CISC 452 Final Report
Due: December 6th 2022
Submission for Group **#41**

Putting the Brake on Distracted Driving using Deep Learning & Computer Vision Techniques.

Authors:

Fiona LeClair-Robertson, #20163405
Farwa Abedi, #20191535
Priyank Gopalbhai Thakkar, #20189336
Sara Litwiniuk, #20175732

1. Motivation

Every day, millions of lives depend on the integrity of vehicle owners to appropriately follow road safety procedures while driving. Though despite these high stakes, “nearly 4 million car collisions are caused by distracted drivers in North America every year”[1]. In Canada, distracted driving is responsible for over 21% of road fatalities[5], and cost car insurance companies $102.3 million annually[3]. Cell phone usage has been condemned more than other forms, leading mobile device companies to introduce “driving mode” and hands-free solutions. Though in reality, your “chances of getting into a collision are doubled” by taking your eyes off the road for merely 2 seconds[2] - regardless of the activity you are doing. As car manufacturers continue to integrate more technology features inside the vehicle, the situation only worsens - prompting the increased need for intervention.

2. Problem Description

Distracted driving refers to any driver that cannot devote their full attention to the road due to either visual distractions, cognitive distractions, or both[2]. For our research, we will be examining minimizing the risk of visual distractions, as body movement indicators can be picked up by convolution neural networks. Our group will be determining which objection detection models are best suited to identify these drivers distracted behind the wheel with the highest level of accuracy. We intend to also consider accessibility of software distribution, and thus will prioritize models that use less computational power. One of the most prominent limitations of current existing models, is that they require a very large amount of computational power & unrealistically high quality images for their training datasets. By creating the most ideal situation for the neural network with multiple available angles, 4k image quality, and tuned lighting, these models are not adequately trained to perform in a standard car environment where a simple dashcam is likely to be mounted. Our team wants to approach this problem while also considering the distribution and real life implementation of our solution, where the model is likely to be run on a compact device with one simple angle. Finally, we want the model to be able to detect various forms of distracted driving, to diversify the capabilities of the network.

3. Contribution

For this experiment, each member has selected a unique object detection model to analyze and extend on, with the goal of improving its accuracy. Given that we want to explore options requiring low computational power, Fiona LeClair-Robertson & Farwa Abedi have been assigned networks that involve depthwise separable convolutions (MobileNet and Xception), which will be contrasted against a VGG-16 model studied by Sara Litwiniuk, and a ResNet50 model contributed by Priyank Thakkar. Each individual part has been researched, analyzed, typed, and added by the respective group member. The general parts of the report have been compiled together by Fiona LeClair-Robertson, and proofread by all members.

4. Related Work

Four different articles were compared and have been summarized in the following comparison table including their purpose, dataset, models implemented, and finally results achieved.

	Title	Dataset	Models Implemented	Conclusions / Results
	***Real-Time Detection of Distracted Driving using Dual Cameras*** *https://ieeexplore.ieee.org/document/9340921*	The researchers created a custom dataset, consisting of **7,000 face images**, and **7,000 body images** collected from **132 videos of** subjects carrying out various **distracted activities** while driving in a created car simulator. 2 different angles are captured per frame, using 2 different cameras.	This team uses a **VGG-16** model and a **MobileNet-v2** model to implement real-time object detection. They trained the models using both a single camera angle, and both cameras angles.	Their study concluded that the model and training set with the highest accuracy was the **VGG-16 with dual camera** angle - resulting in an **accuracy of 96.5%**. Using only a single angle with the VGG-16 yielded **88.9%**, while the MobileNet-v2 combined with dual angle fell behind at **83.5%.**
	**Distracted driving: A Novel Approach towards Accident Prevention** *https://www.ripublication.com/acst17/acstv10n8_42.pdf*	They created their own custom dataset containing **2000 positive images** of eyes, and **1000 negative images** where no eyes are visible. These images are then **converted to greyscale** as they found better results using this coloration.	This group implements object detection using **Haar-Based cascade classifiers** available through OpenCV. For weight training, their model follows an **AdaBoost algorithm** - based off their positive / negative training images.	Given that their model was based on a timing criteria between blinks, their model primarily was tasked with detecting a blink no matter the positioning of the driver's head. It was able to detect blinks with **80% accuracy.** This approach however does not account for many other forms of distracted driving, leaving the problem at large.
	**Real-Time detection of distracted driving based on deep learning** *https://ietresearch.onlinelibrary.wiley.com/doi/full/10.1049/iet-its.2018.5172*	This group both used a **combination of the state farm dataset, as well as their own** captured by their car simulator environment. Both simply depict the driver falling into one of 10 categories, containing **over 10,000 images** in the train/test set. For validation purposes their custom car simulator data was used.	Four different CNN models are used and compared for the purpose of this study; **VGG-16, AlexNet**, **GoogleNet**, and **ResNet** models respectively. Transfer learning is used for training, as they were using pre-trained models, with the last FC layers replaced by their own MLP classifier.	The researchers determine the model with the best speed / accuracy was the **GoogleNet model with 89% accuracy** and a **speed of 11 Hz**. While the **VGG-16 model performed the fastest (14 Hz)** its accuracy fell behind at only **86%**. Conversely, the Res Net model achieved the **highest accuracy of 92%**, though was the slowest with a speed of **8Hz**.
	**Identification of Driver Phone Usage Violations via Object Detection** *https://ieeexplore.ieee.org/abstract/document/9671378*	The group built their own custom dataset for this research project using road-side camera captured footage. They pre-trained twelve different models with their custom dataset	YOLO, SSD, Center-Net and Faster RCNN were used to compare real-time object detection capabilities for the sake of this research.	The researchers determine the model with the best speed / accuracy was the YOLO model with an accuracy of 96% going at 30 FPS.

