Western Engineering

ES 4499 – Integrated Capstone

Distracted Driving

March 28th, 2019

Matthew Clark – 250860456

Eric Cumiskey – 250844593

Pavlo Grinchenko - 250858143

Joerg Oxborough – 250849310

Mark Rabinovich – 250808067

Overview

Distracted driving is a growing epidemic affecting many young and elderly Canadians. Recently many provinces have enacted laws prohibiting the usage of cellular devices while operating a vehicle however, enforcement of these recent laws has proven to be troublesome as it is very difficult to detect when a driver is distracted. Instead of focusing on aiding enforcement of these laws, the group has chosen to reduce the occurrence of distracted driving. This will be achieved by recognizing when the driver is distracted and notifying them, so they can refocus their attention.

To recognize when the driver is distracted, the prototype uses a two-tiered machine learning approach. The visual and sensor data is fed into a Convoluted Neural Network (CNN) which will predict the appropriate steering wheel angle. Then, the predicted steering wheel angle, along with the sensor data is fed into an Extreme Random Trees (ERT) classifier. To simplify development of this project, some substitute products were used. The game "Euro Truck Simulator 2" was used to simulate real world driving visual data. The Logitech G920 Driving Force Race Wheel was used to simulate real world wheel and pedal sensor data.

Through training, the CNN reached a correlation of 0.918 and the ERT model reached 82.6% accuracy. Testing of the entire process shows this model to be fairly accurate and therefore considered a success.

The Github repository of all the code for the project is located at:

https://github.com/Joerg-ffs/Capstone-Project

Demo video can be watched here: https://youtu.be/4tze6cgGxw8

Table of Contents

Introduction

Driving is a very important aspect of modern society. It is used to get from point A to B with relative quickness. However, as cars became faster and more powerful, the rate of vehicular related deaths also increased. To combat this, manufacturers implemented safety features like seatbelts and crumple zones. Governments recognized that driving while drunk led to more accidents and thus passed laws prohibiting such actions. Over the years, cars have gotten safer and more regulations have been passed. It is now illegal to drive under the influence of drugs such as marijuana as well as using as cellphone while driving. The term "distracted driving" is still in its infancy. Some people believe it to only reference using a cellphone while driving. For the purpose of the project "distracted driving" will refer to operating a vehicle while not paying attention to the road. The problem local governments have with enforcing this is the lack of detection options available. It is quite simple to detect a DUI using a breathalyzer. Due to the multitude of ways one can become distracted while driving, it is impossible to quantify an individual's distraction level post incident. Thus, a live detection system is the only solution. The group's first iteration of prototypes yielded wildly different solutions to the same problem. The group identified the strengths and weaknesses and decided on one solution: detecting a driver's distraction level using both a camera and sensor data. The solution was chosen because it can detect all types of distracted driving (other solutions were limited to phone use) and it was proactive rather than reactive.

The final product of this solution would utilize the CAN-BUS system of vehicles as well as a camera mounted on the vehicle to determine whether the driver is distracted or not. However, starting with a first prototype in a vehicle would be too complex, unethical, and dangerous. Therefore, a simulated environment was used. Several options were considered including Grand Theft Auto 5, Forza Horizon 4, and Udacity Car Simulator. The group chose Euro Truck Simulator 2 for several reasons. The game could be played on Windows as well as Linux, the controls were more realistic than other games, and the road environment was the most realistic. The controller options were mouse and keyboard, an XBOX controller, or a Logitech G920 Driving Force Race Wheel. The Logitech G920 was chosen as it provides very realistic steering wheel and pedal data and is shown in Figure 1.

Figure 1: The Logitech G920 Driving Force Race Wheel system

The Model

To detect distracted driving from only driving inputs and image data, the group decided on a two-tiered system. The first tier will use the steering wheel angle as well as the image to predict an ideal angle for the user to drive. The concept is very similar to that used in autonomous driving, but the ideal angle is not used to drive with. Instead, the predicted and actual angle along with the other sensor data (gas and brake pedals) are fed into another model. This model then predicts the distraction level of the user. If the distraction threshold is breached the system will output an auditory clue, so the driver can refocus their attention on the road. The process is visualized below in Figure 2.

