear_range=0.3,
    zoom_range=0.4,
    validation_split=0.2)
```

4. Methodology

Following the implementation of Basic CNN which got 99% accuracy on train data and performed poor on test data, we chose these three architectures to extract features and model from the pre-trained models available to achieve better public score in Kaggle competition.

1. VGG16 model
2. ResNet50
3. EfficientNetB3

We chose these three to train on our dataset based on their performance. Each classifier was tested on 4485 photos after being trained on 17939 samples.

In this notebook, we utilized the [ImageNet dataset]-trained models known as pre-trained models as a feature extractor, these models when they are loaded acts as base models. and its weights into memory to transform images into bottleneck features. Below are the code snippets for loading 3 architectures.

```
model = VGG16(include_top=False)
model.summary()
```

```
Downloading data from https://storage.googleapis.com/tensorflow/keras-applications/vgg16/vgg16_weights_tf_dim_or
58892288/58889256 [==============================] - 1s 0us/step
Model: "vgg16"
_________________________________________________________________
```

```
base_model  = tf.keras.applications.resnet.ResNet50(include_top = False,
                                                     weights = 'imagenet',
                                                     input_shape = model_input_shape)
base_model.summary()
```

```
inputs = Input(shape=(224,224,3))
base_model = EfficientNetB3(include_top=False, weights='imagenet')(inputs)
```

Using these three pretained model, base models are loaded.

This will combine all of the retrieved characteristics into a single picture forecast. We assembled all of our models using Adam optimizer, categorical cross entropy as gradient descent, and accuracy as measurements after modifying a few settings.

We have used both sequential and functional models in defining models.

1. To define VGG16 model, we have applied sequential model technique with global average pooling layer.

```
VGG16_model = Sequential()
VGG16_model.add(GlobalAveragePooling2D(input_shape=train_features.shape))
VGG16_model.add(Dense(10, activation='softmax', kernel_initializer='glorot_normal'))

VGG16_model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
global_average_pooling2d (Gl (None, 512)               0
_________________________________________________________________
dense (Dense)                (None, 10)                5130
=================================================================
Total params: 5,130
Trainable params: 5,130
Non-trainable params: 0
_________________________________________________________________
```

2. For Resnet50 model, we have tried to implement our model using function API of keras which is more flexible and better control over the layers when compared to sequential API. Below figure shows the layers, activation functions used to define the model.

```
#model 2
x = base_model.output
x = tf.keras.layers.Flatten()(x)
x = tf.keras.layers.Dropout(0.5)(x)
x = tf.keras.layers.BatchNormalization()(x)
x = tf.keras.layers.Dense(units = 512, activation = tf.nn.relu)(x)
x = tf.keras.layers.Dense(units = 256, activation = tf.nn.relu)(x)
x = tf.keras.layers.Dropout(0.25)(x)

output =tf.keras.layers.Dense(units = len(class_names),activation = tf.nn.softmax)(x)
model = tf.keras.models.Model(inputs=base_model.inputs, outputs=output)

model.compile(optimizer=tf.keras.optimizers.Adam(0.0001),
              loss=tf.keras.losses.CategoricalCrossentropy(from_logits = False),
              metrics=['accuracy'])

model.summary()
```

Relu is used for hidden layers, whereas softmax is used for output layer.

3. For Our final Pretrained architecture, we have used functions to call the model return output, but the model definition is same as Resnet50. Input shape of image for EfficientNetB3 is 224 x 224 x 3 matrix. Layers defined for classifier are shown in the figure below along with inputs, output, activations.

```
def define_model(num_classes):
    inputs = Input(shape=(224,224,3))
    base_model = EfficientNetB3(include_top=False, weights='imagenet')(inputs)
    x = GlobalAveragePooling2D()(base_model)
    x = BatchNormalization()(x)
    x = Dropout(0.2)(x)
    output = Dense(units=num_classes, activation='softmax')(x)

    model = Model(inputs=inputs, outputs=output)
    model.compile(optimizer=tf.optimizers.Adam(learning_rate=1e-4),
                  loss='categorical_crossentropy',
                 metrics=['accuracy'])
    return model

model = define_model(10)
model.summary()
```

```
Downloading data from https://storage.googleapis.com/keras-applications/efficientnetb3_notop.h5
43941888/43941136 [==============================] - 0s 0us/step
Model: "model"
```

Each model is trained with different epochs based on the performance in training and validation. We have used two regularization techniques while training our models. One is early stopping, and the other is Learning rate annealer. Below figure shows the functioning of early stopping. And Learning rate annealer is to train the model with better learning rate by adapting according to training model and optimizer.

