Capstone: State Farm Distracted Driver Detection

Natraj Rajput January 23, 2018

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, augment the dataset by adding images with noise. Step 5, 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 6, fit the model and save the resulted weights. Step 7, test using the validation subset. Step 8, validate results/accuracy. Step 9, Plot the train and test dataset against the fitted values/epoch to see how much the model is overfitting. Step 10, Predict the labels of the test dataset then submit it to Kaggle. Step 11, compare Kaggle result to see if it meets benchmark results. Step 11, Adjust architecture, model, parameters and data augmentation. Step 12, Repeat until it meets target benchmark results. Step 13, 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.

Sample Kaggle output:

img	c0	c1	c2	c3	c4	c5	c6	c7	c8	c9
img_1.jpg	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
img_10.jpg	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
img_100.jpg	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
img_1000.jpg	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
img_100000.jpg	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
img_100001.jpg	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
img_100002.jpg	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
img_100003.jpg	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
img_100004.jpg	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
img_100005.jpg	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
img_100007.jpg	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1

Multi class log loss example:

```
>LogLossMulti (["bam", "ham", "spam"], [[1, 0, 0], [0, 1, 0], [0, 0, 1]])

[1] 2.1094237467877998e-15

>LogLossMulti (["bam", "ham", "spam"], [[0, 0, 1], [1, 0, 0], [0, 1, 0]])

[1] 34.538776394910684

>LogLossMulti (["bam", "ham", "spam", "spam"], [[0.8, 0.1, 0.1], [0.3, 0.6, 0.1], [0.15, 0.15, 0.7
], [0.05, 0.05, 0.9]])

[1] 0.2990
```

From this example, we can see that when the prediction is completely the same as the actual value, the log loss results to a 2.11e-15 and when the prediction is completely wrong, the log loss reaches 34.54.

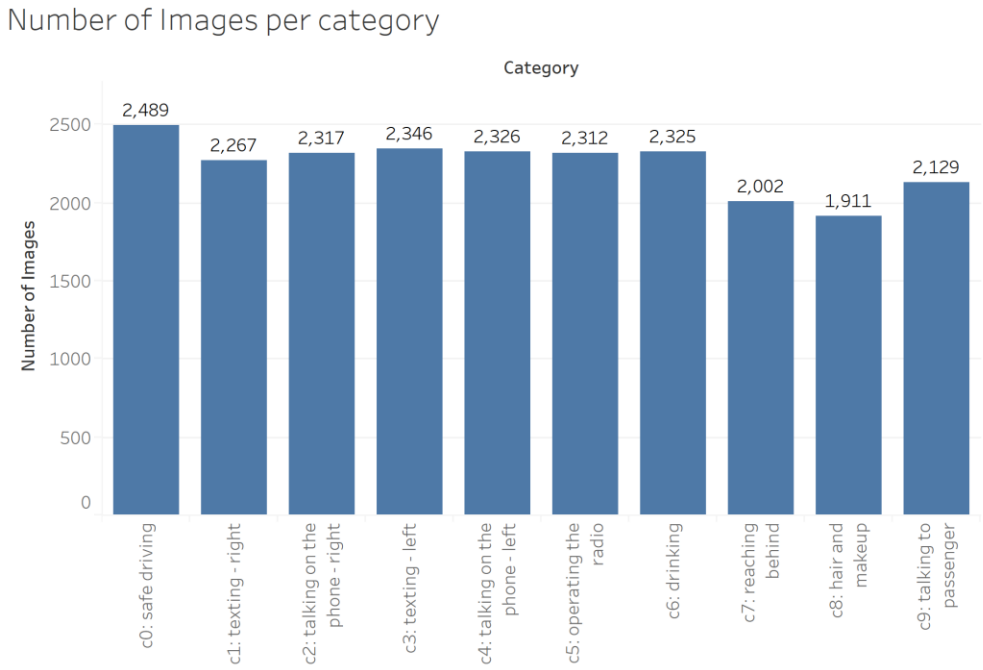
II. Analysis

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. Normalization Layers – These are layers used to counteract the change of direction and distribution affecting our dataset each time a new layer is added. The normalization is done to realign the model input to what it is was originally intended to do – learn new features. Not applying a normalization may affect our model to only learn some features while ignoring other features, this is especially true when new data is being fed. An analogy of the problem would be; studying one lesson so much you end up forgetting to study other areas

for the quiz. So, the solution is reminding yourself the end goal and reassess 'what' you need to study.