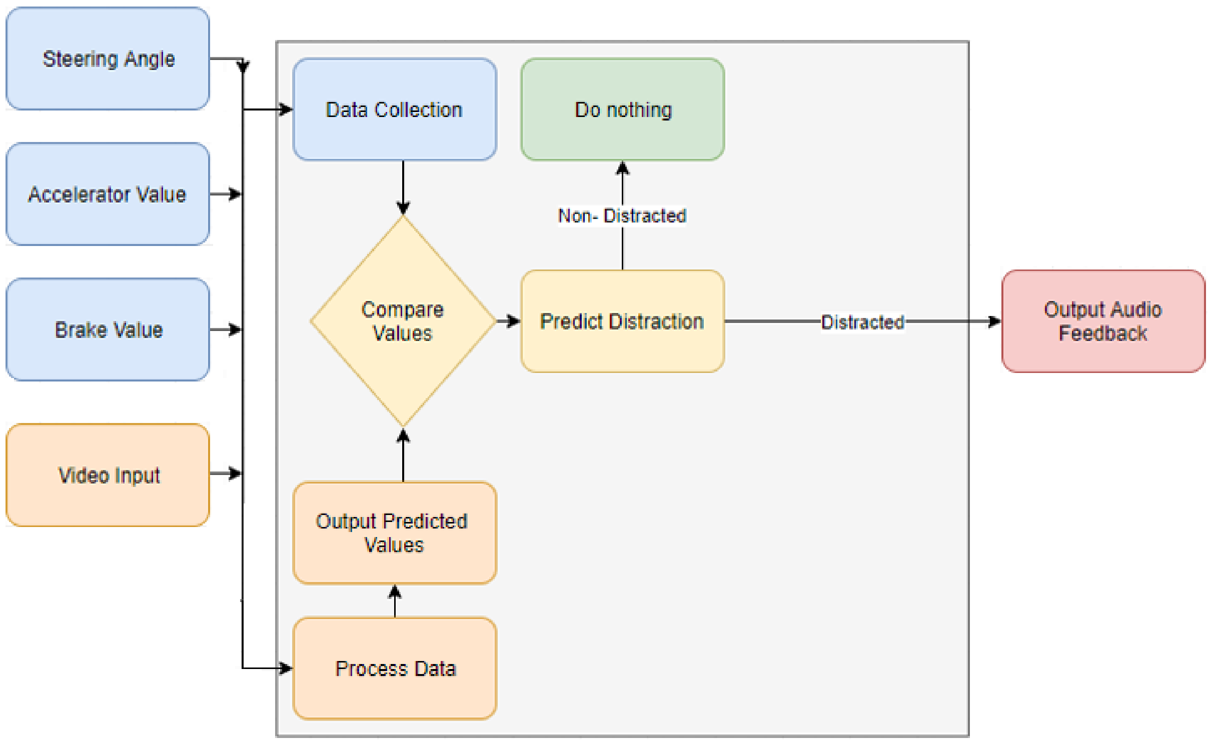


Figure 2: Flowchart of final prototype system

.

Tier One

The first tier of the system functions exactly like an autonomous vehicle except the output is not used for driving but will be fed into another model. To achieve this result, the group utilized a Convoluted Neural Network (CNN). The architecture for this CNN was published and open-sourced by NVIDIA to speed up the development of autonomous vehicles. As shown in Figure 3, the input to the CNN is a 200x66 image. The output of this network is the vehicle control angle.

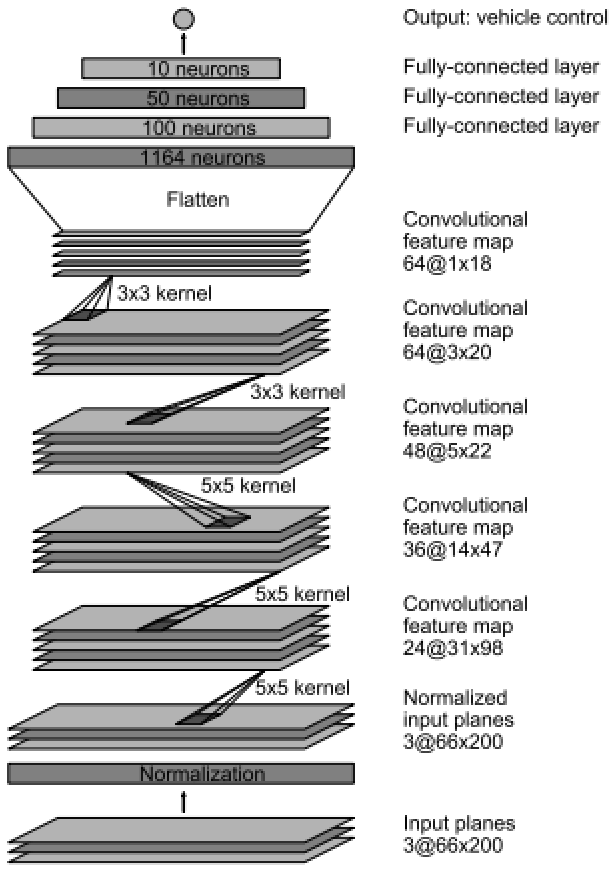


Figure 3: NVIDIA CNN architecture

Data Collection

To collect the data needed to train the CNN, a Linux operating system is needed. Linux offers direct access to the input stream of the all inputs connected to the computer using the Python library “evdev”. This is simpler and easier than solutions offered on Windows. To detect what input stream the racing wheel is connected to, the following command must be run in the terminal.

```
cat /proc/bus/input/devices
```

Once the event stream is identified (i.e. 22), the script can connect to the input stream with:

```
dev = InputDevice('/dev/input/event22')
```

