# Machine Learning Engineer Nanodegree

## Capstone Project

Rohan Patel

March 31, 2018

# I. Definition

## Project Overview

Deep Learning is a hot topic in machine learning and an area of interest of mine. Deep Learning and neural networks have many use cases, but one area I am particularly interested in exploring is image classification and computer vision. Image classification is a type of learning problem in which a trained model must assign an image a category. These categories can range from actions, objects, letters, numbers, etc. One possible use case for image classification is determining if a driver is distracted at the wheel based on images taken from dashboard cameras. Thousands of people are injured and hundreds are killed each year by distracted drivers. Determining if a driver is prone to distraction can help cars better alert drivers as well as provide valuable information to insurance companies.

https://www.kaggle.com/c/state-farm-distracted-driver-detection

## Problem Statement

The CDC motor vehicle safety division states that one out of five car accidents are caused by distracted drivers. The insurance and financial services company, State Farm, is hosting a competition for people to come up with ways to classify a driver's behavior based on 2D dashboard camera images. The goal of this competition is to allow State Farm to help improve the above statistic as well as better insure their customers, by determining whether it is possible to determine bad driving behavior from dashboard cameras.

https://www.kaggle.com/c/state-farm-distracted-driver-detection

To solve this problem, we must first understand what we are trying to achieve. We know that this is a labeled data set, thus it is a form of supervised learning. We also know that we are working with 2D images and want to classify them. In my opinion, the best course of action would be to use a Convolutional Neural Network. These are great at classifying images. In this project, we created our own model to classify driver actions within a vehicle.

## Metrics

For testing the models created in this project we will reference the testing accuracy. This metric will allow us to determine whether the trained model is performing adequately. However, even though we were given a testing set of images, they were not labeled. They did not provide the labels for the test data so there would be no manipulation to perfectly “fit” the testing set. This is because this project was derived from a competition and the evaluation of the model was done through a logloss function on the competition website. To achieve local testing metrics we split the training data of 27 thousand images into train/validation/test sets and tested the models locally. After a sufficient model was found we submitted a csv of our predictions on the actual test set for evaluation on the competition website. The website used the metric described next.

The model will be evaluated on the multi-class logarithmic loss function. Remember that each image has been labeled with the driver’s behavior from a category c0 – c9. The Convolutional Neural Network will supply a vector of probabilities for each of these categories for each supplied image:

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

N = number of images in the test set
M = number of image class labels
Log = natural logarithm
Yij = 1 if observation i belongs to class j and 0 otherwise
Pij = predicted probability that observation i belongs to class j

The predicted probabilities for each given image are not required to sum to one because are rescaled prior to being scored (each row is divided by the row sum). Also, to avoid extremes given by the log function, the predicted probabilities are replaced with the following:

$$max(min(p, 1 - 10^{-15}), 10^{-15}).$$

https://www.kaggle.com/c/state-farm-distracted-driver-detection#evaluation

## II. Analysis

### Data Exploration

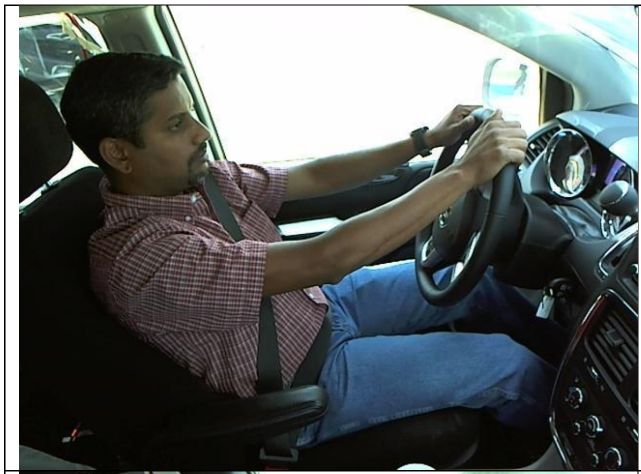
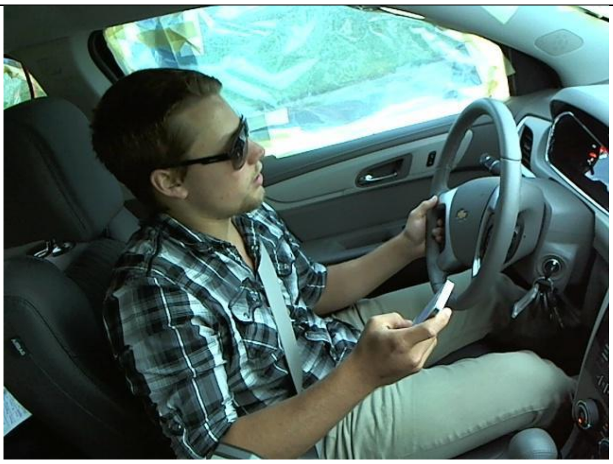
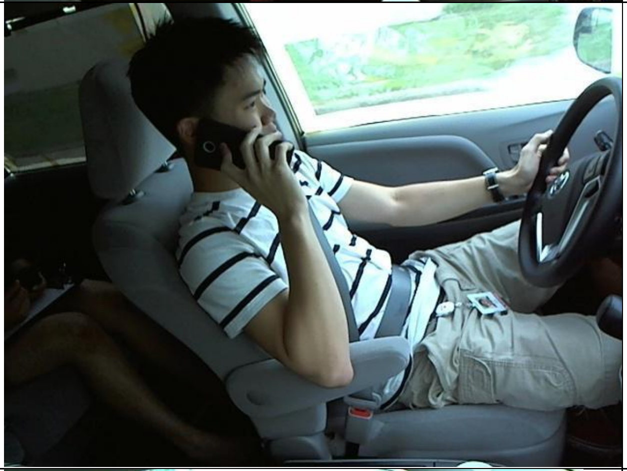
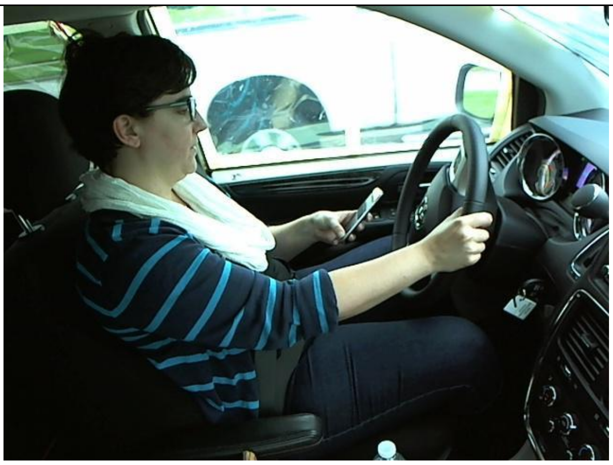
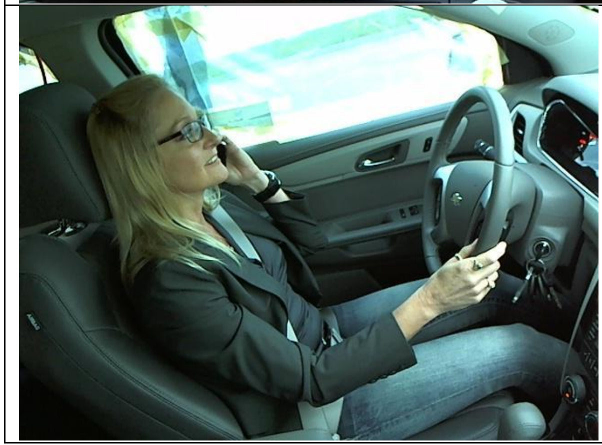
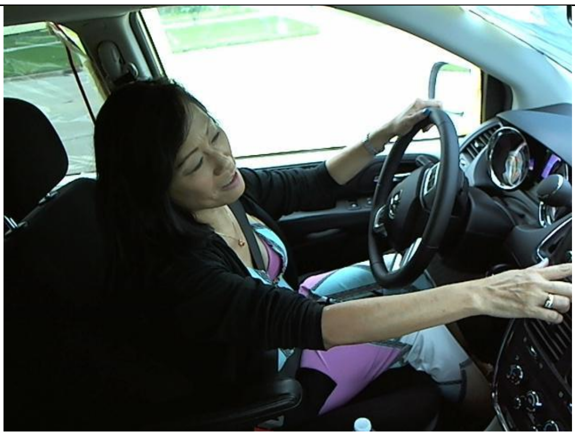
The competition has provided a large dataset of about 100 thousand color images for training/testing data. These are 2D dashboard camera images of drivers either distracted or safely driving. In addition, this is a labeled dataset with one of the 10 following classifications.

