Detection of Distracted Driver using Convolutional Neural Network and VGG-16

Manya Goel, 2310110173 Diya Budlakoti, 2310110466

***Abstract*—The number of road accidents has been steadily increasing in recent years worldwide. According to a survey by the National Highway Traffic Safety Administration, nearly one in five motor vehicle crashes is caused by distracted driving. India accounts for 11 per cent of global deaths in road accidents. We attempt to develop an accurate and robust system for detecting distracted drivers and warn them against it. Motivated by the performance of Convolutional Neural Networks in computer vision, we present a CNN-based system that not only detects the distracted driver but also identifies the cause of distraction. For the scope of this project, we will focus on building a highly efficient ML model to classify different driver distractions at runtime using computer vision. We would also analyze the overall speed and scalability of the model in order to be able to set it up on an edge device. We use CNN and VGG-16 to predict the classes.**

I. INTRODUCTION

According to the World Health Organization (WHO) survey, 1.3 million people worldwide die in traffic accidents each year, making them the eighth leading cause of death, and an additional 20-50 million are injured/ disabled. As per the report of the National Crime Research Bureau (NCRB), Government of India, Indian roads account for the highest fatalities in the world. There has been a continuous increase in road crash deaths in India since 2006. The report also states that the total number of deaths has risen to 1.46 lakhs in 2015, and driver error is the most common cause behind these traffic accidents.

Driver distraction has been identified as the main reason for accidents. Distractions can be caused due to reasons such as mobile usage, drinking, operating instruments, facial makeup, and social interaction. NTHSA describes distracted driving as "any activity that diverts attention of the driver from the task of driving," which can be classified into Manual, Visual, or Cognitive distraction. As per the definitions of the Centers for Disease Control and Prevention (CDC), cognitive distraction is basically "the driver's mind is off the driving". In other words, even though the driver is in a safe driving posture, he is mentally distracted from the task of driving. He might be lost in thoughts, daydreaming, etc. Distraction because of inattention, sleepiness, fatigue, or drowsiness falls into the visual distraction class, where "drivers' eyes are off the road". Manual distractions are concerned with various activities where "driver's hands are off the wheel. Such distractions include talking or texting using mobile phones, eating and drinking, talking to passengers in the vehicle, adjusting the radio, and applying makeup etc.

Nowadays, Advanced Driver Assistance Systems (ADAS) are being developed to prevent accidents by offering technologies that alert the driver to potential problems and to keep the car's driver and occupants safe if an accident does occur. But even today's latest autonomous vehicles require the driver to be attentive and ready to take control of the wheel back in case of an emergency. Tesla Autopilot's crash with the white truck-trailer in Williston, Florida, in May 2016 was the first fatal crash in the testing of an autonomous vehicle. This makes the detection of distracted drivers an essential part of the self-driving cars as well. We believe that distracted driver detection is of utmost importance for further preventive measures.

If the vehicle could detect such distractions and then warn the driver against them, the number of road crashes could be reduced. In this paper, we focus on detecting manual distractions where the driver is engaged in other activities than safe driving, and also identify the cause of distraction. We present a Convolutional Neural Network-based approach for this problem. We would also analyze the overall speed and scalability of the model in order to be able to set it up on an edge device.

II. RELATED WORK

This section summarises review of some of the relevant and significant work from literature for distracted driver detection. Zhang et al. proposed a guided-learning framework for detecting driver mobile phone usage. Instead of relying solely on a CNN to learn features automatically, the authors introduce attention maps that guide the model to focus on semantically important regions such as the driver's hands, face orientation, and phone-relevant areas. In addition, domain priors such as typical hand–phone spatial relations and changes in gaze are incorporated into the feature extraction process to reduce ambiguity and suppress irrelevant background information. This work highlights the importance of integrating guided attention and prior knowledge for improving distraction and phone-usage detection systems.

On analysing other computer vision problems, it was identified that there are multiple pre-trained models such as ImageNet, Sports 19M, IG-Kinetics 19M Images/250M Images and 19 M Videos. In some cases, short, localised videos performed better than long videos/frames for identifying the actions. Also, faster performance can be obtained by starting with a 2D architecture, and inflating all the filters and pooling kernels – endowing them with an additional temporal dimension. Filters

are typically square, and we just make them cubic – N × N filters become N × N × N.