Within infinite loops, the for loop is triggered every time the status of one of the inputs is changed. For example, the steering wheel turning or a pedal/button being pressed. The value of the event can be accessed using event.value and the code, meaning what input was changed, by event.code.

```
while 1:
    for event in dev.read_loop():
            print(categorize(event))
            print("code: " + str(event.code) + "value: " + str(event.value))
```

These inputs can now be captured and saved into a csv file. The road data also needs to be captured and is accomplished using the Python library “mss”. First the screen area is defined.

```
monitor = {'top': 437, 'left': 766, 'width': 400, 'height': 300}
```

Then the area is captured, saved as a .png file, and the path is written to the csv file. Figure 4 shows an example of the screen capture.

```
sct_img = sct.grab(monitor)
mss.tools.to_png(sct_img.rgb, sct_img.size, output=output)
send_data("image", output)
```

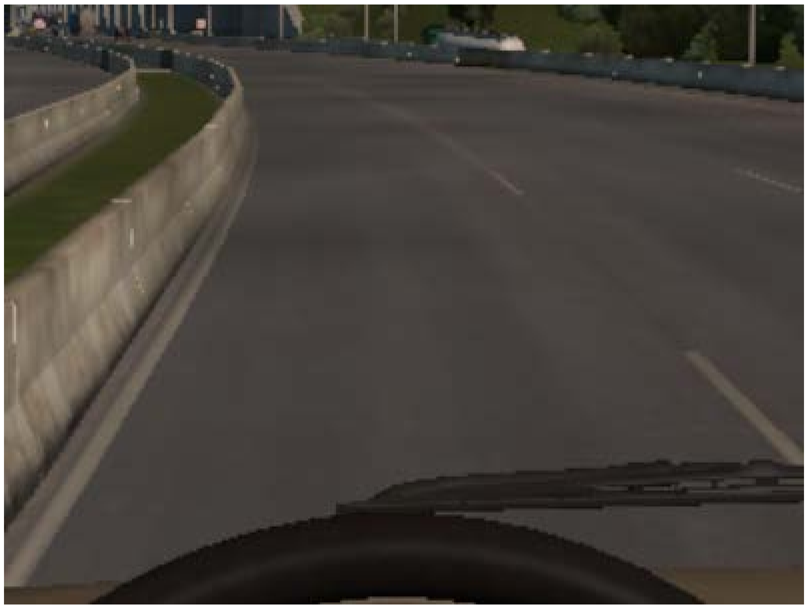


Figure 4: Sample screen capture

As mentioned, the inputs and the image path and written to a csv file.

```
with open(r"tier_1_inputs.csv", "a", newline='') as f:
                writer = csv.writer(f)
                writer.writerow(statusList)
```

This csv file, with the images saved in a folder can then be used to train the CNN. The images are compressed into a .tar.7z file to be transferred easier. The files used in this step were device_config.py and Inputs_G920.py

Training

To train the model, a Jupyter notebook was used in place of individual python scripts. The main reason for this format was to utilize a free platform offered by Google: Google Colabratory. Google Colab offers a cloud based service for machine learning purposes. It gives users access to a NVIDIA Tesla K80, an industry level GPU. By using this, the group's training times were reduced by a factor of 100. This dramatically sped up development as new iterations of models could be trained and tested much faster than before.

Before training the model, the input data must be processed and normalized. The image data is first unzipped into Google Colab.

```
!apt-get install p7zip-full
!p7zip -d frames.tar.7z
!tar -xvf frames.tar!7z e frames.7z
!tar -xvf frames.tar
```

Next the csv is read and the image paths and angles are separated into arrays using the libraries pandas and numpy.

```
data = pd.read_csv("tier_1_inputs.csv")

data_img = np.array(data['Image'])
data_img = data_img.reshape(-1, 1)
data_angle = np.array(data['Angle'])
data_angle = data_angle.reshape(-1, 1)
```

The image array is looped through and the image is read and resized from a 400x300 to 200x66 image for the CNN. An example of a 200x66 resized image is shown in Figure 5.

```
for j in data_img:

    #gets angle and image path from array
    angle = data_angle[i]
    img_string = str(data_img[i,0])
    img_string = img_string[4:]

    #reads image and resizes it from 400x300 to 200x66
    img_array = cv2.imread('frames/' + img_string, 3)
    img_array = img_array[80:212,0:400]
    img_array = cv2.resize(img_array, (200,66))
```

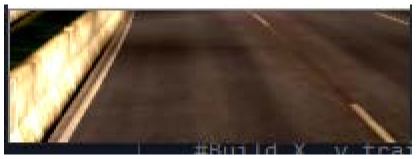


Figure 5: The final 200x66 image to be fed into the CNN

Due to the stability of normal driving, the steering wheel angles are mostly very close to centre which is numerically 0. This is problematic when training the model as the CNN could learn to just predict 0 every time and get good accuracy. The distribution of the data needs to be augmented so it is closer to a uniform distribution.