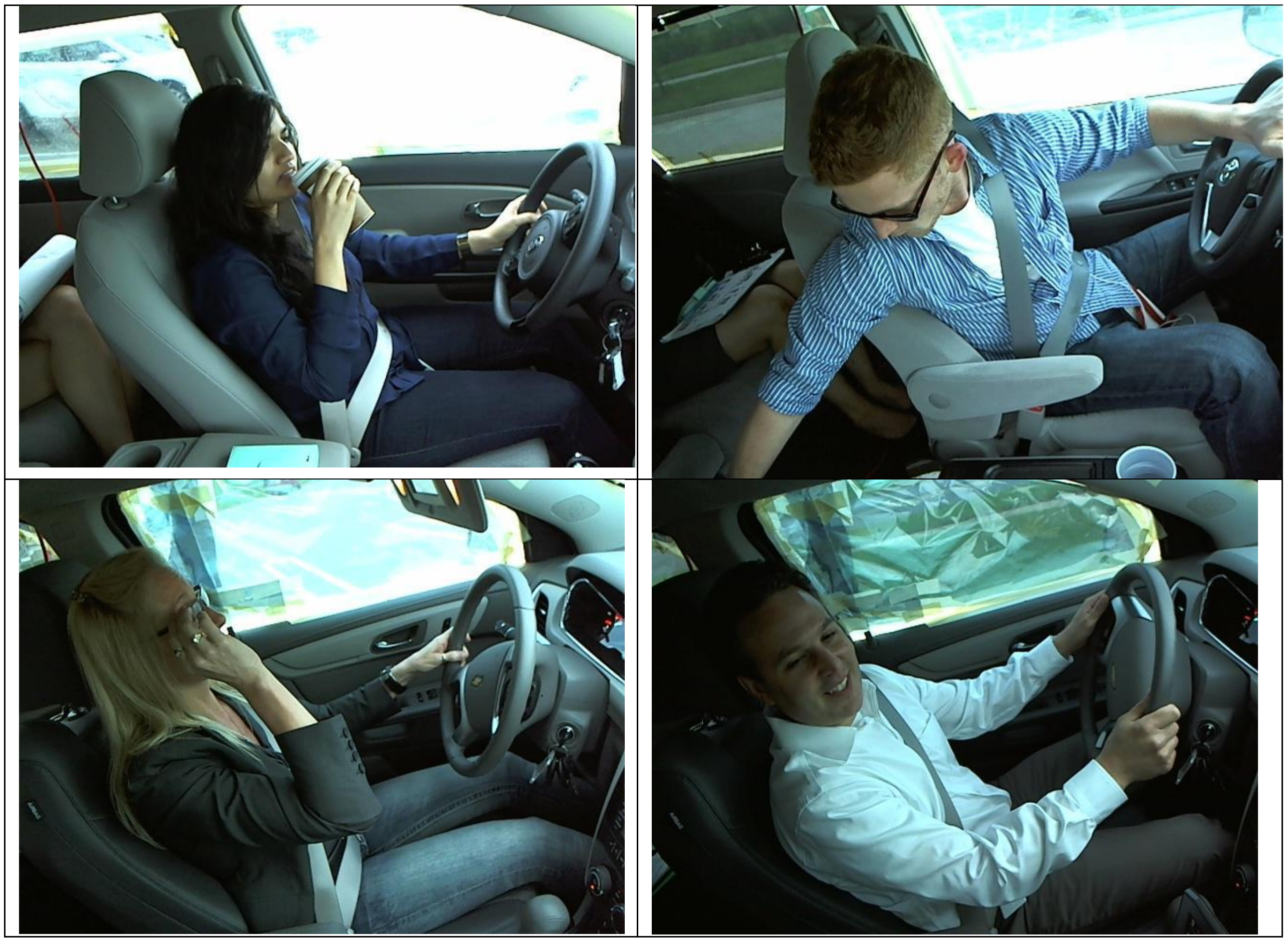
- c0: safe driving
- c1: texting - right
- c2: talking on the phone - right
- c3: texting - left
- c4: talking on the phone - left
- c5: operating the radio
- c6: drinking
- c7: reaching behind

