Distracted Driver Recognition for Insurance Purposes

Mert Bektas [1] **Taras Ivantsiv** [1]

Abstract

This project introduces a Deep Learning model and a Web Application that predicts if drivers are distracted to make accidents more transparent and also decreasing the numbers of accidents caused by distraction, by warning the said drivers.

1. Introduction

Distracted driving is a serious issue that can lead to accidents and injuries on the road. In the United States, over 3,100 people were killed and about 424,000 were injured in crashes involving a distracted driver in 2019.(1)

Our project aimed to develop a system that uses deep learning to detect distracted drivers and alert them. The system would also store information about distracted driving incidents on a server for future use. The ultimate goal of the project was to improve road safety and reduce the number of accidents caused by distracted driving.

In this project, we used a phone with a web app that accessed the camera and took pictures of the driver. We also used a server that was responsible for making predictions about distracted driving and for communication between the car and the insurance company. The phone sends the pictures to a server periodically. The server then uses deep learning technology to classify the snapshots and determine if the driver is distracted. If the driver is detected as being distracted, the web app will notify the driver. This can help prevent accidents by reminding the driver to focus on the road and avoid engaging in distracting activities while driving.

2. Preliminaries

2.1. Dataset

In this project we used the ”State Farm Distracted Driver Detection” dataset to train our model. This dataset is shared on Kaggle by the Insurance company ”State Farm”. (2) It contains over 100.000 images but we used 4806 of them.

Figure 1. An example of a Driving Labeled Photo.

Figure 2. An example of a Talking on the Phone Labeled Photo.

*Equal contribution [1]Institute of Computer Science, University of Tartu, Tartu, Estonia. Correspondence to: Mert Bektas <mert.bektas@ut.ee>, Taras Ivantsiv <taras.ivantsiv@ut.ee>.

There were 10 labels in the dataset and we used 2 of them to train the model. These labels are ”Driving” and ”Talking on the Phone”. Following two figures shows an example of these labels .

2.2. Binarization of Labels

With the help of the library ”scikit-learn”, we converted string labels into binary values. Therefore, after binarization our ”Driving” label is 0 and ”Talking on the Phone” label is 1.

2.3. Preprocessing

Preprocessing starts with cropping the most important part of the images, the drivers' arms. Next, images are getting resized to (270,175) from (480,640) to save from RAM and for a faster training of the model. Following this, backgrounds of the images are being removed. The purpose of this operation is to make the model focus only on arms of the drivers. And finally, images are converted to gray scale to make the model less perfectionist about the colors and focus on the position of the arms.

2.4. Image Augmentation

Image Augmentation process consists of ; rotating, shifting width and height, flipping horizontally, zooming and shearing.

Figure 3. An example from preprocessed images.

3. Web App

3.1. Frontend

The frontend obtains access to the user's webcam through the ‘getUserMedia‘ API, which is part of the HTML5 specification. This video stream is displayed on the page through the use of a ‘video‘ HTML5 element. Every three seconds, a screenshot of the video stream is taken and converted into a base64 encoded image. This image is then sent to the server through an XMLHttpRequest. If the server determines that the image represents someone speaking, an alert sound will be triggered on the page.

3.2. Server

The server is built using Python because it is easier to integrate with the deep learning model that is also written in Python. It uses the Flask framework to handle HTTP requests from the frontend. We chose this framework because it is simple and covers all of the project's requirements. The server uses the same Python libraries that were utilized for training the deep learning model, such as Numpy, OpenCV, and Tensorflow, which allows for seamless integration.

4. Convolutional Neural Network

We used many architectures for deep learning, but we will mention the first and the last of them.

The first model starts with 5 convolutional layer sets. A Convolutional layer set consists of Conv2D with parameters of 32, ReLU activation, MaxPooling2D and BatchNormalization layers. 4 Dense layers come after these sets. Parameters of them are 128, 64, 32, 16 in order. A ReLu activation layer is placed after each of these Dense layers. Finally, the model ends up with a Fully Connected Layer with Sigmoid activation and a single output, as our labels are binary.

The final model is the same as the first model except some additions. We added a Dropout layer with a parameter of 0.25 at the end of each Convolutional layer set and after every Dense layer. The purpose of this operation is to prevent models from overfit and be perfectionist about the unnecessary details in images.

5. Experiments

Experiments were done with the first and the final model by using a mobile phone in different places. Our first tests took place in our faculty. We didn't have a wheel to use to imitate actual driving pictures and this affected our test results. To resolve this problem we used Ivan's car later on.

For the first model we didn't use any preprocessing technique other than cropping and resizing. After training our first model we got the perfect result with 100% accuracy on test images.

For the reason of having 100% accuracy being a rare situation in deep learning, we came up with 2 approaches to prove the model was working correctly and not having bias. The first approach was training the model with only 50 images from the dataset. The second one was training the

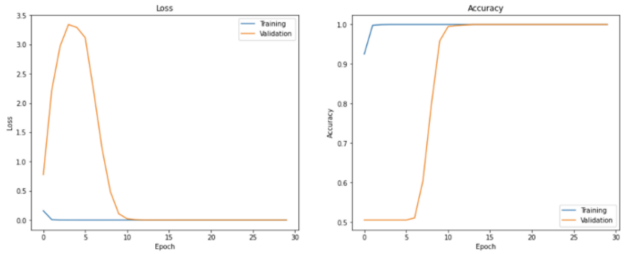

Figure 4. Training Statistics of the First Model.

	precision	recall	f1-score	support
0	1.00	1.00	1.00	510
1	1.00	1.00	1.00	452
accuracy			1.00	962
macro avg	1.00	1.00	1.00	962
weighted avg	1.00	1.00	1.00	962

Figure 5. Classification Report of the First Model.