The earlier datasets concentrated on only a limited set of distractions, and many of them are not publicly available. In April 2016, StateFarm's distracted driver detection competition on Kaggle defined ten postures to be detected (Safe driving + nine distracted behaviours). This was the first dataset to consider a wide variety of distractions and was publicly available. However, CNNs proved to be the most effective techniques, achieving high accuracy. In 2017, Abouelnaga et al. created a new dataset similar to StateFarm's dataset for distracted driver detection. The authors preprocessed the images by applying skin, face and hand segmentation and proposed the solution using a weighted ensemble of five different Convolutional Neural Networks. The system achieved good classification accuracy, but is computationally too complex to be real-time time which is utmost important in autonomous driving.

III. DATASET DESCRIPTION

For this study, we used the StateFarm Distracted Driver Detection Dataset, released on Kaggle in 2016 as part of a large-scale competition. This dataset is one of the most widely used benchmarks for driver distraction detection and has been adopted in numerous research studies. It provides a diverse collection of in-car driver images captured under real driving conditions.

The dataset consists of approximately 79,000 test images and 22,000 training images, all of which are colored and taken from inside a vehicle. The images are categorized into 10 distinct classes, representing different types of distracted-driving behaviours:

- c0: safe driving
- c1: texting (right hand)
- c2: talking on the phone (right hand)
- c3: texting (left hand)
- c4: talking on the phone (left hand)
- c5: operating the radio
- c6: drinking
- c7: reaching behind
- c8: hair and makeup
- c9: talking to passenger

TABLE I
CLASS-WISE DISTRIBUTION OF IMAGES IN THE STATEFARM DATASET

Classes	Driver Actions	Images
C0	safe driving	2489
C1	texting - right	2267
C2	talking on the phone - right	2317
C3	texting - left	2346
C4	talking on the phone - left	2326
C5	operating the radio	2312
C6	drinking	2325
C7	reaching behind	2002
C8	hair and makeup	1911
C9	talking to passenger	2129
Total		**22424**

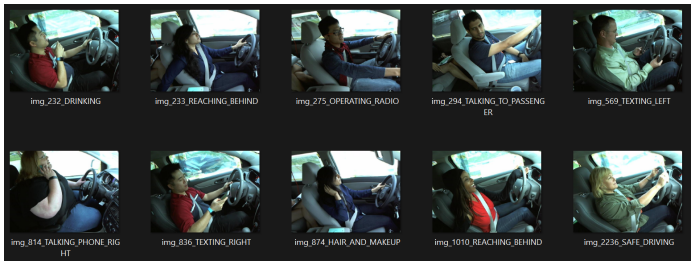

Fig. 1. Image visualization of all 10 classes

In addition to the images, the dataset includes two CSV files containing file paths and their corresponding class labels, which are used to load and preprocess the training data. A separate folder, test_actual, contains test images along with their ground-truth classifications. This folder enables us to compute each model's test accuracy and evaluate its real-world performance.

IV. TECHNICAL APPROACH

The required libraries include TensorFlow 2.16, Keras 3.3, OpenCV, NumPy, Pandas, Matplotlib, sklearn, etc. We have defined paths for training data, testing data, CSV files, pickle files, and the model folder.

A. *CNN ARCHITECTURE*

1) **Convolution Layers**
Each convolution layer uses a set of filters (kernels) to learn features from the image.
   - Conv1 → 64 filters
   - Conv2 → 128 filters
   - Conv3 → 256 filters
   - Conv4 → 512 filters

   Early layers with fewer filters focus on simple, low-level details such as edges, corners, and textures. As the network goes deeper, the number of filters increases, allowing the model to capture more complex and abstract structures such as shapes, object parts, and eventually full objects.
2) **Max Pooling**
Max pooling is applied to reduce the spatial dimensions of the feature maps. This step acts as a summarizer by keeping the most important information through selecting the maximum value in small regions. It helps the network focus on what features are present rather than their exact location, reducing computation and improving robustness.
3) **Dropout Layer**
To prevent overfitting, dropout layers randomly switch off a fraction of neurons during training. This forces the model to learn a broader and more general set of features instead of memorizing the training data.
4) **Flatten Layer**
After feature extraction, the output is passed through a flatten layer, which converts the two-dimensional feature maps into a one-dimensional vector. For example, a

