Machine Learning Engineer Nanodegree

Capstone: State Farm Distracted Driver Detection

I. Definition

A. Project Overview

The domain of this problem is computer vision. Computer vision is a branch of machine learning concerned with the automatic extraction, analysis and understanding of useful information from a single image or a sequence of images (BMVA, n.d.). Computer vision began in the 1960's, when a person named Larry Roberts wrote his PhD thesis on the possibility of extracting 3D details and information from 2D images(T.S. Huang, n.d.). In the 70's, some progress was made on the interpretation of 2d images to 3d images (Hari Narayanan, et al, n.d.). In the 80's, optical character recognition systems that recognize letters, symbols and numbers were used in several industries (Quick history, n.d.). In the 90's, new applications of computer vision were possible as computers became more powerful and common (Quick history, n.d.). In the 2000's, computer vision was used to process large datasets, videos and could understand motion, patterns and predict outcomes (Hari Narayanan, et al, n.d.). The dataset being considered is the one provided in the Kaggle competition. It contains 2 folders, one with the training images and the other with the test images. The images capture the driver from a side-view dashboard camera. An excel file has also been provided and it links the images with its respective drivers for only the Train dataset.

B. Problem Statement

The problem that we are trying to solve is a multi-class classification problem. We are tasked to properly predict and classify driver's behavior given the dashboard images of people doing 10 different actions, 9 of which are considered actions of distracted behavior. The 10 classes are as follows: 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.
A solution to this problem is using machine learning computer vision models to classify driver actions. Working with Convolutional Neural Networks would be a good idea because CNN's are known to yield the most accurate results in the computer vision field. Keras application models with pre-trained weights could reduce the time it takes to train while still maintaining good results. Reducing image size and dividing the images into the RGB channels could make processing of the images more manageable. Pre-processing the images may be necessary to reduce overfitting and improve generalization. Once the model is fit, we will need to predict the labels of the test set to determine which of 10 categories each picture belongs to. The validation results and benchmark result will then tell us if we still need to improve the model. We are trying to achieve a log loss that would be closest to zero and should at least be within the top 10% of the Kaggle public leaderboard.
An outlined solution is, Step 1, import the training data. Step 2, split the training data into 2 subsets, a train subset and a validation subset. Step 3, Reduce and scale images from the dataset to a more manageable size. Step 4, make the CNN architecture but most optimally make use of the keras application and pre-trained model weights such as Xception, VGG-16, Resnet50. Step 5, fit the model and save the resulted weights. Step 6, test using the validation subset. Step 7, validate results/accuracy. Step 8, Plot the train and test dataset against the fitted values/epoch to see how much the model is overfitting. Step 9, compare Kaggle result to see if it meets benchmark results. Step 10, Adjust architecture, model, parameters and data augmentation. Step 11, Repeat until it meets target benchmark results. Step 12, make final adjustments.

C. Metrics

I will make use of the Multi-class logarithmic loss. This metric is used in many computer vision classification problems because it measures the accuracy of a classifier by penalizing false classifications. It is also a good metric for this problem because to calculate log-loss, the classifier must assign a probability to each class rather than yielding the most likely class. Having a good log loss would mean we are generalizing all categories well and not only favoring and generalizing few categories. Since we also need to compare our results to a benchmark, it would be best to stick with the metric made use in the Kaggle competition.

