University of Ontario Institute of Technology

MCSC 6230G Advanced Topics in High-Performance Computing

Distracted Driver Detection Using Convolutional Neural Networks and Transfer Learning

May 11, 2018

Abstract

This work looks at various images of distracted drivers taken from people performing different actions, some of which can be deemed as distracting whilst behind the wheel of a car. A Convolutional Neural Network is used in order to more accurately predict what activity a driver is being distracted by. The VGG16 deep Convolutional Neural Network model, pre-trained on ImageNet, is coupled with a retrained fully-connected model. The newly trained model is able to achieve a 99.396% accuracy on previously unseen distracted driver images.

1 Introduction

Distracted driving is an event that occurs when a driver diverts their primary focus of driving to another task or activity. With our increased dependence on technology and cellphones, distracted driving is becoming an increased concern. Distracted driving does not just involve the use of cellphones but also includes eating, drinking, and even talking to a passenger whilst driving. This work aims to help amend this issue by automatically predicting if drivers are distracted, and what action in specific is resulting in the distracted driving. Three different models are implemented, from simple Convolutional Neural Networks (CNNs) to more complex ones involving Transfer Learning from other pre-trained models. The success of the models will hopefully, one day, aid combat the ongoing and increasing issue of distracted drivers on the roads.

2 Motivation

Distracted driving is an ongoing problem that seems to only be getting worse with the dependence on technology. Not only is the general public becoming more reliant on vehicles, but also on distractions like cell phones and GPS systems within them. Most modern day vehicles come with touch screen navigation that requires some level of interaction to operate. Cellphones, too, are a big part of every day life interactions, with constant incoming notifications and calls. These upward trends in dependence seem to have a correlation to the number of crashes as a result of distracted driving. In the province of Ontario, the deaths from crashes as a result of distracted driving have doubled since the year 2000, with one person being injured every half

hour as a result of a distracted driver[1]. Furthermore, in the United States alone, 425,000 people are injured and 3,000 are killed as a result of distracted driving, according to the Centers for Disease Control and Prevention (CDC) in 2015 [2]. The U.S. Department of Transportation National Highway Traffic Safety Administration (NHTSA) published a Research Note regarding Distracted Driving in 2015 [3]. In the report, it states that 10% of fatal crashes, 15% of injury crashes, and 14% of all police-reported motor vehicle traffic crashes in 2015 were reported as distracted driver collisions.

It is evident that distracted driving is an ongoing issue, and perhaps the very technology that typically causes distractions can be used to alleviate them. With the current advances in machine learning and hardware, computer vision techniques has also followed suit and have shown their powerful capabilities. This work is a stepping stone in the ultimate goal of using computer vision to help fight the distracted driver problem. If a system within the vehicle is able to detect if a driver is distracted, it could perhaps remind the driver to focus their attention on driving. Many drivers may not realize what they do is taking their attention away from driving, possibly because the distractions are so habitual and ingrained in their daily routine. The hypothetical distracted driver detection system in the car could, through feedback, make the drivers consciously aware of their actions, and help correct them over time. This kind of proposed work would, of course, have to eventually be properly engineered, fine tuned, and tested before becoming a successful prevention in distracted driving. However, until then, small steps need to be taken in solving the problem of distracted driver detection.

3 Background

3.1 Convolutional Neural Networks

In order to tackle the distracted driver detection problem, the Convolutional Neural Network (CNN) model is utilized. CNNs have proven to perform remarkably well on classifying images, and as such, are a great fit for this problem. It is worth noting that CNNs typically suffer from overfitting, which occurs when a model adapts too well on trained data, but performs poorly

[1] `https://www.ontario.ca/page/distracted-driving`

[2] `https://www.cdc.gov/motorvehiclesafety/distracted_driving/`

[3] `https://crashstats.nhtsa.dot.gov/Api/Public/ViewPublication/812381`

on new data, and thus is said to generalize poorly. This is an issue that we try to mitigate as much as possible throughout this work.

CNNs rely on the idea that local understanding of an image is good enough, with the practical benefit of having fewer parameters, consequently reducing the computation time and data required to train the model [1]. Rather than have a fully connected layer for every pixel, CNNs only have enough weights to look at small parts of the image at a time. The process typically involves a convolution layer, followed by a pooling and an activation function, but not necessarily in that exact order. These three separate operations can be applied as separate layers to the original image, typically multiple times. Finally, a fully connected layer (or several) are added at the end in order to classify an image accordingly. With the highly variable number of combinations and permutations, it can be difficult to find the exact one that gives the optimal performance. CNN designs are driven by the community and research who thankfully have made some successful results, and made their CNNs publicly available for others to use and improve upon.

