Final Project – Distracted Drivers Classification

Tiffany Duong, MSDS 458(SEC55)

1. Introduction

Insurance companies are always looking for methods to improve pricing for their insurance products. One method of trying to accurately assess the risk of the customer and pricing them appropriately for auto insurance would be through the observation of driving behaviors. Telematics is currently being used as a tool to observe how much and how "safely" an individual drives. With this project, I will be using a different method that may be used to sophisticate insurance pricing. I will be looking at images of distracted and non-distracted driving behavior and will create a model that helps to accurate classify these behaviors into ten different classes.

For the modeling component, a pretrained VGG-16 model is used and the experiment will dive into transfer learning, as well as building on top of the pretrained model to tailor the input dataset.

2. Model & Experiment Design

The dataset that I am using is from the State Farm Distracted Driver Detection competition on Kaggle. The dataset provided is separated into two directories: train (22K images) and test (80K images). The train directory separates the images based on the 10 different driving behavior classes described below, alongside with the count of images per class:

Class	Image Count (Train)
Safe Driving	2,489
Texting – Left	2,346
Talking on the Phone – Left	2,326
Drinking	2,325
Talking on the Phone – Right	2,317
Operating the Radio	2,312
Texting – Right	2,267
Talking to Passenger	2,129
Reaching Behind	2,002
Hair and Makeup	1,911

Figure 1: Sample Image of "Safe Driving"

Figure 2: Sample Image of "Texting - Right"

The experimental design was to follow a train set (80% of training), validation (20% of training), and holdout (testing), however due to the sheer amount of test data, my computer would repeatedly crash. Therefore, the final experimental design would follow a train set (80%) and validation set (20%), both based on the original training dataset but randomly separated into the two sets to prevent bias.

Since this is an image classification problem, I decided that CNNs would be the most appropriate network to train and model on. For the initial experimental design, I wanted to train a VGG-16 network from scratch, but this proved this be extremely memory intensive and time-consuming (training time was estimated to be well over 40 hours). From there, I decided to use a pretrained VGG-16 (trained on ImageNet) model and used this as an opportunity to experiment with transfer learning.

3. VGG-16 Network

For this project, I used a pretrained VGG-16 model and the base model architecture follows:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_21 (InputLayer)        (None, 224, 224, 3)       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
=================================================================
Total params: 14,714,688
Trainable params: 14,714,688
Non-trainable params: 0
_________________________________________________________________
```

The input to the first convolutional layer is a fixed 224x224x3 (RGB) image and it is passed through convolutional layers and max pooling layers, which helps to down-sample the input

representation (from 224x224 to 112x112 to 56x56 to 28x28 to 14x14 and finally 7x7) to reduce computational complexity. While the max pooling layers help to down-sample the input representation, this down-sampling also helps to make assumptions about the features by extracting the most important features (such as edges) from the input. For activations, they all utilized ReLU activation. The width of VGG-16 starts off at 64 and as seen from the architecture, it increases to 512, due to the width increasing by a factor of 2 after every max pooling layer.

For the transfer learning aspect, I decided to not include the top layers that could be included from the pretrained VGG-16. Instead, I chose to add a flattening layer to flatten the data into a 1-D array, which will be used as the input to the two fully connected (Dense) layers (4098 units each), and then the output layer to predict the 10 classes, which utilizes a softmax activation function.

```
from keras.models import Model
#Adding custom layers
x = vgg16.output
x = Flatten()(x)
x = Dense(4096, activation="relu")(x)
x = Dense(4096, activation="relu")(x)
predictions = Dense(11, activation="softmax")(x)
```

The pretrained model was trained using ImageNet, which is considerably different from the distracted driving dataset being used for the project – which could lead to poorer performance accuracy. To tailor the pretrained VGG-16 model to the input data, I decided to freeze all layers, except for the last set of convolutional and max pooling layers, and the fully connected layers – which will be trained based on our dataset of interest. The model was compiled using categorical cross-entropy and Adam's optimizer, with accuracy as the performance metric. For

training, the amount of epochs was set to 20 with a batch size of 64, and early stopping (based on validation) is used to prevent additional training.

4. Results

The model was trained on 17,938 samples and validated on 4,485 samples. The VGG-16 model yielded a performance accuracy of **99.62%** and it took approximately 45 minutes to train, with early stopping activating after only 5 epochs.

The model got to these results through several iterations. Performance accuracy was significantly lower during the first few iterations when I used a batch size of 268. Using a smaller batch size of 64 yielded better results because it looks like using larger batch sizes degrades the ability of the model to generalize better, perhaps due to how using small vs. large can change how the model converges to a minimum. I also decreased the learning rate from 0.001 to 0.001 and the model performed better – this could be due to using an even smaller learning rate would prevent the learning rate from not being able to converge to the minimum directly (i.e. bouncing back and forth over the minimum).

I think one of the leading factors of why the model performed well is because of the architecture itself. Unlike some other CNNs, VGG-16 utilizes smaller filters (starting with 3x3 kernel filters). This is advantageous because having multiple stacks of smaller sized filters allows the model to grow deeper and learn more complex and finer (in detail) features based off the input (due to the use of more filters), while not as being as computationally expensive as using less, large sized filters.

5. Summary/Conclusion

This project was a good exercise in training a very deep convolutional neural network and using transfer learning – both of which I had never had a chance to work with before. A tradeoff between compute power/time with accuracy should always be a consideration when trying to decide whether or not to use a deep network architecture.

The goal of this project was to build a model based off of State Farm's distracted driver data and to correctly identify images into the right "driving behavior" classes. The results of this project demonstrated a 99.62% classification accuracy when using a partially pretrained VGG-16 network. In practice, a new customer may allow an insurance company to have a camera take many photos of them while driving so that the customer could have its auto premium adjusted based on how "well" they demonstrate "safe driving" behavior. Similar models to the one used for this project can be used as a tool to help accurately price customers, leading to customers feeling like they are paying "as they need", as well as allow the insurance company to assess and quantify risk appropriately – a high payoff on both ends.