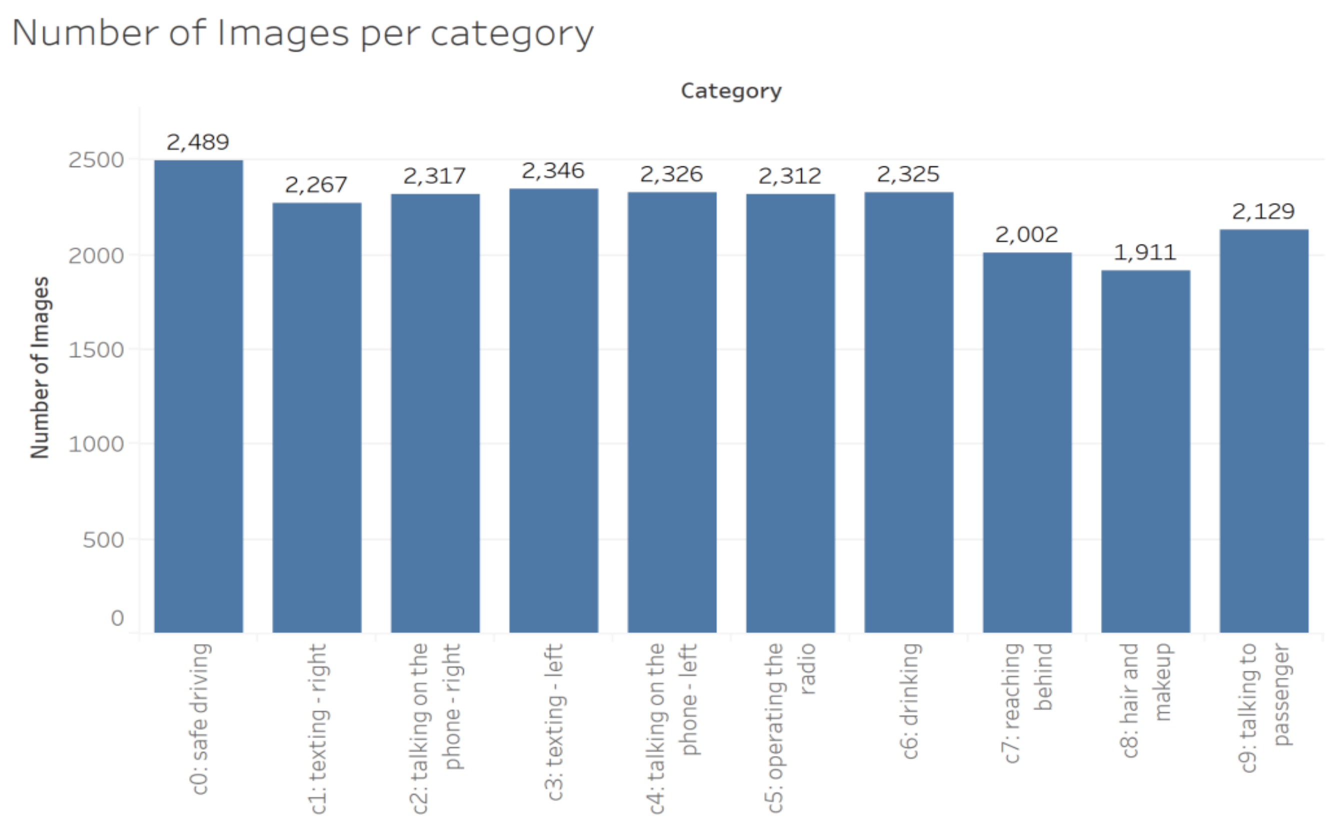
II. Analysis

A. Data Exploration

There is a total of 22424 images in the train dataset and 79726 images in the test dataset. All images are colored and 640 x 480 pixels in size. The images capture the driver from a side-view dashboard camera. An excel file was provided and it links images with their respective drivers as well as each image respective classification. From the excel file, we can come to know that of the 22424 train images, there are only 26 unique drivers. The excel however does not give out any information about the test dataset and because of this, we would have to treat the test dataset as unseen data. Sample image :

:

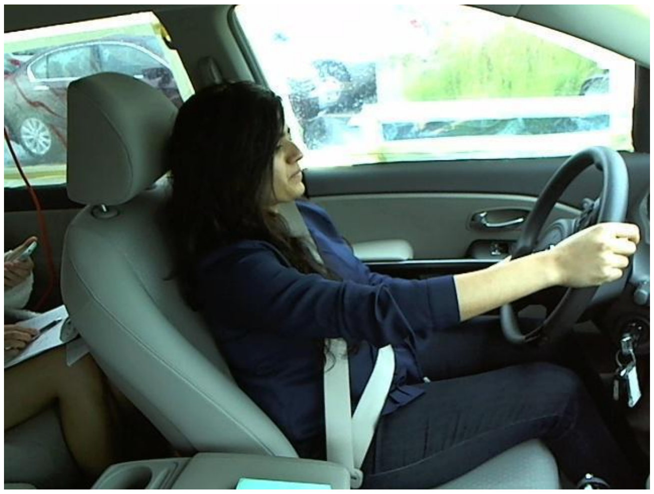
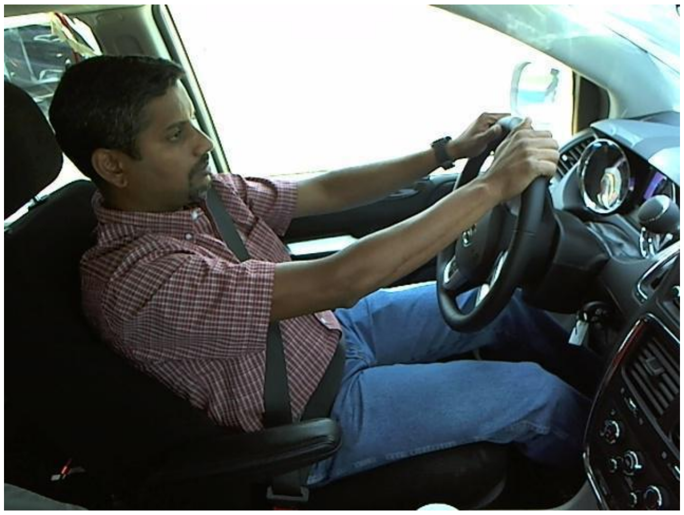
There is one characteristic about the input data that may need to be addressed. I noticed that some categories have more images than the others as we can see from the chart below.