This code checks the last 10 angles and if the sum is too small, they are not included in the final dataset. This helps reduce long stretches of zero angles in the data as well.

```
sum_angle += abs(angle)
if sum_angle > 0.15:
    features.append(img_array)
    labels.append(angle)
if i % 10 == 0:
    sum_angle = 0
```

The extreme angles are also duplicated, and the reversed version is added as well. The reversal of image data and angle data is a very common process in autonomous driving to artificially increase the dataset.

```
if abs(angle > 0.25):
    features.append(cv2.flip(img_array, 1))
    labels.append(angle*(-1))
```

The final arrays of images and angles are stored as pickle files so they can be used for training at any time.

```
with open("features", "wb") as f:
    pickle.dump(features, f, protocol=4)
with open("labels", "wb") as f:
    pickle.dump(labels, f, protocol=4)
```

The library matplotlib allows us to plot the angle data in a histogram to see the distribution

```
plt.hist(labels, bins=20)
plt.show()
```

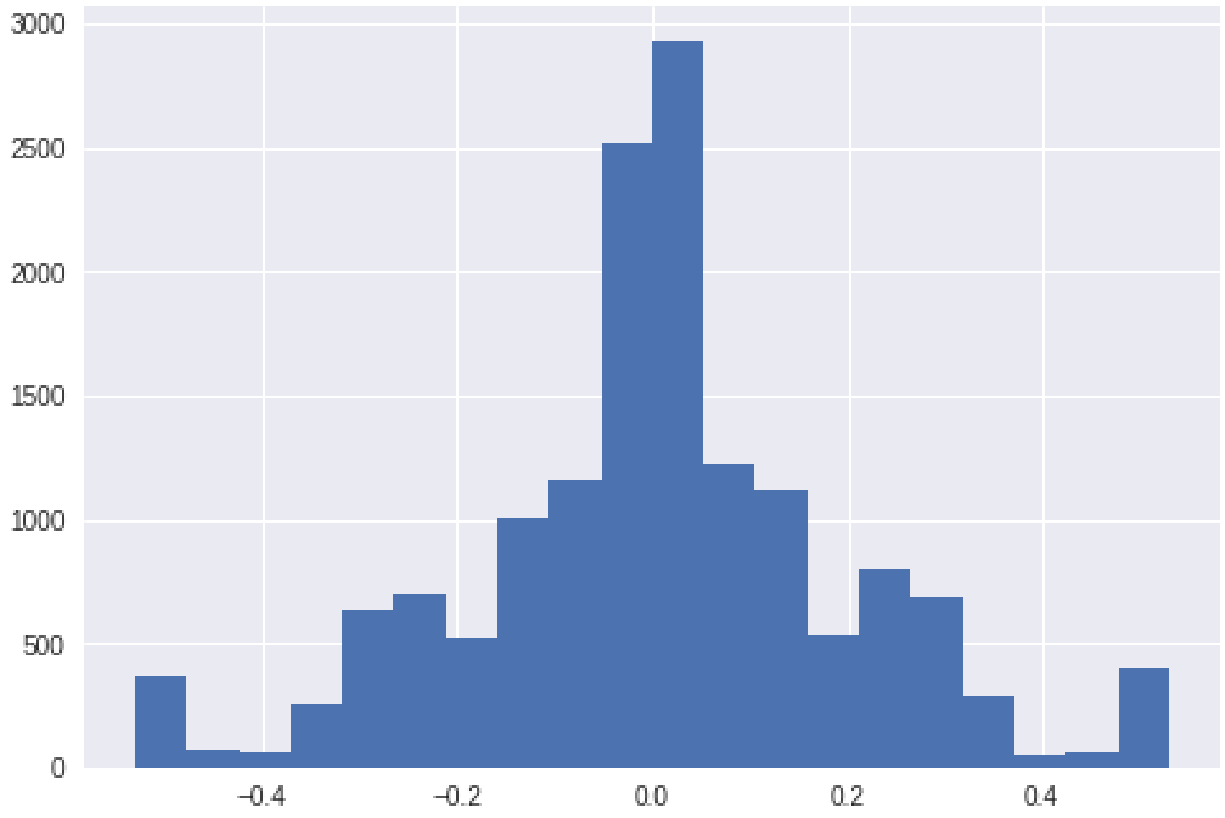


Figure 6: Histogram of augmented angle values

As shown in Figure 6, the data is centered around zero but still has meaningful amounts of data at other angles. In comparison, the raw data in Figure 7 is all around zero.