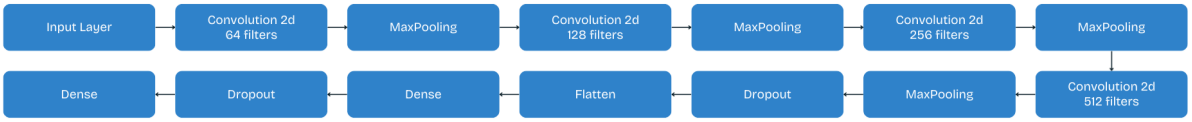

Fig. 2. Basic architecture of a CNN model for both batchwise and non-batch processing.

feature map of size $7 \times 7 \times 512$ becomes a vector of length 25088, making it suitable for fully connected layers.

5) **Fully Connected (Dense) Layers**
Dense layers act as the final decision-making component. The first dense layer learns high-level combinations of all extracted features, and dropout may also be applied here to improve generalization. The final dense layer contains ten neurons (one for each class) and uses a softmax activation function to convert the output into class probabilities.
For example, an output like [cat: 0.9, dog: 0.1] indicates a 90 percent probability that the image belongs to the cat class.

In summary, CNNs hierarchically learn features:

- Early layers: low-level features such as edges and textures.
- Middle layers: shapes and patterns like eyes or noses.
- Final layers: complete objects such as a cat face or a dog face.

Drawbacks of CNNs are that they are not fully invariant to rotation, scaling, viewpoint change, and occlusion. They must manually add data augmentation to learn robustness. They are computationally very expensive and can easily overfit on small datasets.

1) CNN Batchwise: The pixel range here is between [0,255]. Figure 3 shows the model summary of CNN batchwise.

The model processes multiple images together at once (a batch). Batch size = 128 means the network sees 128 images before updating its weights. The following explains how the model works.

1) Images are grouped into batches of 128.
2) Simultaneously run on all 128 images.
3) The average loss across the batch is calculated.
4) Then weight update is performed for the entire batch.
5) The process repeats for the nest batch.

2) CNN Non Batch: The pixel range is between [-0.5,0.5]. Figure 4 shows the model summary of CNN non batch.

The model processes one image at a time, updating weights after every single image. The following explains how the model works:

1) Load 1 image, then run
2) Compute loss for that 1 sample
3) Update weights
4) Repeat for the next image

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 224, 224, 64)	832
max_pooling2d (MaxPooling2D)	(None, 112, 112, 64)	0
conv2d_1 (Conv2D)	(None, 112, 112, 128)	32896
max_pooling2d_1 (MaxPooling2D)	(None, 56, 56, 128)	0
conv2d_2 (Conv2D)	(None, 56, 56, 256)	131328
max_pooling2d_2 (MaxPooling2D)	(None, 28, 28, 256)	0
conv2d_3 (Conv2D)	(None, 28, 28, 512)	524800
max_pooling2d_3 (MaxPooling2D)	(None, 14, 14, 512)	0
dropout (Dropout)	(None, 14, 14, 512)	0
flatten (Flatten)	(None, 100352)	0
dense (Dense)	(None, 500)	50176500
dropout_1 (Dropout)	(None, 500)	0
dense_1 (Dense)	(None, 10)	5010

Total params: 50,871,366
Trainable params: 50,871,366
Non-trainable params: 0

Fig. 3. Model summary of CNN batchwise

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 128, 128, 64)	832
max_pooling2d (MaxPooling2D)	(None, 64, 64, 64)	0
conv2d_1 (Conv2D)	(None, 64, 64, 128)	32896
max_pooling2d_1 (MaxPooling2	(None, 32, 32, 128)	0
conv2d_2 (Conv2D)	(None, 32, 32, 256)	131328
max_pooling2d_2 (MaxPooling2	(None, 16, 16, 256)	0
conv2d_3 (Conv2D)	(None, 16, 16, 512)	524800
max_pooling2d_3 (MaxPooling2	(None, 8, 8, 512)	0
dropout (Dropout)	(None, 8, 8, 512)	0
flatten (Flatten)	(None, 32768)	0
dense (Dense)	(None, 500)	16384500
dropout_1 (Dropout)	(None, 500)	0
dense_1 (Dense)	(None, 10)	5010