4. 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 **ImageNet**. Making use of these pretrained weights can drastically reduce the time it will take to train and optimize the model.

The algorithms and techniques I intend to use or ended up using are:

1. Keras Application Models
    1. Model -This is the main architecture that will be used to train our images. Architectures that I will try are VGG16, VGG19, ResNet50, Xception, Inceptionv3, InceptionResNetv2, MobileNet.
    2. Weights- Random initialization or pretrained on 'ImageNet'.
2. Neural Network Architecture
    1. Number of layers
    2. Layer types – Includes Core, Pooling, Convolutional, Normalization and Noise 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. Weight Decay- Regularization method used to prevent weights from growing too large
    6. Momentum- Technique used for accelerating gradient descent by making use of accumulated velocity.
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.
6. Cross Validation Techniques
    1. Kfold cross validation – Technique used to train and evaluate models by randomly partitioning original samples into k equal sized subsamples. Each subsample is divided into a train and validation dataset.
7. Other techniques

1. Mean/Averaging Predictions- Technique used to improve generalization by averaging out multiple predictions.
2. Multi model training- Technique used to get predictions of multiple models and then apply an averaging technique.

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.
8. Some images are randomly augmented using the following:
    1. Random – Adding a touch of chance is known to improve accuracy and reduce overfitting in deep learning. This can be seen in everything from random weight initializations of models to dropout layers.

2. Augmentations – The model was still memorizing the images despite having some natural uniqueness. These augmentations provide a means of improving generalization:
    a. Zoom
    b. Width Shift
    c. Height Shift
    d. Rotation

Sample Data Preprocessing Outputs:

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.
5. Train model
    a. Implementation: Used Keras fit function to train the model.
    b. Complication: predictions are not good enough.
6. Test model
    a. Implementation: Used the validation set to test the model locally. Then when I was satisfied, I tested the model on the test dataset and submitted to Kaggle to see final results.
    b. Complications: loading the whole test dataset was memory expensive.

Initial Solution Architecture

```
_________________________________________________________________
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, 64)      4160
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 56, 56, 64)        0
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 56, 56, 128)       32896
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 28, 28, 128)       0
_________________________________________________________________
flatten_1 (Flatten)          (None, 100352)            0
_________________________________________________________________
dropout_1 (Dropout)          (None, 100352)            0
_________________________________________________________________
dense_1 (Dense)              (None, 500)               50176500
_________________________________________________________________
dropout_2 (Dropout)          (None, 500)               0
_________________________________________________________________
dense_2 (Dense)              (None, 10)                5010
=================================================================
Total params: 50,218,774
Trainable params: 50,218,774
Non-trainable params: 0
_________________________________________________________________
```

We can see from this architecture that it is a very simple model. To build this model, I just followed the structures and architectures used in some of the examples and exercises of Udacity. I used 3 convolutional 2d layers with incremental filters because I want the model to start learning different features. The 3 max pooling layers are used to reduce the dimensionality of the images. We want to reduce the dimensionality of images, so the layers following it would better interpret our images. A flatten layer is used to combine learnt features (neuron outputs) into one location or array. We want this so the final layers after it can easily access our features. The 2 dropouts are added to counter overfitting by randomly ignoring neurons. The first dense layer is used to filter down/squash our learnt features to 500. The last dense layer is used to filter down our learnt features to 10 because we have 10 classifications. The activations relu and softmax in the dense layers are the ones responsible for squashing and transforming our features into said classifications.

C. Refinement

Please refer to **Solution 1: Simple** Jupyter notebook for my initial solution and **Solution 2: No Iteration** Jupyter notebook for my second solution and lastly **Solution 3: K-fold Implementation** Jupyter notebook for my final solution.

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 7.541.