```
Epoch 9/20
561/561 [==============================] - 431s 769ms/step - loss: 0.0403 - accuracy: 0.9866 - val_loss: 0.0249 - val_accuracy: 0.9915
Epoch 10/20
561/561 [==============================] - 431s 769ms/step - loss: 0.0435 - accuracy: 0.9862 - val_loss: 0.0377 - val_accuracy: 0.9868
Epoch 11/20
561/561 [==============================] - 433s 770ms/step - loss: 0.0379 - accuracy: 0.9880 - val_loss: 0.0262 - val_accuracy: 0.9917
Epoch 12/20
561/561 [==============================] - 432s 770ms/step - loss: 0.0311 - accuracy: 0.9888 - val_loss: 0.0358 - val_accuracy: 0.9893
Epoch 13/20
561/561 [==============================] - 438s 781ms/step - loss: 0.0339 - accuracy: 0.9890 - val_loss: 0.0401 - val_accuracy: 0.9886
Epoch 14/20
561/561 [==============================] - 433s 771ms/step - loss: 0.0284 - accuracy: 0.9900 - val_loss: 0.0309 - val_accuracy: 0.9920
```

Comparison between models

The drawback with this simple VGG16 architecture is that it takes a long time to learn and uses a lot of memory. We come up with higher accuracy by varying the depth from 16–19 layers. Despite the fact that EfficientNetB3 required a long time to train, it provided us with a higher test score. RESNet50 was faster in training than these two models, but we chose EfficientNetB3 because of its performance on test data.

These are the accuracy, loss, and public scores for individual classification model using validation data.

Model	Training accuracy	Validation Accuracy	Kaggle submission score
Basic CNN	99.10	99.50	1.0376
VGG16	95.40	90.97	2.1437
Resnet50	85.16	49.06	3.0974
EfficientNetB3	98,43	98.20	0.4540
EfficientNetB3 after tuning hyper parameters	99.00	0.9920	0.18492

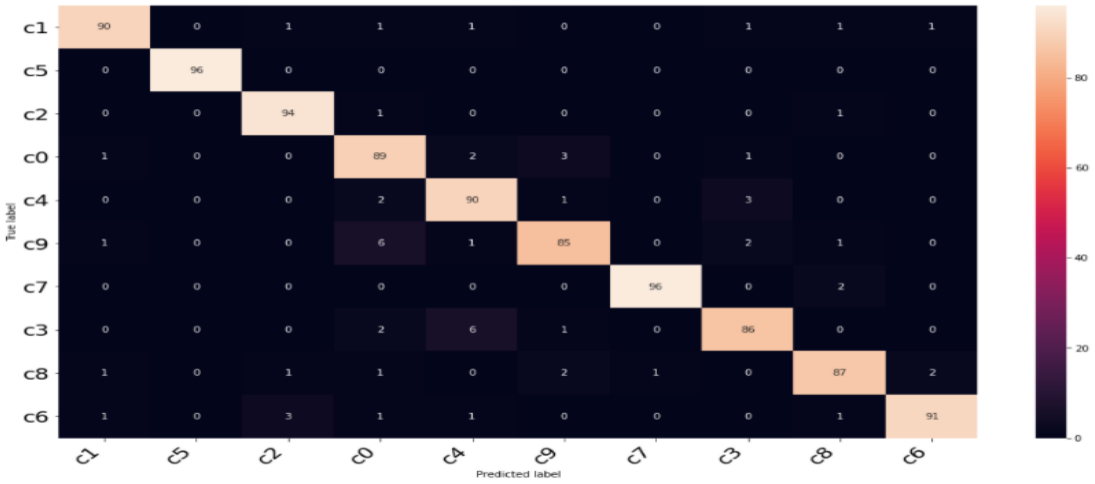
Fig. comparing models based on test score.

4. Models Evaluation

We have used these evaluation metrics for VGG16 model to measure the quality and variance of model.

1. Confusion matrix – to summarize the predicted results