- c8: hair and makeup
- c9: talking to passenger

The supplied driver images are taken in a car while the driver is performing one of the actions above. As the problem statement stated, the goal of this project is to determine/predict the driver's action in a given image.

Here are a few sample images that show the actions c0 to c9 from left to right:

| | |
|---|---|
|  |  |
|  |  |
|  |  |



The images provided have removed all metadata such as creation dates and other marks. In addition, the training and test data are split on the drivers so that drivers can only appear on either train or test. In other words, if they do appear multiple times they will appear on the same set of data. This is good because we want to make sure that our testing data and training are independent and not cross contaminated.

In total the data consists of the following:

- imgs.zip - zipped folder of all (train/test) images
- driver_imgs_list.csv - a list of training images, their subject (driver) id, and class id

The supplied images from the training set will be used to train a classifier which will then be tested on the given training set.

In addition, we may want to preprocess the data so that they are all the same size. Also, color may also be a factor. We may choose to convert all the images to grey scale and provide as input a 2D array of values corresponding to grey scale pixel values. On the other hand, we may choose a 3D array input where we separate the RGB channels.

https://www.kaggle.com/c/state-farm-distracted-driver-detection/data

## Exploratory Visualization

The breakdown of the training data within imgs.zip is as follows:

```
        There are 2489 total images in category c0.
        There are 2267 total images in category c1.
        There are 2317 total images in category c2.
        There are 2346 total images in category c3.
        There are 2326 total images in category c4.
        There are 2312 total images in category c5.
        There are 2325 total images in category c6.
        There are 2002 total images in category c7.
        There are 1911 total images in category c8.
        There are 2129 total images in category c9.
        There are 22424 total images.
```

```
Sample Image:
```

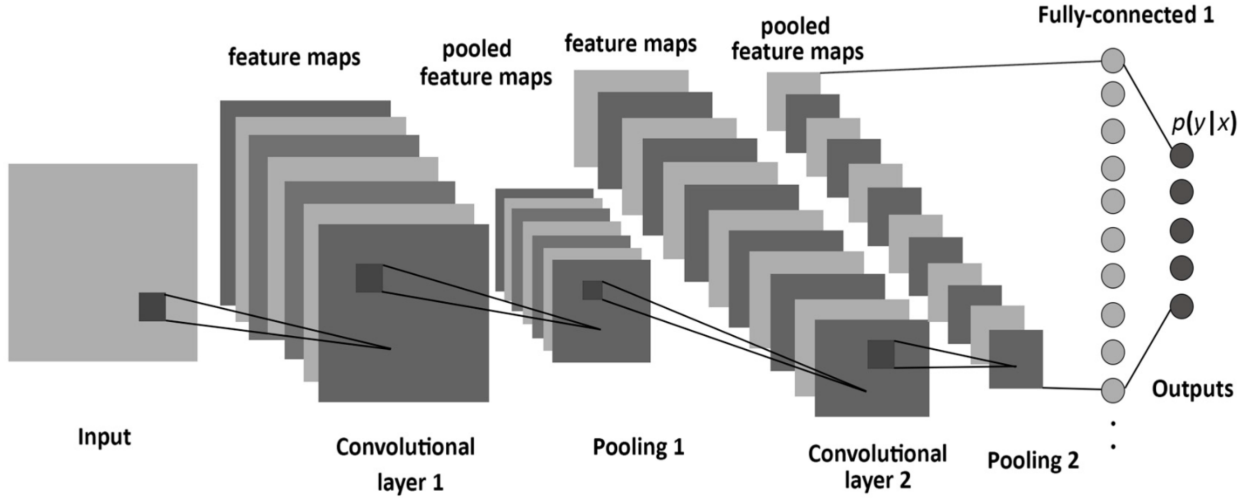
As you can see we have a very large training set and it is also evenly dispersed in terms of categories. This is good as we have a balanced number of images to train on and will make it easier to train the model on all the different classifications. The images provided are all color images so we will have to make use of three dimensions when supplying the data to a Convolutional Neural Net.

## Algorithms and Techniques

To solve this problem, we must first understand what we are trying to achieve. We know that this is a labeled data set, thus it is a form of supervised learning. We also know that we are working with 2D images and want to classify them. In my opinion, the best course of action would be to use a Convolutional Neural Network. These are great at classifying images. In addition, we will be using colored images so we will have input data in that is represented as a 3D array where we have three color channels (BGR).

Essentially, we will convert our images into 3D arrays and possible resize before feeding them into a CNN as training data. We will also have labels so we can properly fit the model and assign appropriate weights. Convolutional Neural Nets will make use of convolutional layers as well as pooling layers. You will see more of the architecture in a later section. The most important step in all of this is data preparation and making sure we can handle this large of a dataset in the training and testing phase.

Model selection will be one of the challenging factors in this project. We will be using a convolutional neural network. We will either design one from scratch or use transfer learning to use a subset of a previously trained model. However, both will output a layer with 10 nodes that correspond to the 10 classifications for drivers (c0 – c9). Here is a sample image of a CNN:



Training the model, evaluating the model, and tuning hyper parameters will be done several times to minimize the training error. We will possibly go through many different convolutional neural network architectures and change several parameters to find the correct formula.

