# Driver Attention Detection using CNN

Arpitha Gurumurthy, Gayathri Pulagam, Surabhi Govil

SJSU Davidson College of Engineering Extended Studies, Department of Software Engineering, San Jose, CA, US

***Abstract*— Distracted driving at the wheel is a phenomenon which happens with 1 in every 5 adults as reported by CDC[1] . There have been 72000 reported crashes on the national highway in 2013. A number of people have been injured due to the driver's distraction. Distraction can hamper one's ability to drive safely.We are not just pointing out the physical state of distraction like talking on a phone but also the brief moments where one might lose attention, for example, briefly turning around in the car. To tackle this problem we aimed to create a solution to detect driver's attention disruption using Convolution Neural Networks based computer vision techniques.**

***Keywords—driver attention detection, deep learning, computer vision, convolutional neural networks, feature extraction, resnet, efficientnet, data augmentation, vggnet, mlops, airflow***

## I. Introduction

A study has shown that there has been a steep increase in the number of fatal accidents recently. According to the CDC motor vehicle safety division, one in five car accidents is caused by a distracted driver. Apart from accidents, distracted drivers using mobile phones also tend to hold up the traffic particularly at traffic signals.

We are working on a dataset where we have images over 10 classes each representing a particular instance of a type of driving behavior.

Reckless driving can be classified into texting or talking on phone while driving. Talking to fellow passengers and not paying attention to the road. Having just one hand on the wheel and reaching the backseats while driving.

Using CNN we can create a model that identifies these classes and then this model can be served in an IOT device for real time predictions.

## II. Related Work

The authors of the paper Driver Fatigue Detection Based on Convolutional Neural Networks Using EM-CNN (Zuopeng Zhao, Nana Zhou, Lan Zhang, Hualin Yan, Yi Xu, Zhongxin Zhang, "Driver Fatigue Detection Based on Convolutional Neural Networks Using EM-CNN"[3] served as the basis of our experiments.

The authors used Multitask cascaded convolution neural networks (MTCNN) to create a ROI (Region of Interest). The ROI was eyes, mouth and measuring their movement.

Facial feature detection can be difficult due to changing driver positions and changes in environmental factors like lightning and background noise.

The authors exploit the depth factor in MTCNN and MTCNN being a hierarchical convolutional network can be easily used to calculate face bounding boxes. They use features of sub models to boost their correlating strengths. This helps detect the feature points of the left eye, right eye, left mouth, and right mouth corner. MTCNN returns bounding boxes and a confidence value. It also returns “key points” which are facial landmarks.

The EM-CNN approach in the paper helps evaluate state of eyes and mouth. As it is depth based it helps avoid misjudgment when only a single eye is visible. The proposed approach uses features extracted using MTCNN and multiple states of the eyes and mouth are acquired, and the fatigue state of the driver is evaluated.

The authors concluded that EM-CNN is more sensitive to facial features and gave better results.

The authors use two measures PERCLOS and POM. PERCLOS calculates eye closure degree and POM finds mouth opening degree.

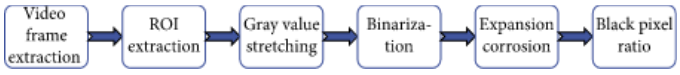



Fig1:Calculation of eye closure based on machine learning

## III. Dataset Preparation

We utilize the dataset provided from State Farm in Kaggle[1]. The dataset contains driver images, each taken in a car with a driver doing some activity in the car (texting, eating, talking on the phone, makeup, reaching behind, etc).