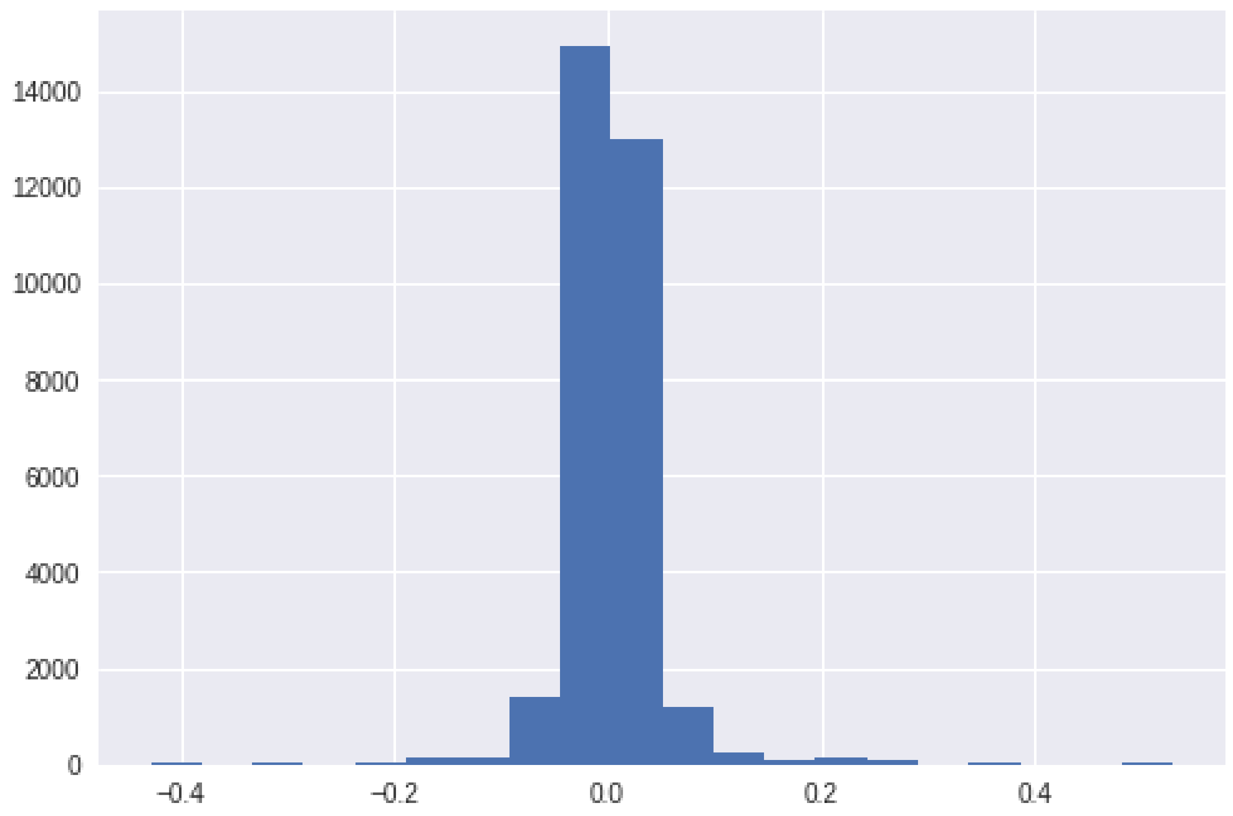


Figure 7: Histogram of raw angle data

Now that the training data is stored in pickle files, the CNN can be trained. The data is first loaded into arrays

```
with open("features", "rb") as f:
    features = np.array(pickle.load(f))
with open("labels", "rb") as f:
    labels = np.array(pickle.load(f))
```

The entire dataset is flipped again to augment it further.

```
features = np.append(features, features[:, :, ::-1], axis=0)
labels = np.append(labels, -labels, axis=0)
```

The dataset is then shuffled and split into training and testing data at an 80/20 split.

```
features, labels = shuffle(features, labels)
train_x, test_x, train_y, test_y = train_test_split(features, labels,
random_state=0,test_size=0.2)
```

The training data is used to train the model and the testing data is used to score the model. This allows the user to see the performance of the model while the model is training and helps reduce overfitting.

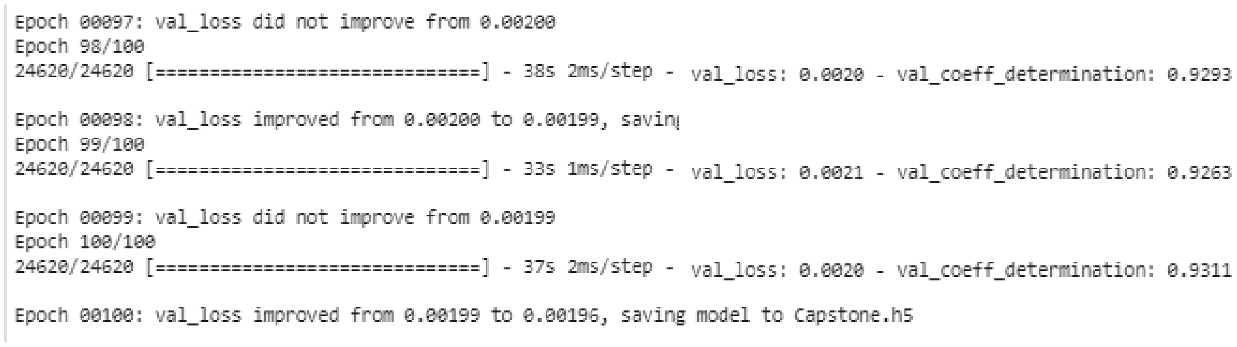
The model is then defined. This is the NVIDIA CNN architecture.

```
model = Sequential()
model.add(Lambda(lambda x: x/127.5-1.0, input_shape=(66,200,3)))
model.add(Conv2D(24, 5, 5, activation='elu', subsample=(2, 2)))
model.add(Conv2D(36, 5, 5, activation='elu', subsample=(2, 2)))
model.add(Conv2D(48, 5, 5, activation='elu', subsample=(2, 2)))
model.add(Conv2D(64, 3, 3, activation='elu'))
model.add(Conv2D(64, 3, 3, activation='elu'))
model.add(Dropout(0.5))
model.add(Flatten())
model.add(Dense(100, activation='elu'))
model.add(Dense(50, activation='elu'))
model.add(Dense(10, activation='elu'))
model.add(Dense(1))
```