The number of images per category may have to be reduced to 1911 because we would not want our model to favor one category over the other.

B. Exploratory Visualization

One interesting quality about our data is that each image differs from one to the other despite belonging to the same classification as we can see from image comparison below:

both images belong to the same category c0: safe driving behavior. In the first image, we can see the seat of the driver and even part of the passenger behind while in the second image, we cannot see the passenger behind or even part of the seat. Another noticeable difference is the hand placement, the first image has the driver holding the wheel with one hand while in the second image the driver is using both hands.

It is interesting because these differences and variations work in our favor and would help the model generalize better without needing too many additional augmentations. Other noticeable differences include:

1. Camera angles
2. Driver
3. Vehicle
4. Seat adjustments
5. Clothing
6. Hand placement
7. Location
8. Lighting

C. Algorithms and Techniques

I have decided to use Deep Learning - Convolutional Neural Networks(CNN) to solve this problem. CNNs are known for their extraordinary potential of solving image classification problems. The reason it is so powerful is because CNN's uses

network of neurons to learn features and patterns similarly to how the brain works.
This is a more in-depth look at Neural networks structure

1. Convolutional Layers – This is the layer responsible for the learning of features. What this layer does is extract features from images by doing matrix calculations. Initial layers extract the more common features such as edges and color. The deeper the network gets(more convolutional layers), the more complex features are learnt (features that are not entirely obvious). The size of the output matrix depends on the set filter size as well as the stride. Stride is what decides how many pixels to skip before it makes the next set of matrix calculations. Every filter has a set of weights that are computed on to the input layer, these weights provide a measure for how a set of pixels closely resembles a feature. Some features are better understood by the model when they are not too compressed while other features would need to be compressed further to learn new features.

2. Pooling Layers – These are the layers responsible for reducing the dimensionality of images. Reducing the dimensionality means we remove excess parameters to make images easier to interpret for the next layer. This is normally done using max pooling, which selects the highest value of the matrix based on the filters and stride. And as such, these layers tend to give a more pixelized version of an image, but these pixels are better interpreted by the machine. It is like blurring out the haystack, in order to make it easier to find the needle.

3. Fully connected Layers- These are layers that convert our complex matrices into more manageable outputs. With this access, we can control the number of outputs we need based on our problem by squishing, transforming and combining features into classifications. Without these layers, the model would not understand ‘how many’ or ‘what’ predictions we actually want or need.

The main model architecture that will be used is a Keras Application Model. These architectures have been perfected to classify images. Every layer and parameter in these architectures were professionally selected. Keras also has the option of making use of pretrained weights based of VGG16 net. Making use of these pretrained weights can drastically reduce the time it will take to train and optimize the model.

1. Keras Application Models

   This is the main architecture that will be used to train our images. Architectures that I will try are VGG16.

2. Neural Network Architecture
   1. Number of layers
   2. Layer types – Includes dense, Pooling, Convolutional layers
3. Layer Parameters

3. Preprocessing Parameters (see the Data Preprocessing section)
4. Training Parameters
   1. Epochs- Training length
   2. Batch size- Number of tensors/images to be trained per epoch.
   3. Activation- The optimization algorithm to use for learning. Activation functions include softmax, elu, selu, softplus,softsign, relu, tanh, sigmoid, hard sigmoid and linear.
   4. Learning Rate- The learning speed of the algorithm

5. Callbacks
   1. Early Stopping- Technique used to reduce the training time when validation loss is not decreasing anymore.
   2. Model Checkpoint- Technique used for saving important weights.

D. Benchmark

The benchmark result will be based on the Kaggle public and private leaderboard as I was unable to find a benchmark model as such. Since everyone in the leaderboard must follow the same rules and evaluation metric, it makes it good for benchmarking. My target result for this project is to reach the top 10% (≤ 144 of

1440) people with a public log loss 0.24859 and/or a private log loss ≤0.25634. The target result is based on the log loss value of the predicted labels against the actual labels of the 79726 test images.

III. Methodology

A. Data Preprocessing

For the number of images per category issue, several attempts were made to balance the number of images by either removing random images or selecting drivers with excess images per category. None of these attempts yield positive results, in fact there was a noticeable dip in performance. I suspect that since we only have 26 drivers, every image plays an important role in building a robust model.
These are the data preprocessing steps that positively impacted results and were applied on the data:

1. Images are converted into 3 channels, RGB – This preprocessing is done so models may use the 3 channels to learn features with the objective of improving accuracy and log loss.
2. Images are resized to 224 x 224 – Resizing images makes it easier to load on memory at the cost of losing some details.
3. Image labels are converted to categorical integer features/vectors – This is done using the one-hot scheme. This encoding is needed for feeding categorical data to our models because it is the most practical way for models to read categorical data.
4. The list of Images is shuffled– Randomizing images is simply done to change the default order.
5. The list of images are divided into a train set and validation set - This division is important so that we could validate if our model is improving or not when it is training.
6. Pixel values are converted to 32-bit floats – This is done so we could rescale our images.