The second solution had reached its peak performance and could not be further improved despite weeks of research, refinements, trial and error. The lowest log loss that it achieved on the whole test dataset is 0.815. The model does well but it was still not enough to beat the benchmark result of 0.248. The refinements done on the first solution to come up with this solution is as follows:

a. Implemented a function that would save read data into cache files.
b. Made a function that would split the train dataset into 3(train, validation and test), this is so my local test would yield a similar log loss to the actual test dataset.
c. Train, validation and test sets are saved in different cache files to reduce memory usage.
d. Data Augmentations and parameters were based on logic, log loss and the train vs validation plot
    a. Data augmentations that did not work: Dividing the whole dataset by the Train mean ,Featurewise center, Samplewise center, featurewise standard deviation normalization, sample wise standard deviation, ZCA epsilon, ZCA whitening,shear range, channel shift range, horizontal flip, vertical flip
    b. Data augmentations that did work: Normalizing data by rescaling values (255), Rotation range, width shift range, height shift range, zoom range.
e. Selected the application model and weight initialization that resulted to the best log loss
    a. Application models that did not work for me in this solution: VGG16, VGG19, ResNet50, InceptionV3, MobileNet.
    b. Weights that did not work: Random Initialization
    c. Application models that worked: InceptionResnetV2.
    d. Weights that worked: ImageNet
f. Selected the number of layers and layer types and layer parameters that yield the best results
    a. Layers types that worked in this solution: Dense, Activation, Dropout, Pooling
g. Selected the best activation based on the problem and trial and error
    a. Activations that did not work: elu, selu, softplus, softsign, tanh, sigmoid, hard sigmoid, linear
    b. Activations that worked: softmax, relu
h. Selected the most effective optimizer based on its default parameters.
    a. Optimizers that did not work: RMSprop, Adagrad, Adadelta, Adam, Adamax, Nadam
    b. Optimizers that did work: SGD
i. Learning rate, weight decay and momentum were selected and updated based on the train vs validation plot
    a. Learning rate that did not work: 0.1,0.001 ,0.0008, 0.0004, 0.0001, 0.00001
    b. Learning rate that did work: 0.0009
    c. Weight decay that did not work: 0.9,0,1,0.001,0.0001, 0.00001, 0.000001
    d. Weight decay that worked: 0.0
    e. Momentum that did not work: 0.5,0.3,0.2,0.1,0.0
    f. Momentum that did work: 0.9
j. Batch size was selected based on GPU memory capabilities
    a. Batch size that did not work: 2,16,64,50,80, 100,128
    b. Batch size that did work: 32
k. Epochs was selected based on the log loss improvement

    a. Epochs that did not work: 2,5,10,20,30,50
    b. Epochs that worked: 15
l. Created a function that splits the test dataset into 5. This is done to reduce the memory usage.

For this final solution, most parameters mentioned in the previous solution had to be tweaked after adding the new refinements. This is due to the better understanding of the problem that was yielding poor results and after months' worth of iterations, trial and error.

To get the final solution, these are the new refinements that were done:

a. Applied the K-fold cross validation technique to maximize the learning of train dataset. I tried different number of K-folds including 2,3,5,7,9,10,12,15. 10 folds yield the best results.
b. Used the Ensembling or Mean/Averaging predictions technique to make predictions. This is used to improve the generalization capabilities by averaging the prediction results of the model with different weights created from using the K-fold cross validation technique.
c. The next refinement was the adding of the Early stopping technique. This technique reduces the time it takes to train when there are no notable validation loss improvements.
d. One other technique was sorting out the images by the drivers to counter overfitting. It appears that the model was overfitting and memorizing the drivers so when exposed to new drivers it would not do as well.
e. Made use of batch normalization technique to provide the final layer of the neural network an input that has zero mean and standard deviation close to one. This is done to train the network faster and generally improve the accuracy.
f. The final refinement is the addition of Multiple models. I trained multiple models then averaged the result.
    a. Combinations that did not work: VGG19+ InceptionResnetV2. VGG19 +InceptionResnetV2 + ResNet50. VGG16+VGG19
    b. Combinations that did work VGG19+ResNet50

IV. Results

A. Model Evaluation and Validation

The final model architecture and parameters were selected based on the validation loss and test results. Multiple tests and refinements had to be done on the model and parameters before I could achieve optimal results.