Figure 1: Literature Comparison and Review

As previously noted, the Real-Time Detection of Distracted Driving using Dual Cameras models achieve very high accuracy results, though are dependent on a very large dataset including a custom dataset created with two different 4k cameras. Their results are noticeably

lower when only using a single camera, which we intend to train our model on while yielding a higher accuracy. The fourth piece of literature, Identification of Driver Phone Usage Violations via Object Detection demonstrates a very high accuracy of 96%, though lacks the diversity in distracted behaviors that our models intend to accommodate. This same issue is present in the second paper Distracted Driving: A Novel Approach Towards Accident Prevention, where only drowsiness is addressed. Real-Time Detection of Distracted Driving Based on Deep Learning will be closely compared with our research as they achieve a fairly high accuracy, using our selected dataset alongside their own custom dataset.

5. Dataset Used

Our group chose to work with the StateFarm Distracted Driver Detection dataset for each of the models implemented to ensure we're maintaining training/testing consistency when comparing them to one another. We found this dataset best fit the needs of our research as it contains an extensive image library depicting a variety of drivers exhibiting inattentive behaviors, which helps us achieve our goal of diversifying the forms of distracted driving able to be detected by our modes. **Ten different categories** are outlined for clustering as follows; Safe Driving, Texting (right hand), Talking on Phone (right hand), Texting (left hand), Talking on Phone (left hand), Operating the Radio, Drinking, Reaching behind, Hair & Makeup, and Talking to Passengers.

There are **22,400 training samples total**, evenly distributed among each of these 10 classes (2,240 each). Each of the samples are .jpeg images captured by a single 2D dashboard camera, of a driver either driving safely or participating in one of the 9 outlined distracted behaviors.

Figure 2: Sample Images from Statefarm Dataset

This further fits the needs of our project, as we emphasize an importance on high performance over lower quality images for more realistic product accessibility. The dataset includes **79,700 unlabelled samples for testing** with different drivers behind the wheel.

6. Individual Parts

6.1 MobileNet Analysis - Fiona LeClair-Robertson

For my implementation of the task, I've chosen to analyze and extend on a MobileNet network. MobileNets are a form of convolutional neural network that are highly compact, and far less computationally expensive than other models recognized in this study, with a rather minimal tradeoff for accuracy. MobileNet uses depth wise separable convolutions to achieve this, created with a combination of depthwise convolutions and pointwise convolutions. This technique largely reduces the number of parameters considered, and ultimately the size of our network.

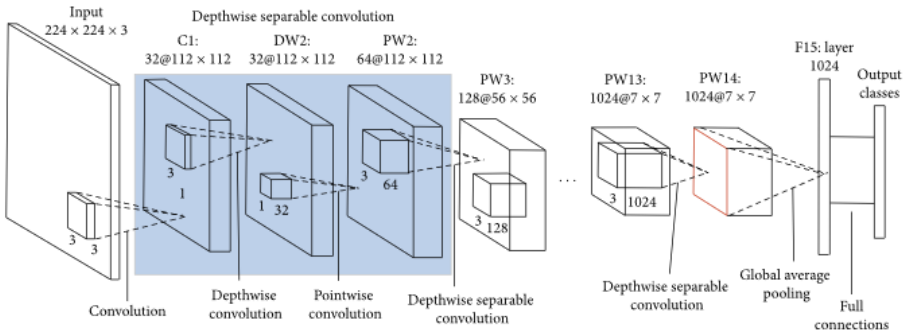

Figure 3: MobileNet Architecture

6.1.1 Data Preprocessing

To improve accuracy and reliability of the network, multiple preprocessing steps were applied to the image dataset. The data was first scaled down from 640*480 to 224*224 to reduce the training time required for each image. Following this, the images were shuffled to randomize the order of appearance for each driver, and respective depicted distracted behavior. To prevent data leakage from occurring, the images were split based on driver ID rather than simply splitting it randomly so the network would encounter different drivers for training vs. testing. Once these separate training sets were established, each image was converted to a feature map by filtering the image 3 times. Global average pooling is then applied to each image, to reduce

dimensionality by simply taking the average of each specified area of the feature map as demonstrated in the figure below.