3.2 VGG16

VGG is a CNN model proposed by Karen Karen Simonyan, and Andrew Zisserman, in their paper *Very Deep Convolutional Networks for Large-Scale Image Recognition*[2]. VGG16, also commonly referred to as *OxfordNet*, is named after the *Visual Geometry Group* at the University of Oxford. More specifically, VGG16 is a deep 16-layer Convolutional Neural Network, with 13 convolutional layers, and 3 fully connected ones at the very top. A visual representation of the VGG16 architecture is shown in Figure 1. The model is able to achieve up to 92.7% top-5 accuracy[2] on ImageNet[4], which contains nearly 14 million images with over 1000 categories. The authors have not only made the model publicly available, but also its pre-trained ImageNet weights. This is something that is utilized in our work in order to not only decrease the computational cost, but also increase the accuracy of the model.

3.3 Transfer Learning

Transfer learning embodies the idea that knowledge gained whilst solving one problem, can be applied in solving a different, but related, problem.

[4] `http://image-net.org/`

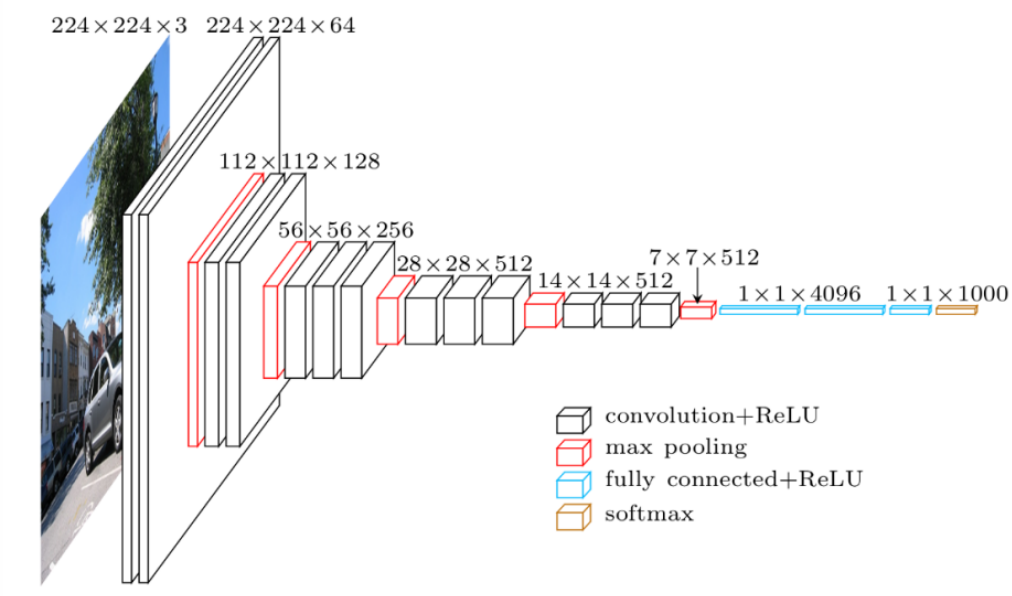


Figure 1: VGG16 Architecture Visualized[3]

Typically, Transfer Learning can be applied to machine learning, or more specifically CNNs. This is usually most appropriate when there is insufficient data or computational power, in which case a model with pre-trained weights on a much larger database to solve one problem can be applied. One strategy typically employed is to use the pre-trained convolutional network as a fixed feature extractor. This is done by completely removing the last fully-connected layers, and then using the rest of the CNN as a feature extractor for a different problem. A linear classifier can then be trained using those features, as well as the new dataset to solve a different problem.