The model architecture and parameters are as follows:

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 ResNet50 and VGG19.
3. This decision was entirely based on the validation results. I tried and tested all Keras application models from VGG16 to MobileNet.
4. Made use of the 'ImageNet' (pre-trained) weights for the Keras application model because the random initializations option did not provide results anywhere near that of 'ImageNet'.

Also using pre-trained weights option is known to yield decent results when working on similar classification problems without needing to spend too much time training the model.

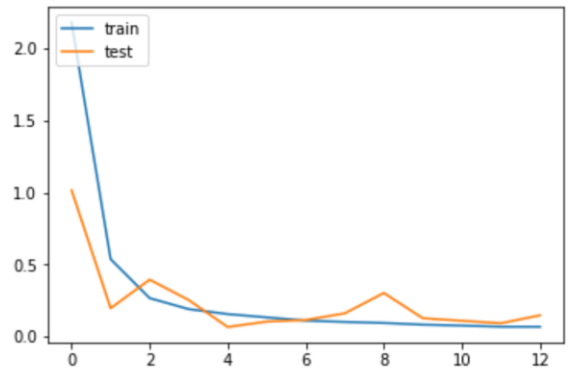
5. The output of the Keras application model is then applied a Global Max Pooling 2D layer because it used to reduce the dimensionality of the data and partly reduce overfitting. Using a Global Average Pooling 2d does only slightly worse and not using either would significantly (negatively) affect the end results.
6. Applied a dense layer with 512 units. 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
7. 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. I tried all other available activations like 'selu' and 'softplus'. Some did at times provide better test results, but the results differed significantly from one test to the other despite setting a random seed.
8. The next layer added is a Batch Normalization layer. This layer is used to normalize the data. This is a technique helps train the network faster and improve the results.
9. The last layer is a dense layer with 10 units. The units should match the number of classification we need. This is common practice in CNN's.
10. 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.
11. The model architecture is then compiled with the optimizer 'SGD'. This optimizer was selected after looking through all available optimizers.
12. The 'SGD' optimizer has the learning rate of 0.001. At this speed the optimizer learns important features rather quickly. If we were not to apply Batch Normalization, we would need to reduce the learning rate to 0.0009.
13. Applying a Momentum of 0.9 to our optimizer simply aims to speed up the learning process.
14. Applying the Nesterov parameter enhances the momentum parameter into reaching the best results even faster.

To verify the robustness of the final model, I first look at each of the 10 K-fold train vs validation plots like the ones we can see below.

Example Train vs Validation plot of 2 different K-folds:

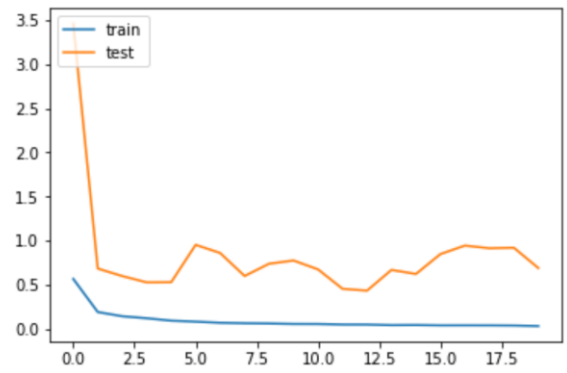
VGG19 9th Kfold

```
Epoch 00012: val_loss did not improve
```



VGG19 10Th Kfold

```
Epoch 00019: val_loss did not improve
```



We can see that the plot results differ from one iteration to the other, but they both are satisfactory (it does not completely overfit or underfit, this is especially True when we ensemble the results of all K-folds). Maintaining a low validation and train log loss is the first indication that proves we have a robust model. The next thing I look at is at each model's validation set log loss (as seen below), the validation set is the very same data used to validate our model during training. I did not split the data to create a separate test dataset just because doing so, gave me poor results. I used this validation score as my basis to changing or tuning variables.

```
***FINAL MODEL RESULTS/SUMMARY***
VGG19 Final log_loss: 0.2803574772810449, nfolds: 10 epoch: 20
ResNet50 Final log_loss: 0.40494291567483126, nfolds: 10 epoch: 20
Multi Model Log_loss Validation Score: 0.24721512995793982
```