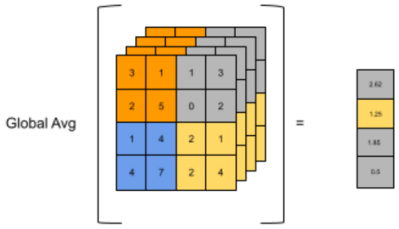

Figure 4: Global Average Pooling

Finally, in order to create a larger dataset image augmentation was used on the existing training images. This was done to prevent overfitting given the number of parameters in the neural network, and implemented by shifting the height, width, and zoom by 0.5, along with a 30° rotation. This enables us to double the size of our training data as each image creates a new child image as shown below.

Figure 5: Before vs After Image Augmentation (Parent/Child image)

To summarize this process, a full data flow diagram showing each of the transformations applied to the dataset is included in the figure below.

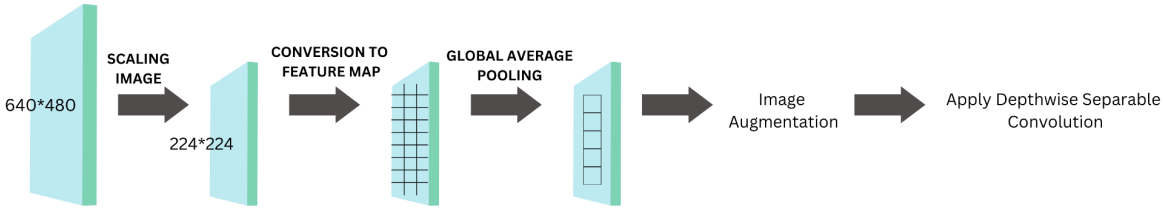

Figure 6: Data Flow Diagram Overview

6.1.2 Experimental Setup

To run this experiment, the MobileNet50 base model was created using Tensorflow and Keras in Jupyter Notebook. The model extensions were trained & tested using an AMD Ryzen 5 CPU on a device with 11.4 usable GB of RAM. Due to the low amount of available memory,

there were hardware limitations preventing further layers (beyond the two I already added) or a larger dataset from being used.

6.1.3 Training, Testing, & Validation

The training parameters are where most of my extensions to the code were applied. Given that we chose to use accuracy as our evaluation metric, I had to make quite a few adjustments as they had initially chosen logarithmic loss as theirs. Whereas the original code included only a single hidden layer, I added two more to the network so that it could further identify patterns and improve results. In the second hidden layer, I normalized the data values to reduce major inconsistencies and increase efficiency of the network. Additionally, I added dropout parameters to avoid overfitting. Both the second and third added layer used a ReLU activation function, as it demonstrates optimal results for hidden layers of convolutional neural networks.

Although I tried a couple different learning rates, 0.005 yielded the best results with the most efficient convergence. This way the learning was fairly gradual, without risking non-convergence in the network. Stochastic gradient descent was chosen optimization by the original authors, and I left their choice as using Adam resulted in a high volatility descent pattern. Finally, I increased both the number of epochs the network would run from 20 to 30, and the early stopping patience from 10 to 20. Using a lower number, the original authors were able to better preserve logarithmic loss though at the expense of further accuracy. I chose to let the model run slightly longer, in hopes of allowing the accuracy to improve. Thus the stopping criteria was set to either reach 30 epochs of 292 batches with the batch size set to 64, or fail to improve logarithmic loss over 20 iterations.

Using the above parameters I was able to slightly increase the training accuracy from 90.5% to 91.55% though noticed a slight decrease in testing accuracy from 84.5% to 82.64%. When further investigating this drop, I noticed that my model was likely overfitting and should have had stricter stopping criteria as it performed better on the 23rd epoch over the 25th - where it ultimately terminated. Results obtained during training vs testing in both the original code, and my edited version based on accuracy as a validation criteria are depicted in the figure below.

