Eyes on the Driver:
A Machine Learning Approach to Detecting Distracted Driving

I. Abstract

Distracted driving is dangerous and difficult to monitor. Timely detection of those behaviors is critical to mitigating injury and fatalities. But that is always a challenge. This project creates an AI model that allows distracted driving detection from an image of the driver alone, which provides a solution to let the car alert the driver of their dangerous behavior. The model is a lightweight Convolutional Neural Network that has been trained with over 13,000 images in 10 different categories. The model has classification accuracies of over 90% and has been fine-tuned to adapt to real world conditions to be as effective as possible in actual cars.

II. Table of Contents

I. Abstract 1
III. Introduction 1
IV. Purpose and Hypothesis 2
V. Background 3
VI. Procedures 3
VII. Experiment Summary 4
VIII. Data summary 7
IX. Conclusions 10
X. Application Discussion 10
XI. References 11
XII. Acknowledgements 13

III. Introduction

More than 20% of fatal collisions in Canada are caused by distracted driving, according to the Canadian Association of Chiefs of Police [1]. The CAA also reported 80% of surveyed drivers admit to being distracted while driving [2]. The growing numbers of distracted drivers are becoming a heavy workload for human police to handle or detect on their own. That is why AI should step in.

This project aims to detect when a driver is distracted by analyzing an image taken within the vehicle. A distracted driver could be doing one of many things - such as using the radio, drinking coffee, or talking to someone in the back seat. This makes it difficult to identify for certain whether a driver is distracted or not.

It was, therefore, necessary to find the right dataset of distracted driver images. These images would be used to identify common trends between different distracted drivers and flag them.

This project is also timely in that self-driving technology, especially the popular level 2 systems, may allow drivers to become more complacent and therefore easily distracted while they are on the road [3] [4]. However, this extra technology also allows the same computer systems in the vehicle to monitor the driver for distracted behavior. For example, webcams on the dashboard of the car may be installed to watch the driver.

In order to process image data in a machine learning environment, this project explores Convolutional Neural Networks (CNNs) for their ability to study what is going on within a 2D image. I designed and implemented experiments around this type of model in Python. Python has a rich set of libraries for machine learning (including CNNs) and is efficient in data processing, making it an ideal language for this type of project. In the end, I came up with a lightweight yet highly accurate network architecture, being useful for detecting distracted driving in the computers of a car rather than a dedicated Google server.

IV. Purpose and Hypothesis

Given both the timely and relevant need for AI detection of driver distraction, the research hypothesis was as follows: “A machine learning model is able to detect when a driver is distracted by analysing an image taken inside the vehicle.”

Using my existing knowledge of machine learning and which models were most likely to succeed with this task, my active research question that I was trying to answer was “What does an optimal CNN for detecting distraction look like?” Given the large image datasets that this project used, there was the assumption that if the task was not possible using an image specific model, it was very unlikely that it would be possible to use another model.

This in turn allows us to assess the feasibility of implementing this technology inside vehicles to reduce the risk of distracted driving on the road, and thus lower the number of accidents related to it. Therefore, the research question was appropriate for testing my hypothesis, and both the question and the hypothesis are justified by the important purpose of keeping roads safe.

V. Background

As recently as last year, there was a study researching the impact of self-driving on distractedness [3] [4] [5]. This study showed a worrying trend; that semi-autonomous driving could increase distracted driving and potentially more accidents. Reducing accidents in self-driving cars is clearly an important goal for that reason, and one that will become more relevant over time.

Other studies have already investigated the use of pre-made CNNs, such as VGG-16 and Alexnet, to detect distracted driving. [6] However, the accuracy of those models comes at the cost of a very large size, making them harder to run on a device without a dedicated GPU or other processor specifically for the model.

VI. Procedures

In order to establish a robust methodology for conducting systematic experiments, it was first needed to find a relevant dataset and construct an initial model to experiment on. As previously mentioned in the research question, a CNN would be the most appropriate model structure for me to develop my procedures around.