The last step in this project will be to check our ranking against our benchmark which is the kaggle leaderboard. We will try to keep improving our log loss function until we reach a satisfactory value.

## Benchmark

The Kaggle website provides a leaderboard for scores of previous submissions obtained using the evaluation metric (multi-class logarithmic loss) described below. The goal of this capstone is to place as high on the leaderboard as possible by minimizing the log loss function. As a standard for this project a log loss of 3.0 is the minimum acceptable to place in the top 1400. However, the real goal and challenge is to place much higher.

# III. Methodology

## Data Preprocessing

Data preprocessing was one of the most crucial steps in this project, because we needed to get the data into a stage where we can feed it into our model as training data. Luckily for us, Kaggle has supplied two sets of data; one for testing, and one for training. The testing set was divided into 10 categories, one for each classification. The testing data was left unlabeled. This was done for competition reasons, and to allow a server side application score our predictions.

Our first step involved gathering all the data from the provided zip file. We loaded the training data image names and stored them into a list as tuples.

```
Sample format of train data. label and filepath.
('c2', 'imgs/train/c2\\img_100029.jpg')
```

This list was shuffled so that its contents were placed in a random order. This was done because the original order contained all the categories grouped together. By shuffling the data we can maximize the number of different labels we supply to a model if we partition this list containing training data.

The next step involved splitting the training data into train/validation/test sets. We could not use the supplied testing data because we did not have the labels (as explained previously). So we split the around 20K training images into the three sets. This is where we met our first major hurdle.

The issue came about when we tried reading the images in order to convert them as into 3D arrays. My computer has 16 GB of RAM, but that wasn't enough to hold the entire training data set, even if we resized them. So the decision was made to only train on a subset of the training data. We used 3500 images for training data, 500 for Validation data, and 500 for testing data. This split seemed to be sufficient as shown later.

In addition, an attempt was made to use the entire data set, but was unsuccessful. I tried supplying the model with data in batches (using "model.fit_generator" instead of "model.fit") instead of all at once, but the training time was too significant to switch techniques. Also, as shown later, the performance of the model with the reduced data set was significant enough to overlook this setback.

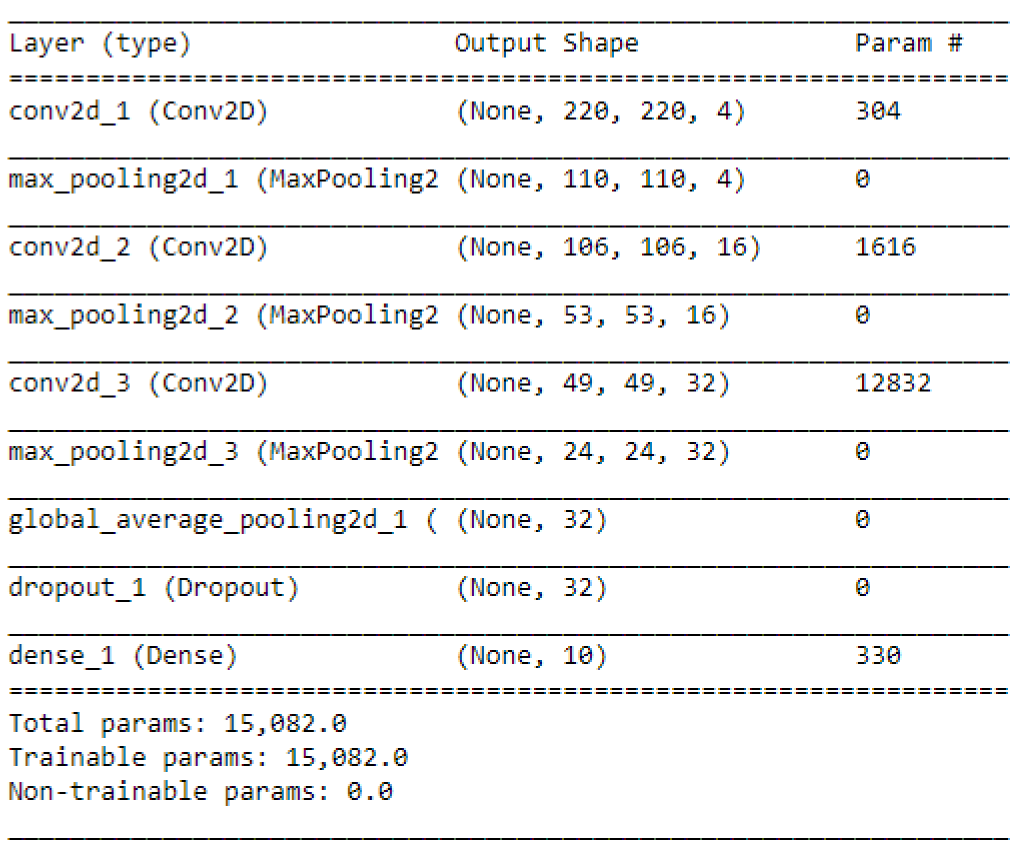
After numerous attempts at processing the data, the final version was to use a subset of the original 20K images. These images were converted into 3D arrays in the shape (224, 224, 3) where 224 is the width and the height, and 3 represents the three color channels. We also divided every cell in the array by 255 to make every pixel value between 0 and 1. This is because color values in decimal code range for 0-255 for each channel R, G, and B. Lastly, we made each of our labels an array where each index is the probability of that particular classification. Since we have 10 categories, each label will be an array of size 10 where every value is 0 except the actual label which will be a 1. This array is used in comparison with that generated by the model during training.

## Implementation

As explained earlier, we used a Convolutional Neural Net as our proposed solution to this classification problem. Our strategy was to first create our own models and see how we progress. Then, if necessary, we would use transfer learning in an effort to increase the model accuracy.