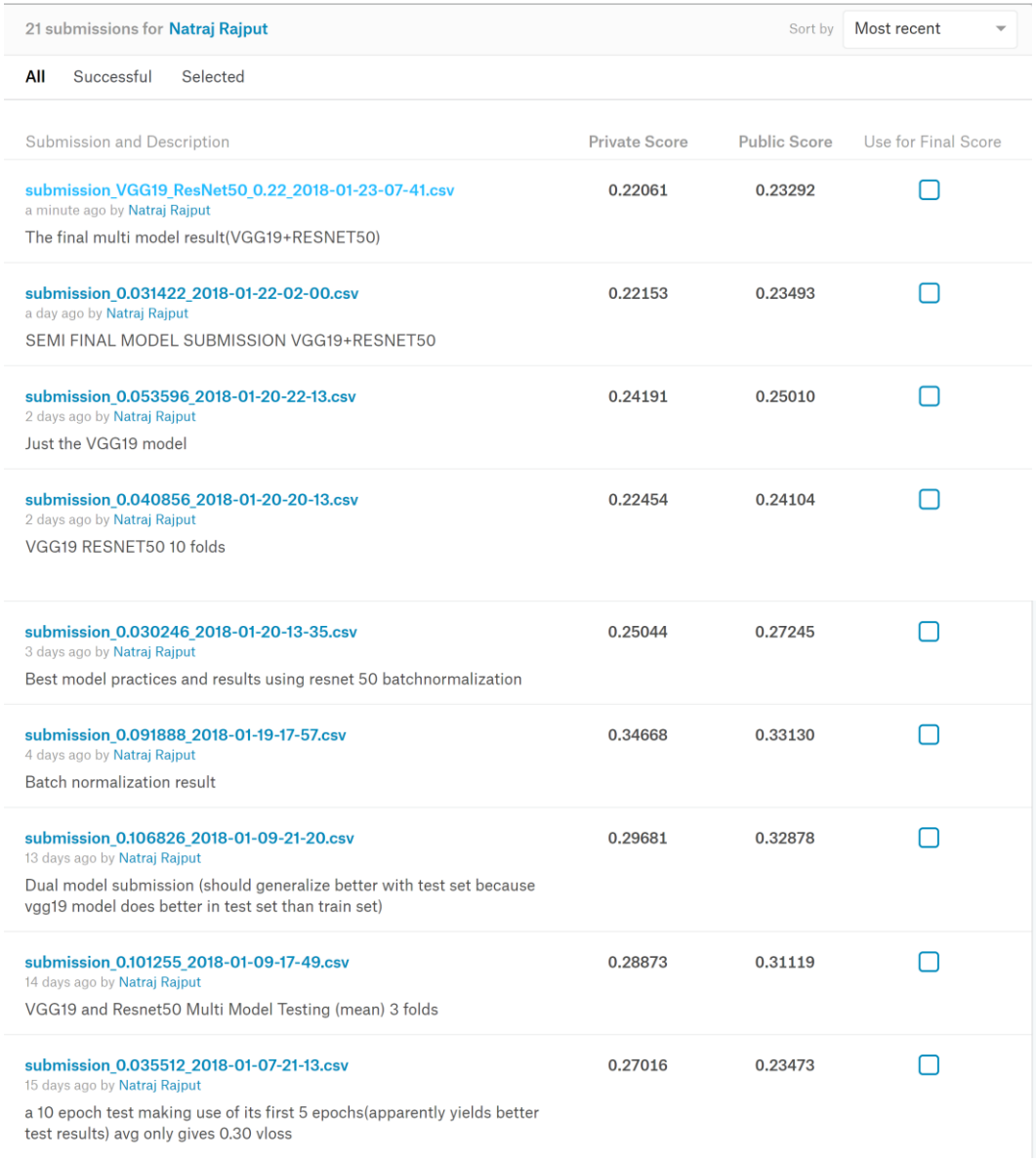
We can see that VGG19 model has a good validation loss. On the other hand, the ResNet50 model falls behind quite a bit. The summary also shows that the validation score improves after ensembling the results of both models. Once I was satisfied with the validation scores, I then tested the model on the Kaggle test dataset and submitted it. The Kaggle Test Dataset contains 79726 unseen images. The Kaggle results show that our model has a public log loss score of 0.23292 and private score of 0.22061. As you can see, the score is not far off our local validation score which is good. After taking all these results into consideration, we can be confident that our solution is robust and good at generalizing unseen data. I was unable to provide my own images because I just recently moved to a different country and here I do not own a vehicle. Small changes (aside from the necessary data augmentations) on the train data does significantly impact (negatively) the end results because these changes would affect the learning of key features, start learning arbitrary features and ultimately yield bad results.

