Computer Vision Capstone Project

Distracted Driver Posture Classification

Ali Attaran

Contents

1 Definition

1.1 Project Overview

The Center for Disease Control and Prevention (CDC) found that nearly one in five vehicle accidents are caused due to distracted driving [1]. This statistics has led to more than 3,000 fatal injuries and 425,000 injuries every year in the USA. This project is taken from a Kaggle competition [2] and the purpose of this project is to classify the action of the drivers which they are doing in the provided images and whether they are distracted or not[3]. This is actually a multi-class classification problem for this image recognition problem.

The major cause of these accidents was the use of mobile phones. The National Highway Traffic Safety Administration (NHTSA) defines distracted driving as “any activity that diverts attention from driving”, including: a) talking or texting on one’s phone, b) eating and drinking, c) talking to passengers, or d) fiddling with the stereo, entertainment, or navigation system [4]. The CDC provides a broader definition of distracted driving by taking into account visual (i.e. taking one’s eyes off the road), manual (i.e. taking one’s hands off the driving wheel) and cognitive (i.e. taking one’s mind off driving) causes [5]. State Farm created a computer vision competition on Kaggle, a platform that provides data science projects and company sponsored competitions. The company is challenging competitors to classify the driver’s behavior.

1.2 Problem Statement

Many publications have used pre-trained CNN models and the most popular are VGG-19 and ResNet, which were state-of-the-art one or two years ago and had been improving. They most widely used models are nearest neighbors (KNN), Ensemble, K and data augmentation [6]. First, because the test set size is about four times as large as the training set. I will explore different CNN models with fine tuning their parameters to find the highest accuracy. I will explore CNN with and without dropout layers, VGG16, InceptionV3 with different epochs and batch sizes. Many top participants reported that apart from VGG and GoogleNet, ResNet achieves promising performance for this challenge [7] [8] [9] [10] [11].

This project is sponsored by State Farm on Kaggle website. State Farm aims to reduce these alarming statistics, and better insure their customers, by detecting the driver’s distracting activity from the dashboard cameras. Given a dataset of 2D dashboard camera images, State Farm is challenging Kagglers to classify each driver's behavior [2]. Are they driving attentively, wearing their seatbelt, or taking a selfie with their friends in the backseat?
The goal of this project is to implement and train a classifier that can predict the likelihood of what the driver is doing in each picture with an accuracy of over 90%. Using computer vision and deep learning algorithms, I will give each image scores with range [0-1] for each behavior listed above. To classify these images, I have read several articles and reports on this Kaggle competition and most of them used verities of CNN and KNN pre-trained models [12] [13]. I plan to experiment on several CNN models and fine tune their parameters to achieve as high accuracy as possible.

The work in the distracted driver detection field over the past seven years could be clustered into four groups: multiple independent cell-phone usage detection publications, Southeast University Distracted Driver dataset. Laboratory of Intelligent and Safe Automobiles in University of California San Diego (UCSD) datasets and publications, and affiliated publications, and recently, State Farm’s Distracted Driver Kaggle competition [3].

1.3 Metrics

1.3.1 Accuracy and loss – history callback

For multi-class classification problems, f1-score, loss and accuracy scores are very common metrics to determine and evaluate the model's performance.

The F1 score can be interpreted as a weighted average of the precision and recall, where an F1 score reaches its best value at 1 and worst score at 0. The relative contribution of precision and recall to the F1 score are equal. The formula for the F1 score is:

F1 = 2 * (precision * recall) / (precision + recall)

In the multi-class and multi-label case, this is the weighted average of the F1 score of each class.

Accuracy measures how often the classifier makes the correct prediction. It's the ratio of the number of correct predictions to the total number of predictions (the number of test data points).

Recall (sensitivity) is a ratio of true positives (classes classified correctly) to all the classes that were actually classified wrongly, in other words it is the ratio of

[True Positives/ (True Positives + False Negatives)]

For classification problems that are not skewed in their classification distributions, accuracy is not a good metric. For other types of classification cases, precision and recall come in very handy. These two metrics can be combined to get the F1 score, which is weighted average (harmonic mean) of the precision and recall scores. This score can range from 0 to 1, with 1 being the best possible F1 score (we take the harmonic mean as we are dealing with ratios) {, 2017 #46}.

Precision tells us what proportion of classes we classified as (for example c0, actually were c0). It is a ratio of true positives to all positives (all classes classified as c0, irrespective of whether that was the correct classification), in other words it is the ratio of

[True Positives/(True Positives + False Positives)]

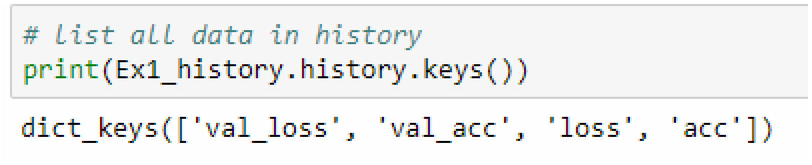
I will plot the loss and accuracy of each CNN model with respect to number of epochs. With this visualization one can see clearly if the system loss and accuracy is improving or not with respect to number of epochs. A perfect model will show an increment in accuracy and decrement in loss. We can see the point of overfitting if the accuracy starts to decrease after a certain epoch.

Keras provides the capability to register callbacks when training a deep learning model.

One of the default callbacks that is registered when training all deep learning models is the History callback [14]. It records training metrics for each epoch. This includes the loss and the accuracy (for classification problems) as well as the loss and accuracy for the validation dataset, if one is set [14, 15].

The history object is returned from calls to the fit() function used to train the model. Metrics are stored in a dictionary in the history member of the object returned.

For example, you can list the metrics collected in a history object using the following snippet of code after a model is trained:



```
# list all data in history
print(Ex1_history.history.keys())
```

```
dict_keys(['val_loss', 'val_acc', 'loss', 'acc'])
```

1.3.2 F-score and confusion matrix

Next, I will use f-score function in sklearn to calculate the f-score of each class for each model. Also, I will plot the confusion matrix for the validation set on each model.

2 Analysis

2.1 Data Exploration

The State Farm supplied the data for a Kaggle challenge. The dataset consists of 22,400 training and 79,727 testing images (640 × 480 with RGB colors) of drivers either driving attentively or doing one of 9 classes of distracting behaviors [3] [16]. An example input image is shown in Fig. 1.

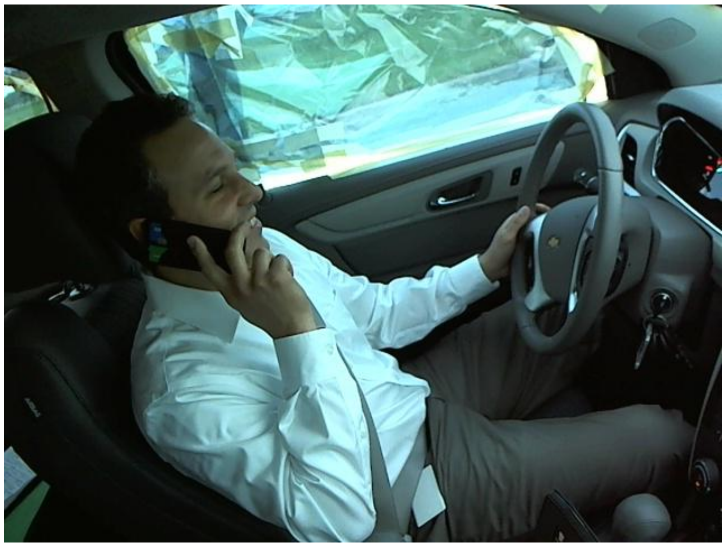


Fig.1 . Driver talking on a phone which is one of the distracting activities.

The training images come with correct labels and the challenge is to make the best multi-class classifications possible. State Farm provided a dataset of driver images where each image is taken inside a car with a driver engaging in some activity such as texting, eating, and etc. The given dataset is in csv format. The following files/folder were given to tackle this problem:

- imgs.zip - zipped folder of all (train/test) images
- sample_submission.csv - a sample submission file in the correct format
- driver_imgs_list.csv - a list of training images, their subject (driver) id, and class id

The types of activities State Farm wants the competitors to predict the likelihood is listed below:

The 10 classes to predict are:

- c0: safe driving
- c1: texting - right
- c2: talking on the phone - right
- c3: texting – left
- c4: talking on the phone - left
- c5: operating the radio
- c6: drinking

- c7: reaching behind
- c8: hair and makeup
- c9: talking to passenger

To evaluate the success of models, the training images were split into train and validation sets.

The dataset consists of an “imgs” folder that has a test and train folder of 640 x 480 jpg files. The images were taken by a dashboard camera. Each image consists of a driver performing a task from one of the distracted tasks. There are no duplicate images in the dataset. Also State Farm removed metadata from each image (e.g. creation dates). State Farm set up these experiments in a controlled environment. While performing each task, the drivers were not driving as a truck dragged the car on the streets.