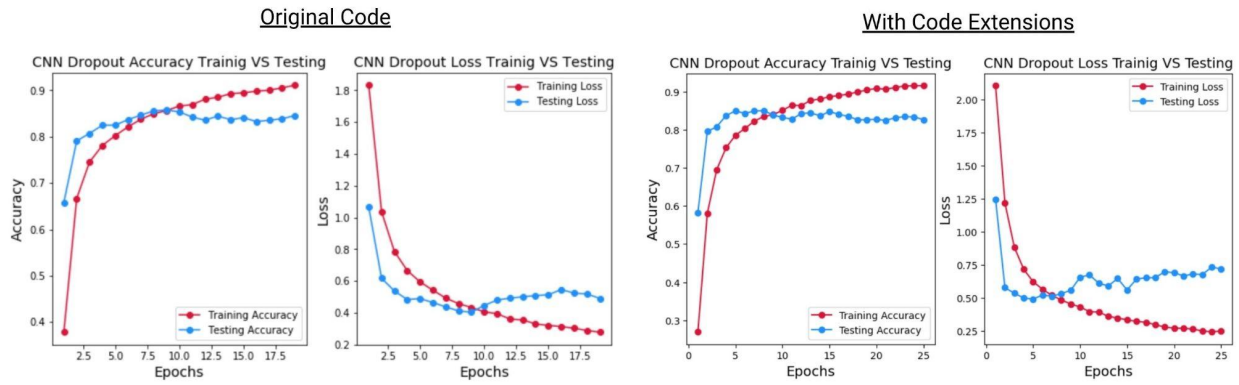

Figure 7: Testing and Training Result Comparison in Original and Extended Code

To avoid this issue in future work, I would reduce the number of training epochs, and increase the dropout rate. Additionally, should the hardware be permitting, I would use a larger dataset depicting similar actions or apply further image augmentation to synthetically produce this effect.

6.2 Individual Part 2 - Priyank Thakkar

RESNET 50 is a Convolutional Neural Network that was published in a 2015 paper called “Deep Residual Network for Image Recognition” which was published by He Kaiming, Zhang Xiangyu, Ren Shaoqing, and Sun Jian. Convolutional Neural Networks are a type of Neural Network which are used while handling image recognition or any sort of visual data. RESNET stands for Residual Networks and is made of individual residual blocks which are then connected in a network to form the Neural Network . RESNET 50 has 50 layers in the neural network out of which 48 layers are convolutional neural layers and 2 are pooling layers; out of the 2 pooling layers , 1 layer is an average pooling layer and the other layer is a maxpool layer.

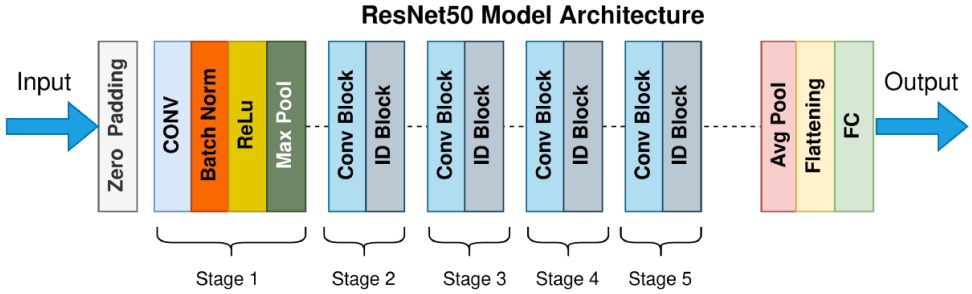

Figure 8: ResNet50 Model Architecture

Pooling in RESNET 50

Pooling is a method which is used in Convolutional Neural Networks to downscale the feature map while keeping the important features in the feature map . Pooling greatly reduces the training time of the model as it reduces the dimensionality of the feature map while not affecting the accuracy of the model. RESNET 50 uses two types of pooling layers ; 1) Average pool layer and 2) Max Pool layer.

Average & Max Pool Layer

Average Pool Layer takes the average of the specified area of the feature map and uses that value to resemble that area of the feature map. Max Pool Layer takes the maximum value of the specified area of the feature map and uses that value to represent that area of the feature map.

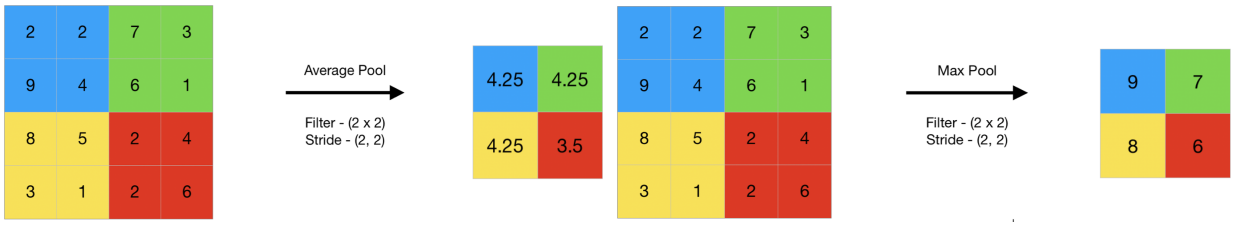

Figure 9: Max Pooling vs Average Pooling

Data Pre-Processing

The Data is scaled down from 640*480 pixels to 224*224 pixels using the Average Pool Layer and the Max Pool Layer to reduce the training time and to focus on only the area of the image which is useful while training the model. Image augmentation was also used to counter the problem of having less data which would result in overfitting; more images were created by rotating the current images horizontally and vertically which would increase the samples that RESNET 50 could train on and hence prevent overfitting. Image augmentation also helped us to differentiate between actions which looked similar as we had more data of a person doing the same action and hence our accuracy went up and our false positives went down.

Data Split And Parameters

The Data was Split the ratio of the 80-20 as 80 being the training data and 20 being the testing data. The Parameters used were learning rate = 0.001, Beta = 0.9, Beta2= 0.999, Decay = 0, Epochs = 40 and a batch size of 64.