To construct this CNN, prior data processing was essential. The data that I chose contained 10 different categories of driver. Here is a sample image from the dataset. (Figure 1) One category was "safe driving" and the other nine were different forms of distracted driving. This dataset was generated and provided by the insurance company State Farm, as part of their research into driver safety [7]. The dataset contained over 13,000 images of resolution 360 x 240 pixels. Importantly, the images also contained a diverse set of drivers

Figure 1: a sample image from the State Farm dataset

regarding gender, ethnicity and age, meaning we can be confident that the model is, or at least very close to, unbiased and will work for the whole population.

During this initial processing, the dataset was divided into two sets. One set of images would be used to train the model, meaning the CNN was able to learn and adapt based on the images and the outputs from the CNN. The other set would be kept hidden from the model, so that the model could never learn from its contents. This allowed for a bank of 'never before seen' images that we could use to validate how the model might perform on brand new data. 80% of the dataset was assigned to the 'training' set, and the other 20% of images was assigned to the 'validation' set for these tests.

Due to the model's numerical inputs, each image must be processed into its RGB values. This means that each image, rather than being processed as a single file, is actually read into the machine as a list of 230,400 numbers (320x240 = 76,800 pixels, each of which needs an R, G and B value). Doing this for all our 13,000 images was very computer memory intensive, so I had to design a special generator class that can read these numbers in batches from the computer harddrive. In summary, the generators first read out the images and added any image transformations needed, then cached the transformed images on disk to be quickly read back later, rather than transforming the images every time the network needed to see them.

The State Farm dataset represents nearly perfect images, but a model trained purely on that might struggle with more difficult conditions, such as if the camera is lower resolution or if the person is simply not in the same place as the sample images. To build in resilience to the model, I transform the dataset using rotations, panning, and zoom to simulate different driver positions, and adding noise to the images to simulate real-life distortion and to prevent overreliance on specific pixels.

The CNN initial model was made using Tensorflow Keras, which is a library in Python published by TensorFlow [8]. This model allowed me to build out an initial attempt, which I would then refine in a series of experiments and reverse engineering of the model.

VII. Experiment Summary

The set of experiments occurred in two phases. The first was to get a general understanding of how different machine learning approaches responded to the image data. This would help in

validating my initial assumption that CNN's were the optimal machine learning technique for this project. The second phase was to optimize and configure the CNN model to better fit in the real world for detecting distraction, both in regard to accuracy and stability. I experimented with these different configurations to arrive at my final CNN model.

As part of my phase one model investigation, I tried a linear regression model on a reduced dataset of 64x64 images and 4 categories. A linear regression model isn't supposed to be used on images or classification problems in the first place. The linear model did better than expected, with an accuracy of 82.3%. This was surprising, as essentially all the model was doing was drawing lines of best fit corresponding to the colours in the image. Based on these lines alone, the model was able to sort the image into one of ten categories correctly over 80% of the time. However, despite being surprisingly high, the accuracy was significantly lower than any other model, even with the reduced dataset. When looking at the model's coefficients, there appeared to be little pattern in those, suggesting that the model might have been focusing in on a few very specific pixels. I did not add any transformations to the reduced dataset, but I suspect that they would have affected the linear model even more than any other type of model.

Given the unusual success of the linear model, I decided to investigate a special kind of neural network that essentially builds a linear model for each of the different output categories, referred to as a Perceptron. I used this Perceptron model on the full-scale dataset, with a higher resolution of 128x128. It had nearly perfect accuracy, though I suspect that again transformations would have affected this more than the CNN, given how it focused on specific regions of the image which would have been shifted around and become inaccurate. Something I noticed when increasing the resolution was how the RAM usage went up drastically, to the point that I likely wouldn't be able to store the entire dataset in RAM. Every time I wanted to run the neural network, I had to convert the RGB 128x128 image into over 49,000 numbers which I then inputted into the model.

This allowed me to visualize what the model's relationship was to the input for each of the unique 10 output categories. If an area in the image for a specific category is bright, then brightness

Figure 2: Image representations of the Perceptron's weights, with one image per category.

in that area will make the model more likely to output that category. Color in these images represent colors in the real input images. These are shown below (Figure 2):

While quite messy, these outputs do seem to show areas of interest, such as the coffee cup in the 7th image being used to identify that form of distraction, or the extra relevance of the bottom left region in the 8th image that corresponds to reaching to the back seat.

