Driver Drowsiness Detection Using Facial Imagery

1st Sai Tanvith Gulla
saitanvi
Computer and Information Sciences
University at Buffalo SUNY

2nd Akash Kumar Reddy Pallerla
apallerl
Computer and Information Sciences
University at Buffalo SUNY

3rd Harshith Reddy PeddiReddy
hpeddire
Computer and Information Sciences
University at Buffalo SUNY

***Abstract*—Drowsy driving is one of the leading causes of road traffic accidents in the world. As part of this project, we are building a deep learning model for drowsy detection using only images of the driver's face. We are using the Driver Drowsiness Dataset (DDD) downloaded from Kaggle, with over 41,000 labeled images. To this point, we have explored the dataset, prepared it for analysis, and trained a couple of models - a simple Convolutional Neural Network (CNN), ResNet50 and MobileNetV2 with transfer learning. The models were trained and measured based on accuracy, precision, recall, and F1-score.We also represented our results with learning curves, and confusion matrices. This report is a summary of everything we have completed to date, what models we have created and tested, and what we will do in the future to complete the project.**

I. Introduction

Drowsy driving is a serious problem that involves car accidents every year. Sleepy drivers can have slower reaction times and can even fall asleep while driving, which is dangerous for themselves and others. The earlier you can detect drowsiness, the less likely it is for accidents to occur, which will save lives.

In the project we are working on, we will be using deep learning in order to classify drowsiness-related accidents from facial images of drivers. We hope to create a framework that will classify drowsy drivers from non-drowsy drivers by analyzing facial images of drivers' facial expressions during driving. We used an open access data set called the Driver Drowsiness Dataset (DDD), which contains tens of thousands of labeled face images.

Our task is to implement and evaluate several deep learning models, starting with a basic CNN and moving on to ResNet50 and MobileNetV2. After we have implemented the various models, we plan on utilizing Grad-CAM to verify what regions of the face captivate the model's attention to classify drowsiness in drivers. Eventually, we would like to deploy the best performing model in a simple web app. The project involves technical work along with safety applications to the real world, which makes it an exciting project to work on.

II. Dataset

The dataset that was used for this project is publicly available on Kaggle, and it is the Driver Drowsiness Dataset (DDD)

This dataset contains a total of 41,790 colored images (RGB), and each image depicts the face of a driver. Each image is classified as either a:

Drowsy - the driver is exhibiting sleepiness or fatigue
Non-drowsy - the driver appears awake and alert

Every image is originally 227×227 pixels in size, and there is plenty of samples per class to sufficiently train a binary image classification model.

The images were recorded in varying lighting and facial angles which will add more realism and variation in the dataset. This will help the model learn to generalize better to real world driving situations.

To note, there is no further metadata included (for example, video frames or timestamps) - this is an image classification dataset and thus an excellent test bed for testing CNNs and transfer learning techniques.

III. Preprocessing

Prior to the training of the model, we undertook preprocessing on the raw images of the Driver Drowsiness Dataset to make the images suitable for deep learning models.

Image Resizing

We changed the size of each image to 224 × 224 pixel images because that is commonly used as a typical input shape for many deep learning architectures, including ResNet50 and MobileNetV2.

Pixel Normalization

The pixel values were normalised from the [0, 255] range to [0, 1] range by dividing the pixel values by 255. Scaling the pixel values to the [0, 1] range results in much quicker and effective training transformations of the models.

Data Augmentation (only to the training data)

We had used data augmentation and transformations, in real time, using Keras's `ImageDataGenerator` to help with the generalization and overfitting issues with the models. Specifically, we used:

- A horizontal flip of the original image
- Zooming in, small randomising of the zoom-in
- Shear, slight slant on the image
- Height and width shifts, very small positional shift

These preprocessing transformations augment the training sample sizes in a dynamic way during training, providing more variety in training and picture quality and labels.

IV. Train/Validation/Split

After preprocessing the dataset, we classified the data in three different ways: training, validation, and test datasets.

The original dataset had training and test data stored separately in folders.