The model is then finally trained on the dataset, with a batch size of 13 and 100 epochs.

```
model, callbacks_list = keras_model()
model.fit(train_x, train_y, validation_data=(test_x, test_y), epochs=100,
batch_size=13, callbacks=callbacks_list)
```

Figure 8 shows example epochs of the model during its training process. The loss is defined as mean squared error and the metric is correlation.



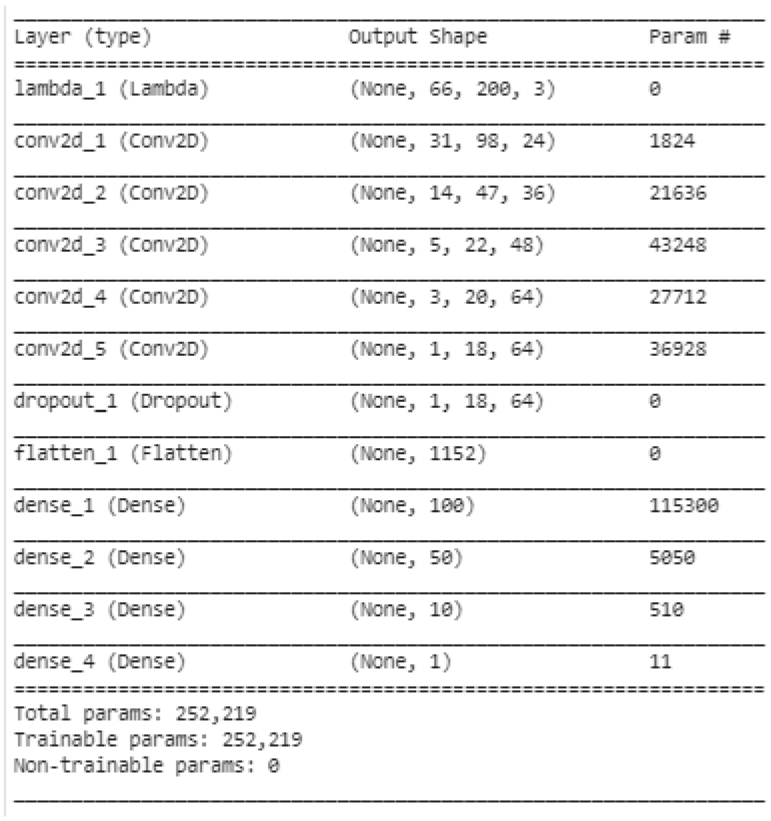
```
Epoch 00097: val_loss did not improve from 0.00200
Epoch 98/100
24620/24620 [==============================] - 38s 2ms/step - val_loss: 0.0020 - val_coeff_determination: 0.9293

Epoch 00098: val_loss improved from 0.00200 to 0.00199, savin
Epoch 99/100
24620/24620 [==============================] - 33s 1ms/step - val_loss: 0.0021 - val_coeff_determination: 0.9263

Epoch 00099: val_loss did not improve from 0.00199
Epoch 100/100
24620/24620 [==============================] - 37s 2ms/step - val_loss: 0.0020 - val_coeff_determination: 0.9311

Epoch 00100: val_loss improved from 0.00199 to 0.00196, saving model to Capstone.h5
```

Figure 8: Metrics from final epochs of training



```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lambda_1 (Lambda)            (None, 66, 200, 3)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 31, 98, 24)        1824
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 14, 47, 36)        21636
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 5, 22, 48)         43248
_________________________________________________________________
conv2d_4 (Conv2D)            (None, 3, 20, 64)         27712
_________________________________________________________________
conv2d_5 (Conv2D)            (None, 1, 18, 64)         36928
_________________________________________________________________
dropout_1 (Dropout)          (None, 1, 18, 64)         0
_________________________________________________________________
flatten_1 (Flatten)          (None, 1152)              0
_________________________________________________________________
dense_1 (Dense)              (None, 100)               115300
_________________________________________________________________
dense_2 (Dense)              (None, 50)                5050
_________________________________________________________________
dense_3 (Dense)              (None, 10)                510
_________________________________________________________________
dense_4 (Dense)              (None, 1)                 11
=================================================================
Total params: 252,219
Trainable params: 252,219
Non-trainable params: 0
_________________________________________________________________
```

Figure 9: Model architecture, showing shape of data as it progresses through layers

The Jupyter notebook used for Tier 1 training is Tier_1_Training.ipynb.