While the accuracy was extremely high, 99.3%, due to the 1:1 relationship between the input and the output, the model would be very bad at handling noise. For instance, if the coffee cup moved to a different part of the image, the model would ignore it, because it was not exactly where it was when it was trained. Additionally, it was very time and computationally expensive to train and test this model because the input to the model was so large. I therefore concluded that I needed to use a different model type, which led me to the CNN.

To understand what a CNN is, a few terms need to be introduced. A Convolutional Neural Network generally has a convolution step and a neural step. Kernels and neurons are two types of functions that help in those steps. In the convolution step, a kernel takes a small part of the image, multiplies it with a special, trainable convolutional kernel (number by number), and adds all the products. This essentially compresses multiple pixels into a single value. Example here (Figure 3):

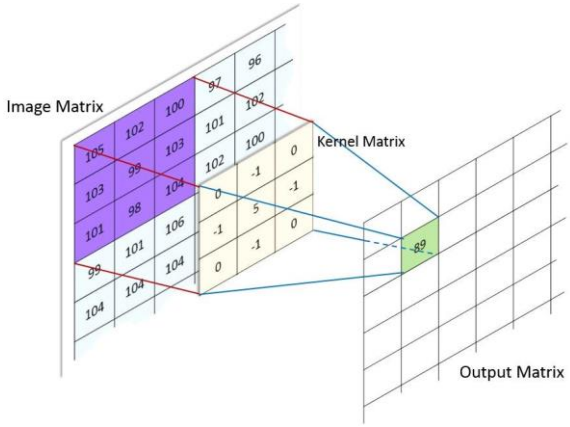

Figure 3: A simple visual representation of how a convolution works

A neuron in the neural step takes a series of values (generally from other neurons), multiplies by some trainable weights, and adds the products together. This is quite similar to a convolutional kernel, but often looks at all the preceding neurons, while a kernel only looks at a small patch of a grid (or matrix, in computer science terms). A CNN functions by having multiple layers of convolutional kernels and neurons, with each layer having multiple kernels or neurons to learn different aspects of the input. Eventually, with enough adjusting by a loss function (how wrong the answers are) and an optimizer (how to be less wrong), this collection of multipliers and adders will "learn" (be less wrong) to "activate" (set high output value) certain kernels and certain neurons for some category, and then it will be useful in the task it is put to.

Other than the better-than-expected performance of the linear regression, it seemed apparent that the CNN was going to be in the sweet spot between complexity and time efficiency, as well as having the ability to deal with nosier and shifted images, making it more relevant for real-world use. During this second phase, I made use of visualisation techniques that would allow for 'seeing' inside the model and get a sense for how it was learning, much like with the Perceptron. This allowed more accurate and more effective modifications to the model. For instance, if an image representation of the model's network showed that many of the neurons weren't being used, a good modification would be to reduce the number of neurons so that it would train and run more efficiently. If it looked like the model could benefit from more neurons, I was able to add them and see how it reacted.

The reason that CNNs are able to use fewer neural network inputs than the Perceptron is because part of the model is trained to 'pull out' what is important from the image, using a convolutional layer, hence the model's name. This allows the model to identify areas of interest regardless of where they are in the image and is much better at finding patterns in the image, which it identifies and uses in a more traditional neural network layer. So, over the course of the experimentation, I tried different kernel configurations alongside different neuron configurations to see which combination worked best. Increasing the number of kernels was able to offload some of the pattern recognition from the neural network, which made the model much faster. Large kernels were both able to pick up complex patterns and also deal with noise added to the dataset to simulate real world conditions.

VIII. Data summary

After training without any processing (noise, rotations, or image shifts), the accuracy of the model ends up around 99%. (over 4 trials) Here is the confusion matrix for one of the trials. A confusion matrix compares the predicted results with the actual results, showing the predicted results on the X axis with the actual results on the Y axis.