Due to the nature of convolutional networks, the deeper the layers, the more specific they become based on the data they were trained on. This can be problematic when using Transfer Learning to solve another problem, since some layers will be oriented towards the original dataset. For example, the pre-trained ImageNet VGG16 CNN, might have some generic first layers that detect things like lines and edges; however, in the deeper layers, other features are learned, which are only specific to classes in the original data. If the more generic layers of the pre-trained CNN could be frozen, then the more specific layers could be re-trained on the new dataset and eventually build a model specific to the new data. This process is known as Fine-Tuning

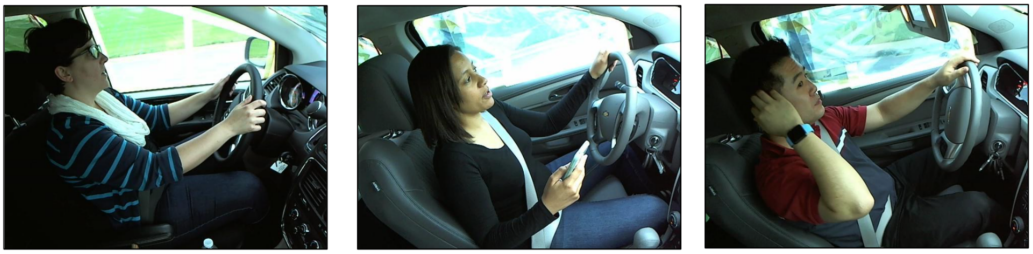


Figure 2: Examples of Distracted Drivers [4]

[5] and is a commonly used strategy in Transfer Learning.

The aforementioned Transfer Learning strategies are also something that will be utilized throughout in this work. The primary reason for this is to overcome the hardware limitations and to more importantly achieve a higher accuracy.

4 Data

In order to train a model to detect distracted drivers, a dataset of images of both distracted and non-distracted drivers is required. This would only require a binary classifier model that would predicted if a driver is distracted or not. A more interesting problem would be one where the drivers are distracted in different ways, like eating or fixing their hair for example. This is a harder problem and requires a multi classifier model and a more specific dataset.

4.1 Source

Kaggle [6] is an online platform for data science tutorials and competitions. One of those competitions is the *State Farm Distracted Driver Detection* [4] competition hosted by State Farm. This is an event that started and ended in 2016, however, the dataset remains available to anyone who wishes to take part in a late submission. The dataset consists of images of drivers

[5] `http://cs231n.github.io/transfer-learning/`

[6] `https://www.kaggle.com/`

CLASS	DESCRIPTION
c0	Safe Driving
c1	Texting (right hand)
c2	Talking on the phone (right hand)
c3	Texting (left hand)
c4	Talking on the phone (left hand)
c5	Operating the radio
c6	Drinking
c7	Reaching behind
c8	Hair and makeup
c9	Talking to passenger(s)

Table 1: Different classes of driver images

performing one of 10 possible activities, one of which is safe driving. The rest of the images belong to classes where the driver can be considered distracted (e.g. texting). An example of the images used can be seen in Figure 2. In these three examples, *c0* represents **safe driving**, while *c1* and *c8* represent **texting with their right hand** and **doing their hair and/or makeup**, respectively. Its worth noting that all the images are typically taken from the same angle, in the same environmental conditions (i.e. daytime) which is both good and bad in training the model. It could be considered a positive event because, if a subset of this data is used as the test set, then the model will most likely perform well since the test images have the same conditions. However, this posses an issue when the model is generalized to images that don't necessarily meet the same conditions as the ones in the original dataset. The hypothesis is that a drastic change in angle, or incoming light will cause the model to perform very poorly. This hypothesis is hard to confirm without finding a new dataset of distracted drivers in varying conditions, which, to our knowledge, does not exist.

4.2 Classes

All of the different classes of images in the dataset can be summarized in Table1. They are labeled from *c0* to *c9*, each representing a specific distracted driving activity. The only exception is *c0*, which represents safe driving. The remaining classes vary in the sense that some are distinctly different, like **c5: Operating the radio** and **c7: Reaching behind**, while others are very

related, **c1: Texting (right hand)** and **Texting (left hand)**, for example. This makes some classes easier to distinguish from others, while other predictions might be consistently labeled wrong due to their similarity to other classes.

4.3 Data Split

The overall dataset provided by State Farm [4] contains close to 100,000 images that fall under the 10 classes shown in Table 1. More specifically, the dataset was originally split up into training and testing data, with **22,424** and **79,726** images in each, respectively. The main difference between the two is that the training data contained the ground truth labels of each image, while the testing images did not. This was done deliberately since it was a competition and like such, it allowed for some measure of success between different submissions. In this work, however, in order to train and evaluate the performance of the models, the **22,424** training labeled images were split up further into 60% actual training data, 20% validation, and 20% testing data. This allows us to evaluate the model offline without having to submit to the Kaggle [4], but also test the model against unseen data and apply our own testing measures. The initial concern is that it might not be enough data to train the model; however this will be supplemented by data augmentation and transfer learning techniques.