Layers Added

We added more dense layers to the original Resnet 50 to compare if the accuracy increases while preventing the model from overfitting the data. We connected 4 extra Dense Layers to the model with the activation function RELU. We found our accuracy increasing from 87 percent to 89 percent. More work can be done in adding extra filters to increase more accuracy which we didn't touch upon. The comparisons between the two models are shown in the graph below.

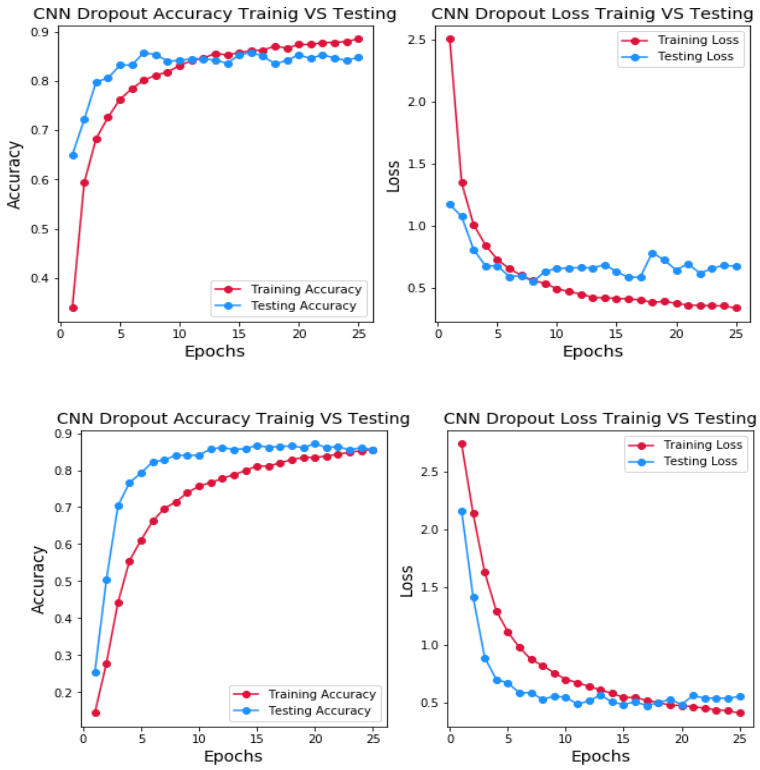

Figure 10: Testing & Training Results in Original and Extended Code

Conclusion

RESNET 50 is known to produce high accuracy results while also having less training time. We found that more accuracy can be achieved by adding more layers to RESNET 50 while preserving low validation loss. More work can be done by upscaling the image and considering a larger data set. Dataset of lower samples was used because of technological constraints.

6.3 Individual Part 3 - Sarah Litwiniuk

VGG-16 Model

In VGG-16 there are actually 21 layers, 13 convolutional layers, 5 max pooling layers and three dense layers. The reason it is called VGG16 is because there are only 16 weight layers that relate to learnable parameters. In order to deepen the number of network layers and to avoid too many parameters, a small 3x3 convolution kernel is used in all layers. The preprocessing done for the model is to normalize the RGB values for every pixel, achieved by subtracting the mean value from every pixel.

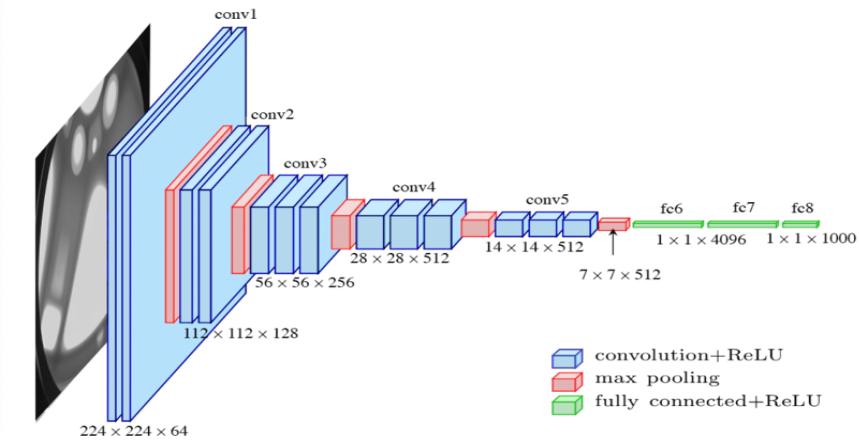

Figure 11: VGG-16 Architecture

Above, you will find an image representing the VGG-16 model. The model begins with taking the input image and processing it through 2 convolutional layers, the information from these convolutional layers is then processed by the max pooling layer. The pooling layer reduces the dimensions of the feature map by halving the size of the activations; which is why in the upper image we see the sizes of the layers getting smaller as we move through the model. The first stack of 2 convolution layers contains 64 filters, the second has 128 filters, and the next is 256. As the max pooling layers decrease the size of the feature map, the number of filters being used increases. This process of convolutional layers being processed and reduced by a max pooling layer continues until we reach the 3 dense layers, denoted in the image by the three green layers, which connect the previously passed data into an output for the system. The first two dense layers have 4,096 neurons each, and the last fully connected layer serves as the output layer and has 1,000 neurons corresponding to the 1,000 possible classes for the ImageNet dataset.