Results

As shown in Figure 10, the CNN had a correlation coefficient of 0.93 with the testing data. When used in real time, the predicted angle closely follows the actual angle. In practice, the model tends to predict larger absolute angles better than angles close to 0. This could be due to the dataset being augmented, however the group believes it is due to slight under and overcorrections that drivers make during normal driving. The model was trained on imperfect driving as the model is using behavioural cloning by copying the group's driving performance. However, the data in Figure 7 shows a 0.74 R^2 value which means high correlation.

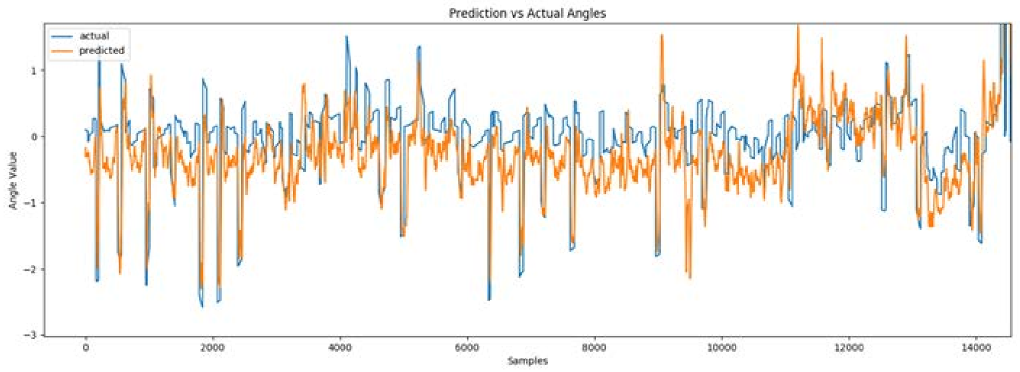


Figure 10: Actual vs Predicted angles over a 20-minute period

Tier Two

The second model of the prototype system is an Extremely Random Trees classifier. The input data from the racing wheel is fed into the model as well as the predicted angle from Tier 1 of the system. This model was chosen as it had the best performance in the initial prototype phase as well as for this dataset. The output of this model is a confidence value between 0 and 1 of how distracted the driver is.

Data Collection

The data collection for the second tier is quite similar to the collection for the first. The key differences are the image data is not stored and the predicted angle is recorded.

The predicted angle is output from the Tier 1 model.

```
pred_angle = model.predict(X, batch_size=1, verbose=0)[0][0]
```

The angle data is normalized to better compare against the normalized angles the model outputs. The wheel output values from 0 to 69504 so this function puts most angle between -1 and 1.

```
angle = (event.value - 32752) / 32752 * 2
```

This function creates a rolling array of the last ten predicted angles to smooth out the prediction values. This helps with predicative accuracy.

```
if len(prev_angles) < 10:
    prev_angles.append(pred_angle)
else:
    prev_angles.pop(0)
    prev_angles.append(pred_angle)
```

Depending on if the user is recording distracted or non-distracted data the last index of the list statusList. A 0 means not distracted and a 1 means distracted. These values are then written to a csv file.

```
with open(r"tier_2_training_data.csv", "a", newline='') as f:
            writer = csv.writer(f)
            writer.writerow(statusList)
```

The script used for Tier 2 data collection is Inputs_G920_Distraction.py.

Training

A Jupyter notebook is used again to take advantage of the Google Colab performance capabilities.

The csv file is read into a dataframe and then split into x and y arrays.

```
data = pd.read_csv("tier_2_training_data.csv")
```

```
data_y = data['Distraction']
data_x = data.drop("Distraction", axis=1)
```

Then the input array, which consists of real angle, predicted angle, gas, and brake values is split from a 2D array of shape (242100, 4) into an array of shape (2421, 4, 100). This chunks the array into segments of 100 which help improve the accuracy of the model.

```
X = np.array_split(data_x, 2421)
y = np.array_split(data_y, 2421)
```

The model only accepts 2D arrays however, so the array must be flattened.

```
X = X.reshape(2421,400)
y = y.reshape(2421,100)
```

The x and y data is split into training and validation datasets with a split of 80/20.

```
X_train, X_validation, Y_train, Y_validation =
model_selection.train_test_split(X, y,
test_size=validation_size, random_state=seed)
```

The model is then trained on the data sets with a scoring metric of accuracy and 50 n_estimators. The mean and standard deviation of the score is then printed.

```
ERF = ExtraTreesClassifier(n_estimators=50, max_depth=None, min_samples_split=2,
random_state=0)
ERF.fit(X, y)
kfold = model_selection.KFold(n_splits=90, random_state=seed)
cv_results = model_selection.cross_val_score(ERF, X_train, Y_train, cv=kfold,
scoring=scoring)
results.append(cv_results)
msg = "%f (%f)" % (cv_results.mean(), cv_results.std())
```

Lastly, a pickle of the model is saved so that the model can be used for real time prediction.

```
pickle.dump(ERF, open( "Tier_2_Model.pkl", "wb" ) )
```

The notebook used for training is Tier_2_Training.ipynb.