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 0.22061 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. As we can see, our solution does not do significantly

better than the benchmark, but it is at least good enough to solve the problem of categorizing drivers into the 10 behaviors with a smaller error margin.

Kaggle Results:



21 submissions for Natraj Rajput — Sort by: Most recent

All | Successful | Selected

Submission and Description	Private Score	Public Score	Use for Final Score
submission_VGG19_ResNet50_0.22_2018-01-23-07-41.csv a minute ago by Natraj Rajput The final multi model result(VGG19+RESNET50)	0.22061	0.23292	☐
submission_0.031422_2018-01-22-02-00.csv a day ago by Natraj Rajput SEMI FINAL MODEL SUBMISSION VGG19+RESNET50	0.22153	0.23493	☐
submission_0.053596_2018-01-20-22-13.csv 2 days ago by Natraj Rajput Just the VGG19 model	0.24191	0.25010	☐
submission_0.040856_2018-01-20-20-13.csv 2 days ago by Natraj Rajput VGG19 RESNET50 10 folds	0.22454	0.24104	☐
submission_0.030246_2018-01-20-13-35.csv 3 days ago by Natraj Rajput Best model practices and results using resnet 50 batchnormalization	0.25044	0.27245	☐
submission_0.091888_2018-01-19-17-57.csv 4 days ago by Natraj Rajput Batch normalization result	0.34668	0.33130	☐
submission_0.106826_2018-01-09-21-20.csv 13 days ago by Natraj Rajput Dual model submission (should generalize better with test set because vgg19 model does better in test set than train set)	0.29681	0.32878	☐
submission_0.101255_2018-01-09-17-49.csv 14 days ago by Natraj Rajput VGG19 and Resnet50 Multi Model Testing (mean) 3 folds	0.28873	0.31119	☐
submission_0.035512_2018-01-07-21-13.csv 15 days ago by Natraj Rajput a 10 epoch test making use of its first 5 epochs(apparently yields better test results) avg only gives 0.30 vloss	0.27016	0.23473	☐

submission_0.101981_2018-01-03-21-51.csv 19 days ago by Natraj Rajput A simple kfold with only 10 epochs and 3 folds	0.32049	0.34019
submission_0.05_2018-01-03-15-45.csv 20 days ago by Natraj Rajput Kfold application using VGG19 Model GlobalMaxPooling 25 epochs 10 folds	0.24915	0.24118
submission_0.890_2017-12-02-10-21.csv 2 months ago by Natraj Rajput Uses the inception v3 model with an added 512 dense and 10 dense layers	0.81543	0.98299
submission_1.564_2017-11-16-20-53.csv 2 months ago by Natraj Rajput Trial submission with enhanced training dataset and improved memory efficient test dataset	1.62077	1.95230
submission_2.311962_2017-11-10-17-51.csv 2 months ago by Natraj Rajput overfitted ??? validation just makes use of a different dataset. the preprocessing of this one is improved	2.28788	2.27233
submission_0.003512_2017-11-10-04-23.csv 2 months ago by Natraj Rajput separated train,test and val drivers OVERFITTED VERSION	7.52116	7.20146
submission_0.017225_2017-11-09-00-38.csv 2 months ago by Natraj Rajput This submission aims in the reduction of ram memory usage but maintaining accuracy	11.18825	9.86915
submission_0.050671_2017-11-08-16-54reordered.csv 2 months ago by Natraj Rajput as name is stated, reordered submission of trial 1(not resized nor dtype float32)	29.07587	28.91903
submission_1.050671_2017-11-08-16-54.csv 2 months ago by Natraj Rajput dataset with float 32 and resize	7.54143	8.27916
submission_0.050671_2017-11-08-16-54.csv 2 months ago by Natraj Rajput dataset without resize and float32	29.07587	28.91903
submission_0.035573_2017-11-08-01-09.csv 3 months ago by Natraj Rajput v2 with better pre processing	31.41828	31.29326