7. Pixel values are divided by 255 – We divide our data by 255 because it is the maximum RGB value and we want our data to be within the range of 0 and 1. This is done to normalize our data.

B. Implementation

This first implementation followed most of the steps outlined in the problem statement and the solution is as follows:

1. Import data
    - a. Implementation: Created a function that would read all images. One function for reading the train dataset and another function for reading the test dataset.
    - b. Complications: It is a time-consuming and memory expensive process.

2. Split train dataset into train and validation subsets
    - a. Implementation: Split the dataset using train_test_split function of sklearn.

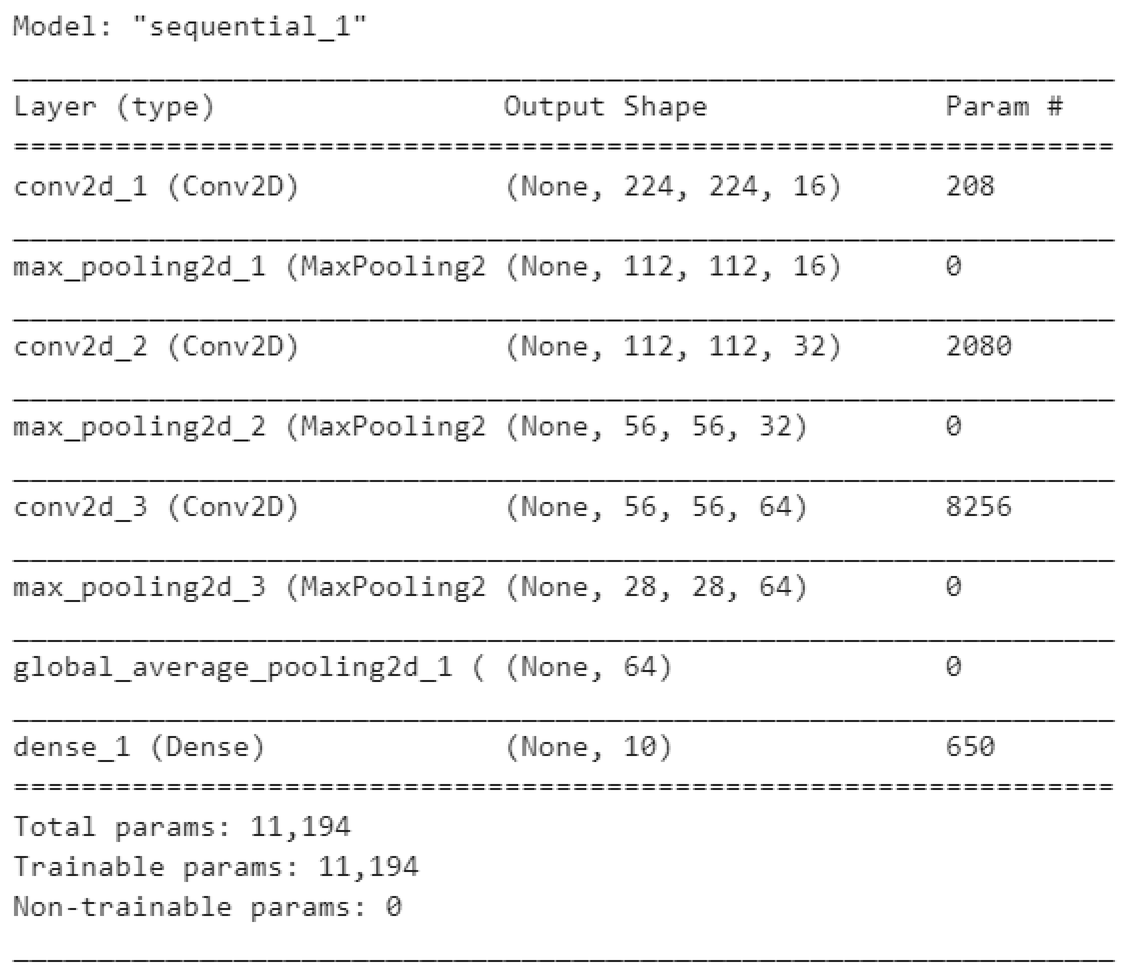
3. Preprocess dataset
    - a. Implementation: Data preprocessing is done as explained in the previous section.
    - b. Complications: The data preprocessing is memory expensive.

4. Create Model Architecture
    - a. Implementation: Built my own architecture without using any Keras Application Models.
    - b. Built another model using transfer learning (VGG16 net).

5. Train model
    - a. Implementation: Used Keras fit function to train the model.

6. Test model
    - a. Implementation: Used the validation set to test the model locally. Then when I was not satisfied, I tested the model on the test dataset and I got bad results , accuracy = 16.9441% .