Our first attempt used the following architecture:

Model V1:



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_1 (Conv2D)            (None, 220, 220, 4)       304       
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 110, 110, 4)       0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 106, 106, 16)      1616      
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 53, 53, 16)        0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 49, 49, 32)        12832     
_________________________________________________________________
max_pooling2d_3 (MaxPooling2 (None, 24, 24, 32)        0         
_________________________________________________________________
global_average_pooling2d_1 ( (None, 32)                0         
_________________________________________________________________
dropout_1 (Dropout)          (None, 32)                0         
_________________________________________________________________
dense_1 (Dense)              (None, 10)                330       
=================================================================
Total params: 15,082.0
Trainable params: 15,082.0
Non-trainable params: 0.0
_________________________________________________________________
```

This model had over 15K parameters. We supplied it a input made up images in the size (224, 224, 3). The layers are 3 sets of pairs of convolutional and pooling layers followed by a global average pooling layer. We also added a dropout layer to reduce the chance of overfitting. Lastly we added a dense layer to map to contain the 10 possible classifications.

After training the model with a validation set, we determined its accuracy by making predictions on our “testing set”. Unfortunately, it only had about a 30.8% accuracy on a set of 500 test images. This is good for our first attempt as we are doing better than random chance (1/10 = 10%), but not good enough in terms of usability.

```
print(test_targets[0])
print(model.predict(np.expand_dims(test_images[0], axis=0)))
```

```
[ 0.  1.  0.  0.  0.  0.  0.  0.  0.  0.]
[[ 0.15588532  0.14716297  0.07078248  0.21618374  0.14734071  0.01396769
   0.08366183  0.03734739  0.05895158  0.06871618]]
```

As you can see the label says this image is 'c1: texting – right'. However, the prediction array has values that don't necessarily lean to a specific value. For example index 0, 1, and 4 are almost evenly possible. So this model isn't doing the best job.

## Refinement

Much other architecture for CNN were tested, and two of them had testing accuracies over 96%. Below you will find these two CNNs, an explanation of the changes made, and their resulting accuracies.

<u>Model V2:</u>

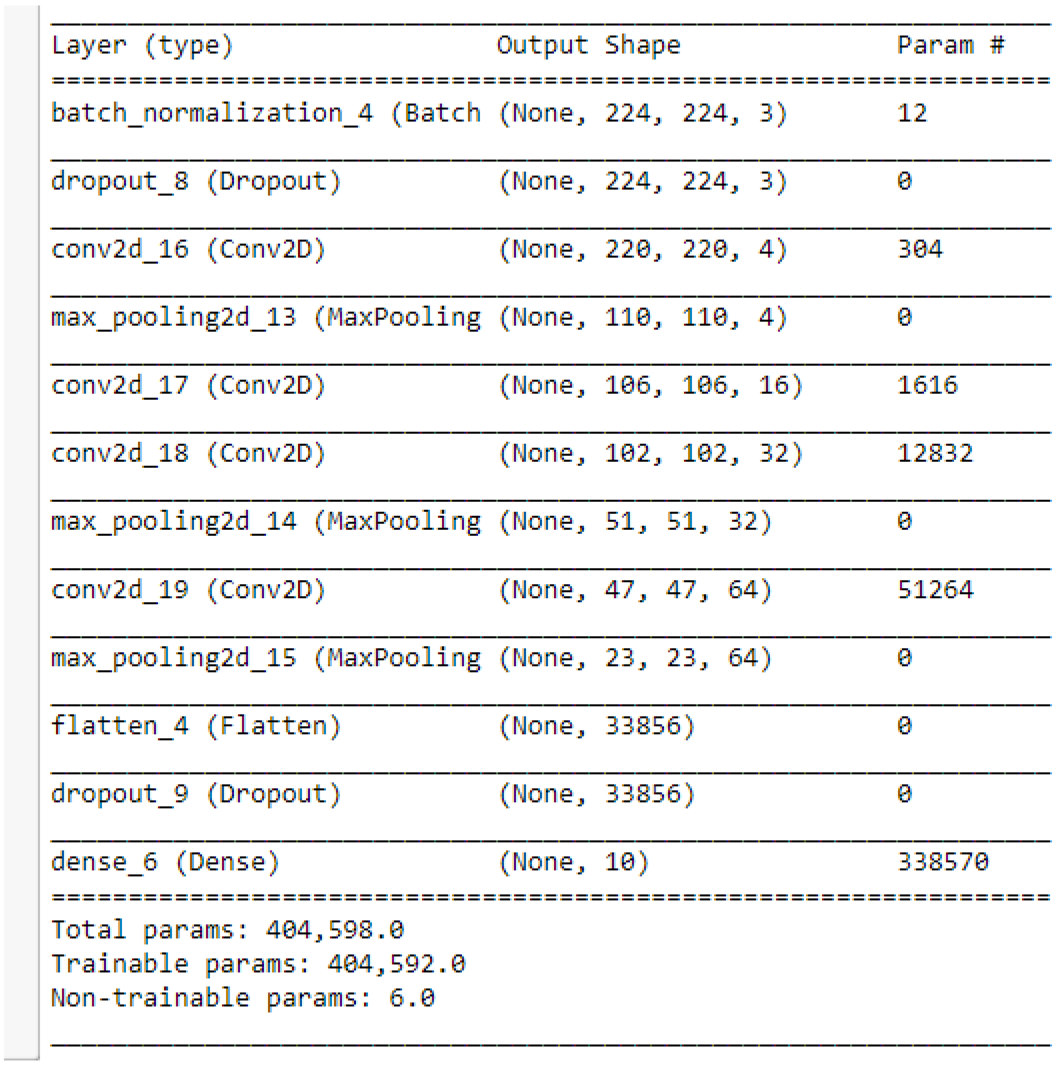
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
batch_normalization_5 (Batch (None, 224, 224, 3)       12        
_________________________________________________________________
dropout_10 (Dropout)         (None, 224, 224, 3)       0         
_________________________________________________________________
conv2d_20 (Conv2D)           (None, 220, 220, 4)       304       
_________________________________________________________________
max_pooling2d_16 (MaxPooling (None, 110, 110, 4)       0         
_________________________________________________________________
conv2d_21 (Conv2D)           (None, 106, 106, 16)      1616      
_________________________________________________________________
max_pooling2d_17 (MaxPooling (None, 53, 53, 16)        0         
_________________________________________________________________
conv2d_22 (Conv2D)           (None, 49, 49, 32)        12832     
_________________________________________________________________
max_pooling2d_18 (MaxPooling (None, 24, 24, 32)        0         
_________________________________________________________________
flatten_5 (Flatten)          (None, 18432)             0         
_________________________________________________________________
dropout_11 (Dropout)         (None, 18432)             0         
_________________________________________________________________
dense_7 (Dense)              (None, 10)                184330    
=================================================================
Total params: 199,094.0
Trainable params: 199,088.0
Non-trainable params: 6.0

_________________________________________________________________
```