The images are segregated into 10 folders each representing the class of which the image belongs to. The 10 classes in the dataset are - *safe driving, texting-right, talking on the phone-right, texting-left, talking on the phone-left, operating the radio, drinking, reaching behind, hair and makeup, talking to the passenger.*

Fig 2. Sample image shows driver texting on the left

The images are divided into train (22,784 images) and test (79,700 images) folders in such a way that a driver can only appear in either train or test set. Along with the images, the dataset contains *driver_imgs_list.csv* file, which consists of a list of training images, subject (driver id) and class id.

| | subject | classname | img |
|---|---|---|---|
| 0 | p002 | c0 | img_44733.jpg |
| 1 | p002 | c0 | img_72999.jpg |
| 2 | p002 | c0 | img_25094.jpg |
| 3 | p002 | c0 | img_69092.jpg |
| 4 | p002 | c0 | img_92629.jpg |

Fig 3. driver_imgs_list.csv file

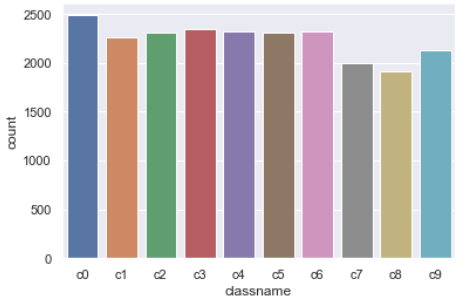



Fig4: Distribution of images

## IV. Methodologies

### A. Model Training

We trained CNN, VGG, EfficientNet models to gain an insight into which one performs the best. We trained all of our models on collabs or local setup with GPUs. The details of each model are explained under the *Experiments* section.

After our experiments, we found that CNN and VGG fared well in comparison to EfficientNet. We tweaked these models and eventually deployed them using Apache Airflow for a real-time inference from the model.

### B. MLOps Pipeline setup

An ML Pipeline facilitates us to run all the required steps of an ML system, starting from Data collection to model deployment and serving.

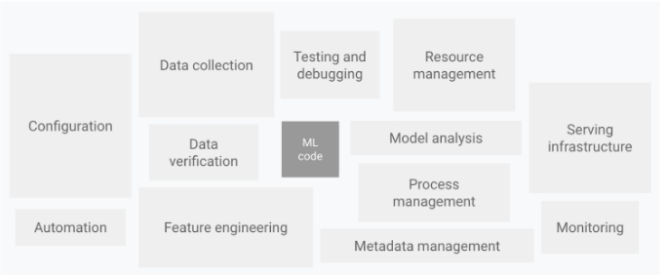



Fig 5: A typical MLOPS architecture

We used Apache Airflow as our ML pipeline orchestrator.
In order to achieve this, below are the steps we followed:

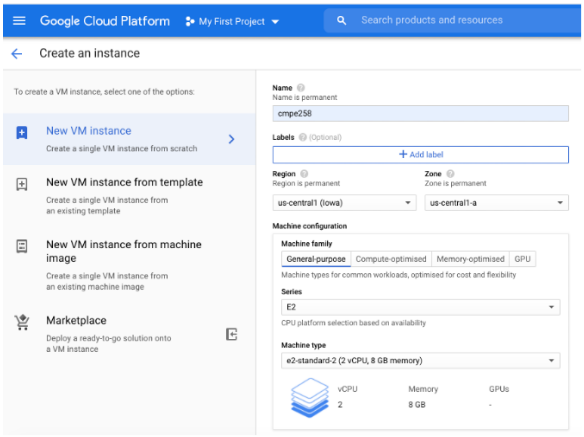
1. We built a docker container with installations for airflow, tensorflow,

OpenCV, tensorboard etc. We packaged our python scripts into the docker as well.

2. We brought up the docker container on GCP which deploys our trained model and offers predictions.

*GCP setup*

We created a compute instance on the GCP console with the below shown configurations.



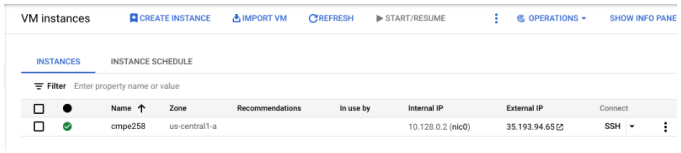
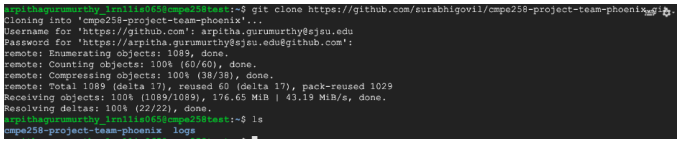
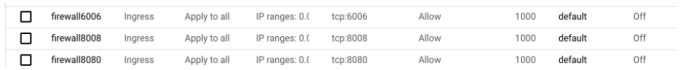




Fig 6: GCP Setup

Then, we ssh into the instance and cloned our github repository which has all the necessary files.