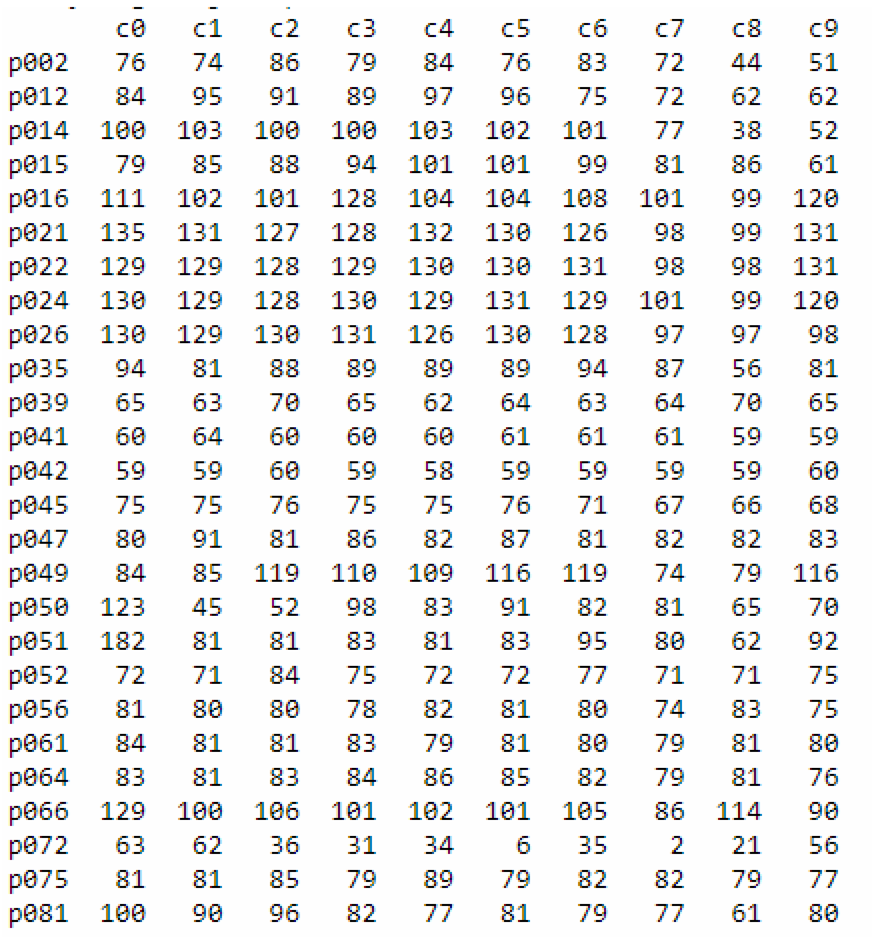
Below I listed the number of files for each category in the training data and the test data.

C0	C1	C2	C3	C4	C5	C6	C7	C8	C0
1689	1467	1517	1546	1526	1512	1525	1202	1111	1329

In total, there are 22,424 training examples. “Safe driving” has the most examples, and “Hair and makeup” the least. It makes sense to have a lot of “safe driving“examples because State farm is in general interested in finding out if the driver is driving safely or not. They would rather have false negatives than false positives when labeling safe driving. More examples would improve the classifier’s performance in decreasing false positives for safe driving. “Hair and makeup” may be less because it is a task mostly performed by women.

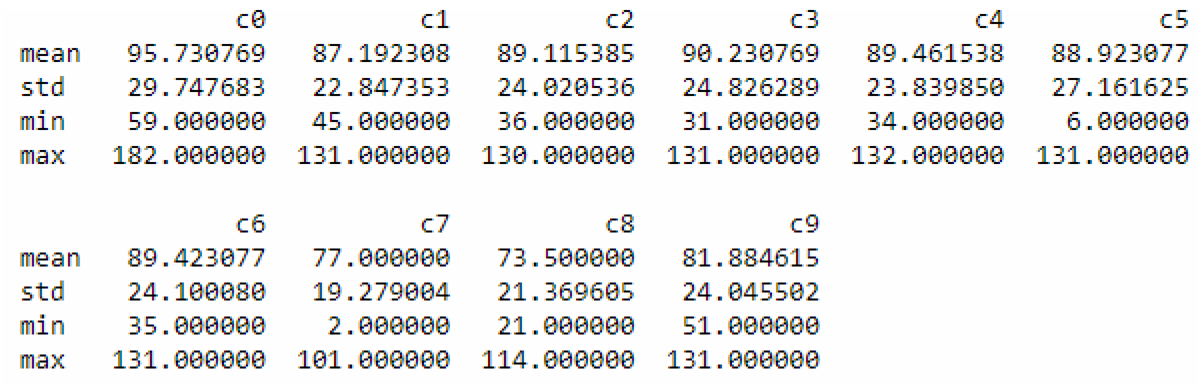
2.2 Exploratory Visualization

There is also a csv file that lists training images’ filenames, their subject (driver) id, and class id. It may be hard to accurately train a classifier with 26 drivers due to the differences in driver’s physical traits, especially if the dataset is relatively small for each driver. However, in the end we can create a robust classifier that will make good predictions on people with different sizes, gender etc.



```
      c0   c1   c2   c3   c4   c5   c6   c7   c8   c9
p002   76   74   86   79   84   76   83   72   44   51
p012   84   95   91   89   97   96   75   72   62   62
p014  100  103  100  100  103  102  101   77   38   52
p015   79   85   88   94  101  101   99   81   86   61
p016  111  102  101  128  104  104  108  101   99  120
p021  135  131  127  128  132  130  126   98   99  131
p022  129  129  128  129  130  130  131   98   98  131
p024  130  129  128  130  129  131  129  101   99  120
p026  130  129  130  131  126  130  128   97   97   98
p035   94   81   88   89   89   89   94   87   56   81
p039   65   63   70   65   62   64   63   64   70   65
p041   60   64   60   60   60   61   61   61   59   59
p042   59   59   60   59   58   59   59   59   59   60
p045   75   75   76   75   75   76   71   67   66   68
p047   80   91   81   86   82   87   81   82   82   83
p049   84   85  119  110  109  116  119   74   79  116
p050  123   45   52   98   83   91   82   81   65   70
p051  182   81   81   83   81   83   95   80   62   92
p052   72   71   84   75   72   72   77   71   71   75
p056   81   80   80   78   82   81   80   74   83   75
p061   84   81   81   83   79   81   80   79   81   80
p064   83   81   83   84   86   85   82   79   81   76
p066  129  100  106  101  102  101  105   86  114   90
p072   63   62   36   31   34    6   35    2   21   56
p075   81   81   85   79   89   79   82   82   79   77
p081  100   90   96   82   77   81   79   77   61   80
```

As you can see, most of the drivers performed each task equally. However, we can see some outliers here. For example, subject p072 has quite a variance in the images that were taken for each category. The subject has only 6 images for c5 (Operating the radio) and 2 images for c7 (Reaching behind) whereas for c0 (Safe driving), there are 63 images. When training the classifier, I may experiment with omitting data from subject p072 to see if it affects the performance.



```
              c0          c1          c2          c3          c4          c5
mean   95.730769   87.192308   89.115385   90.230769   89.461538   88.923077
std    29.747683   22.847353   24.020536   24.826289   23.839850   27.161625
min    59.000000   45.000000   36.000000   31.000000   34.000000    6.000000
max   182.000000  131.000000  130.000000  131.000000  132.000000  131.000000

              c6          c7          c8          c9
mean   89.423077   77.000000   73.500000   81.884615
std    24.100080   19.279004   21.369605   24.045502
min    35.000000    2.000000   21.000000   51.000000
max   131.000000  101.000000  114.000000  131.000000
```

2.3 Algorithms and Techniques

2.3.1 Transfer Learning

Transfer learning is the idea of using a CNN model pre-trained on a large dataset as an initialization [9]. It gave us a significant boost in terms of speed and performance. For each model, the benchmark, CNN, VGG16 and InceptionV3, I modified the last dense layer to output 10 class predictions. Then I used the given training set as the input images to train the whole neural network.

2.3.2 Convolutional Neural Network

Convolutional neural networks or CNN networks are similar to conventional neural networks (NN) except they are built to take 2D arrays such as images as inputs [9]. In other words their neurons are 3D structured as an 3D volume. Layers of a CNN transforms one volume to another. I will go over common CNN layers in next sections.

2.3.3 Input Layer

The input layer consist of the pixel values of each input images. In this project, full color images are used, so the input layer is $640.0 \times 480.0 \times 3.0$ (for R, G, B channels).

Convolutional (CONV) Layer: The CONV layer's parameters are a set of learnable filters of small dimensions (e.g. $5 \times 5 \times 3$). Each filter is convolved with the input 2D arrays (across height and width in 2D) to select small areas (e.g. 5×5) and in each of these small local areas, it computes the dot products with weights/parameters. Each filter corresponds to a slice or a depth of one in the output volume (i.e. the depth of the output volume of a CONV layer is determined by the number of filters).

2.3.4 Rectified Linear Units (ReLU) Layer

The Rectified Linear Unit or ReLU layer applies an element-wise non-linear activation function to increase nonlinearity in the model.

2.3.5 Pooling Layer

Now, In order to reduce the dimension of the model, pooling layer is introduced to reduce the 2D dimensions of the input volume (leaving the depth unchanged) to make the training easier and to prevent the model from overfitting and getting too large (too many weights) to compute. We can add the pooling layer to each input by applying a small filter (e.g. 2×2). Tow most popular pooling functions are average and max functions (e.g. outputting the average and maximum values of 2× 2 regions, thus reducing the dimension by 75%).