Initial Solution Architecture

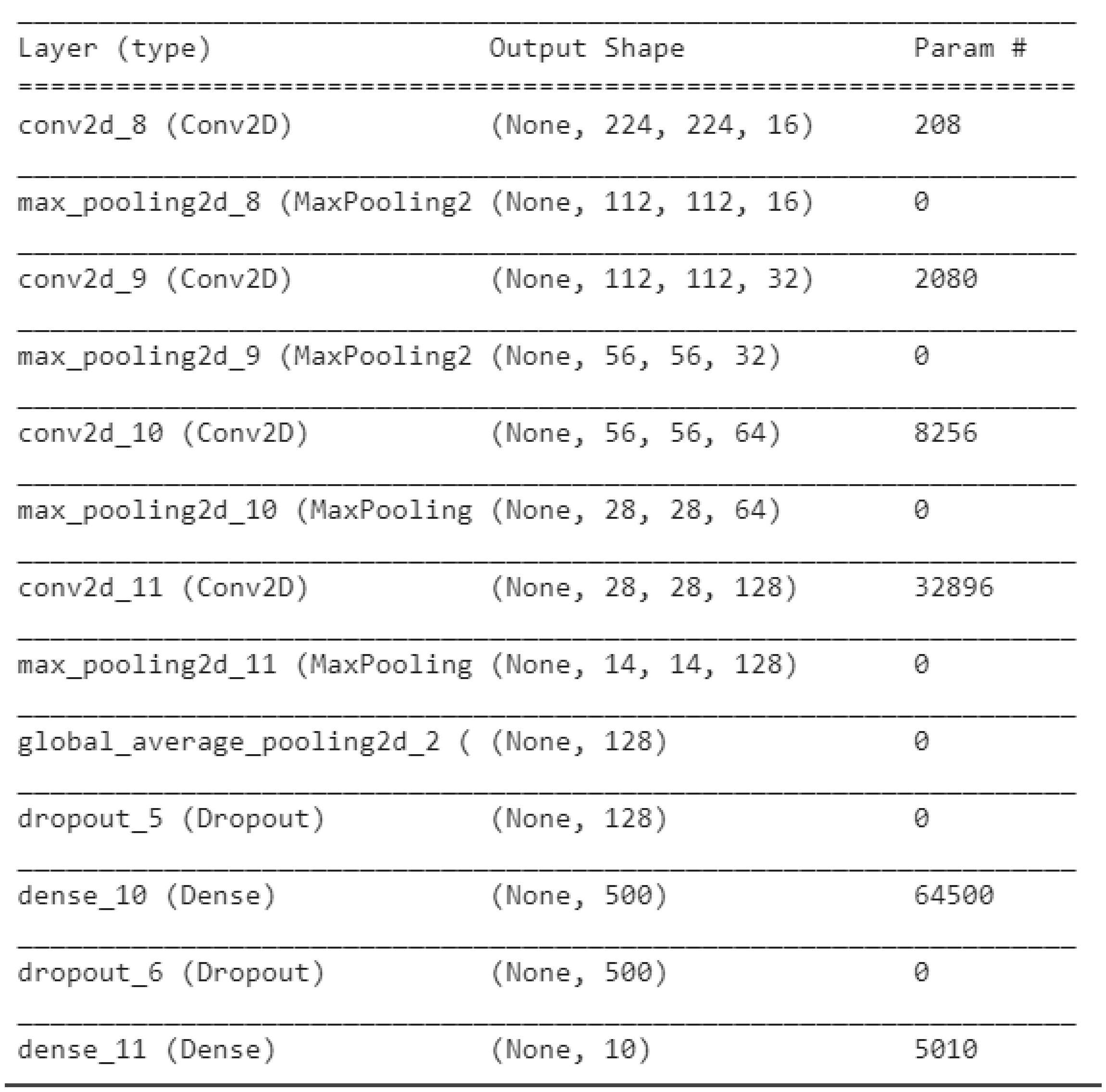


```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 224, 224, 16)      208
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 112, 112, 16)      0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 112, 112, 32)      2080
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 56, 56, 32)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 56, 56, 64)        8256
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 28, 28, 64)        0
_________________________________________________________________
global_average_pooling2d_1 ( (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 10)                650
=================================================================
Total params: 11,194
Trainable params: 11,194
Non-trainable params: 0
_________________________________________________________________
```

C. Refinement

The first solution as seen in the previous section was a solution that left all parameters in its default state. Parameters that had no default state were selected at random. This solution was simply created to get things started and begin the refinement process. The best log loss this implementation achieved in the whole test dataset is 2.23708.