V. Conclusion

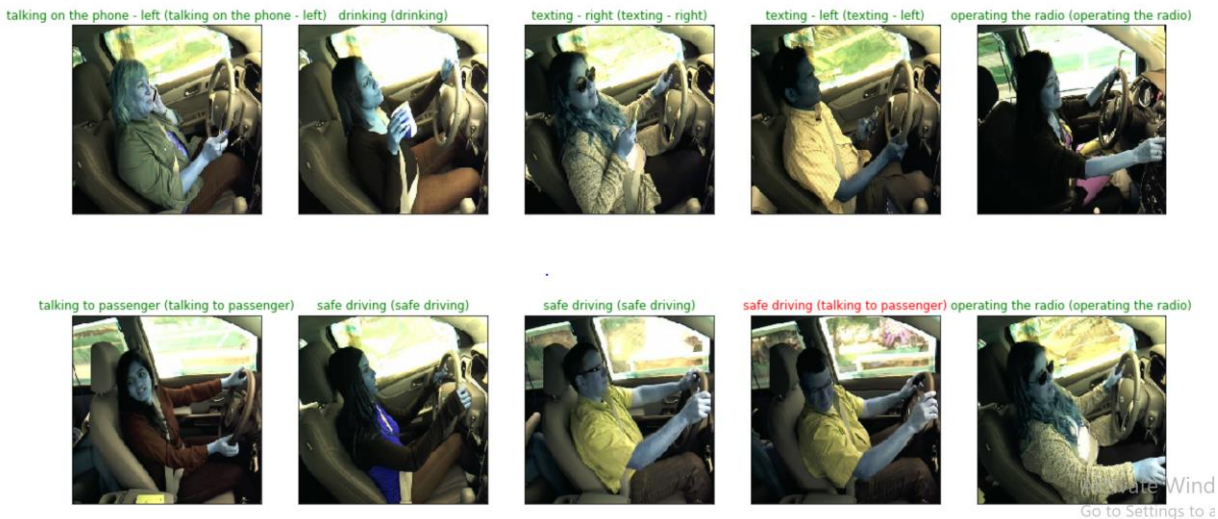
A. Free-form Visualization

There is one interesting quality about the result that can be understood better by looking at the plot below.

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. Now from this graph, we can see that there are times when one model captures features better that the other but it's important to note that neither model outperforms the other. When both models have their validation scores averaged per K-fold, we can see a slight improvement in validation loss. A bigger improvement would be seen when we merge all K-fold predictions into one. Averaging would help bump generalization by neither overfitting on the validation data nor the train data.

These are 10 random images and their predictions per model:

VGG19



ResNet50

VGG19 ResNet50

We can see that all 3 models yield satisfactory results however, we can also see that all models are committing the same mistake. This just goes to show that solution is not perfect. The combined model is simply better at generalizing data, that's why it has better log loss but that does not mean it fixes all the mistakes of the two models.

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 months 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. That's when I ended up trying K-fold, and when I almost gave up there again I found batch normalization and multi model techniques. Never give up.
2. Training the model takes too long. My GPU(GTX 1060) takes around 40 minutes per K-fold iteration to train on just half of the train set and that is using all the best practices to reduce training time. It takes 80 minutes on the whole train set multiplied by the number of K-folds. This would not have been an issue if I had more experience. This coupled with my memory RAM issues made this process even slower.
3. 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.
4. Another difficult aspect of this project was fitting data into memory. I spent weeks trying to implement all sorts of memory optimization techniques, but they were all just not enough. Of course, I could have just reduced the image size, removed some image augmentations and submitted the project with sub-optimal results. But because I really wanted to achieve the results I committed to, I resorted to buying an additional 8 GB of RAM and when that wasn't enough during the K-fold implementation I bought another 8 GB. Even then I used to still run into memory issues, but it was at least enough to get the job done.
5. The validation score does not exactly reflect the Kaggle score. This became especially true when I started implementing K-fold cross validation. It does yield similar results to the

Kaggle score when I use a separate Test dataset or Kfold-1 but doing so affected my log loss significantly.