We created new firewall rules on GCP to open traffic on ports – 8080, 8008 and 6006 respectively for airflow UI, our application UI and tensorboard.

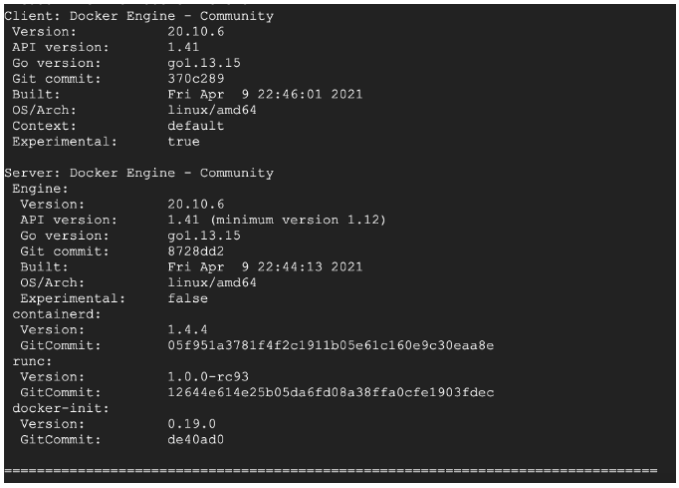



*Docker setup*

We installed docker engine on GCP server using the command –

`bash <(curl -s https://get.docker.com/)`

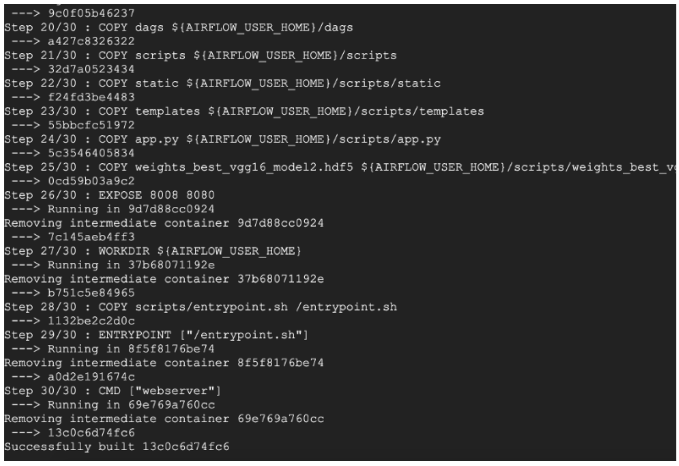
We can see successful docker installation below:



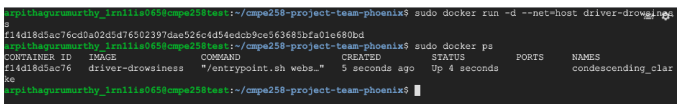

We have all the instructions required to set up our application written in the “Dockerfile” and also all the dependencies under “requirements.txt”.

We now build our docker using the following command –

` docker build -t driver-drowsiness:latest .`




On running the docker, we can see the docker container getting created:




We can also see that the Airflow UI now comes up on port 8080. We can see that our pipeline has been listed:

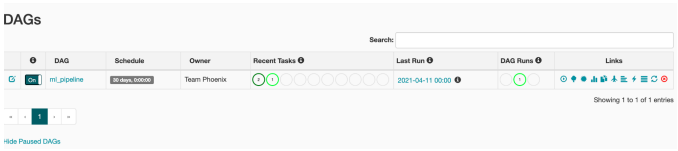



Fig 7: Starting a pipeline in Airflow using DAGs

Below is the graph view of the sequence of steps in the DAG:

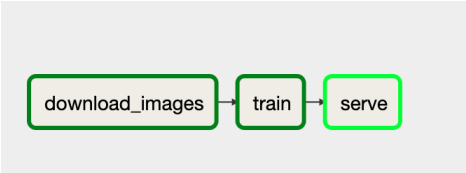
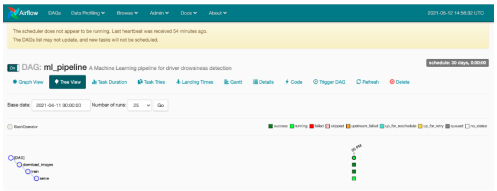


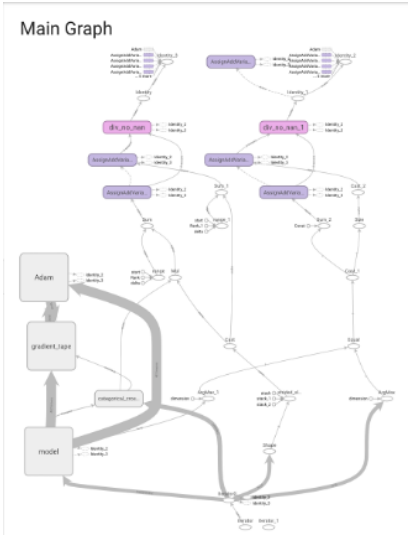

Fig 8: Airflow MLOps Pipeline

Below is the Tree view of the same:




We can set the scheduler to periodically run our custom dag - defined by downloading images, training and serving.

We train our model on the image data to observe the logs on tensorboard. Since we are training our model on the fly, we run 4 epochs. The tensorboard comes up on port 6006 displaying the metrics on accuracy and loss:



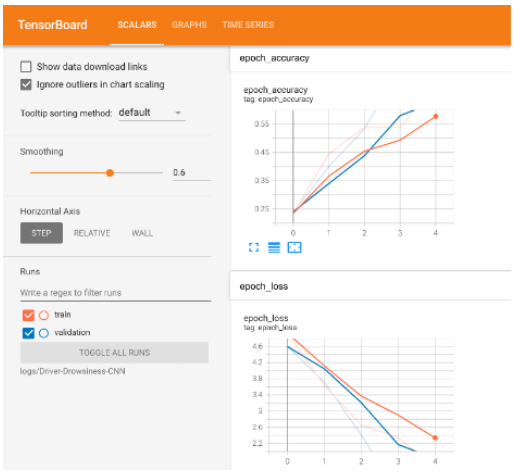
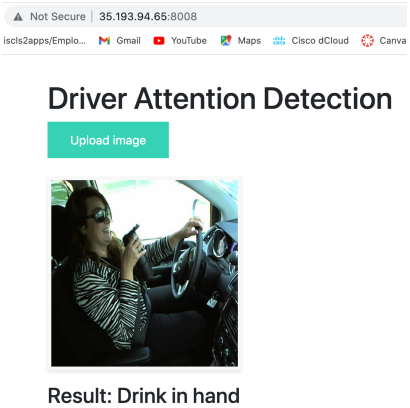
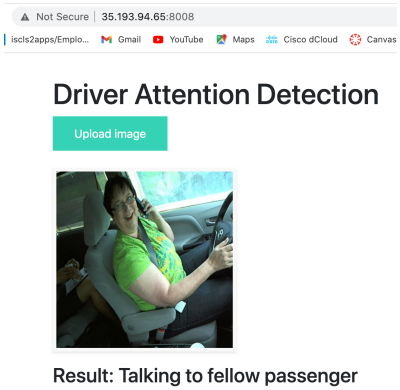






Fig 9: Tensorboard outputs from pipeline execution in Airflow

Some of the inference examples of classifications can be seen below in the example images -






## V. EXPERIMENTS

### A. CNN

CNN has evolved to be the state of the art computer vision technique. CNN has become increasingly popular among other neural networks like RNN, LSTM, ANN etc.

They are a preferred choice for computer vision tasks like image classification, object detection, image classification, etc.

A Convolutional Neural Network is a powerful neural network that uses filters to extract features from images. It also does so in such a way that position information of pixels is retained

.

A convolution is a mathematical operation applied on a matrix. This matrix is usually the image represented in the form of pixels/numbers. The convolution operation extracts the features from the image.