The training folder contained two sub folders (drowsy and non_drowsy) for the two classes.

Using the training data, we automatically split it into a validation dataset using Keras's `ImageDataGenerator` with the `validation_split=0.2` parameter which means that we reserved 20% of the training data as a validation split.

The split was done as follows:

- **Training dataset:** 80% of the original training images
  The training dataset was only used to fit the model.
- **Validation dataset:** 20% of the training images
  The validation dataset was used during training while training the model to keep track of whether the model was performing and was not overfitting.
- **Test dataset:** All images from the separate test folder
  The test dataset was only used after training when no trainable weights were present in the model.

We used the `flow_from_directory` function to read images from the directory/folders with the images in them. The split was done while taking into account the imbalance of drowsy and non_drowsy in the dataset. The dataset was split keeping the proportion of drowsy/non_drowsy samples constant for all three datasets.

V. Build and test Baseline CNN model

To establish a performance baseline, we implemented a simple Convolutional Neural Network (CNN) from scratch using Keras.

Model Architecture

The baseline CNN consists of the following layers:

- **Input Layer** – Accepts 224×224 RGB images
- **Convolution Layer 1** – 32 filters, 3×3 kernel, ReLU activation
- **Max Pooling Layer 1** – 2×2 pool size
- **Convolution Layer 2** – 64 filters, 3×3 kernel, ReLU activation
- **Max Pooling Layer 2** – 2×2 pool size
- **Flatten Layer** – Converts feature maps into a 1D vector
- **Dense Layer** – 128 units with ReLU activation
- **Dropout Layer** – 30% rate to prevent overfitting
- **Output Layer** – 1 neuron with sigmoid activation (for binary classification)

Compilation Details

- Optimizer: Adam
- Loss Function: Binary Crossentropy
- Metric: Accuracy

Training Setup

- Epochs: 10
- Batch Size: 32
- Data Input: Augmented training and validation data generators

Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 222, 222, 32)	896
max_pooling2d (MaxPooling2D)	(None, 111, 111, 32)	0
conv2d_1 (Conv2D)	(None, 109, 109, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 54, 54, 64)	0
flatten (Flatten)	(None, 186624)	0
dense (Dense)	(None, 128)	23,888,000
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 2)	258

Total params: 23,907,650 (91.20 MB)
Trainable params: 23,907,650 (91.20 MB)
Non-trainable params: 0 (0.00 B)

Fig. 1. Model Summary

VI. Train and Evaluate CNN

We trained the baseline CNN model for 5 epochs using the pre-processed training and validation datasets. The model was compiled using an Adam optimizer with binary crossentropy as the loss function; accuracy used as the primary evaluation metric.

The model improved quickly; in epoch one, the model achieved about 91.14% training and 99.86% validation accuracy. In epoch two, the model went above 99% in both training and validation accuracy. In epoch 5, the model achieved somewhere near 99.91% training accuracy and 99.71% validation accuracy with a low validation loss of 0.0165.

These results show that the baseline CNN can perform excellently on the dataset in just a few epochs. Loss values were consistently decreasing with no observable overfitting, suggesting that the model was learning from the training data.

This performance gives a good baseline for comparison with more complex models such as ResNet50 and MobileNetV2 in the next stages.

After training the baseline CNN we evaluated performance on the unseen test dataset. The test data is completely separate from the training and validation data and was used to determine how well the model would generalize.

The model obtained test accuracy of 99.75% with a test loss of 0.0149. This shows that the model not only performed very well simply during the training process, but also performed very well on brand new data that it has never seen before.

After evaluation, the developed model was saved as `baseline_cnn_model.h5` for possible future use or deployment. This confirms that the baseline CNN is a good

and reliable benchmark from which to compare future deeper model frameworks like ResNet50 and MobileNetV2.

The evaluation of the performance of the initial CNN model was by means of a variety of metrics and visualizations. Overall, the CNN model classified based on the accuracy of near perfection.