5 Detecting Distracted Driving

5.1 Initial Attempt

In the first attempt at solving the distracted driver problem, a simple Convolutional Network was implemented on the *TensorFlow* framework. Much of the image manipulation had to be done manually and prior to the machine learning process, since it did not fit into memory using the limited hardware at our disposal. As a result, the images had to be reduced significantly from **640 x 480** to **24 x 24** and greyscaled as depicted in Figure 3. This was the only feasible solution at the time, since the model was being trained on a laptop with a CPU and only this method allowed it to run in a reasonable amount of time. This reduction in quality greatly reduces the amount of information available to the model, and as we will see later, had a significant

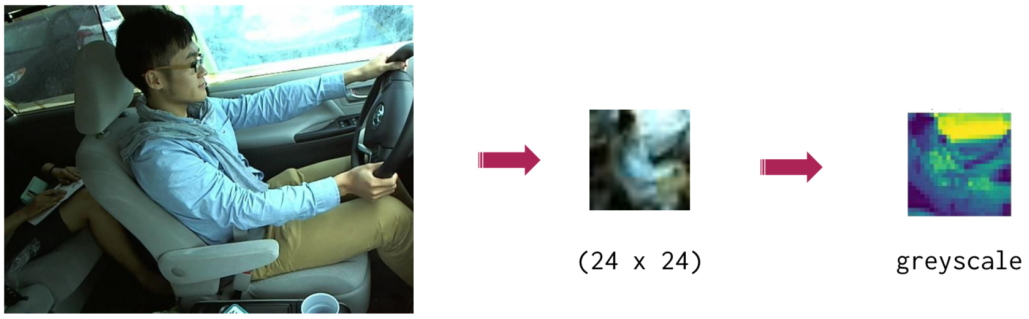


Figure 3: Image Resizing and Greyscaling [4]

impact on the results.

The Convolutional Neural Network architecture implemented consists of 2 convolutional layers and 2 fully connected ones. As CNNs were a fairly new concept to us, several resources were utilized including [1] and several *TensorFlow* Tutorials[7]. The architecture of the model can be seen in Figure 4. The **conv** layers represent a convolution followed by a Rectified Linear Unit (ReLU) activation. The **pool** layers represent pooling layers, and **norm** layers depics local response normalization. Finally, the **fc** layers represent a fully connected layer with a ReLU activation. The model has previously achieved 86% accuracy on CIFAR-10 with only a few hours of training on a GPU, which showed that the model was viable. We were hoping to achieve a similar accuracy when trained on the distracted driver dataset. Not having access to a GPU at the time this model was implemented, training on a laptop was unfortunately the only option. After training the model on a CPU overnight for several hours and 25 epochs, it achieved an all-time-high accuracy of **10.418%**. The results were not up to par with our expectations, as the model performed only slightly better than a random model, which would have a 10% accuracy. There were a couple of takeaways from this implementation, mainly a different strategy was required, and a hardware update was imperative.

[7] `https://www.tensorflow.org/tutorials/deep_cnn`

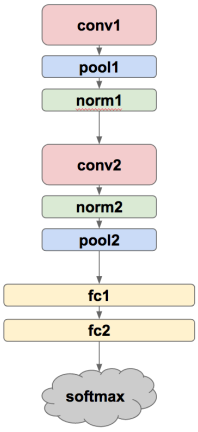


Figure 4: Initial CNN Architecture

5.2 Simple CNN

In this attempt at solving the distracted driver problem, a server with a single *GeForce GTX 1080 Ti* graphics card was utilized. Additionally, the framework of choice was the *Keras*[8] open source neural network library, with *TensorFlow* as the backend. There seemed to be more resources surrounding the library. It was higher-level of abstraction compared to *TensorFlow*, and most importantly, it supported GPU computation. In this implementation we aimed to build a simple CNN by using Keras documentation[5]. The architecture of the simple convolutional network is shown in Figure 5. Similarly to the initial network, the **conv** layers represent a convolution followed by a ReLU activation layer, **pool** layers represent pooling layers, and **fc** layers represent fully connected layers followed by a ReLU activation. The new addition to the model is the **dropout** layer which has replaced the previous **norm** layers. Dropout is mainly introduced to deal with regularization by introducing noise into the model and consequently reducing the amount of overfitting. The noise introduced by dropout is done by, with some probability, turning off some perceptrons to prevent the model from memorizing patterns that are specific only to the training data. Lastly another convolutional and pooling layer were added to add some complexity to the model since training was done on a GPU, and computation wasn't as much of a constraint. It is worth noting that one of the most important changes was