2.3.6 Dropout Layer

Some times to avoid overfitting, we can use dropout layer to generalize the model. Dropout layers is a regularization method to prevent overfitting. We can set how many perceptron we want to turn off in a particular layer and only other perceptron would be trained. A dropout layer randomly sets some unit activations to zero and thus removes some feature detectors. Because some feature detectors make the model adapt to very complex functions in the context of some other very specific feature detectors. By dropping out some activations, some of the high variance due to this is removed [3].

2.3.7 Fully-connected (FC) Layer

Fully-connected layers, as the name suggests, is like ordinary NN where each neuron is connected to all the outputs from the previous layer. The last FC layer computes probabilities for each class. For multi-class classification, softmax is a popular choice. Softmax regression has the following log-likelihood function:

$$l(\theta) = \sum_{i=1}^{m} log \prod_{l=1}^{k} \left(\frac{exp(\theta_l{}^T x^{(i)})}{\sum_{j=1}^{k} exp(\theta_j{}^T x^{(i)})} \right)^{1\{y^{(i)}=l\}}$$

That is, Softmax trains the final layer to correctly predict with maximal confidence for each image.

2.3.8 Outline of tasks

1. Analyze the data
2. Separate validation set from training set
3. Primarily visualizations
4. Benchmark: linear model
5. CNN networks experiments
    a. CNN linear model benchmark
    b. CNN with dropout layer
    c. VGG16 without pre-trained weights
    d. InceptionV3 with pre-trained weights
    e. InceptionV3 with further fine-tuning

6. Plot each network's loss and accuracy scores

2.4 Benchmark

2.4.1 CNN linear model (benchmark)

I will start with the simple model: a fully connected network with no hidden layer, i.e., linear model. This is to provide a benchmark for subsequence development.

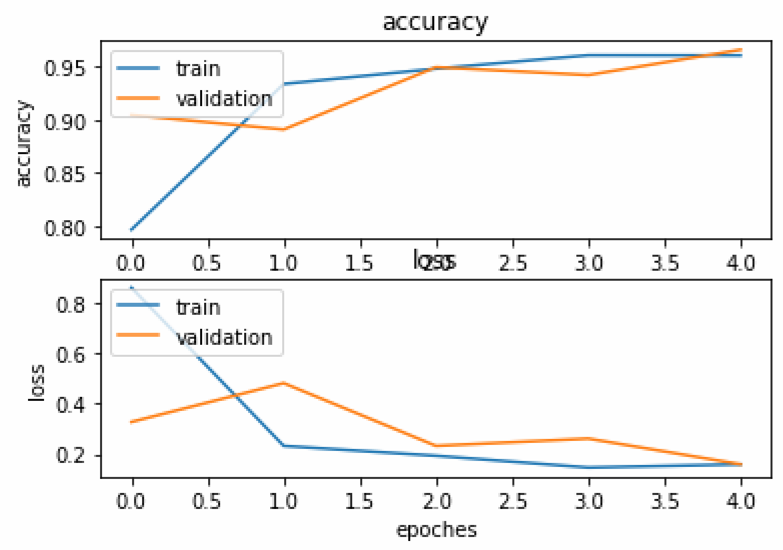
- Use batchnormalization() right at the input layer to avoid any domination input values that could skew the output.
- Activated the output with a softmax layer for 10 classes.
- 224x224 input shape, as the results we will have 1.5+ million parameters and easily over fitted with a linear model, hence, l2 regularization is used to minimize impact of overfitting.

Even though the trainable parameters are only 1,505,296. The tensorflow-gpu version could not handle computing this model. So I used 20% of the GPU memory to prevent overflowing memory usage. Then I used the fit method to the training and validation sets with 5 epochs and 32 batches.

The accuracy is increasing be each epoch which means the model is working.

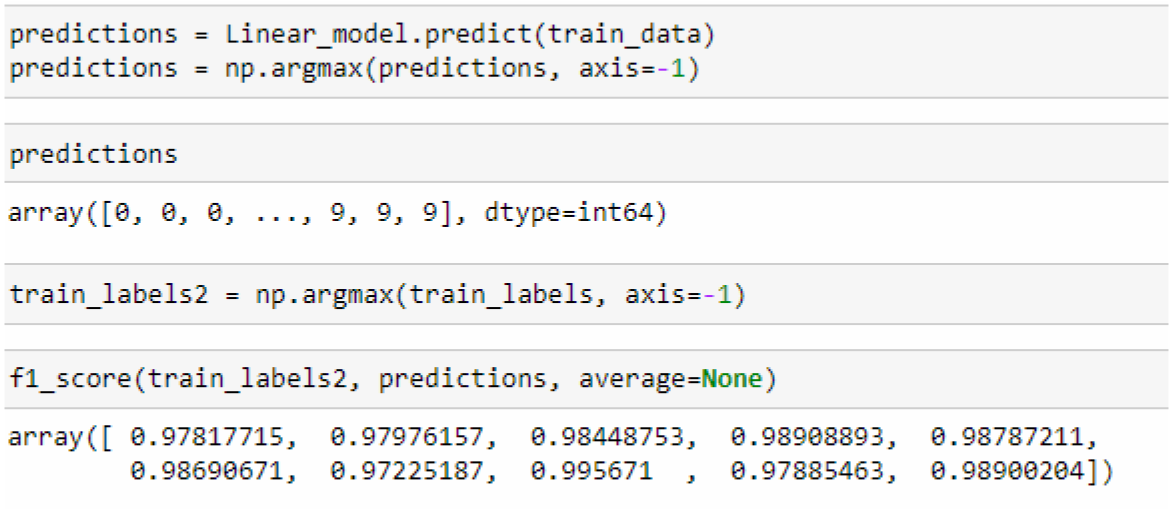
Plot_history is a simple functions to plot the accuracy and loss of the model.

```
plot_history(Ex1_history)
```



From the plots we can see that the training and validation accuracy are not very stable. But it could achieve 94% accuracy which is certainly better that random guessing.

Next, I used predict function to predict the training and validation sets. This piece of code shows the f1-score of each class.



```
predictions = Linear_model.predict(train_data)
predictions = np.argmax(predictions, axis=-1)
```

```
predictions
```

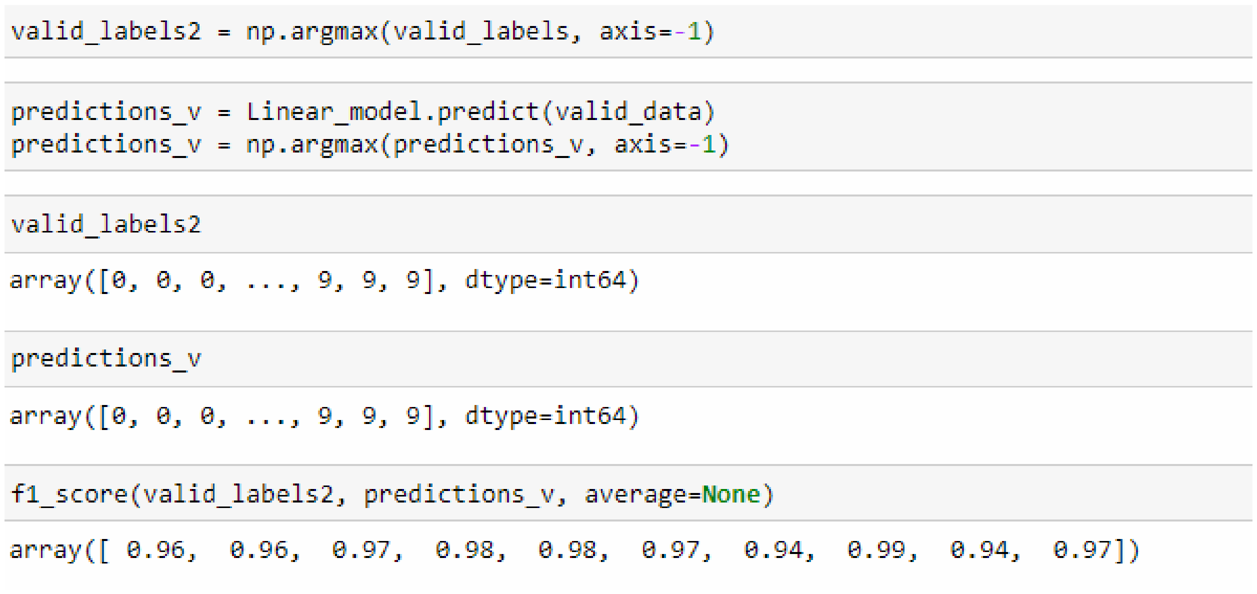
```
array([0, 0, 0, ..., 9, 9, 9], dtype=int64)
```

```
train_labels2 = np.argmax(train_labels, axis=-1)
```

```
f1_score(train_labels2, predictions, average=None)
```

```
array([ 0.97817715,  0.97976157,  0.98448753,  0.98908893,  0.98787211,
        0.98690671,  0.97225187,  0.995671  ,  0.97885463,  0.98900204])
```