Total params: 17,079,366
Trainable params: 17,079,366
Non-trainable params: 0

Fig. 4. Model summary of CNN non batch

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, None, None, 3)	0
block1_conv1 (Conv2D)	(None, None, None, 64)	1,792
block1_conv2 (Conv2D)	(None, None, None, 64)	36,928
block1_pool (MaxPooling2D)	(None, None, None, 64)	0
block2_conv1 (Conv2D)	(None, None, None, 128)	73,856
block2_conv2 (Conv2D)	(None, None, None, 128)	147,584
block2_pool (MaxPooling2D)	(None, None, None, 128)	0
block3_conv1 (Conv2D)	(None, None, None, 256)	295,168
block3_conv2 (Conv2D)	(None, None, None, 256)	590,080
block3_conv3 (Conv2D)	(None, None, None, 256)	590,080
block3_pool (MaxPooling2D)	(None, None, None, 256)	0
block4_conv1 (Conv2D)	(None, None, None, 512)	1,180,160
block4_conv2 (Conv2D)	(None, None, None, 512)	2,359,808
block4_conv3 (Conv2D)	(None, None, None, 512)	2,359,808
block4_pool (MaxPooling2D)	(None, None, None, 512)	0
block5_conv1 (Conv2D)	(None, None, None, 512)	2,359,808
block5_conv2 (Conv2D)	(None, None, None, 512)	2,359,808
block5_conv3 (Conv2D)	(None, None, None, 512)	2,359,808
block5_pool (MaxPooling2D)	(None, None, None, 512)	0
sequential (Sequential)	(None, 10)	5,130

Total params: 14,719,818 (56.15 MB)
Trainable params: 14,719,818 (56.15 MB)
Non-trainable params: 0 (0.00 B)

Fig. 5. Model summary of VGG16 fine tuned batchwise

Layer (type)	Output Shape	Param #
input_layer (InputLayer)	(None, None, None, 3)	0
block1_conv1 (Conv2D)	(None, None, None, 64)	1,792
block1_conv2 (Conv2D)	(None, None, None, 64)	36,928
block1_pool (MaxPooling2D)	(None, None, None, 64)	0
block2_conv1 (Conv2D)	(None, None, None, 128)	73,856
block2_conv2 (Conv2D)	(None, None, None, 128)	147,584
block2_pool (MaxPooling2D)	(None, None, None, 128)	0
block3_conv1 (Conv2D)	(None, None, None, 256)	295,168
block3_conv2 (Conv2D)	(None, None, None, 256)	590,080
block3_conv3 (Conv2D)	(None, None, None, 256)	590,080
block3_pool (MaxPooling2D)	(None, None, None, 256)	0
block4_conv1 (Conv2D)	(None, None, None, 512)	1,180,160
block4_conv2 (Conv2D)	(None, None, None, 512)	2,359,808
block4_conv3 (Conv2D)	(None, None, None, 512)	2,359,808
block4_pool (MaxPooling2D)	(None, None, None, 512)	0
block5_conv1 (Conv2D)	(None, None, None, 512)	2,359,808
block5_conv2 (Conv2D)	(None, None, None, 512)	2,359,808
block5_conv3 (Conv2D)	(None, None, None, 512)	2,359,808
block5_pool (MaxPooling2D)	(None, None, None, 512)	0
sequential_1 (Sequential)	(None, 10)	5,130

Total params: 14,719,818 (56.15 MB)
Trainable params: 14,719,818 (56.15 MB)
Non-trainable params: 0 (0.00 B)

Fig. 6. Model summary of VGG16 fine tuned non-batch

B. VGG16 ARCHITECTURE

VGG Net is one of the most influential CNN architecture from literature. It reinforced the idea that networks should be deep and simple. It works well on both image classification as well as localization task.

- use only 3×3 convolution filters
- stack many of them
- increase filters as the network goes deeper

It has 13 convolution layers + 3 fully connected layers = 16 learnable layers.
Since VGG16 is also a CNN, it uses the same types of layers, but not in the same quantity, depth, or arrangement as a simple/custom CNN.