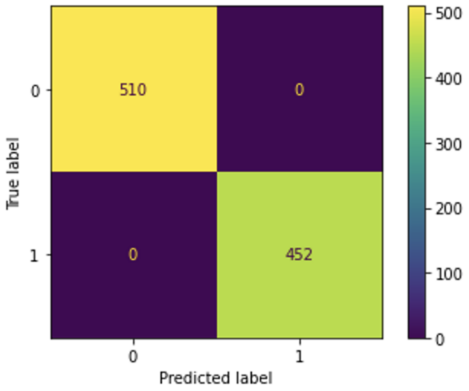

Figure 6. Confusion Matrix of the First Model.

model with all the images but assigning labels randomly, to confuse the model. After establishing these two and training the models, we found out they got bad results as you can see from the following figures. This meant our original model was real and working correctly.

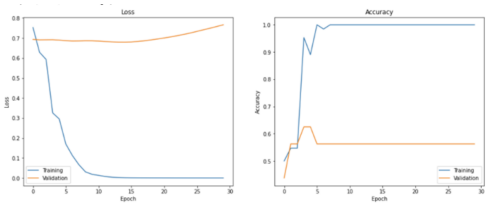

Figure 7. Training Statistics of the Model Trained with 50 Images.

	precision	recall	f1-score	support
0	0.00	0.00	0.00	11
1	0.45	1.00	0.62	9
accuracy			0.45	20
macro avg	0.23	0.50	0.31	20
weighted avg	0.20	0.45	0.28	20

Figure 8. Classification Report of the Model Trained with 50 Images.

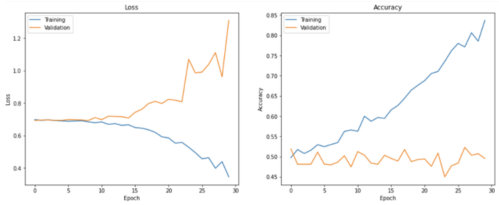

Figure 9. Training Statistics of the Model with Random Assigned Labels.

	precision	recall	f1-score	support
0	0.49	0.46	0.47	485
1	0.48	0.51	0.49	477
accuracy			0.48	962
macro avg	0.48	0.48	0.48	962
weighted avg	0.48	0.48	0.48	962

Figure 10. Classification Report of the Model with Random Assigned Labels.

In our first tests with our first model, we realised that our model was predicting every image of ours as driving, re-

gardless of if we are driving or not. To understand what is wrong with our model, we used Gradient-weighted Class Activation Mapping (Grad-CAM). Grad-Cam is a technique used to visualize which parts of images models focus on. After implementation and running of Grad-CAM we found out that our model was focusing on unnecessary parts of the images such as, back of the driving seat, top of the wheel, outside of the car, drivers' knees. To eliminate these, we came up with the idea of adding background removal to our preprocessing procedure and adding dropouts to our model.

Figure 11. An Image after Grad-CAM used on it. Red areas are the focus points.

Figure 12. An Image after our new preprocessing procedure.

After changing the model and preprocessing procedures, our model's training performance got lower but its in real life performance got higher.

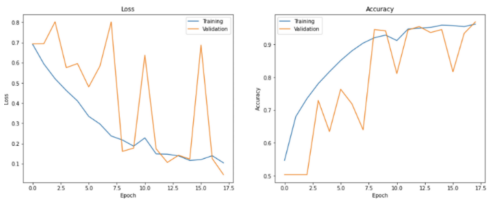

Figure 13. Training Statistics of the Final Model.

	precision	recall	f1-score	support
0	0.96	0.98	0.97	489
1	0.98	0.96	0.97	473
accuracy			0.97	962
macro avg	0.97	0.97	0.97	962
weighted avg	0.97	0.97	0.97	962

Figure 14. Classification Report of the Final Model.

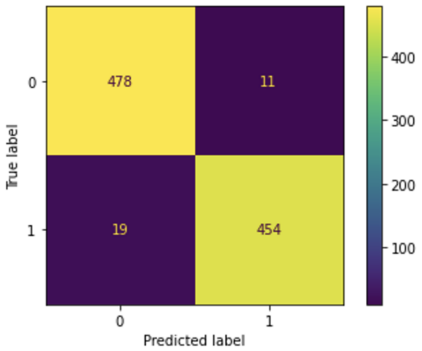

Figure 15. Confusion Matrix of the Final Model.

6. Conclusion

In conclusion, this project has demonstrated the potential of deep learning technology to reduce the number of accidents caused by distracted drivers. By using a deep learning model to detect and alert distracted drivers, it is possible to minimize the risks of distracted driving and improve road safety. This shows that deep learning technologies can play a valuable role in preventing accidents and protecting drivers and passengers on the road. The project's codes are available on: https://github.com/ivantsiv-ut/distracted-driver-recognition

References

[1] https://www.cdc.gov/transportationsafety/distracted_driving/index.html

[2] https://www.kaggle.com/competitions/state-farm-distracted-driver-detection/data