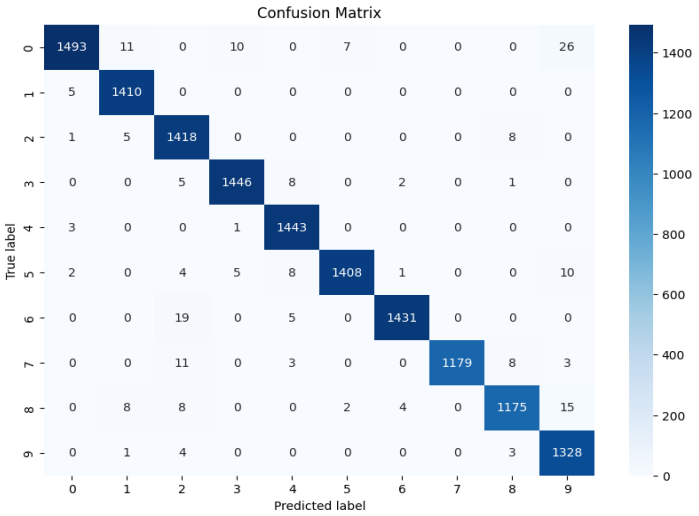

Figure 4: Sample Confusion matrix of model (no transformations)

A quick glance at the confusion matrix (Figure 4) will show that the model is very accurate, with the error rates between categories never being greater than 30 in a testing set of 14,000 images.

An interesting aspect of the confusion matrix is how many errors category 0 (Driving Safely) has. Category 0 has the greatest number of false positives of any category, which may lead to annoying false negatives that lower the user's confidence in the system. This might be mitigated when the driver is actually distracted, since they will quickly realize that they are in fact distracted.

Accuracy of final model with noising and transformation ends up around 85%, with around a percent of variation. Notably, the train accuracy ends up significantly higher than the validation accuracy, suggesting overfitting. (Confusion matrix: Figure 5)

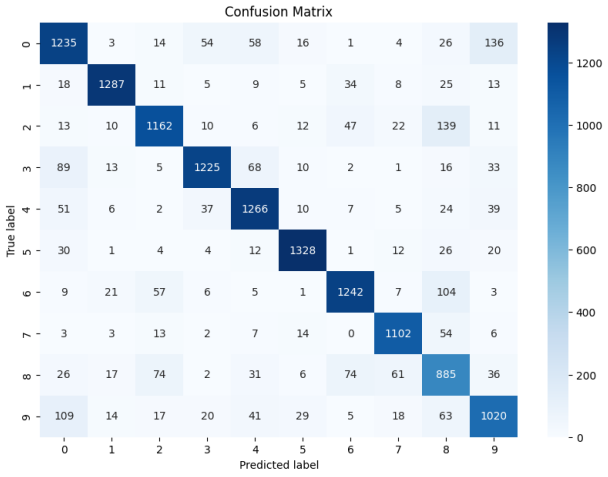

Figure 5: Sample Confusion matrix of model (with transformations)

It is remarkable how the model can adapt to different positions of the driver and not have a too steep accuracy decrease. Testing a "transformed" model with untransformed data (and vice versa) was not done, though I do expect the "transformed" model to have accuracy comparable to the "untransformed" model.

With the final model architecture, there were only 735 thousand trainable parameters, or a 2.8MB file size for the stored model. One other model has 15 million parameters, which is around

20 times more than this project's. [6] A benefit of small model size is that a dedicated GPU likely won't be needed to run the model at a decent frame rate. Rather, the computer inside a car would be able to run the model, depending on performance.

I didn't test greater or lesser amounts of transformation, I simply switched on and off certain parts of the pre-processing. This could have been interesting to find different compromises between model accuracy and model resilience and see how much I could push the model before it broke down.

IX. Conclusions

To answer the hypothesis, it appears that a CNN is able to detect distracted driving with higher accuracy. Even in more challenging conditions, it still functions well, though with reduced accuracy.

A noticeable effect of the transformations was that the rotations and the image shifts would decrease the accuracy of the model, likely caused by repositioning the studied objects in the image. No transformations would have the best accuracy, only one of rotations or image shifts would have middling accuracy, and both would have the worst accuracy. However, noise seemed to have little effect on the model, which suggests that the convolutional layers were able to filter out the noise in some way. However, noise might make a model less dependent on single pixels, so it might still be worthwhile to add to images.