The reason the model has two 3x3 or three 3x3 convolutions is because they are the equivalent to a 5x5 receptive field and 7x7 receptive field, respectively. There are a few advantages to doing it this way, one being that including three ReLu layers instead of one, makes the decision function more discriminative; and another being that it allows us to reduce parameters

VGG-16 Code

The model was built using tools from Keras. The pre-trained VGG-16 model (trained on ImageNet) was used as a basis on which to train our dataset. When it came to modifying the base model code, I could not simply add more parameters/layers. This is because for VGG-16 the more layers you add the heavier the model becomes and the longer the inference time is. Adding more layers does not mean you will get better results or model performance. I chose to add batch normalization to the model, which allows the network layers to learn a bit more independently. Batch normalization normalizes the output of a previous activation layer by subtracting the batch mean and dividing by the batch standard deviation. Batch normalization makes sure that there's no activation that's gone really high or really low, allowing us to use a large learning rate and train things previously untrained. Although due to technical limitations I wasn't able to run this new code, I theorize that adding a batch normalization layer, and increasing the learning rate will improve the accuracy of the model.

VGG-Testing and Results

To train the model, the training image folder from the image database was used, and shuffled to put them in a random order. 4 "participants" (those whose images were taken and used for the database) were chosen arbitrarily to be used for the test data, and the rest of the participants (22) were used for the training set. Training values for the model are as follows;

- Epochs: 25
- Steps per epoch: size of training set / 64
- Model Checkpoint: Kept track of only the best model weights and saved them
- Early Stopper: The model looked at the val loss, checking at the end of every epoch whether the loss was no longer decreasing. If it found that to be the case the training would be stopped

The parameters used to analyze the results of the system were validation accuracy and validation loss. The trends of the accuracy and dropout loss had a negative correlated relationship, meaning that as the accuracy increased, the dropout loss decreased. The training data for both the accuracy and loss was a fairly consistent and smooth progression, whereas the testing accuracy experienced more localized jumps. You can see the trends of the accuracy and loss for testing and training data in the figures below. By epoch 20 the changes stopped being significant enough to track and the stopping condition halted the code.

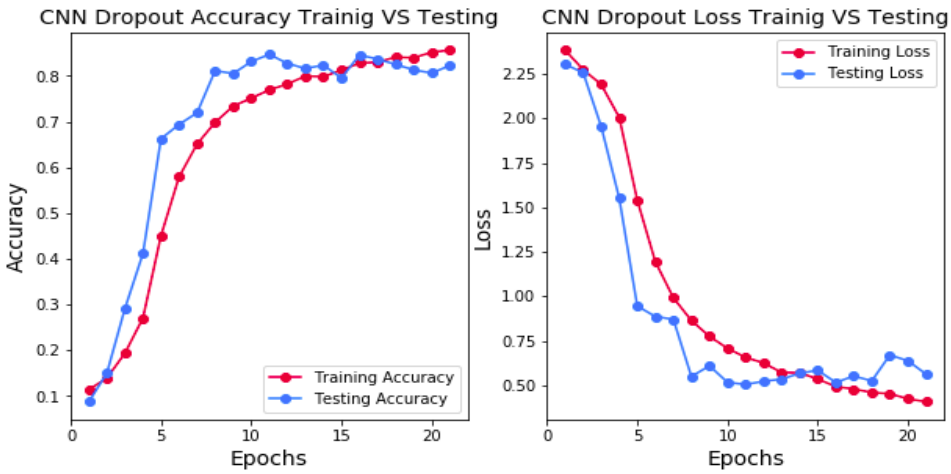

Figure 12: Accuracy and Loss Training vs. Testing Results

6.4 Individual Part 4 - Farwa Abedi

XCEPTION Model

The Xception model is an extension of the inception model and contains 71 layers. The images are converted from size 640x480 to 224x224 so that the dimensions match the dimensions of the pre-trained convnets. The model augments the images by using the ImageDataGenerator which shifts the height, width, zoom range and rotation of the images. This aims to transform the data images so the model can train different variations of each image.