1) **Convolution Layers** VGG16 is divided into 5 blocks, each containing 2–3 convolution layers followed by max-pooling.
   - Block 1
     - Conv1-1 → 64 filters
     - Conv1-2 → 64 filters

     Purpose: learns simple edges, corners, color gradients.
   - Block 2
     - Conv2-1 → 128 filters
     - Conv2-2 → 128 filters

     Purpose: learns slightly more complex textures, curves.
   - Block 3
     - Conv3-1 → 256 filters
     - Conv3-2 → 256 filters
     - Conv3-3 → 256 filters

     Purpose: learns patterns like circles, repeated textures, mid-level shapes.
   - Block 4
     - Conv4-1 → 512 filters
     - Conv4-2 → 512 filters
     - Conv4-3 → 512 filters

     Purpose: learns structured shapes: eyes, wheels, stripes, geometry.
   - Block 5
     - Conv5-1 → 512 filters
     - Conv5-2 → 512 filters
     - Conv5-3 → 512 filters

     Purpose: learns full object parts (faces, legs, silhouettes) and high-level features.

As the image is compressed through pooling, the network increases the number of filters so it can capture more and more abstract patterns (therefore, the deeper the blocks, the more object-level concepts).

2) **Max Pooling Layers** Each block ends with a Max Pooling layer to reduce the spatial size it keeps only the strongest feature activations, makes the network more robust to small movements of the object and reduces computation by lowering image resolution block-by-block.
3) **Flatten Layer** After the final max pooling, the feature maps are converted into a 1-dimensional vector using a Flatten layer, enabling the transition from feature extraction to classification.
4) **Fully Connected (Dense) Layers** Since VGG16 is being fine-tuned, the original classification head (4096 → 4096 → 1000) is replaced with a custom dense head in the sequential block. It also has a Final Dense layer with softmax for the number of classes (Softmax converts the previous layer's raw scores into probabilities).
5) **Fine Tuning** Using Transfer Learning, load VGG16 without the top classification layer, use its convolutional layers as a feature extractor and add our own classifier top.
   When we fine-tune VGG16 for our own dataset, we modify only the top of the model.
   The original 1000-class ImageNet classifier is replaced with:
   - Global Average Pooling or Flatten
   - Dense (e.g., 512 units)
   - Dropout(0.5): Prevents overfitting.
   - Softmax

   We also lock all convolution layers, so VGG16's learned features remain unchanged. Only the new dense layers learn from the data. This gives a specialised classifier using VGG16's powerful feature extractor.

The major drawback of VGG-16 is total number of parameters which counts to be nearly 140M. Fully connected layers are computationally too expensive and also consume most of these parameters. Also, the network with fully connected layer can be applied to input of fixed size only. Replacing fully connected layer with convolution layer saves the number parameters and it can be applied to varying input size. In the modified network architecture number of parameters are reduced to 15M that is only 11% of the original VGG-16 parameters. All the regularization parameters remain unchanged.

1) VGG16 FINE TUNED BATCHWISE: The pixel range here is between [0,255]. Figure 5 shows the model summary of VGG16 batch- wise. Batchwise training divides the dataset into mini-batches, and updates the model parameters after each batch. In our case, a batch size of 128 was used. The concept of batchwise models works the same way here as it did in section 2.1.

2) VGG16 FINE TUNED NON-BATCH: The pixel range is between [-0.5,0.5]. Figure 6 shows the model summary of VGG16 fine-tuned non-batch. Non-batch training includes two extremes: batch size = 1 or full-batch training (using the entire dataset in a single gradient update).

V. RESULTS

In this section, we evaluate the performance of four different models: CNN Non-Batch, CNN Batchwise, VGG16 Fine-Tuned Non-Batch, and VGG16 Fine-Tuned Batchwise on a dataset consisting of 10,000 test images across the 10 classes. The objective was to compare the effect of batchwise vs. non-batchwise training on both a custom CNN and a deep transfer-learning model (VGG16). The evaluation metrics include training accuracy, test accuracy, learning curves, and confusion matrices.