This modeled has some similarities to the previous layer, but is vastly different in performance. The first change is the introduction of a Batch Normalization input layer followed by a dropout layer. The batch layer is used to normalize the activations of previous layers. We also include a dropout layer to reduce overfitting. Lastly, we added a flatten layer and removed the global pooling layer. This layer is used to flatten the input before passing it onto the dense output layer.

These changes worked remarkably and the testing accuracy rose to an incredible 96 % after just 11 epochs.

<u>Model V3:</u>



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
batch_normalization_4 (Batch (None, 224, 224, 3)       12
_________________________________________________________________
dropout_8 (Dropout)          (None, 224, 224, 3)       0
_________________________________________________________________
conv2d_16 (Conv2D)           (None, 220, 220, 4)       304
_________________________________________________________________
max_pooling2d_13 (MaxPooling (None, 110, 110, 4)       0
_________________________________________________________________
conv2d_17 (Conv2D)           (None, 106, 106, 16)      1616
_________________________________________________________________
conv2d_18 (Conv2D)           (None, 102, 102, 32)      12832
_________________________________________________________________
max_pooling2d_14 (MaxPooling (None, 51, 51, 32)        0
_________________________________________________________________
conv2d_19 (Conv2D)           (None, 47, 47, 64)        51264
_________________________________________________________________
max_pooling2d_15 (MaxPooling (None, 23, 23, 64)        0
_________________________________________________________________
flatten_4 (Flatten)          (None, 33856)             0
_________________________________________________________________
dropout_9 (Dropout)          (None, 33856)             0
_________________________________________________________________
dense_6 (Dense)              (None, 10)                338570
=================================================================
Total params: 404,598.0
Trainable params: 404,592.0
Non-trainable params: 6.0
_________________________________________________________________
```

This modeled has some similarities to the previous layer, but is slightly different in performance. The only difference is the addition of a convolutional layer in-between the second set of convolutional and pooling pairs. This change worked slightly as the training accuracy rose to 95.8%.

I also tried a final attempt but that model was more complex and yielded a lower accuracy of 95%. In my opinion, it is better to go with simpler approach with higher accuracy. This thinking was validated when we discovered the scoring of these two refined models using our benchmark model.

# IV. Results

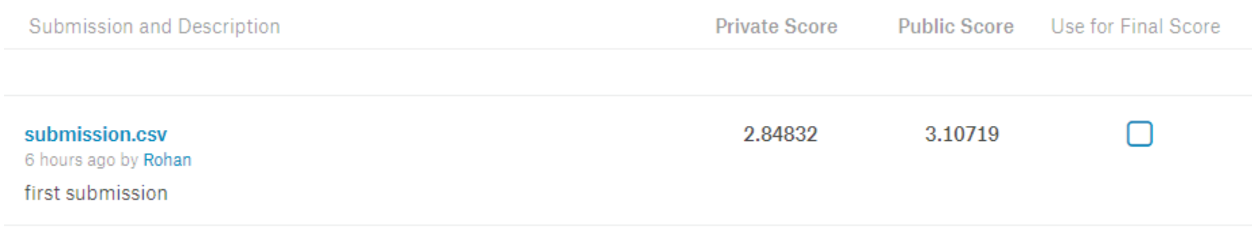
## Model Evaluation and Validation

Our initial model didn't meet the standards set by our benchmark model in which we wanted a logloss of less than 3.0. We didn't even need to test with the actual testing set as the model accuracy on the "test" set was roughly 30%. On the other hand, the refined models achieved accuracies of 96% and 96.8% on the "test" set. In addition, all of these models were trained on over three thousand images, validated on 500 images, and tested on 500 images to determine those accuracies. The validation sets were used to prevent overfitting.

The second and third models were the ones we used to compare against our benchmark model. This was done in a unique way as we couldn't score the model ourselves. We had to make predictions on the roughly 80 thousand test images (actual test images w/o labels) and supply the predictions within a csv file. This file was submitted on the kaggle site where the predictions were put through a logloss function supplied with the actual test labels to determine your score. Both models scored better than the benchmark model score of 3.0 but one did significantly better.