[8] `https://keras.io/`

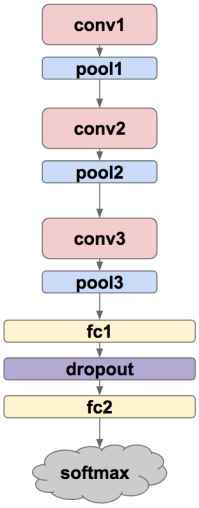


Figure 5: A Simple CNN Architecture [5]

that the image size was resized to **150 x 150** as opposed to the previous **24 x 24**. This was significant because a lot more information was retained in the image. Additionally a random rotation of 10° and zooom of 10% was applied at random on the training images to help alleviate the problem of overfitting.

The model depicted in Figure 5 was ran with 13,447 training, 4,490 testing and 4,487 validation samples on a machine with a *GeForce GTX 1080 Ti* GPU. The model was set to run with 50 epochs and a dropout rate of 50%. The main challenge in this step was to not overfit the model, which is why we opted to use the validation set. At every epoch, the model was trained and tested against the validation data that it had not encountered during training. If the accuracy against the validation set increased, that would indicate the model is improving; however, if the validation accuracy began to drop, this meant that the model was starting to overfit and it should be stopped. This strategy was employed: as soon as the validation accuracy stopped improving, or got worse, the model would terminate training after 3 epochs of constant outcome. Finally, the best recorded weights up to that point would be saved. After training for less than an hour, the model finished after 11 epochs and recorded the best weights with a **96.808%** accuracy. These were significantly higher results than the initial attempt, which could be largely due to increased image size, random rotations, zooming, dropout layer, and

model complexity. This was a significant enough result to end the experiments, but as per our initial proposal, we were determined to implement a Transfer Learning model and observe how it compared to the simple model.

5.3 Transfer Learning with VGG16

The strategy in this implementation of classifying the distracted driver images was to utilize the VGG16[2] CNN. This network was pre-trained on the ImageNet dataset and is easily accessible through the *Keras* library. The idea is that the pre-trained network would have learned features from previous data that might prove useful in predicting images of distracted drivers. This won't work out-of-the-box, of course, because the network was trained to predict things like cats and dogs, and doesn't directly transfer to the driver images. The strategy is to disconnect the the fully connected layers from VGG16, since they contain weights very specific to the original ImageNet dataset. In order to augment the model to fit the distracted driver dataset, a new sequential model was created with a similar fully-connected classifier as the Simple CNN in section 5.2. The model was built with the help of *Keras* documentation [5] and its architecture is shown in Figure 6. The training and validation images were propagated through the VGG16 network a single time, the output bottleneck features were saved into files and were then fed into the newly created full-connected classifier top model. It is important to note that only the top model was trained on the new dataset, and as such, it learned features only prevalent in the distracted driver dataset. The key takeaway is the retrained fully-connected classifier that was stitched to the top of the VGG16 model to be trained on the distracted driver images. It is also worth noting that, due to the reduced training time, we could afford to increase the image sizes to **224 x 224**, which enabled the model to retain even more of the information than our previous implementation5.

The training of the top model was done similarly to the training method applied in the Simple CNN model. The validation set was used with a patience of 3 and ran with 50 epochs. After 14 epochs, there were no more (positive) changes to the accuracy of the validation set, so the training process terminated and the best weights were saved. Running the model on the test set achieved an accuracy of **99.396%**. These results were quite surprising, as it took only a few minutes to train, and it achieved an even higher score than the previous implementation. These results will be further analyzed in the sections to follow.