```
print_heatmap(ytest,ypred,class_names)
```



From the above figure we can anlayse that model made more mistakes while predicting c9 and c0, c3 and c4 classes.

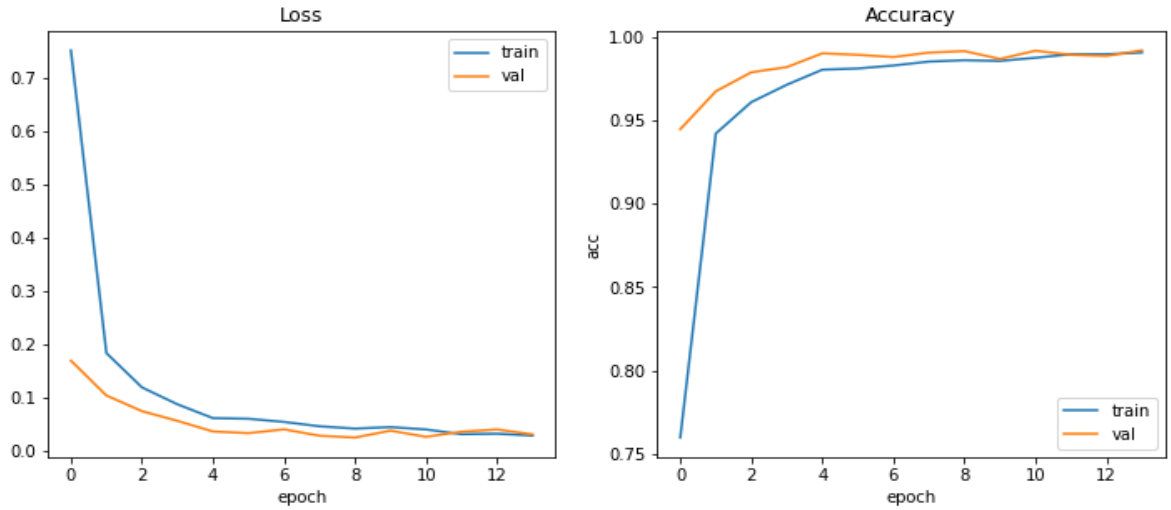
2. Accuracy, Precision, Recall, and F1 score – all these values are calculated for all models to know that error. Below code shows the error metrics calculated for our final model.

```
accuracy = accuracy_score(ytest,ypred_class)
print('Accuracy: %f' % accuracy)
# precision tp / (tp + fp)
precision = precision_score(ytest, ypred_class,average='weighted')
print('Precision: %f' % precision)
# recall: tp / (tp + fn)
recall = recall_score(ytest,ypred_class,average='weighted')
print('Recall: %f' % recall)
# f1: 2 tp / (2 tp + fp + fn)
f1 = f1_score(ytest,ypred_class,average='weighted')
print('F1 score: %f' % f1)
```

```
Accuracy: 0.909699
Precision: 0.910446
Recall: 0.909699
F1 score: 0.909820
```

Testing the model with validation data:

Learning graphs shows the difference between training and validation loss and accuracy for EFficientNetB3 model.



These are the graphs for EfficientNetB3 model showing that model is performing well in both datasets. It is inferred that model is not overfit.

5.Results

In our tests, the CNN Network model with 4 convolutional layers with some layers had the greatest validation accuracy of 99.5 percent, which matched our prediction that CNN-based models would outperform others. EfficientNetB3 was the next model to perform well enough on validation data, with 99.2 percent accuracy. But the log score after submission in Kaggle shows that EfficientNetB3 performed well on test data.

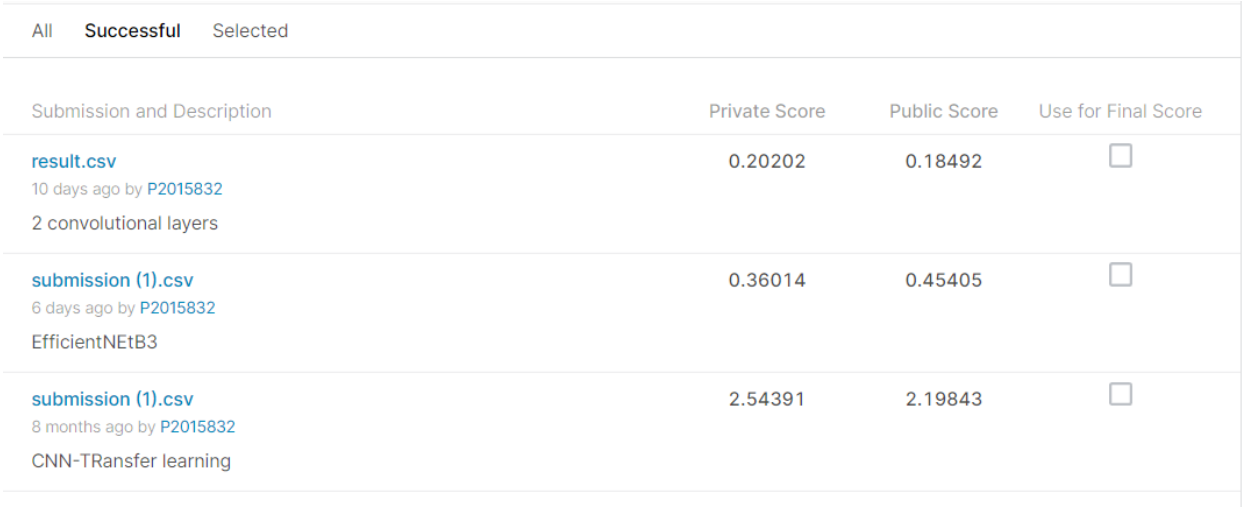
Predicted image:

```
print(predicted)
new_img = cv2.resize(testing_data[1001][1],(img_size2,img_size1))
plt.imshow(new_img,cmap='gray')
plt.show()
```

```
talking on the phone - left
```

Kaggle submission:

These are the submissions we have made, and our least log score was 0.18492. As the competition was ended, we have calculated based on the scores in the leaderboard. Our score is 48 rank out of 1438 teams.



All Successful Selected

Submission and Description	Private Score	Public Score	Use for Final Score
result.csv 10 days ago by P2015832 2 convolutional layers	0.20202	0.18492	☐
submission (1).csv 6 days ago by P2015832 EfficientNEtB3	0.36014	0.45405	☐
submission (1).csv 8 months ago by P2015832 CNN-TRansfer learning	2.54391	2.19843	☐

7. Conclusion and Future Work

Finally, we were able to effectively develop a deep learning model that performed very well on the distracted driver detection challenge. In addition, the effect of data augmentation, drop out layer, Batch normalization on the performance of the architecture is studied. Although there is chance to use ensemble models, we did not observe much difference than with single Convolutional neural networks. More complex models took lot of training and test time. While training out models in GPU, we have observed that basic CNN and VGG16, and ResNet50 are faster than EfficientNetB3 model.

We have learnt lot of machine learning models from this competition, we have got least score of 0.18 with single convolutional layers. Machine learning techniques may be used in the future to aid in the understanding of the self-driving operation. ML algorithms can also be fueled with data which could take charge of the steering wheel under situations where the driver is in a compromised position. This process would not only help in promoting safe driving but also reduce the loss of lives from careless driving.

References

Anthony D. McDonald, Thomas K. Ferris, Tyler A. Wiener. 2019. *Classification of Driver Distraction: A Comprehensive Analysis of Feature Generation, Machine Learning, and Input Measures*, first published at June 25, 2019. [e-journal] Available at: <https://doi.org/10.1177/0018720819856454>

Koesdwiady A., Bedawi S.M., Ou C., Karray F. 2017. *End-to-End Deep Learning for Driver Distraction Recognition*. In: Karray F., Campilho A., Cheriet F. (eds) Image Analysis and Recognition. ICIAR. Lecture Notes in Computer Science, vol 10317. Springer, Cham. Available at: <https://doi.org/10.1007/978-3-319-59876-5_2>

Liuming Zhao, Taiming Zhang, Lingzi Guo, 2017. "*Classification Models of Driving Distraction: Analysis and Comparison"*. (online), Available at: <http://cs229.stanford.edu/proj2017/final-reports/5241763.pdf>

Prof. Pramila M. Chawan, Shreyas Satardekar et al., April, 2018. *"Distracted Driver Detection and Classification",*Int. Journal of Engineering Research and Application ISSN : 2248-9622, Vol. 8, Issue4 (Part -III), pp60-64, Available at: < https://www.academia.edu/36964710/Distracted_Driver_Detection_and_Classification>

T. Liu, Y. Yang, G. Huang, Y. Yeo, and Z. Lin, 2016. "*Driver Distraction Detection Using Semi-Supervised Machine Learning"*, IEEE Transactions on Intelligent Transportation Systems, [e-journal] 17(4), 1108-1120; doi: 10.1109/TITS.2015.2496157

W. Kim, H. -K. Choi, B. -T. Jang and J. Lim, 2017. *Driver distraction detection using single convolutional neural network*, 2017 International Conference on Information andCommunication Technology Convergence (ICTC), pp. 1203-1205, doi: 10.1109/ICTC.2017.8190898.

Appendix

GitHub link for project:

https://github.com/devabhakthiniprathyusha/State-Farm-Driver-distraction-detection.git