As seen in the diagram, the data goes through three stages: the entry flow, the middle flow, and the exit flow. After the data goes through the entry flow, it goes through the middle flow eight times before going through the exit flow. As opposed to applying regular convolutions to the data, as seen in other models, the Xception model applies depthwise separable convolutions to the data. Depthwise separable convolutions are more efficient in terms of computation time compared to regular convolutions. A depthwise separable convolution involves

a depthwise convolution followed by a pointwise convolution. Depthwise convolution applies a single convolution filter per input channel instead on all channels and therefore, reduces computational power. A pointwise convolution uses a 1x1 kernel with a depth of the number of input image channels and it which iterates through every single point. These layers in the Xception model are then followed by a layer of batch normalization. This layer regulates the model and make it more stable during training. It works by normalizing the layers by recentering and rescaling the data from the previous layers, decreasing the number of epochs needed to train the model. The data in the Xception model is preprocessed as the data images are reshaped and converted into a np array. The Xception model initializes its weights using the imagenet data set weights and uses a transfer learning approach to adapt the weights during training. This allows for a much faster convergence rate. The model uses a ReLU activation function and is applied after the convolution.

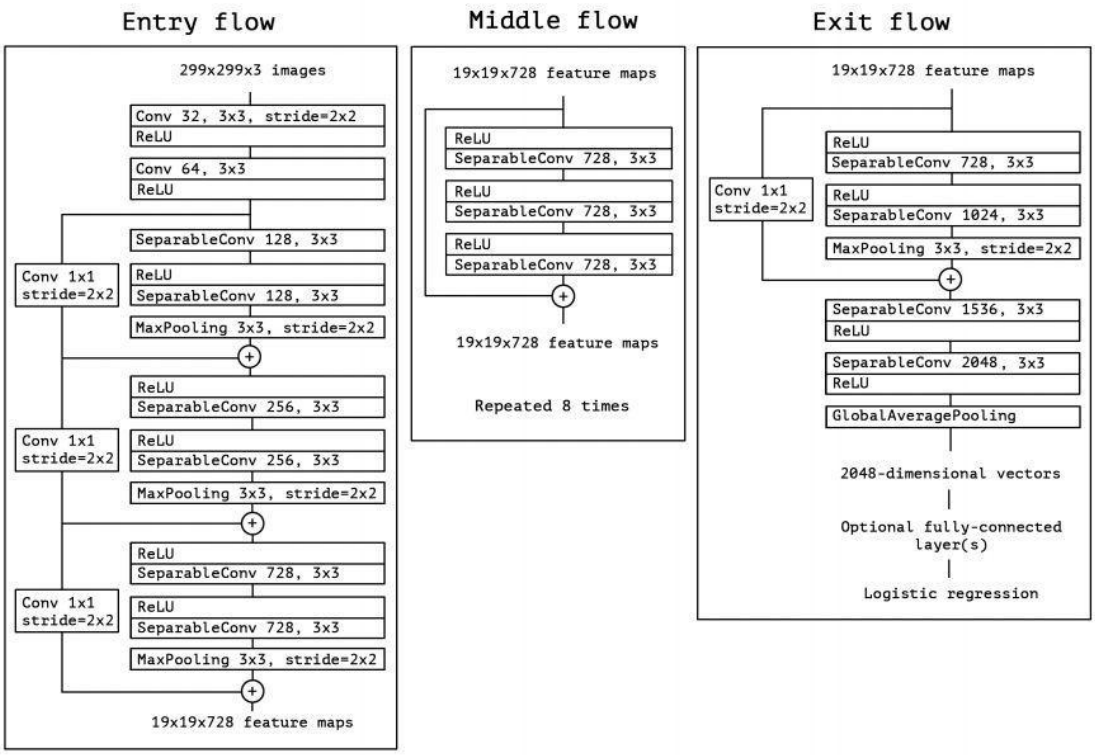

Figure 13: Data Flow Diagram

The model applies an average pooling 2D layer. The default matrix size is 2x2. This layer selects the average value of each 2x2 matrix and records it in a new smaller matrix. This method obtains the average of the features in the region.

The model is trained by applying a couple of layers. One layer the Xception model applies to the data is a dense layer. This layer receives input from every neuron from the previous layer and combines the results by performing matrix-vector multiplication of row and column vectors.

After the dense layer, the Xception model applies a dropout layer whose purpose is to eliminate random output layers in order to prevent overfitting of the model. The model then applies two more dense layers, one with a ReLU activation function and the other with a softmax activation function.

The model used an Adam optimizer function. There were 24 million total parameters and slightly less trainable parameters. There were 56K non trainable parameters. There were 30 epochs. Each epoch contains a set of 292 images. The accuracy of the model is 0.86. The images from the state farm data set were divided into x train, y train, x test and y test. There are 18732 images in x train and y train and 3692 in x test and y test. The train and test data are divided based on the drivers, meaning a driver can only show in either the train or test set. The validation criteria used is accuracy. The batch size is 64. The validation set contains the x and y images being tested. The stopping criteria used for the model is a callback called early-stopper. This monitors the value of the validation loss and aims to stop training the model once the value stops decreasing. In other words, when the error stops decreasing, the model can stop training. Another parameter the callback takes is patience which is set to 10. This means that after the model reaches a minimum validation loss value, the model trains for another 10 epochs before stopping. This is done because sometimes a model's validation loss increases before decreasing again. The modelcheckpoint callback ensures that the model with the best performance is saved when it stops training.