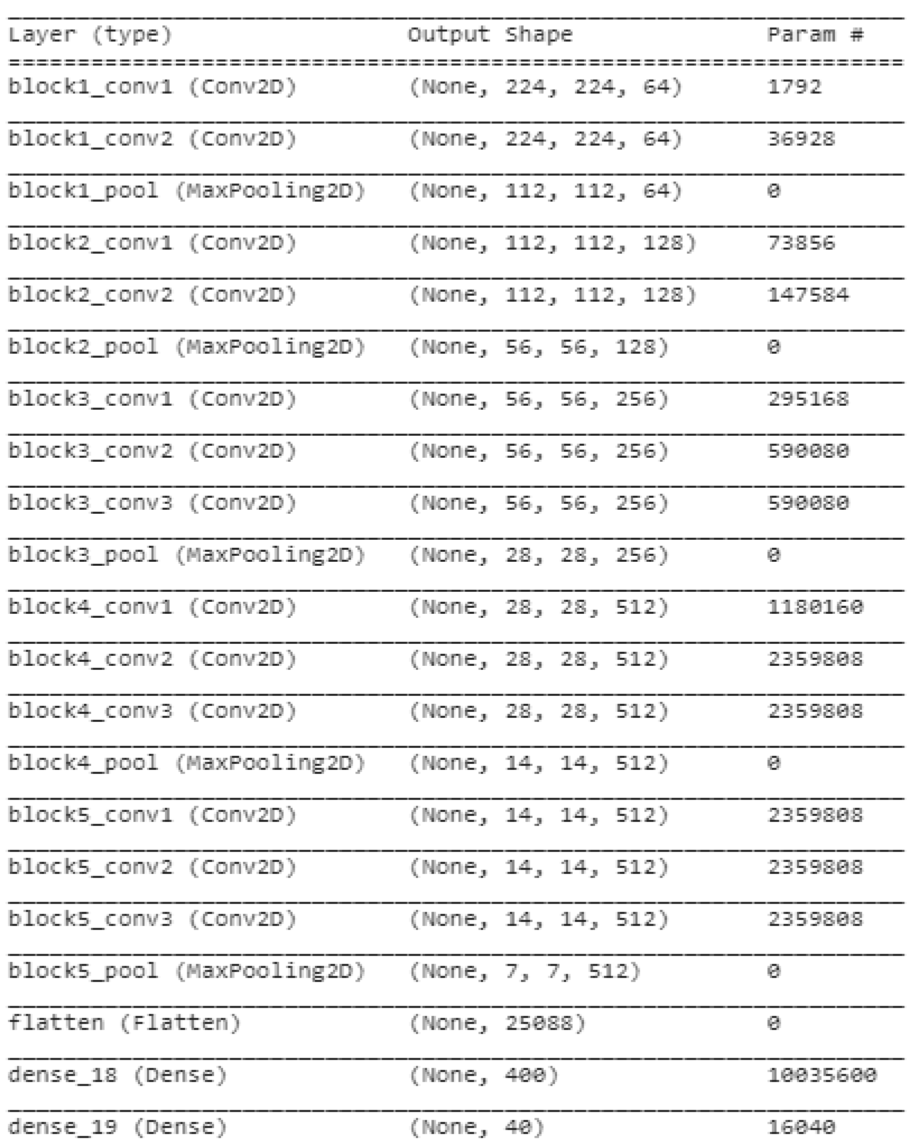
I made some modifications to this model , First I added another convolution layer . Second , I added another dense layer with relu activation . I also added dropout to the mode to decrease overfitting .



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_8 (Conv2D)            (None, 224, 224, 16)      208       
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 112, 112, 16)      0         
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 112, 112, 32)      2080      
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 56, 56, 32)        0         
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 56, 56, 64)        8256      
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 28, 28, 64)        0         
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 28, 28, 128)       32896     
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 14, 14, 128)       0         
_________________________________________________________________
global_average_pooling2d_2 ( (None, 128)               0         
_________________________________________________________________
dropout_5 (Dropout)          (None, 128)               0         
_________________________________________________________________
dense_10 (Dense)             (None, 500)               64500     
_________________________________________________________________
dropout_6 (Dropout)          (None, 500)               0         
_________________________________________________________________
dense_11 (Dense)             (None, 10)                5010      
```

The lowest log loss that it achieved on the whole test dataset is 0.34090, and I got accuracy = 89.7295%. The model does well but it was still not enough to beat the benchmark result of 0.248.

Then , I tried transfer learning using VGG16 net , I got very good results.
I removed the last three layers of the VGG16 net and added three dense layers. The first two of them has relu activation and the third has softmax. I freezed the pre-trained layers and trained only the last 3 layers with 5 epochs , batch size 20. This lowest loss = 0.01525, and the highest accuracy = 99.5392% .



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
dense_18 (Dense)             (None, 400)               10035600
_________________________________________________________________
dense_19 (Dense)             (None, 40)                16040
```

IV. IV. Results

A. Model Evaluation and Validation

1. The model input uses the default shape of our keras application model. The shape is 224, 224, 3.
2. The Keras application model I used and selected is VGG16.
3. This decision was entirely based on the validation results

4. The output of the Keras application model is then applied to 3 dense layers
5. Applied 3 dense layers with 400 , 40 , 10 units , respectively. This is done to change the dimensionality of the vectors and is basically used to filter out

features. Tried with several other units and combination of layers (dropout and other dense layers) but none yield better results