A logical next step in improving would be to add more layers to the model. So far, the model does not appear to use any neurons that are added. So, if another layer is added instead, that could let the model learn more features and adapt to transformations better.

X. Application Discussion

The obvious application of this lightweight CNN is to use it in a real car, with a camera watching the driver and some sort of warning (e.g an audio chime) to tell the driver that they are distracted. Such a device wouldn't have to be complex or large in size, though it would need to have the processing power to run the CNN at a high enough frame rate for timely detection. A slightly different approach would be to integrate the CNN into the car itself, something that may be possible with newer cars having built-in cameras to watch the driver and built-in computers to assist in driving. While this may require recollecting data and retraining the model to a new perspective, this approach wouldn't require a separate device installed in the car. Instead, it would come with the car and be able to reach many more people, especially those with new semi-autonomous self-driving cars and who might need this kind of CNN the most.

XI. References

Journal Articles:

[1] Canadian Association of Chiefs of Police. "CANADA ROAD SAFETY WEEK – NATIONAL FACTS AND STATS May 16-24, 2023." *CACP*, 24 May 2023, www.cacp.ca/_Library/Documents/20230510152555172895795 5_crsw2023may1624nationalfactsstats.pdf.

[2] "Distracted Driving Statistics." *CAA National*, 16 Nov. 2022, www.caa.ca/driving-safely/distracted-driving/statistics/.

[3] Biondi, Francesco N., and Noor Jajo. *Human Factors Assessment of On-Road L2 Driving: Recommendations for the Implementation of Partially-Automated Vehicles.*, autoevolution, Oct. 2023, www.autoevolution.com/pdf/news_attachements/attention-span-goldfish-new-research-reveals-driving-an-l2-vehicle-can-be-twice-as-deadly-223496.pdf.

[4] Waddell, Dave. "Preliminary Findings of University of Windsor Study Shows Driver ..." *Windsor Star*, 12 Jan. 2023, windsorstar.com/news/local-news/preliminary-findings-of-university-of-windsor-study-shows-driver-assistance-systems-reduces-attentiveness.

[5] Dunn, N., Dingus, T.A. & Soccolich, S. (2019). *Understanding the Impact of Technology: Do Advanced Driver Assistance and Semi-Automated Vehicle Systems Lead to Improper Driving Behavior?* (Technical Report). Washington, D.C.: AAA Foundation for Traffic Safety.

[6] Baheti, Bhakti, Suhas Gajre, and Sanjay Talbar. "Detection of distracted driver using convolutional neural network." *Proceedings of the IEEE conference on computer vision and pattern recognition workshops.* 2018.

[7] SEOK JEONG-RO. "State Farm Distracted Driver Detection." *Kaggle*, State Farm, 2021, www.kaggle.com/datasets/rightway11/state-farm-distracted-driver-detection.

[8] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Mike Schuster, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng.
TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

Images:

https://kreizerlaw.com/distracted-driving-a-bigger-problem-than-you-might-think/

https://www.scottsfortcollinsauto.com/top-10-most-dangerous-driving-distractions/

https://pngtree.com/freebackground/digital-roadscape-with-many-vehicles-driving-on-it_2504932.html

https://www.analyticsvidhya.com/blog/2022/01/convolutional-neural-network-an-overview/

https://www.mychoice.ca/blog/distracted-driving-stats-canada-2022/

https://colab.research.google.com/

https://github.com/tensorflow/tensorflow

https://codechalleng.es/bites/331/

https://www.siliconrepublic.com/machines/autonomous-driving-tesla-ford-amazon-baidu

https://news.abplive.com/technology/frt-what-is-facial-recognition-technology-and-how-does-it-work-1467714

Webpages:

https://www.mathworks.com/videos/introduction-to-deep-learning-what-are-convolutional-neural-networks--1489512765771.html

https://www.techtarget.com/searchenterpriseai/definition/convolutional-neural-network

XII. Acknowledgements

I would like to thank my mentor, Simeon Sayer, for giving me guidance to follow and the tools to start building the CNN model.

I would also like to thank my family, for giving me advice and support on the project.