And this piece of code is to calculate the f-score for each class on validation set.



```
valid_labels2 = np.argmax(valid_labels, axis=-1)
```

```
predictions_v = Linear_model.predict(valid_data)
predictions_v = np.argmax(predictions_v, axis=-1)
```

```
valid_labels2
```

```
array([0, 0, 0, ..., 9, 9, 9], dtype=int64)
```

```
predictions_v
```

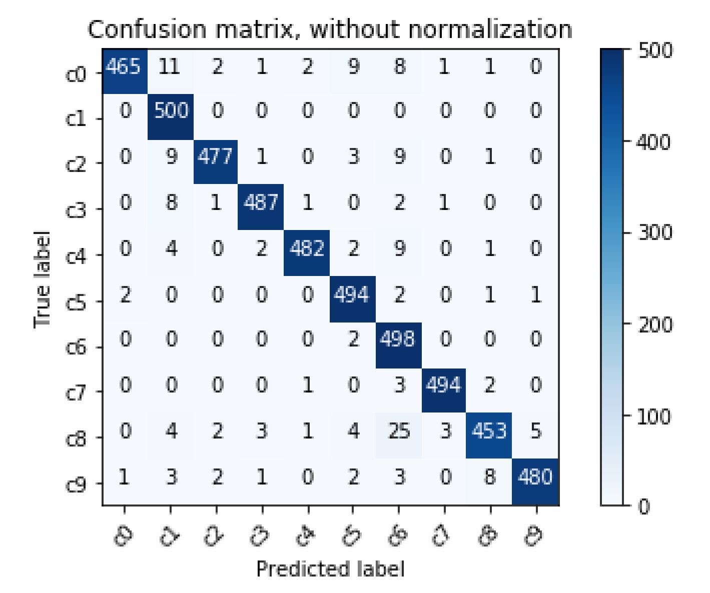
```
array([0, 0, 0, ..., 9, 9, 9], dtype=int64)
```

```
f1_score(valid_labels2, predictions_v, average=None)
```

```
array([ 0.96,  0.96,  0.97,  0.98,  0.98,  0.97,  0.94,  0.99,  0.94,  0.97])
```

Next, I used the confusion matrix function to plot the 2D confusion matrix of training and validation sets. The plots below are the confusion matrices with and without normalization. The diagonal axis in the confusion matrices is the correctly labeled predicted classes. We can see class c0 and c8 which are the safe driving and hair and makeup classes are the most poorly predicted class with 96% accuracy.

The confusion matrices on validation set are:



	c0	c1	c2	c3	c4	c5	c6	c7	c8	c9
c0	465	11	2	1	2	9	8	1	1	0
c1	0	500	0	0	0	0	0	0	0	0
c2	0	9	477	1	0	3	9	0	1	0
c3	0	8	1	487	1	0	2	1	0	0
c4	0	4	0	2	482	2	9	0	1	0
c5	2	0	0	0	0	494	2	0	1	1
c6	0	0	0	0	0	2	498	0	0	0
c7	0	0	0	0	1	0	3	494	2	0
c8	0	4	2	3	1	4	25	3	453	5
c9	1	3	2	1	0	2	3	0	8	480

The accuracy on the validation set is obviously worse than on training set. Again c0 and c8 are the worse predicted classes. It shows the model is underfitted. So I will use Conv2D layers in the CNN model next to train it better.

3 Methodology

3.1 Data Preprocessing

3.1.1 Data preparation

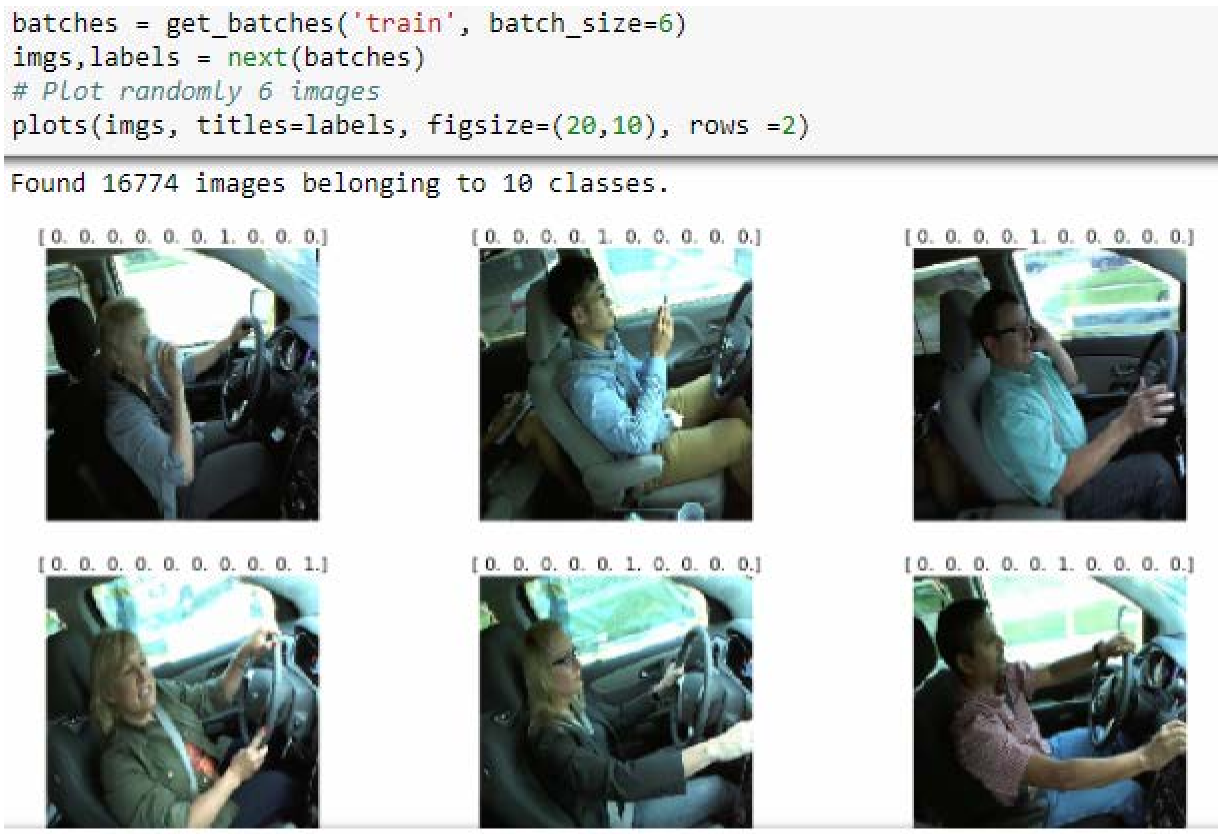
First I separated the validation set from training set by moving around 25% of the training sets in different folders. First I set the paths for training, validation and results and moved 500 images from each class to validation folder.

```
# moving ~20% data from train sets to validation sets
for label in class_labels:
    g = glob(label+'/*.jpg')
    shuffle = np.random.permutation(g)
    for i in range(500): move(shuffle[i], valid_path+shuffle[i])
```

These codes should only be run once. Otherwise we might move more than 500 images to validation sets.

3.1.2 Data visualization

In order to investigate the data more thoroughly, it’s a good idea to plot some of the images form training and validation sets to see how they are different. We need to make sure the training and validation sets are actually similar but not exact. For example we want them to be taken from the same angle, have same background, same lighting, and so on. The features we are interested to bring out are the driver’s activity and we don’t want other parts of the images to interfere with the data that gives us information about the driver’s activity. In the plotted images we are looking for position of the driver’s hands, face, and his/her behavior. Also we need to make sure the drivers are different in both training and validation sets and the activities are distributed uniformly in validation set as well.



```
batches = get_batches('train', batch_size=6)
imgs,labels = next(batches)
# Plot randomly 6 images
plots(imgs, titles=labels, figsize=(20,10), rows =2)
```

```
Found 16774 images belonging to 10 classes.
```

```
[0. 0. 0. 0. 0. 0. 1. 0. 0. 0.]    [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]    [0. 0. 0. 0. 1. 0. 0. 0. 0. 0.]

[0. 0. 0. 0. 0. 0. 0. 0. 0. 1.]    [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]    [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
```

```
imgs,labels = next(batches)

# Plot randomly 6 images
plots(imgs, titles=labels, figsize=(20,10), rows =2)
```

Now, in order to fetch the images from their directory and load them in numpy array, we need 2 functions as below. I used the image processing package in keras [17]. Now we can use the get_data to fetch images from their directory. I used save_array function to convert the images into array shape. This is only for submission to kaggle. Now we can see how many training and validation images we have prepared.

```
(valid_classes, train_classes, valid_labels, train_labels, valid_filenames, train_filenames) = get_cla
valid_data = load_array(path+'results/valid_data.dat')
train_data = load_array(path+'results/train_data.dat')
```

```
Found 16774 images belonging to 10 classes.
Found 5000 images belonging to 10 classes.
```

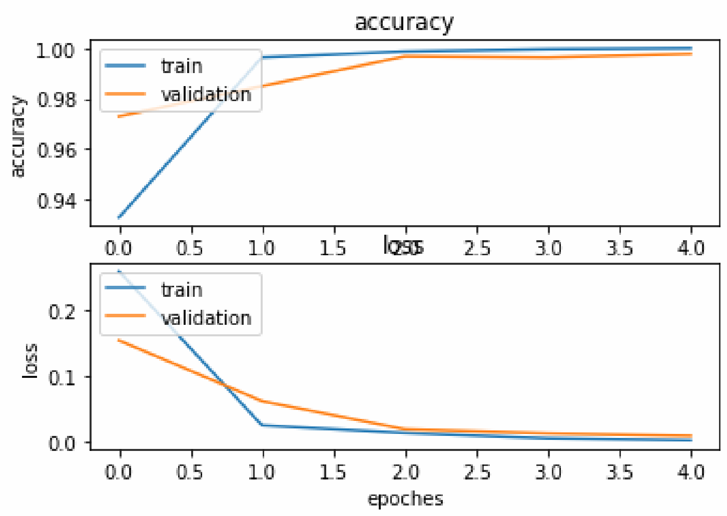
During this section of the project, I faced a lot of issues with making the functions behave as they are supposed to. Working with numpy in conjunction with other library which do not take numpy array as input was the problem. I fixed these bugs by going step by step and see the type and shape of the output after every line. That way I could see where was the problem and what method can I use to fix them.