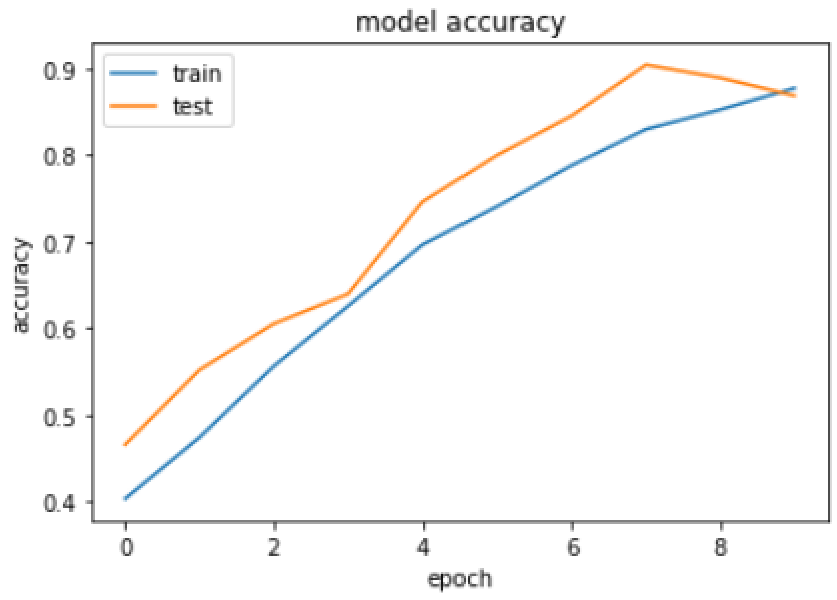
6. The dense layer makes use of the ‘relu’ activation. This activation was recommended in the Udacity course because training is faster and overall gives the best results

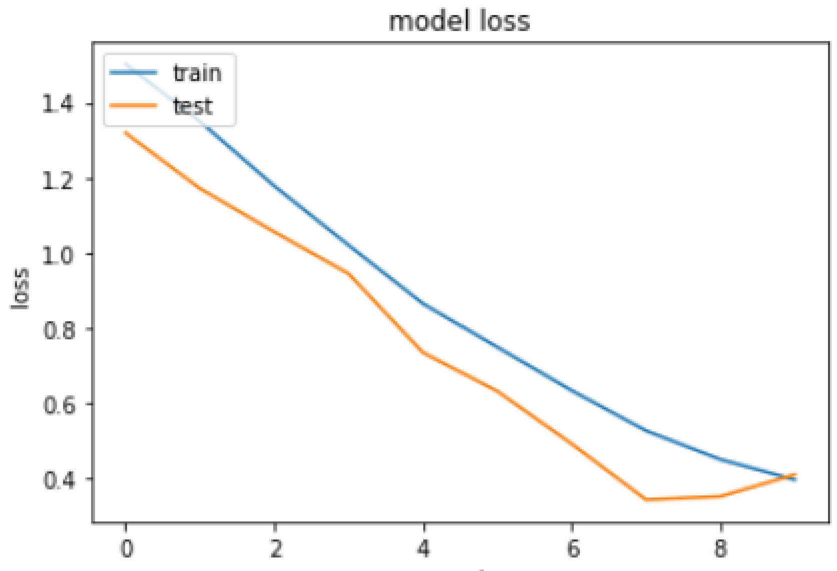
7. A ‘softmax’ activation is used in the last dense layer because it yields the best results for a multi classification problem. I tried and tested with other activation functions, but no activation does better.

8. The model architecture is then compiled with the optimizer ‘Adam. This optimizer was selected after looking through all available optimizers.

9. The ‘Adam’ optimizer has the learning rate of 0.0001. At this speed the optimizer learns important features rather quickly.

To verify the robustness of the final model, I plotted the model accuracy vs epocs and train and test loss vs epocs.





We can see that VGG19 model has a good validation loss and accuracy.

B. Justification

The final model does better than benchmark result. The Kaggle results show that the final solution achieved a public log loss of 0.23292 while the benchmark has a public log loss of 0.24859. For the private leaderboard, the solution got a log loss of 0.01525 in comparison to the benchmark log loss of 0.25634. In terms of placement, I would have been in the top 8.34% of the public leaderboard and top 6.32% of the private leaderboard.

V. Conclusion

A. Free-form Visualization

The importance of using K-fold cross validation to solve this problem can be seen, as well as the slight improvements of using 2 models. The validation loss differences of one iteration to the other shows us, that all pictures have important features. If the images were very much alike, the validation loss would have stayed constant and it’s because of this that we were not able to get good results by to removing images to maintain an equal sample size across categories.