Live Prediction

The live prediction program is similar to the program used for data collection. The screen area and inputs are captured in the same manner as well as the predicted angle.

Both models are loaded into the program.

```
model = load_model('Tier_1_Model.h5', custom_objects={'coeff_determination':
 coeff_determination})
    ERF = pickle.load(open('Tier_2_Model.pkl', 'rb'))
```

As explained above, the Tier 2 model is trained in sets of 100. This function creates a rolling array of the past 100 sets of inputs. This output is then fed into the predict statement.

```
if np.size(inputs_array, 0) < 400:
                            inputs_array = np.append(inputs_array, inputs)
                        else:
                            inputs_array = np.array(inputs_array)
                            inputs_array = np.delete(inputs_array, [0, 1, 2, 3])
                            inputs_array = np.append(inputs_array, inputs)
                            inputs_array = inputs_array.reshape(1, -1)
```

The inputs are fed into the model and the confidence of distraction value is the output.

```
prediction =  ERF.predict_proba(np.array(inputs_array))[0][0][1]
```

This function looks at the last 50 predictions and if they are over the threshold (currently set at 75), will output the text 'DISTRACTED'.

```
sum_pred += prediction
if j % 50 == 0:
   if sum_pred*2 > DISTRACTION_THRESHOLD:
       print('DISTRACTED')
   sum_pred = 0
j += 0.5
```

In addition, a scrolling plot of the sum_pred variable is displayed with this function as shown in Figure 11.

```
y = np.append(y, sum_pred*2)
y = y.reshape(-1, 1)
plt.plot(y, '.r-')
plt.pause(0.0000000000000000000000001)
```

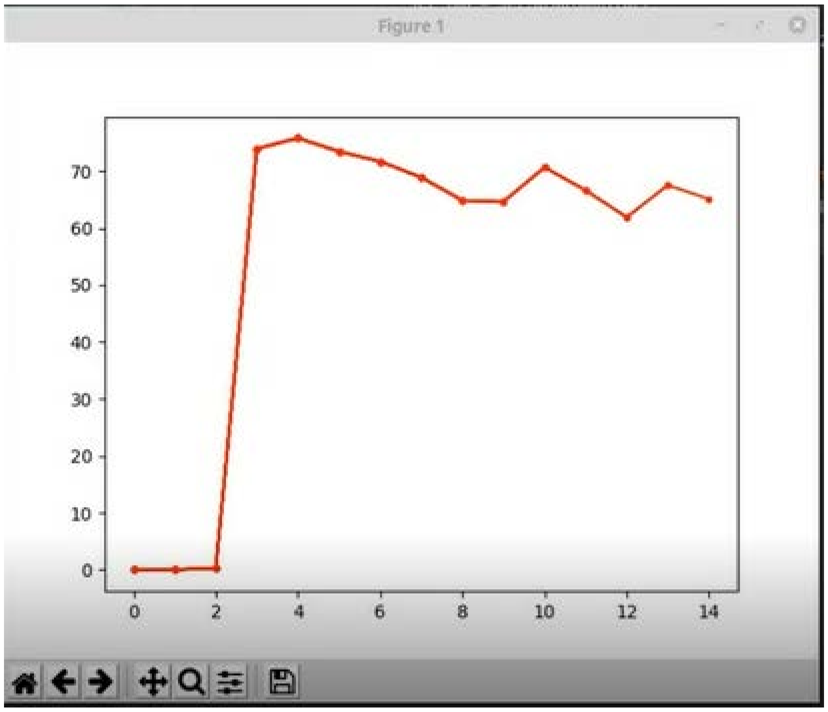


Figure 11: Real time graph of prediction values

As well a steering wheel showing the predicted angle of the model is displayed using the following code and the wheel is shown in Figure 12.

Figure 12: The real time rotating steering wheel showing the predicted angle

Conclusion

The results from the CNN indicate a 93% correlation between real and predicted angle data. In practice, the model has a R^2 value of 0.74 which is very accurate. The ERT model has an 82% accuracy and when smoothed, is quite accurate at detecting distracted data in real time. Based on these results, the group believes this project to be a success.

The final product would involve using a vehicle's CAN-BUS and OBDII system to send the steering wheel angle as well as gas and brake pedals to an onboard computer. A mounted camera would capture images and both inputs would be fed into the live prediction program. This was unfeasible to do as a capstone group as the process could be dangerous and the group was unable to access a car with a mapped-out CAN-BUS system. Only new, popular models of cars like 2016+ Honda Civics have had their CAN-BUS system mapped and made publicly available. Mapping out the system of an available vehicle is outside the scope of this project as the timeline would be too great. Therefore, the group could not achieve detecting distracted driving within an actual vehicle. However, distracted driving can be detected at a reasonable rate inside a simulated environment.

A short demonstration video of the final prototyped can be viewed here:

https://youtu.be/4tze6cgGxw8

The Github repository of all the code for the project is located at:

https://github.com/Joerg-ffs/Capstone-Project