## Justification

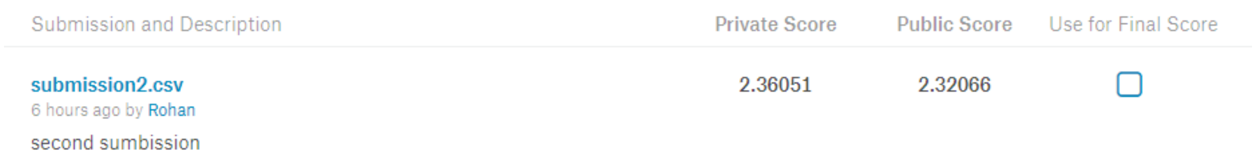
My initial thought was to use the model with the 96.8 accuracy. And in turn these were the results for that model:



| Submission and Description | Private Score | Public Score | Use for Final Score |
|---|---|---|---|
| submission.csv<br>6 hours ago by Rohan<br>first submission | 2.84832 | 3.10719 | ☐ |

Although we achieved our benchmark goal of 3.0 in the private leaderboard we failed to do so in the public leaderboard.

In my second submission we used the simpler model that scored a 96% in "test" accuracy. Its score is as follows:



| Submission and Description | Private Score | Public Score | Use for Final Score |
|---|---|---|---|
| submission2.csv<br>6 hours ago by Rohan<br>second sumbission | 2.36051 | 2.32066 | ☐ |

As you can see this model is much better in terms of its public and private score. And we surpassed the benchmark score of 3.0.

It is also important to mention that the training set and testing set did not have an overlap in drivers. This was essential in removing this driver factor dependency on the model. By not having overlap in the testing and training set, we didn't violate the golden rule in machine learning: "don't use testing data for training". By introducing the same drivers in both set, we may introduce a type of bias in which specific drivers are distracted behind the wheel. Instead, we can focus on determining features that suggest distracted driving.

## V. Conclusion

### Free-Form Visualization

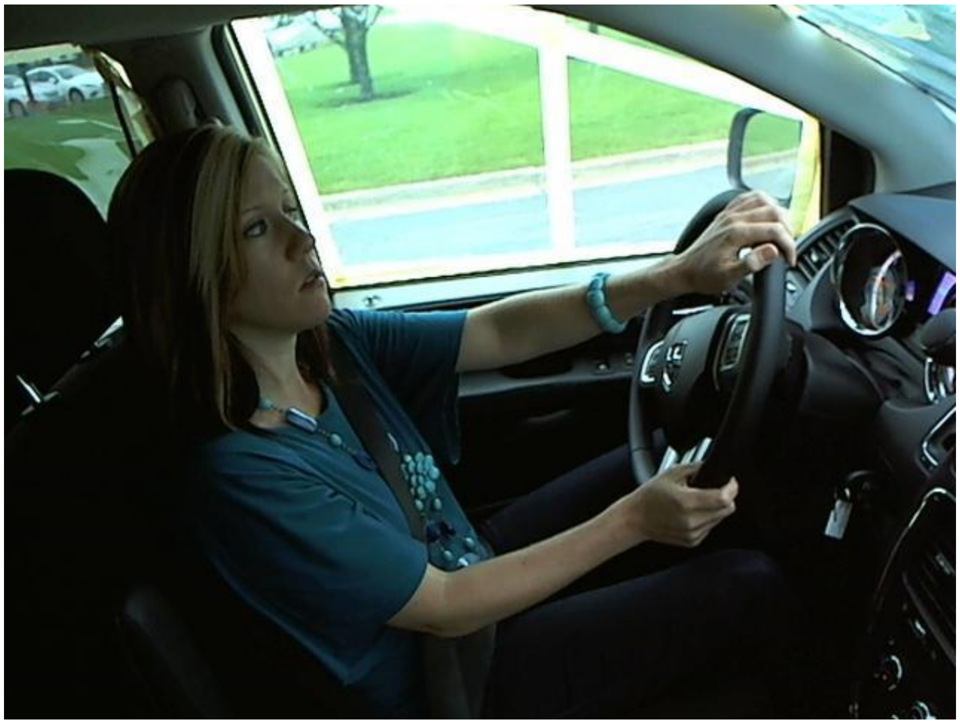
Lets see how our model is at detecting distracted driving first hand. Below you will see images of drivers and the output array of the CNN. The indexes in the array 0-9 relates to c0-c9 which are the classifications. Here is a reminder of those values:

- 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

The first image is of a woman who is driving safely (c0). So we expect that the first element in the array is the largest probability. And that is exactly what we see below. Although the probability isn't close to 1 it is by far the largest value in the array.

| Classification | Probability |
|---|---|
| • c0: safe driving | **0.7408380508422852** |
| • c1: texting - right | 0.007946318946778774 |
| • c2: talking on the phone - right | 0.04649019613862038 |
| • c3: texting - left | 0.0005890822503715754 |
| • c4: talking on the phone - left | 0.00034316786332055926 |
| • c5: operating the radio | 0.020734010264277458 |
| • c6: drinking | 0.00010067865514429286 |
| • c7: reaching behind | 0.11349882185459137 |
| • c8: hair and makeup | 0.025346212089061737 |
| • c9: talking to passenger | 0.04411342367529869 |

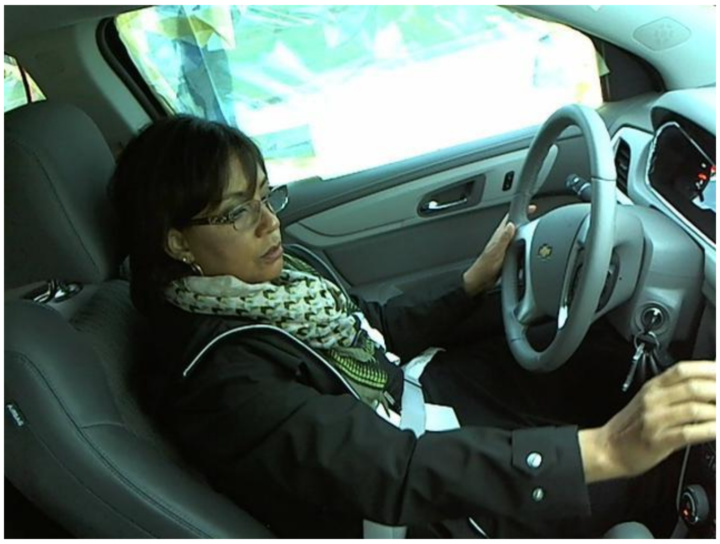
img_41.jpg



The first image is of a woman who is operating the radio (c5). So we expect that the sixth element in the array is the largest probability. And that is exactly what we see below. This value is very close to one (approx. 0.99).

| **Classification** | **Probability** |
|---|---|
| • c0: safe driving | 2.8829515485995216e-07 |
| • c1: texting - right | 1.1862861981626338e-07 |
| • c2: talking on the phone - right | 2.5270250262110494e-05 |
| • c3: texting - left | 2.1437011810121476e-07 |
| • c4: talking on the phone - left | 6.587623033738055e-07 |
| • c5: operating the radio | **0.9998001456260681** |
| • c6: drinking | 2.4104372187139234e-07 |
| • c7: reaching behind | 8.494596841046587e-06 |
| • c8: hair and makeup | 2.501423068679287e-06 |
| • c9: talking to passenger | 0.00016204385610762984 |

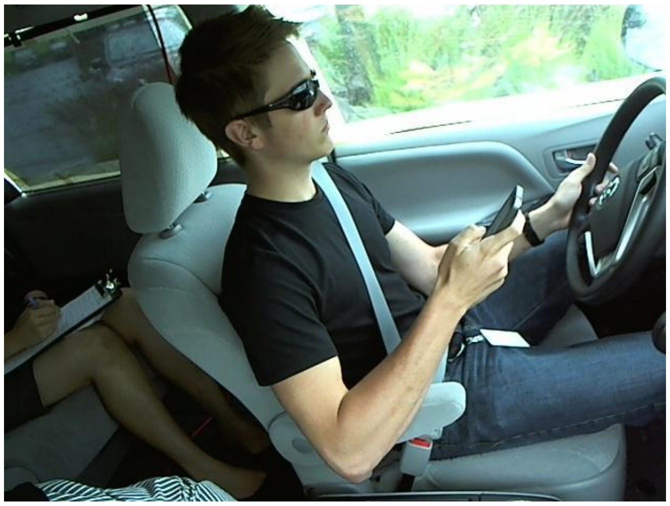
img_1.jpg