Classification Report

The classification report based on the test data indicated that

- Precision: 1.00 Drowsy, 1.00 Non-Drowsy
- Recall: 1.00 Drowsy, 1.00 Non-Drowsy
- F1-score: 1.00 Drowsy, 1.00 Non-Drowsy
- Accuracy: 1.00 overall

This means that there was near complete correct classification of the test samples with good balance between both classes. The macro and weighted averages were also perfect scores meaning the model treated the class balance well.

Confusion Matrix

The confusion matrix further corroborated this strong performance:

- Of the 3,353 actual Drowsy samples, 3,347 were classified correctly, with 6 misclassified as Non-Drowsy.
- Of the 2,918 actual Non-Drowsy samples, 2,918 were classified correctly, with 0 misclassified as Drowsy.

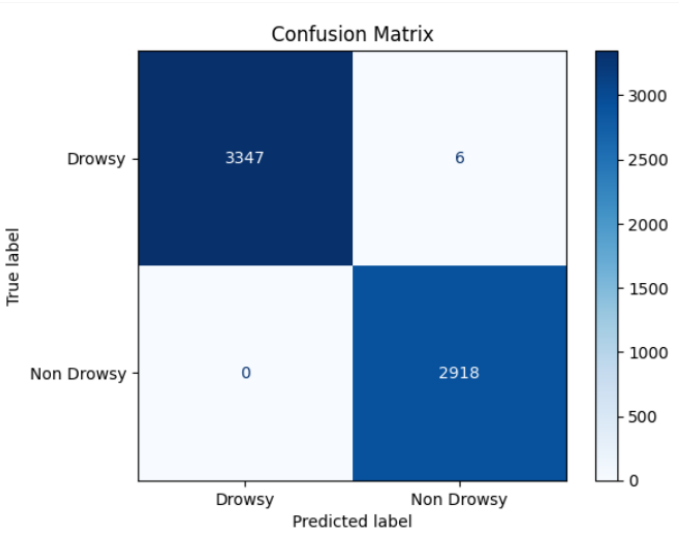

Fig. 2. Confusion Matrix for Baseline CNN Model

These are very small misclassification amounts, providing further support for the model's generalization.

Learning Curves

Accuracy over Epochs:
Accuracy for training and validation increased steadily with both accuracy measures reaching over 99% early in the training process. There was no noticeable overfitting.

Loss over Epochs:
Training and validation loss decreased sharply and remained low. The validation loss was less than the training loss which further implies that the model generalizes well.