3.2 Implementation- CNN model experiments

3.2.1 CNN without dropout layer

Next, I will experiment a neural network with 2 convolutional layers. This experiment will give us an idea on how this dataset behave under convolutional actions. I will try to monitor overfitting and later adding some regularization or data augmentation. This CNN model has 2 Conv layer with "relu" activation, MaxPooling2D layer of size (3,3), the number of filters are 32, 64 in the first and second layer. I added Batch Normalization layer in both layes. It normalize the activations of the previous layer at each batch, i.e. applies a transformation that maintains the mean activation close to 0 and the activation standard deviation close to 1 [18] [15]. The last layer is dense layer with 10 ports because we have 10 classes and softmax activation. Now we just need to fit the model to our training and validation sets. Figure below shows the loss and accuracy plot versus number of epoch. It can be seen that the accuracy reaches 1 at epoch = 5. During training (loss) dropout is on. For validation (val_loss) dropout is off. Dropout basically shuts down a few weights of the network randomly. Think of it like a system where part of it fails regularly. The other parts have to learn how to do their best, even if they can't count with their peers. During validation the full system is on. This helps to deal with overfitting.

```
plot_history(Ex2_history)
```



In epoch = 4, it reaches 99% accuracy but after that it starts to overfit. We can use dropout layer to avoid overfitting.

Below it shows the f-score calculation for training and validation sets.

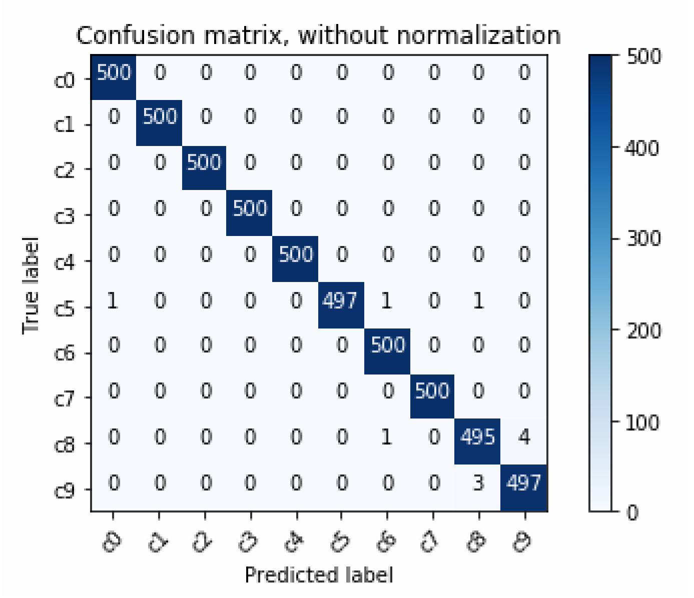
```
predictions_Ex2 = CNN_simple.predict(train_data)
predictions_Ex2 = np.argmax(predictions_Ex2, axis=-1)
train_labels_Ex2 = np.argmax(train_labels, axis=-1)
f1_score(train_labels_Ex2, predictions_Ex2, average=None)
```

```
array([ 1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.,  1.])
```

```
predictions_Ex2_v = CNN_simple.predict(valid_data)
predictions_Ex2_v = np.argmax(predictions_Ex2_v, axis=-1)
valid_labels_Ex2 = np.argmax(valid_labels, axis=-1)
f1_score(valid_labels_Ex2, predictions_Ex2_v, average=None)
```

```
array([ 1.  ,  1.  ,  1.  ,  1.  ,  1.  ,  1.  ,  1.  ,  1.  ,  0.99,  0.99])
```

Below shows the confusion matrices on validation matrices.



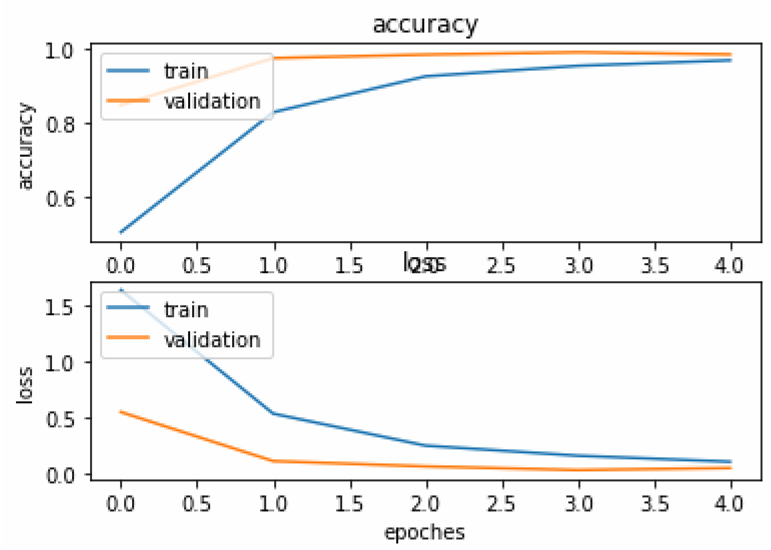
	c0	c1	c2	c3	c4	c5	c6	c7	c8	c9
c0	500	0	0	0	0	0	0	0	0	0
c1	0	500	0	0	0	0	0	0	0	0
c2	0	0	500	0	0	0	0	0	0	0
c3	0	0	0	500	0	0	0	0	0	0
c4	0	0	0	0	500	0	0	0	0	0
c5	1	0	0	0	0	497	1	0	1	0
c6	0	0	0	0	0	0	500	0	0	0
c7	0	0	0	0	0	0	0	500	0	0
c8	0	0	0	0	0	0	1	0	495	4
c9	0	0	0	0	0	0	0	0	3	497

The added conv2D layers in the CNN model work very well and we can see the f-score and confusion matrices show very high scores, but the model is overfitted as we saw from accuracy and loss plots. So I will use dropout layers to generalize the model more.

3.2.2 CNN with dropout layer

In this experiment, I will add dropout layer in order to simplify the model at the cost of losing some information. It also causes the model to be more generalized and less sophisticated. To compensate for losing some of the data computation, I added more layers to the model. Next, I fit this model to our training and validation sets with 5 epochs. The dropout layers clearly simplified the model and reduced the overfitting as can be seen from the plots below.

```
plot_history(Ex3_history)
```



At epoch 4, it starts to loss accuracy, so it starts to overfit again; we can stop the fitting at epoch 4.

Below shows the f-scores on training and validation sets for the CNN model with dropout layers. As expected, since we are losing some information due to dropout layers, the f-scores on the training set are not as high as previous model.

```
predictions_Ex3 = CNN_dropout.predict(train_data)
predictions_Ex3 = np.argmax(predictions_Ex3, axis=-1)
train_labels_Ex3 = np.argmax(train_labels, axis=-1)
f1_score(train_labels_Ex3, predictions_Ex3, average=None)
```

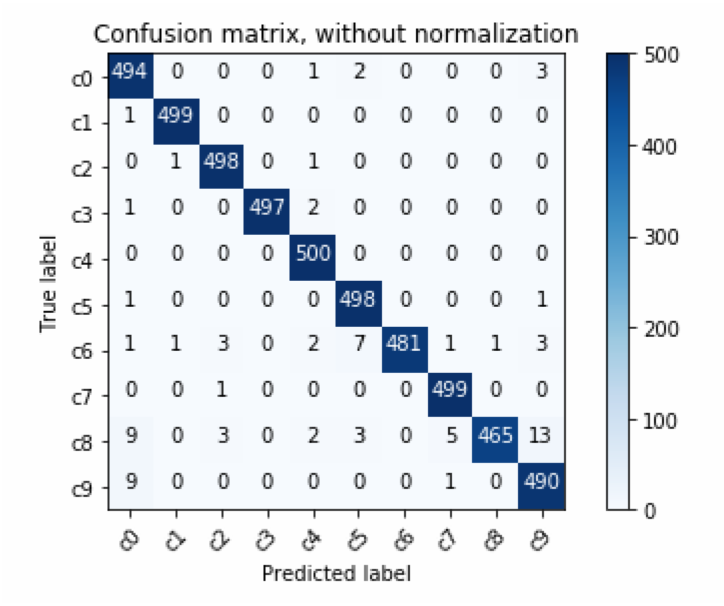
```
array([ 0.99,  1.  ,  1.  ,  1.  ,  0.99,  0.99,  0.99,  1.  ,  0.98,  0.99])
```

```
predictions_Ex3_v = CNN_dropout.predict(valid_data)
predictions_Ex3_v = np.argmax(predictions_Ex3_v, axis=-1)
valid_labels_Ex3 = np.argmax(valid_labels, axis=-1)
f1_score(valid_labels_Ex3, predictions_Ex3_v, average=None)
```

```
array([ 0.97,  1.  ,  0.99,  1.  ,  0.99,  0.99,  0.98,  0.99,  0.96,  0.97])
```

Below shows the confusion matrices on training set. We can see it's performing not as good as previous model because of dropout layers, but it should perform better on testing data. Unfortunately the submission on kaggle competition is closed and I don't have access to the test data classes.