The third image is of a man who is texting with his right hand (c1). So we expect that the second element in the array is the largest probability. However, instead of we a different value that is almost one. That value if talking on the phone – left. Our model is scoreing 96% accuracy and this may be one of those areas of error. Both texting and talking on the phone involve the use of a cell phone, so the model may be grabbing that feature but just mapping it incorrectly. Or we need to do more training.

| **Classification** | **Probability** |
|---|---|
| • c0: safe driving | 0.0002057491656159982 |
| • c1: texting - right | 0.031331565231084824 |
| • c2: talking on the phone - right | 7.057366815388377e-07 |
| • c3: texting - left | 0.01158074475824833 |
| • c4: talking on the phone - left | **0.9549766182899475** |
| • c5: operating the radio | 7.157983645811328e-07 |
| • c6: drinking | 0.0005713162245228887 |
| • c7: reaching behind | 1.0653084245859645e-06 |
| • c8: hair and makeup | 0.00043564787483774126 |
| • c9: talking to passenger | 0.0008960370905697346 |

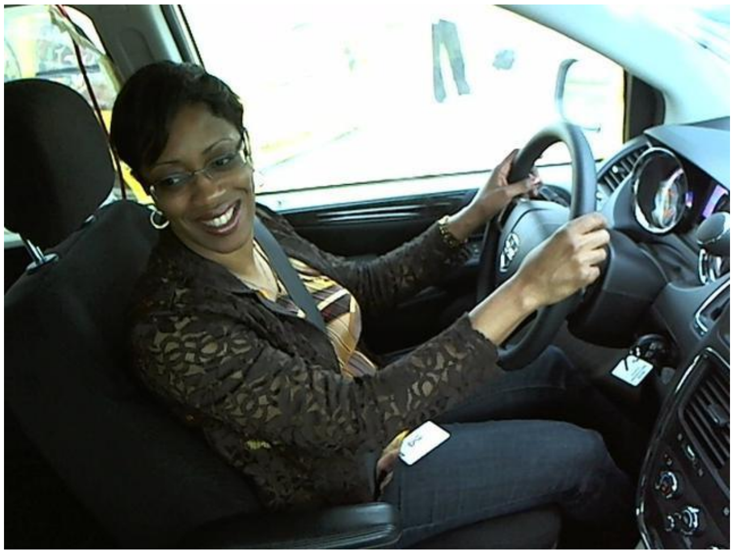
img_39.jpg



The final image is of a woman who is talking to the passenger (c9). So we expect that the last element in the array is the largest probability. That is exactly what we see, but it isn't as clear as the previous correct predictions. The highest and second highest values are only off by .30. So this shows that even though we got the correct classification, we can definitely improve.

| **Classification** | **Probability** |
|---|---|
| • c0: safe driving | 0.0694221779704094 |
| • c1: texting - right | 0.07034522294998169 |
| • c2: talking on the phone - right | 0.09682472795248032 |
| • c3: texting - left | 0.00010799306619446725 |
| • c4: talking on the phone - left | 0.0007729366770945489 |
| • c5: operating the radio | 0.031338151544332504 |
| • c6: drinking | 0.0013899386394768953 |
| • c7: reaching behind | 0.003219669684767723 |
| • c8: hair and makeup | 0.03488872945308685 |
| • c9: talking to passenger | **0.6916904449462891** |

img_178.jpg



## Reflection

In this project, we took images of drivers and converted them into 3D arrays of size (224,224,3). We then fed them into a Convolutional Neural Net made up of convolutional and pooling layers who would in turn map the image to an array of probabilities representing 10 driver classifications. We used a training set of 20K images and a testing set of 80K images supplied to us by Kaggle. In the end, we were able to generate a model with 96% “test accuracy” and a benchmark model score (logloss) of about 2.3.

Overall this project was very satisfying, as I got to challenge a real world problem, as well as an interest are of mine, computer vision. Image Classification in this project proved to be difficult with the sheer size of date, and my limited computational resources. Although I had a pc with 16GM of ram and a dedicated video card able to run tensorflow locally. Training times on large datasets very impossible as I would run out of memory either training the model or loading all the images as 3D arrays. I tried strategies such as batch training using fit_generator but in the end the simpler approach with a reduced training set size seemed the most efficient.

## Improvement

There are several areas of improvement in this project that could be implemented with better resources. However, there are also things that could take this project to the next level.

Implementing batch training to train our model on the entire 20K training images rather than a subset is one place to start. In addition, we can try to use transfer learning to get a better accuracy and extract more features. I decided not to do in this project, due to resource issues.

Furthermore, we could try to make this model more robust, by first detecting for drivers, and then detecting distracted driving. This would involve creating a detector that would in essence crop out the scene to only include driver.