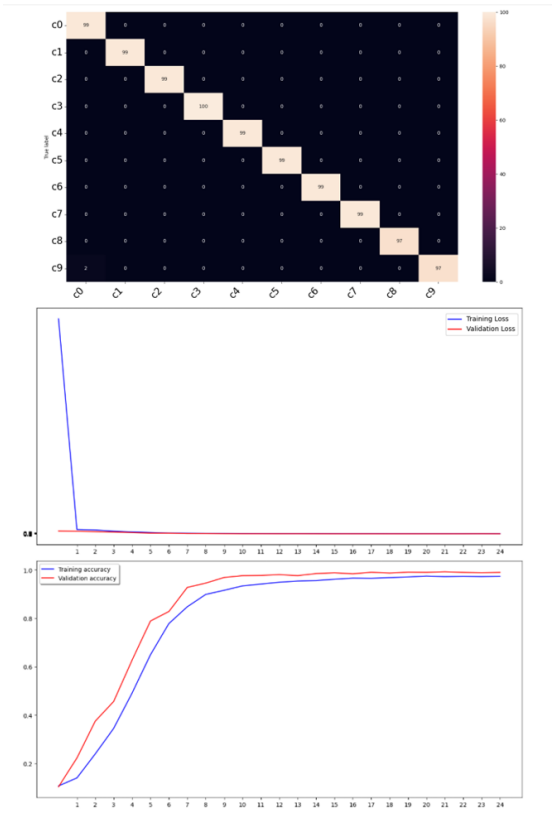

Fig. 7. Graph and Confusion Matrix of CNN Batchwise

- All models achieved 99% test accuracy, with batchwise versions slightly outperforming non-batch versions.
- Batchwise training provided smoother optimization and more stable learning curves.
- Non-batch training produced noisier gradients but occasionally better peak accuracy.
- VGG16 models outperformed custom CNN in terms of training vs validation graph. But CNN performed better on test data.
- Confusion matrices and real-world demos confirm the reliability of all four models for practical deployment.

VI. CONCLUSION

Driver distraction is a serious problem leading to large number of road crashes worldwide. Hence detection of distracted driver becomes an essential system component in self driving cars. Here, we present a robust Convolutional Neural Network-based system to detect distracted driver and also identify the cause of distraction.

Overall, both CNN and VGG16 architectures perform exceptionally well, with training strategy primarily affecting training

TABLE II
TEST ACCURACY VS TRAIN ACCURACY

Model	Total Images	Correct	Test Accuracy	Train Accuracy
CNN NON BATCH	10000	9988	99.88%	99.5095%
CNN BATCHWISE	10000	9979	99.79%	99.0627%
VGG16 FINE TUNED NON BATCH	10000	9986	99.86%	99.1081%
VGG16 FINE TUNED BATCHWISE	10000	9946	99.46%	99.0723%