For validation set:



	c0	c1	c2	c3	c4	c5	c6	c7	c8	c9
c0	494	0	0	0	1	2	0	0	0	3
c1	1	499	0	0	0	0	0	0	0	0
c2	0	1	498	0	1	0	0	0	0	0
c3	1	0	0	497	2	0	0	0	0	0
c4	0	0	0	0	500	0	0	0	0	0
c5	1	0	0	0	0	498	0	0	0	1
c6	1	1	3	0	2	7	481	1	1	3
c7	0	0	1	0	0	0	0	499	0	0
c8	9	0	3	0	2	3	0	5	465	13
c9	9	0	0	0	0	0	0	1	0	490

3.3 Refinement

The model fits the validation and training sets well but it could be improved by generalizing the model a bit more. It's starting to overfit the data which can be dealt with by more dropout layers or using other pre-trained models. In this section, I will try to use VGG16 with pre-trained ImageNet weights to predict the validation set.

3.3.1 VGG16 with pre-trained ImageNet weights

VGG-16 is a 16-layer CNN developed by the University Of Oxford Visual Geometry Group (VGG). I read from literature that ImageNet weights in junction with VGG16 model performs well in classification problems [19-22]. I set up the pre-trained VGG-16 model in jupyter notebooks and used the ImageNet weights to compute the VGG16 model outputs.

```
from keras.applications.vgg16 import VGG16
vgg_model_orig = VGG16(weights='imagenet', include_top=True)
```

To add the batch normalization layers to VGG16, first I popped all the layers and extracted all the conv2D layers. Then created a CNN model with 30 layers. Initially, this model had more than 100 million trainable parameters. It would take very long time to finish the computation, so I added few MaxPooling2D and AveragePooling2D layers to reduce the parameters.

To make the VGG16 layers non-trainable, I set the trainable parameter in all of them to false. This way only the added layers will be trained.

```
# Set all the convolutional layers to nontrainable
for layer in conv_model.layers:
    layer.trainable = False
vgg_model.summary()
```

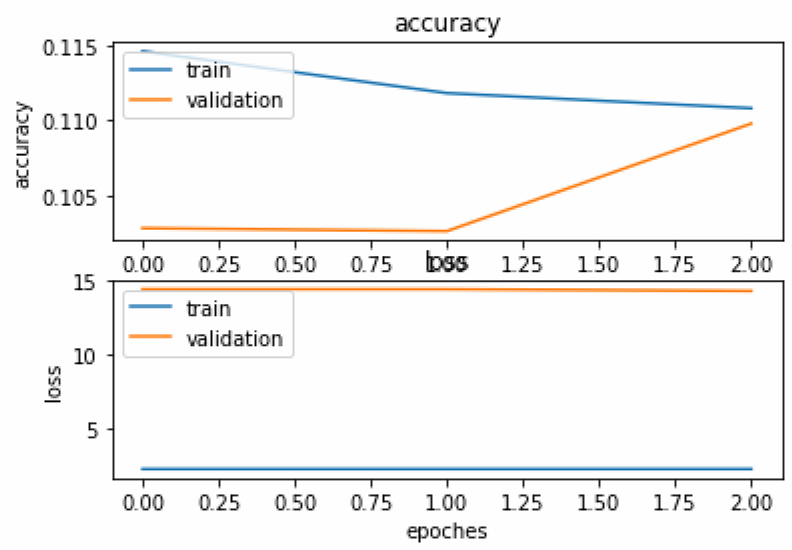
```
vgg_model.compile(Adam(lr=10e-5), loss='categorical_crossentropy', metrics=['accuracy'])
```

```
batch_size = 1
Ex41_history = vgg_model.fit(train_data,train_labels, batch_size=batch_size, epochs=3,
                              validation_data =(valid_data,valid_labels))
```

```
Train on 16774 samples, validate on 5000 samples
Epoch 1/3
16774/16774 [==============================] - 14618s - loss: 2.2926 - acc: 0.1146 - val_loss: 14.4584
- val_acc: 0.1028
Epoch 2/3
16774/16774 [==============================] - 15513s - loss: 2.2930 - acc: 0.1118 - val_loss: 14.4540
- val_acc: 0.1026
Epoch 3/3
16774/16774 [==============================] - 14967s - loss: 2.2929 - acc: 0.1108 - val_loss: 14.3356
- val_acc: 0.1098
```

Next, we fit the model to the training and validation sets and plot the accuracy and loss of the model. I wanted to set the batch size as high as possible to save training time. However, due to the memory limitation of my laptop, I set the batch size to 1. For the learning rate, [13] I experimented with learning rates of 0.001 to 0.00001. And with learning rate of higher than 0.00001, it will not converge.

```
plot_history(Ex41_history)
```



By plotting the test and training accuracies, I found that our VGG model suffered from overfitting - i.e. low bias and high variance. The model is not working as well as I hoped. The validation accuracy with 3 epochs is very low around 0.11. I will attempt another pre-trained model in next section.

4 Results

4.1.1 InceptionV3 with pre-trained weights

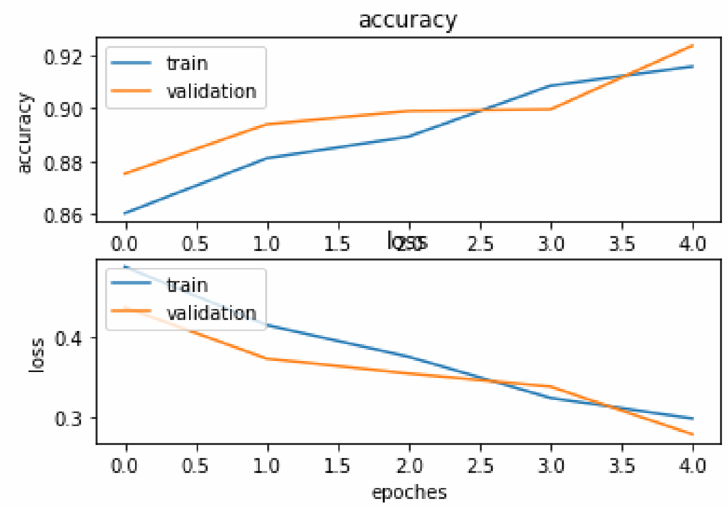
ImageNet is a common academic data set in machine learning for training an image recognition system. In [23], proposed Inception-v1 architecture which is a variant of the GoogleNet in the paper going deeper with convolutions, and in the meanwhile they introduced Batch Normalization to Inception (BN-Inception) [13] [24].

The main difference between V2 and V3 is the network described in [25] that the 5x5 convolutional layers are replaced by two consecutive layer of 3x3 convolutions with up to 128 filters. And in the paper rethinking the Inception Architecture for Computer Vision, the authors proposed Inception-v2 and Inception-v3. In the Inception-v2, they introduced Factorization (factorize convolutions into smaller convolutions) and some minor change into Inception-v1. Note that we have factorized the traditional 7x7 convolution into three 3x3 convolutions As for Inception-v3, it is a variant of Inception-v2 which adds BN-auxiliary. BN auxiliary refers to the version in which the fully connected layer of the auxiliary classifier is also-normalized, not just convolutions [7, 20, 24].

4.2 Model Evaluation and Validation

In this experiment, I will use InceptionV3 model with pre-trained ImageNet weights. I will first start with the base model with slightly modification of the top layer to adapt with 10 classes instead 1000 classes. Next, I will fit the model to the training and validation sets and plot the loss and accuracy of the model.

```
plot_history(Ex4_history)
```



The model is over fitted, and the validation accuracy stays around 91% for 1 to 4 epochs. Next I will set the trainable parameter of the latter layers to true and freeze the first several layers.

4.2.1 InceptionV3 with further fine-tuning

In this section, I set the trainable parameter of the latter layers to true and freeze the first several layers.