CNN takes an imager raw pixel and draws inferences from it to learn what objects it can extract from the image.

We have used a Sequential model from keras
We used RELU as an activation function to introduce non linearity into the model. Next Pooling layer was used to downsample convoluted features and save processing time.

### B. VGG

VGG was introduced in a 2014 paper titled “Very Deep Convolution Networks for Large-Scale Image Recognition” by KAren Simonyan and Andrew Zisserman

VGG using small receptive fields of (3x3 with a stride of 1). The small size of the convolution filter allows VGG to have more weight layer thus increased performance.

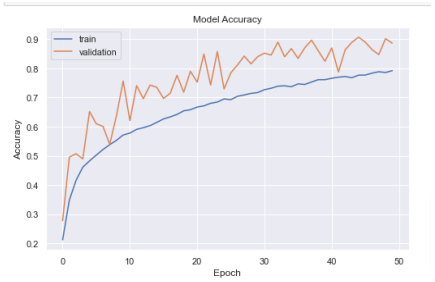



Fig10: Model Accuracy using a VGG

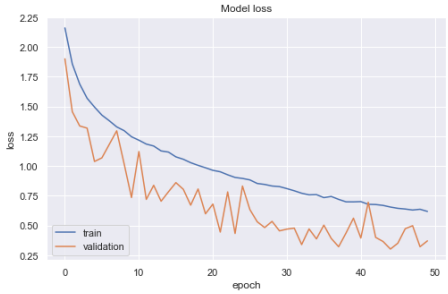
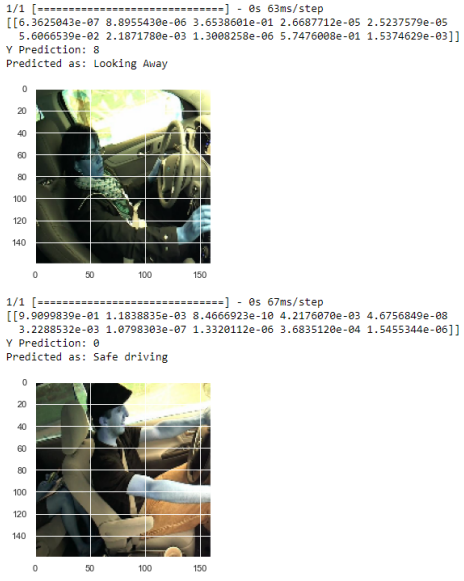



Fig11: Model Loss using a VGG

Predictions from VGG based model



*Methodology we adopted -*

For transfer learning using VGG we used imagenet weights and removed the fully connected top layer. We replaced it with a Dense layer with Softmax to predict the 10 classes.

We then freezed all of the layers of the pertained model. This was important as the convolutional layers will be initialized with weights based on a training on ImageNet dataset.

As we can confidently say that the convolutional layers work as feature extractors.

Training the whole network with our 10 classes, will train the new fully-connected layers and fine-train the convolutional layers. Freezing the layer helps keep the exact same feature extractor.

And to make the best use of feature extractor images were normalised using standard deviation while augmenting images using keras image data generator.

*C. EfficientNet*

As part of our experiments, we also tried transfer learning with EfficientNet. First, we did not include the top layer of EfficientNetB3 and added 3 dense layers. The results were around 25% train and val accuracy.

In the second round of experimentation, we froze the trainable parameters of the EfficientNetB3 model. We added Global Average Pooling layer, dropout with a value of 0.2 and batch normalization. In this context, the model performed much better at 60% train accuracy and 45% validation accuracy at 25 epochs.

We also tried using different optimizers like SGD, rmsprop and Adam. In our case, Adam optimizer gave us the best performance. In terms of image preprocessing, in 2 out of 3 trials, we used image augmentation and in the third experiment, we tried image augmentation along with image normalization, horizontal flip and converting the images into grayscale. Image normalization along with augmentation gave us the best results.