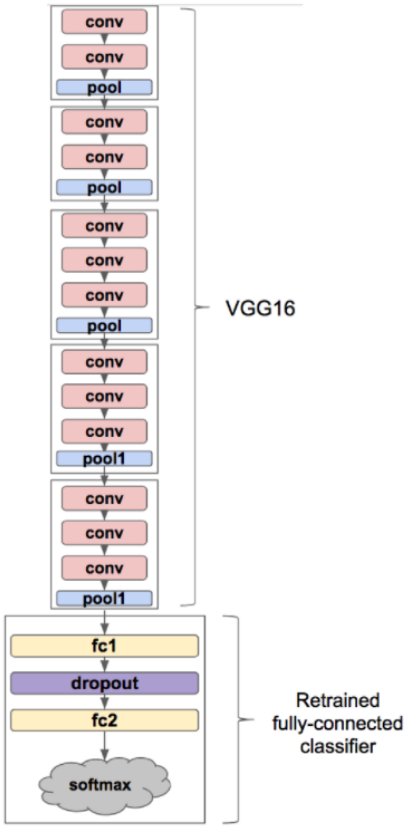


Figure 6: VGG16 with retrained fully-connected classifier [5]

CNN	Accuracy
Initial 4	10.418 %
Simple CNN 5	96.808 %
VGG16 Top Retrained 6	99.396 %

Table 2: Accuracies of the different models implemented

6 Results

To summarize the results found thus far, Table 2 shows the VGG16 model with a retrained fully-connected classifier to be the best performing model out of the three. We will divert our attention to this model for the remainder of the results, as it yields the most optimal outcomes thus far.

The VGG Top Retrained model seems to have some overhead cost in creating the bottleneck features, but since they can be stored offline and loaded at any time, the subsequent training of the fully connected layers takes significantly less time. It seems as though this is the best overall model thus far, since it not only results in the highest accuracy, but also trains in the least amount of time. It is so powerful that its training can even be done on a regular laptop CPU with very reasonable training time. Let us further analyze the model by looking at the Receiver Operating Characteristic (ROC) curve shown in Figure 7, plotting the True Positive Rate (TPR) against the False Positive Rate (FPR). An optimal prediction will have a TPR of 1 and FPR of 0, which corresponds to the top left corner of the graph. A steeper and quicker increase to the top left corner indicates a close to optimal solution, which can be seen in Figure 7. Another characteristic to take note of in the graph is its sharp bend, which indicates that there is a very clear and near-perfect separation.

The change in accuracy and the categorical cross entropy [9] (referred to as simply *Loss*) can be observed in Figure 8a as the model is trained through the different epochs. Both, the training and validation accuracies, progressively increase through the number of epochs in Figure 8 until they reach some threshold, indicating no further improvements are made and causing training to terminate. Similarly, in Figure 8b, both the validation and training loss decrease as the model learns, until eventually terminating when the model begins to overfit.

[9] `https://keras.io/losses/`

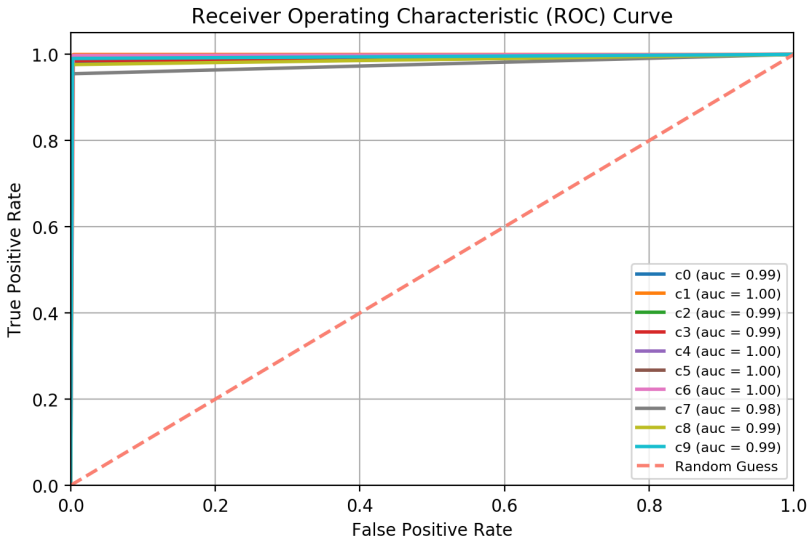
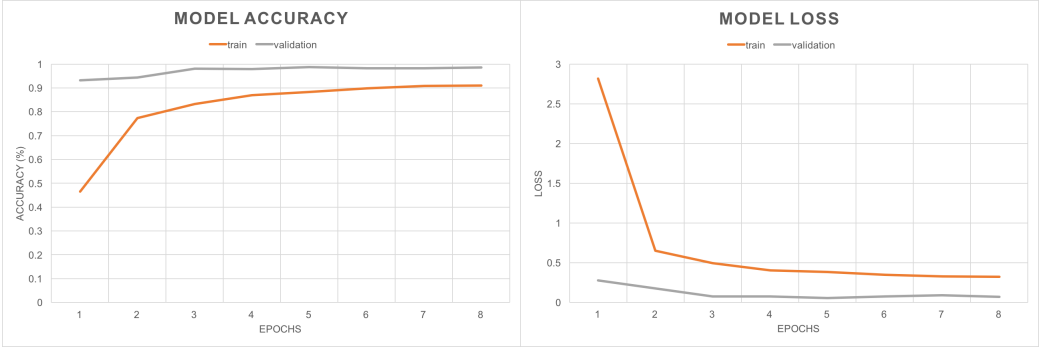