Then I complied and fit the model to the training and validation sets.

```
# the first 172 layers and unfreeze the rest:
for layer in IcepV3_model.layers[:172]:
    layer.trainable = False
for layer in IcepV3_model.layers[172:]:
    layer.trainable = True
```

```
IcepV3_model.compile(Adam(lr=10e-5), loss='categorical_crossentropy', metrics=['accuracy'])
```

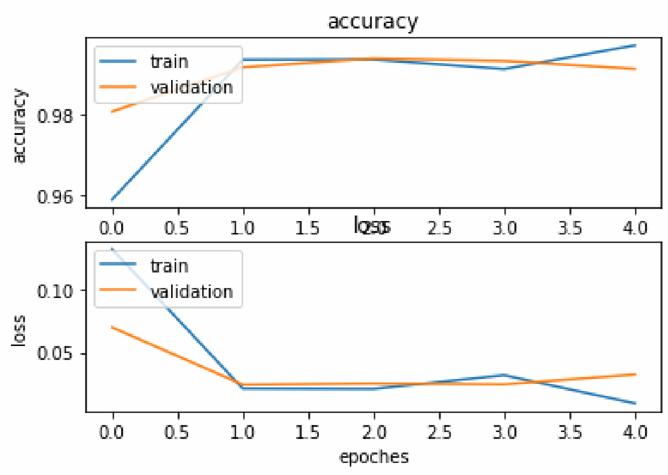
```
Ex5_history = IcepV3_model.fit(train_data,train_labels, batch_size=batch_size, epochs=5,
                           validation_data =(valid_data,valid_labels))
```

```
Train on 16774 samples, validate on 5000 samples
Epoch 1/5
16774/16774 [==============================] - 5627s - loss: 0.1324 - acc: 0.9589 - val_loss: 0.0700 -
val_acc: 0.9808
Epoch 2/5
16774/16774 [==============================] - 5203s - loss: 0.0212 - acc: 0.9937 - val_loss: 0.0243 -
val_acc: 0.9918
Epoch 3/5
16774/16774 [==============================] - 4885s - loss: 0.0209 - acc: 0.9937 - val_loss: 0.0252 -
val_acc: 0.9940
Epoch 4/5
16774/16774 [==============================] - 4807s - loss: 0.0318 - acc: 0.9914 - val_loss: 0.0245 -
val_acc: 0.9934
Epoch 5/5
16774/16774 [==============================] - 4819s - loss: 0.0093 - acc: 0.9973 - val_loss: 0.0324 -
val_acc: 0.9914
```

4.3 Justification

The model's training and validation accuracy and loss score are improved compared to previous models. But at epoch 4 it is starting to overfit, which can be avoided by using dropout layer and/or using another pre-trained model.

```
plot_history(Ex5_history)
```



The accuracy and loss plots show significant improvements over other experiments. The model could achieve 99.14% validation accuracy, which is an acceptable result.

I calculated the f-score on training and validation sets. The model did pretty well on both sets. The f-score codes are:

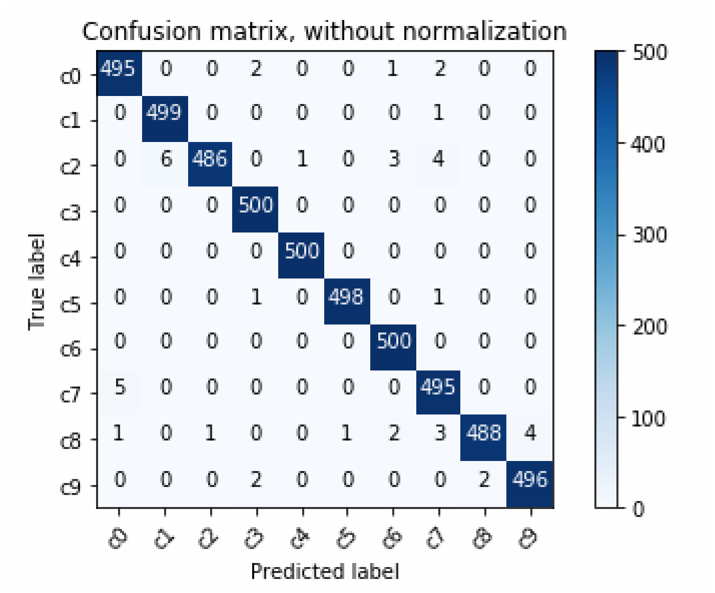
```
predictions_Ex4 = IcepV3_model.predict(train_data)
predictions_Ex4 = np.argmax(predictions_Ex4, axis=-1)
train_labels_Ex4 = np.argmax(train_labels, axis=-1)
f1_score(train_labels_Ex4, predictions_Ex4, average=None)
```

```
array([ 1.  ,  0.99,  0.99,  1.  ,  1.  ,  1.  ,  1.  ,  0.99,  1.  ,  1.  ])
```

```
predictions_Ex4_v = IcepV3_model.predict(valid_data)
predictions_Ex4_v = np.argmax(predictions_Ex4_v, axis=-1)
valid_labels_Ex4 = np.argmax(valid_labels, axis=-1)
f1_score(valid_labels_Ex4, predictions_Ex4_v, average=None)
```

```
array([ 0.99,  0.99,  0.98,  1.  ,  1.  ,  1.  ,  0.99,  0.98,  0.99,  0.99])
```

Below are confusion matrices on validation set, because of fine tuning and freezing first several layers, the model is simpler and more generalized and fitted the validation set better.



	c0	c1	c2	c3	c4	c5	c6	c7	c8	c9
c0	495	0	0	2	0	0	1	2	0	0
c1	0	499	0	0	0	0	0	1	0	0
c2	0	6	486	0	1	0	3	4	0	0
c3	0	0	0	500	0	0	0	0	0	0
c4	0	0	0	0	500	0	0	0	0	0
c5	0	0	0	1	0	498	0	1	0	0
c6	0	0	0	0	0	0	500	0	0	0
c7	5	0	0	0	0	0	0	495	0	0
c8	1	0	1	0	0	1	2	3	488	4
c9	0	0	0	2	0	0	0	0	2	496

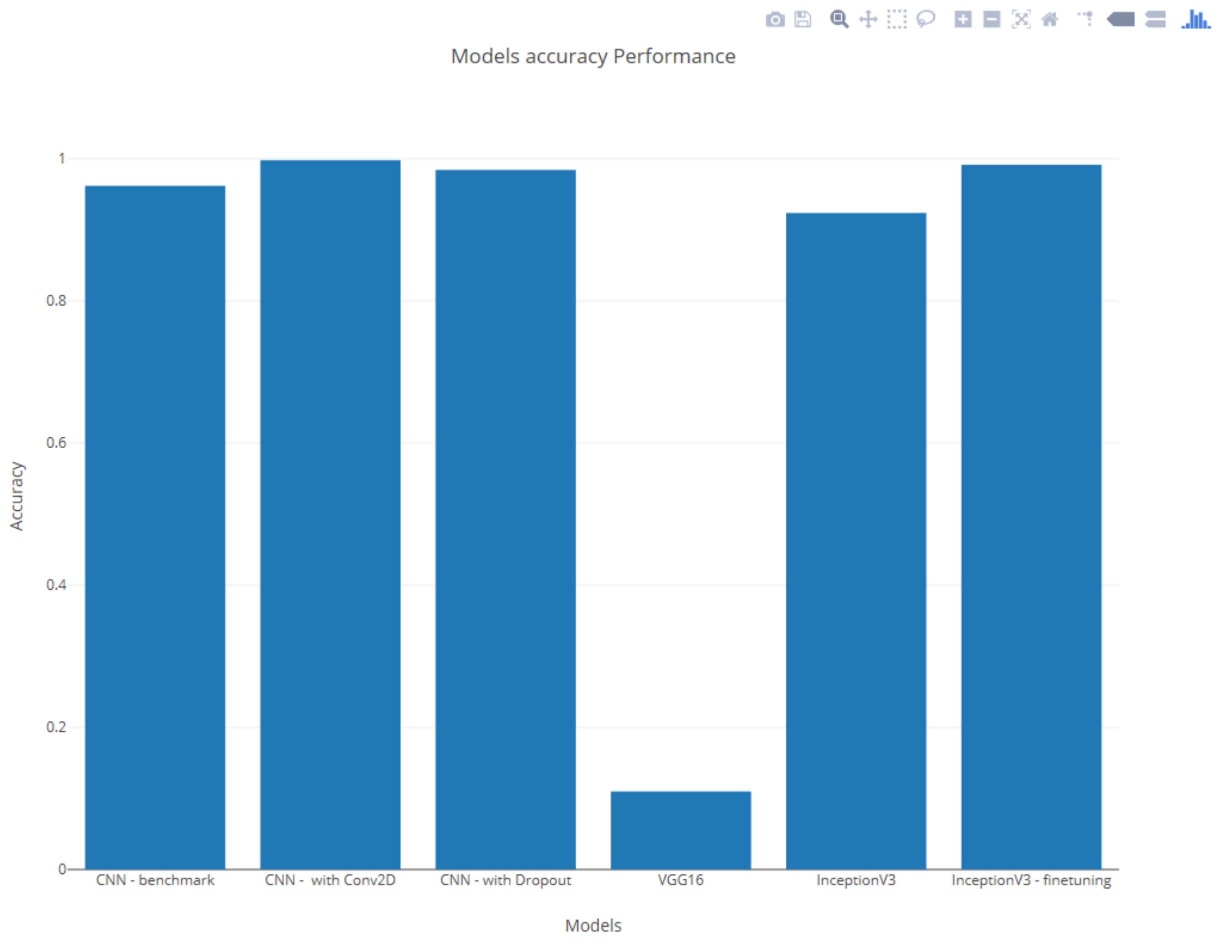
Compare to the benchmark, the InceptionV3 model has higher F1-scores for each class in validation set, which is what we are looking for. Also it shows better confusion matrix, where each class is predicted.