## VI. Evaluation

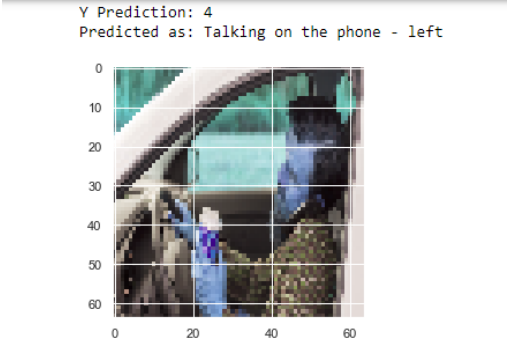



Fig12: Prediction on images different from test set

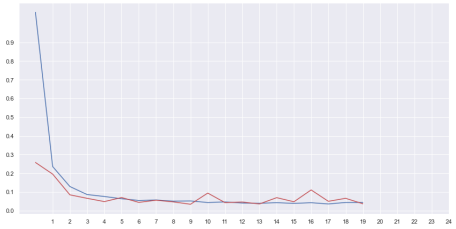


Fig13: Training Loss v/s Accuracy Loss

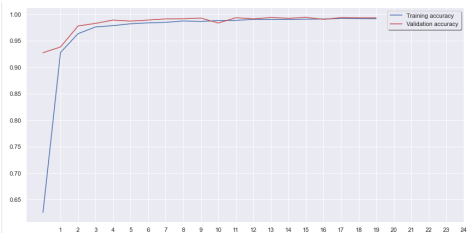


Fig14: Training Accuracy v/s Validation Accuracy

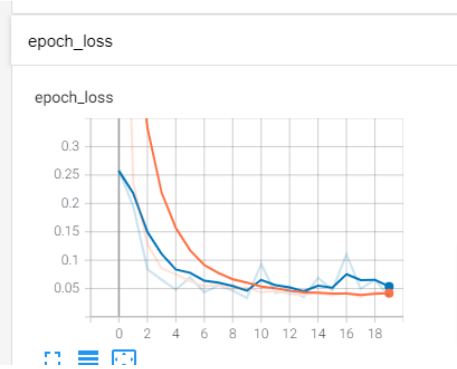



Fig15: TensorBoard Logs of model training

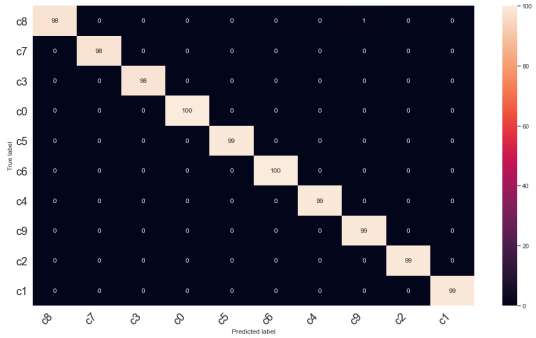


Fig16:Confusion Matrix of Predicted labels from CNN

## VII. Challenges

We faced challenges when trying to run the model due to the large set of data and we faced compute power issues. To make the model more generalizable, a large varied and balanced dataset is required.

Another learning experience was that the training and validation accuracies in general were low and this owed to the construction of various layers and right hyperparameters.

We also observed that the images in the training data were composed of the same driver in various images. This could cause the model validation to be wrong in case the same driver appears in validation data.

## VIII. Conclusion

Distracted driving is a real concern and is estimated to cause a lot of accidents on the road. We aimed to curb this problem by developing a model which classifies the attentiveness of a driver into one of the 10 classes defined.

We used CNN architecture based models to solve this problem. For future work, we aim to fine tune the model to perform better on unseen test data.

## Acknowledgment

We would like to acknowledge and thank our professor, Vijay Eranti for aiding us throughout our project.

## References

[1] StateFarm Distracted Driver Detection Dataset https://www.kaggle.com/c/state-farm-distracted-driver-detection

[2] Machine Learning Pipeline using Apache airflow https://towardsdatascience.com/10-minutes-to-building-a-machine-learning-pipeline-with-apache-airflow-53cd0926897

[3] Driver Fatigue Detection Based on Convolutional Neural Networks https://doi.org/10.1155/2020/7251280