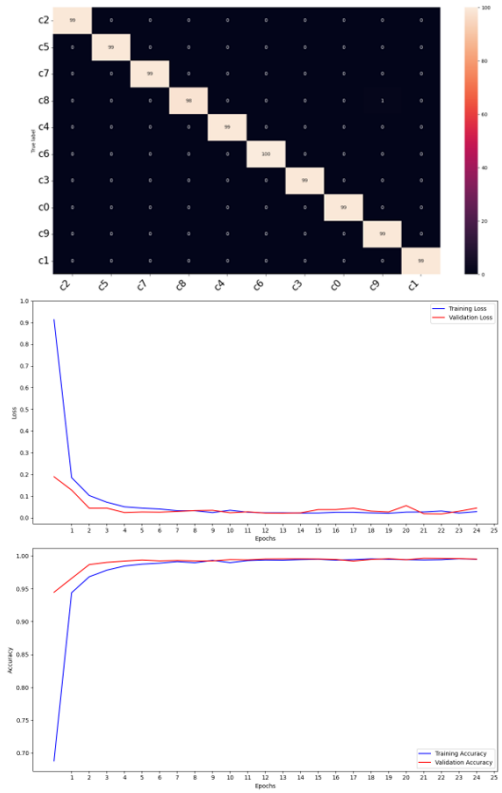

Fig. 8. Graph and Confusion Matrix of CNN Non-Batch

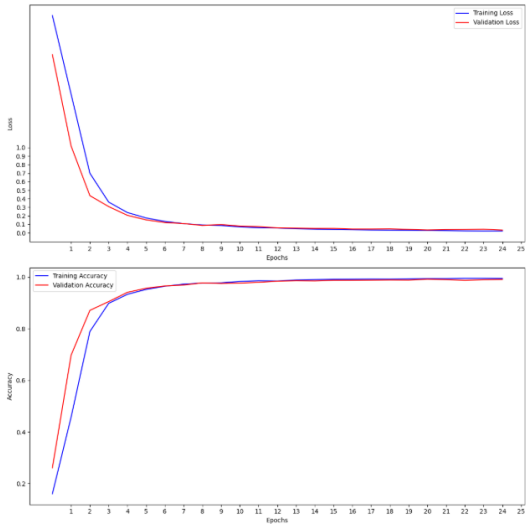

Fig. 9. Graph of VGG16 Fine Tuned Batchwise

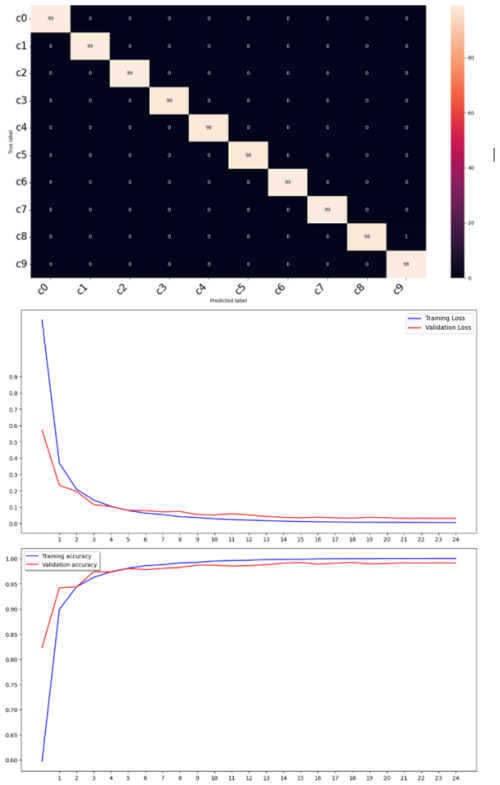

Fig. 10. Graph and Confusion Matrix of VGG16 Fine Tuned Non-Batch

dynamics rather than final accuracy. Additional qualitative demonstrations on images and videos were performed and are provided in the project presentation and supplementary drive folder. Refer to fig, 7, 8, 9, 10 for the confusion matrices and graphs of each model.

VII. FUTURE DIRECTIONS

Although the proposed models achieve high accuracy in driver-state classification, several promising directions remain for future development and deployment. These enhancements aim to improve real-world applicability, robustness, and scalability.

1) Real-Time Driver Sleepiness Detection
   A key extension of this work is the development of an end-to-end driver drowsiness detection system. Such a system would provide timely alerts to prevent accidents caused by driver drowsiness.
2) Hardware Implementation for In-Vehicle Deployment
   Future work also includes converting the proposed model into a lightweight embedded system suitable for integration within a car's interior. The goal is to develop

a compact, cost-effective device that can be mounted on the dashboard and perform on-device inference without requiring external computation or internet connectivity.

3) Integration with Speeding and Traffic Cameras
Another important direction is the integration of the model into speeding cameras or roadside surveillance systems. This requires training and adapting the model to handle images captured from greater distances and under varying lighting, weather, and camera quality conditions.

REFERENCES

[1] Driver's mobile phone usage detection using guided learning based on attention features and prior knowledge, 2022

[2] Detection of Distracted Drivers using Convolution Neural Network, 2022

[3] Detection of Distracted Driver using Convolutional Neural Network, 2018

Example of conference paper proceedings:

[5] Drive&Act: A Multi-modal Dataset for Fine-grained Driver Behavior Recognition in Autonomous Vehicles, 2019

[6] Ghadiyaram, D., Feiszli, M., Tran, D., Yan, X., Wang, H., & Mahajan, D. 2019. Large-scale weakly-supervised pre-training for video action recognition (pp. 12046–12055)

[7] Carreira, J., & Zisserman, A. (2017). Quo Vadis, action recognition? A new model and the Kinetics dataset (pp. 6299–6308)

[8] Using machine learning to understand driving behavior patterns, 2024.