Figure 7: ROC Curve of distract driver predictions



(a) Model Accuracy (b) Model Loss

Figure 8: VGG16 Top Retrained - Model Accuracy and Model Loss

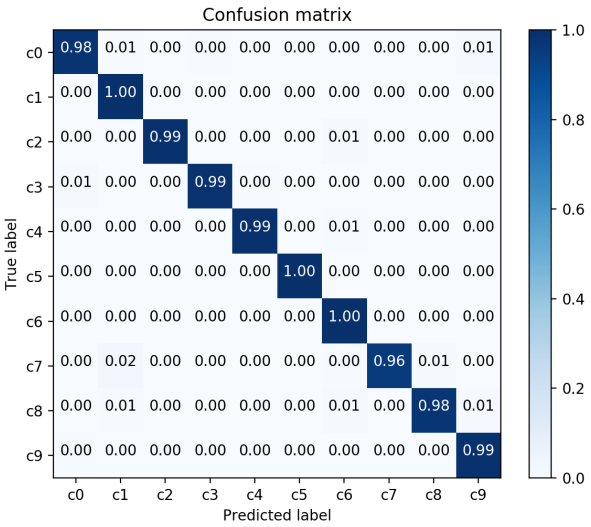


Figure 9: Confusion Matrix of The Model

Another interesting evaluation metric we can see is the Confusion Matrix shown in Figure 9. Most of the predicted labels are close to 100% accuracy. One, however, stands out: c7 with *0.96* accuracy is relatively low, compared to the rest of the classes. This class corresponds to *reaching behind*, as seen in Table 1. What we can conclude from the confusion matrix is that predicting if a driver is reaching behind is the most miss-classified, or most difficult category to predict. Furthermore, we can see from the confusion matrix that c7 is most commonly mislabeled as c1, meaning that reaching behind is typically mislabeled texting with the right hand. This is understandable, since drivers that are typically reaching behind perform this action with the right hand raised. As such, the model will, at times, think they are talking on the phone with the right hand if captured at a specific frame of motion.

Lastly, we were able to save the weights of the top model, run never-seen images of distracted drivers, and display the top-5 predictions that the model made. The results of a sample image are shown in Figure 10, where the actual label of **c0: Safe driving** is predicted as **99.978%** of being a safe driver.

```
1. safe_driving: 99.978167%
2. talking_to_passanger: 0.015883%
3. texting_right: 0.005701%
4. texting_left: 0.000247%
5. operating_radio: 0.000004%
```

Figure 10: A driver image with ground truth of class 0: "Safe Driving"

7 Conclusion

This work has looked at solving the detection of distracted drivers through images obtained from the *State Farm Distracted Driver Detection*[4] competition on Kaggle. By using a pre-trained VGG16 network, extracting the bottleneck features, and retraining a new set of fully-connected layers, the model was able to achieve 99.978% accuracy on test data. Despite given the task of classifying very specific classes, the model is evidently able to accomplish that with great success. Further evaluation revealed that the most miss-labeled class was reaching behind, often confused with the driver talking on their phone with the right hand. Overall, the model has proven to be very effective at predicting distracted drivers, and will hopefully, one day, aid in preventing further injuries and deaths resulting from distracted driving. In regards to future work, we hope to fine tune a few of the lower layers of the VGG16 network by freezing them and retraining the remaining ones on the distracted driver dataset.

References

[1] Nishant Shukla. *Machine Learning with TensorFlow*. Manning Publications, 2017.

[2] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[3] Leonard Blier. A brief report of the heuritech deep learning meetup 5, 2016.

[4] Kaggle Inc. State farm distracted driver detection, 2016.

[5] Francois Chollet. Building powerful image classification models using very little data, 2016.