5 V. Conclusion

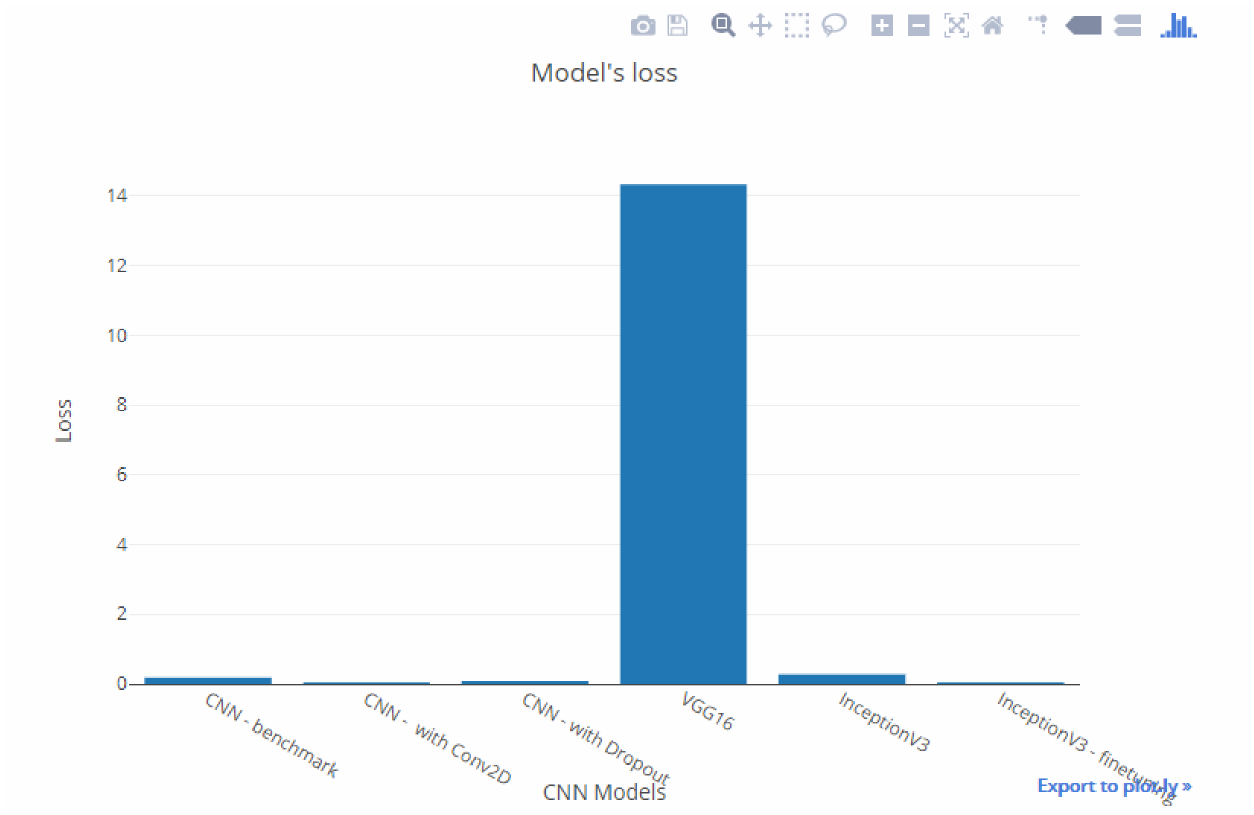
5.1 Free-Form Visualization

To predict few samples from the test images, I wrote a simple function to predict the driver's activity in the test images and writes its class and its activity. The model is predicting the images correctly; in future I would let it predict more training images. Below shows all models' accuracy and loss scores on validation set. It is clear that VGG16 model predicted the validation set the worse. The top models are CNN with 2 conv2D layer and inceptionV3 with fine-tuning.

Thought out this project I learned a lot about different CNN structures and how to stack them to get the best result possible. I also learned working with tensorflow GPU version is not as straight forward as it seems. It has many limitations and bugs. I either could use AWS could computing or use tensorflow CPU version. I chose tensorflow CPU version to speed my work overall not just the computing part. I also had some issues with keras image processing library, to fix the bugs and errors, I made it very simple and basic and build up on the functions and modules as I went on. Another interesting aspect of this project was fine tuning the InceptionV3 model. I read several fine tuning methods and the solution that was easy to implement on this model was to freeze the first several layers and train the rest.



Models accuracy Performance
Accuracy
1
0.8
0.6
0.4
0.2
0
CNN - benchmark
CNN - with Conv2D
CNN - with Dropout
VGG16
InceptionV3
InceptionV3 - finetuning
Models



5.2 Reflection

The project was to train CNN models to predict what the driver is doing by input images from the camera installed on the dashboard in a car. The model is supposed to detect if the driver is driving safely or not and classify the driver's activity out of 10 given classes. First, I used around 25% of the training images as validation set. And used several CNN models to train the model based on training set and test the on validation set. Then I plotted the accuracy, loss score and confusion matrix for each model and each class. It was interesting to see how easy a model is overfitted and how important it is to keep it simple and in compliance with the data.

5.3 Improvement

For further improvements, I would try simpler models with less layers to avoid overfitting the model. By adding more layers and complicating the CNN structure, some of the models became too sophisticated and overfit the data which signs were seen in accuracy and loss scores. Training more than 5 epochs is also contributed to overfitting some of the models. In future I would get feedback from accuracy and loss score plots to not to train the model for more than certain epoch to avoid overfitting, because adding dropout layer does not always fix the problem, by adding dropout layers, and simplifying the model, we would loss valuable information from the data with we need to use in training. Another future improvement ideas are to use data augmentation, ensemble to take advantage of several models or use KNN.

6 References

1. T. M. Pickrell, H. R. Li, and S. KC, *TRAFFIC SAFETY FACTS*. 2016: https://www.nhtsa.gov/risky-driving/distracted-driving.
2. Kaggle, *State Farm Distracted Driver Detection* 2017: https://www.kaggle.com/c/state-farm-distracted-driver-detection/data.
3. Colbran, S., *Classification of Driver Distraction*. 2016: https://pdfs.semanticscholar.org/cb49/ac9618bb2f8271409f91d53254a095d843d5.pdf.
4. Services, U.D.o.H.H., *Distracted Driving*. https://www.cdc.gov/motorvehiclesafety/distractedf gdriving/.
5. Organization, W.H., *Global status report on alcohol and health* W.H. Organization, Editor. 2014.
6. Singh, D., *Using Convolutional Neural Networks to Perform Classification on State Farm Insurance Driver Images*. 2016: cs229.stanford.edu/proj2016spr/report/004.pdf.
7. *https://github.com/alireza-a/kaggle-statefarm*. 2017.
8. Wakabayashi, M., *State Farm Distracted Driver Detection*. 2016: https://github.com/mwakaba2/Computer-Vision-Capstone-Project.
9. CS231n, G., *Convolutional Neural Networks for Visual Recognition*. 2016: http://cs231n.github.io/.
10. Hinton and Geoffrey, *Improving neural networks by preventing co-adaptation of feature detectors.* arXiv preprint arXiv, 2012. **1207.0580**.
11. Berri, R., et al., *A Pattern Recognition System for Detecting Use of Mobile Phones While Driving*. Vol. 2. 2014.
12. Bertin, M., *Kaggle State Farm* 2015: https://github.com/MarvinBertin/Kaggle_State_Farm.
13. Abouelnaga, Y., H. Eraqi, and M. Moustafa, *Real-time Distracted Driver Posture Classification*. 2017.
14. *https://keras.io/callbacks/*. 2017.
15. *https://machinelearningmastery.com/display-deep-learning-model-training-history-in-keras/*. 2017.
16. Simonyan Karen and A. Zisserman, *Very deep convolutional networks for large-scale image recognition.* arXiv preprint 2014: p. 1409-1556
17. *https://keras.io/preprocessing/image/*. 2017
18. *https://github.com/fchollet/keras/blob/master/keras/layers/*. 2017.
19. *BatchNormalization*. 2017: https://keras.io/layers/normalization/.
20. *https://datascience.stackexchange.com/questions/15328/what-is-the-difference-between-inception-v2-and-inception-v3*. 2017.

21. *https://github.com/BVLC/caffe/wiki/Model-Zoo*. 2017.
22. *https://github.com/alireza-a/kaggle-statefarm/blob/master/src/fine_tune_caffe_net.ipynb*. 2017.
23. Sergey Ioffe and Christian Szegedy. *Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift*. in *ICML*. 2015.
24. Asawa, C., *Deep Learning Approaches for Determining Optimal Cervical Cancer Treatment*. 2016: cs231n.stanford.edu/reports/2017/pdfs/922.pdf.
25. Szegedy, C., et al., *Rethinking the Inception Architecture for Computer Vision*. 2016.