6. Results & Discussion

Upon completing each experiment, our models yielded quite successful results against the existing state-of-the-art models as depicted in *Figure 14*. In the table below, we have compared our models (far right column) to the models discussed in our literature reviews - specifically the first and third paper in *Figure 1*, as their models were also aimed at identifying body movements linked to visual distractions.

Model	Real-Time Detection of Distracted Driving using Dual Cameras	Real-Time Detection of Distracted Driving Based on Deep Learning	Putting the Brake on Distracted Driving Using Deep Learning & Computer Vision Techniques **(Our work/models)**
ResNet50	N/A	92%	89%
VGG-16	96.5%	86%	85.64%
MobileNet	83.5%	N/A	91.55%
Xception	N/A	N/A	80.38%

Figure 14: Comparison Result Table to State-of-the-Art Models

An important factor we must keep in mind when analyzing these results, is that our models were able to achieve these accuracies with a much smaller training dataset and far less computational power than the state-of-the-art models. With this in mind, the MobileNet performed exceptionally well as it surpassed the accuracy of its Real-Time Detection of Distracted Driving Using Dual Cameras competitor, despite using only a single angle of a dashcam image. Similarly, the ResNet50 may yield a slightly lower accuracy though than its competitor Real-Time-Detection of Distracted Driving Based on Deep Learning, though this can be justified by the fact that their model was trained on a whole additional custom dataset on top of the exact same dataset we used. We can expect its performance to surpass the other model should it be trained on a similarly sized dataset with the same computational power, as the model will be exposed to more features in our data. The VGG-16 model and Xception models selected for our paper demonstrate strong implementations, as they also produce similar results to the state-of-the-art models using far smaller networks. This reduces training time and computational power exerted to accomplish these results, while maintaining a high accuracy.

7. Conclusions & Future Work

In this paper we compared ResNet50, VGG-16, MobileNet, and Xception models against one another to determine which is best suited for object detection with the goal to determine whether a driver is distracted behind the wheel or not. In analyzing our results, we determined that the MobileNet best performed this task, as it both used the least amount of parameters while still yielding the highest accuracy. We determined that using images captured by a simple

dashboard camera can still attain reliable accuracies from our convolutional neural networks, as proven by the comparison table. This is an important takeaway, as it highlights the potential deep neural networks have towards realistically intervening with distracted driving, given that one in ten cars in Canada are registered with a dashboard camera[7].

Due to hardware limitations, many of our models were limited to a maximum of 3 hidden layers. This leaves the opportunity for future work to explore even more optimal approaches to these models, using a deeper network. Additionally, the ability to detect drivers who are distracted in real time rather than having to use snapshots could be implemented by further investigating YOLO, R-CNN, or even MobileNet SSD models. This could save valuable time, where a split second on the road can be the determining factor of a life-or-death collision.

8. References

1. Battista, M. (2019, April 11). *Distracted driving facts and stats you should know*. isure. Retrieved November 28, 2022, from https://isure.ca/inews/distracted-driving-facts-and-stats/#:~:text=21%25%20of%20all%20fatal%20car,sometimes%20use%20smartphones%20in%20traffic.
2. *Dangers of distracted driving*. Travelers Canada. (2021). Retrieved December 1, 2022, from https://www.travelerscanada.ca/prepare-prevent/car/dangers-of-distracted-driving#:~:text=Imagine%20driving%20100km%2Fh%2C%20that%27s,rink%20with%20your%20eyes%20closed.&text=Your%20chances%20of%20getting%20into,road%20for%20merely%20two%20seconds
3. Desoutter, A. (2022, March 25). *Distracted driving costs more than $102 million every year in Canada (2022)*. HelloSafe. Retrieved November 26, 2022, from https://hellosafe.ca/en/car-insurance/distracted-driving-cost#:~:text=Obviously%2C%20the%20road%20injuries%20and,insurance%20in%20the%20near%20future
4. Pachigolla, S. N. (2019, December 15). *Distracted driver detection using Deep Learning*. Medium. Retrieved November 29, 2022, from https://towardsdatascience.com/distracted-driver-detection-using-deep-learning-e893715e02a4
5. *Public education campaigns*. OACP. (2021). Retrieved November 27, 2022, from https://www.oacp.ca/en/public-safety-and-awareness/public-education-campaigns.aspx
6. *State Farm distracted driver detection*. Kaggle. (2016). Retrieved November 23, 2022, from https://www.kaggle.com/c/state-farm-distracted-driver-detection
7. Writer, S. (2022, June 28). *Dash Cam Statistics & Trends in Canada*. The Seeker Newsmagazine Cornwall. Retrieved December 4, 2022, from https://theseeker.ca/2022/06/dash-cam-statistics-trends-in-canada/
8. Zulkernine, F. (2022). *Cisc 452 Neural and Genetic Computing*. *Course Content*. Online.