B. Reflection

This may serve as a summary of the entire process I used for this project

1. Find a relevant problem that machine learning can solve
2. Understand the data input
3. Research similar problems and their solutions
4. Identify the most efficient high-level solution for the problem
5. Create a benchmark result
6. Implement a simple solution

7. Identify problems areas that stop the implementation from reaching desired results.
8. Research and list down solutions to the problems listed
9. Select the top 5 solutions
10. Test each solution
11. Apply and refine working solutions
12. Iterate steps 6 to 11 until satisfied with results

The most interesting aspects of this project are:

1. Understanding all the different techniques and optimizations to solve the problem. I used to think that just applying random changes to the model would help me get good results, but in this field, we need to make sure we know exactly what we are doing so we could achieve desirable results.

2. Importance of understanding the logic behind each parameter. I am a bit more confident in my Deep Learning skills

3. All the difficult aspects

The most difficult and interesting aspects of this project are:

1. Trying to achieve optimal results was a real challenge yet the most fun part. I had to make use of all the best practices and optimizations I could find before I could get even close to the benchmark result. I spent 3 weeks working on this problem because each time I got closer to the benchmark, It got a lot harder to improve results. I was at the brink of giving up because I had exhausted all types of optimizations, normalization and regularization techniques and parameter combinations with no significant positive results. Never give up.
2. Another complicated aspect of this project was the cause and effect. An example of this is that by changing parameter A, I would have space to optimize parameter B but doing so would negatively affect parameter C. Trial and error was key to helping me understand the relationship of one parameter to the other.

The final model met my expectations for the problem and I believe there are many techniques and optimizations used here that would work on similar problems however I also believe that there are several improvements that could be made to come up with better predictions and this is discussed in the next section.

C. Improvement

If I were to use my solution as the new benchmark, beating it would certainly be a challenge. However, there are some solutions that are significantly better than this one as we can see from the Kaggle leaderboard, but it would require someone with either more experience, better equipment or more time.
There are some improvements that would have worked had I known how to implement them. I would have tried studying these methods, but I ran out of

money to extend the course. I will however try to properly implement them after passing this project. These are the improvements that would have worked:

1. One technique is exploring further the multiple models. This technique is used to improve the generalization and overall prediction accuracy. I implemented a version of this, but it could have done better. In my case both models were learning the same features and used similar parameters. Training multiple models on different portions of the image would have yield better results.
2. Another technique that would have worked is unsupervised learning for our neural networks. We have a lot of test data that was not used to train our images, if we could make use of these images, our model would have done significantly better.
3. Using other types of merging/ average techniques. Results could have improved had I known how to implement something like the K-nearest neighbor average or the Segment/category average for deep learning. These techniques are known to do better than the simple average especially when using K-fold or Multiple Models.
4. Hiding unimportant portions of the image. This would help the model learn only the important parts of the image. In theory it should improve the training speed and increase the accuracy.

6. Rescaling images closer to its original size. Important features may have been lost by rescaling them to a 224 x 224 and I did not have the necessary hardware to effectively test this.
7. Decreasing the batch size. In theory, the model would generalize better at the cost of training time.

Sources:

Motor Vehicle. (n.d.). In Merriam Webster’s online dictionary. Retrieved October 25, 2017, from https://www.merriam-webster.com/dictionary/motor%20vehicle

Distracted Driving. (2017). In Centers for Disease Control and Prevention. Retrieved October 25, 2017, from https://www.cdc.gov/motorvehiclesafety/distracted_driving/

Road Traffic Injuries. (2017). In World Health Organization. Retrieved October 25, 2017, from http://www.who.int/mediacentre/factsheets/fs358/en/

Distracted driving global fact sheet. (n.d.). In U.S. Department of Transportation National Highway Traffic Safety Administration. Retrieved October 25, 2017, from http://usdotblog.typepad.com/files/6983_distracteddrivingfs_5-17_v2.pdf

Annual Global Road Crash Statistics. (n.d.). In Association for safe international road travel. Retrieved October 25, 2017, from http://asirt.org/initiatives/informing-road-users/road-safety-facts/road-crash-statistics

Ward, Marry. (2007). In Offaly History. Retrieved October 25, 2017, from http://www.offalyhistory.com/reading-resources/history/famous-offaly-people/mary-ward-1827-1869

Automobile History. (2010). In History. Retrieved October 25, 2017, from http://www.history.com/topics/automobiles

Texting and Driving Accident Statistics. (n.d.). In Edgar Snyder. Retrieved October 25, 2017, from https://www.edgarsnyder.com/car-accident/cause-of-accident/cell-phone/cell-phone-statistics.html

Andrew B. Collier(2015)Making sense of Logarithmic Loss. In Exegetic. Retrieved Octover 26,2017 from http://www.exegetic.biz/blog/2015/12/making-sense-logarithmic-loss/

I State Farm Distracted Driver Dataset. (2016). In Kaggle. Retrieved October 25, 2017, from https://www.kaggle.com/c/state-farm-distracted-driver-detection/data