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. Another technique is the freezing of layers and retraining them.
5. 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/

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

Ritchie Ng. (n.d.). Evaluating a classification model. Retrieved October 30,2017, from http://www.ritchieng.com/machine-learning-evaluate-classification-model/

What is computer vision? (n.d.). In BMVA. Retrieved October 30,2017, from http://www.bmva.org/visionoverview

Quick History of Machine Vision. (n.d.). In EPIC systems. Retrieved October 30,2017, from https://www.epicsysinc.com/blog/machine-vision-history

Hari Narayanan, Libin Sun, Greg Yauney, et al. (n.d.). Introduction to computer vision. Retrieved October 30,2017, from https://cs.brown.edu/courses/cs143/lectures/01.pdf

T. S. Huang. (n.d.) Computer vision: Evolution and Promise. Retrieved October 30,2017, from https://cds.cern.ch/record/400313/files/p21.pdf

Vsmolyakov.(July 5, 2017) Ensemble Methods. Retrieved January 15, 2018 from https://github.com/vsmolyakov/experiments_with_python/blob/master/chp01/ensemble_methods.ipynb

Eric Jang. (July 2016). Why randomness is important for Deep Learning. Retrieved January 15,2018 from http://blog.evjang.com/2016/07/randomness-deep-learning.html

Ilya Sutskever, James Martens, George Dahl, et al. (n.d.). On the importance of initialization and momentum in deep learning. Retrieved January 16, 2018 from http://www.cs.toronto.edu/~fritz/absps/momentum.pdf

Data preprocessing. (n.d.). Setting up the data and the model. Retrieved January 16, 2018 from http://cs231n.github.io/neural-networks-2/#datapre

Unsupervised Feature Learning and Deep Learning (April 2013). Data preprocessing. Retrieved January 16, 2018 from http://ufldl.stanford.edu/wiki/index.php/Data_Preprocessing

Venelin Velkov.(July 11, 2017). Credit card fraud detection using autoencoders in keras. Retrieved January 16, 2018 from https://medium.com/@curiousily/credit-card-fraud-detection-using-autoencoders-in-keras-tensorflow-for-hackers-part-vii-20e0c85301bd

ZFTurbo. (November 14, 2016). Kaggle Distracted Driver Keras Simple. Retrieved January 16, 2018 from https://github.com/ZFTurbo/KAGGLE_DISTRACTED_DRIVER/blob/master/run_keras_simple.py

Karel Zuiderveld. (April 27, 2017). Improving the speed of augmented data training using Keras 2. Retrieved January 16,2018 from https://github.com/kzuiderveld/deeplearning1/blob/master/Improving%20training%20speeds%20using%20Keras%202.ipynb

Jason Brownlee. (September 21, 2016). How to improve Deep Learning Performance. Retrieved January 16, 2018 from https://machinelearningmastery.com/improve-deep-learning-performance/

Adit Deshpande. (July 20, 2016). A beginner's guide to understanding convolutional neural networks. Retrieved January 23, 2018 from https://adeshpande3.github.io/adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/

Johannes Kuhn. (February 9, 2017). Batch Normalization. Retrieved January 23, 2018 from https://wiki.tum.de/display/lfdv/Batch+Normalization

Dishashree Gupta. (July 29, 2017). Architecture of Convolutional Neural Networks demystified. Retrieved January 23, 2018 from https://www.analyticsvidhya.com/blog/2017/06/architecture-of-convolutional-neural-networks-simplified-demystified/

Chip Huyen. (n.d.). Playing with convolutions in TensorFlow. Retrieved January 23, 2018 from http://web.stanford.edu/class/cs20si/lectures/notes_07_draft.pdf

Petar Velickovic. (March 20, 2017). Deep learning for beginners: convolutional neural networks in Keras. Retrieved January 23, 2018 from https://cambridgespark.com/content/tutorials/convolutional-neural-networks-with-keras/index.html

Cs231n. (n.d.). Convolutional Neural Networks (CNNs / ConvNets). Retrieved January 23, 2018 from http://cs231n.github.io/convolutional-networks/#conv