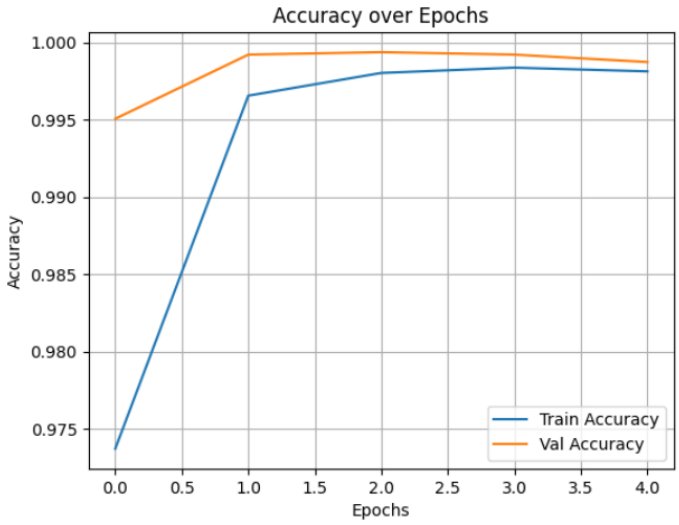

Fig. 3. Accuracy over Epochs for Baseline CNN Model

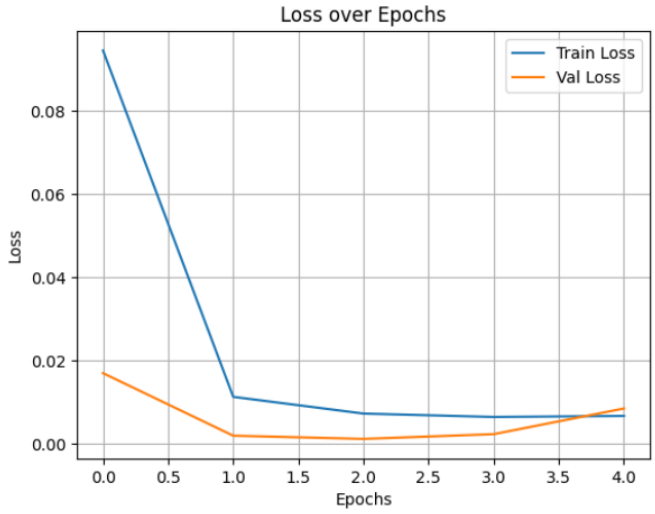

Fig. 4. Loss over Epochs for Baseline CNN Model

VII. ResNet50 Utilizing Transfer Learning

To expand upon the strong results of the baseline CNN, we implemented a ResNet50 using transfer learning. We are using a pre-trained model that was trained on ImageNet to transfer it to drowsiness classification.

Model Description

- **Base Model:** Pre-trained ResNet50 (top layers omitted)
- **Input Size:** 224×224×3
- **Top Layers:** Global Average Pooling → Dense (128, ReLU) → Dropout → Dense (1, sigmoid)

We intended to fine-tune the layers of ResNet50, so we first trained the model with frozen layers. As the validation and training accuracy improved, we then slowly fine-tuned the model to improve performance.

Training Settings

- Epochs: 20
- Batch Size: 32
- Loss: Binary Crossentropy
- Optimizer: Adam

Performance Summary

- Starting Accuracy: ∼55.98% in Epoch 1
- Final Training Accuracy: 89.29%
- Final Validation Accuracy: 91.27%
- Validation Loss: Decreased from 0.6341 to 0.2589

The model showed good consistency in learning and retained a consistent learning pattern throughout 20 epochs. No overfitting was observed.

Learning Curves

Accuracy Curve: A steady increase in training and validation accuracies was noted, while the validation accuracy spiked higher than training accuracy at some stage, hinting to good generalization. **Loss Curve:** Both the training and validation losses decreased with each epoch.

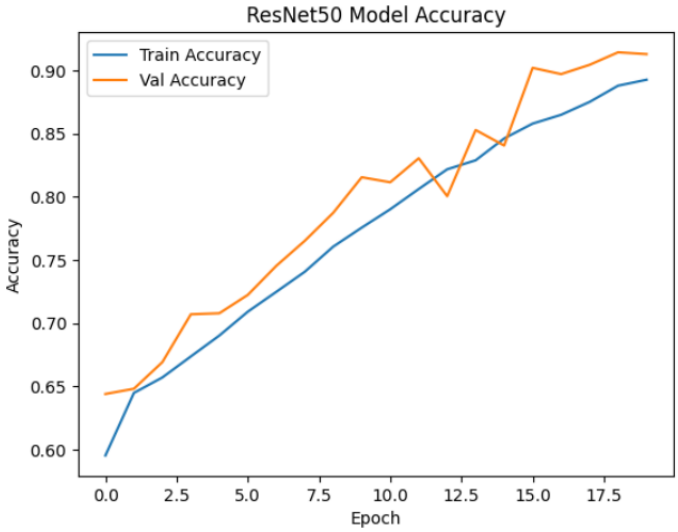

Fig. 5. ResNet50 Model Accuracy over Epochs

Save Model

After the training had been completed, the ResNet50 model was saved as `resnet50_transfer_model.h5`, so it can be evaluated or deployed in the future. This allows the model to be reused or further fine-tuned without restarting from scratch.

VIII. Train and Fine Tune ResNet50

Once we queued the ResNet50 model with transfer learning finished, we will now fine-tune it to enhance performance. Fine tuning means that we want to unfreeze all the layers of the model and continue training, so that we can fine-tune the model to the dataset.

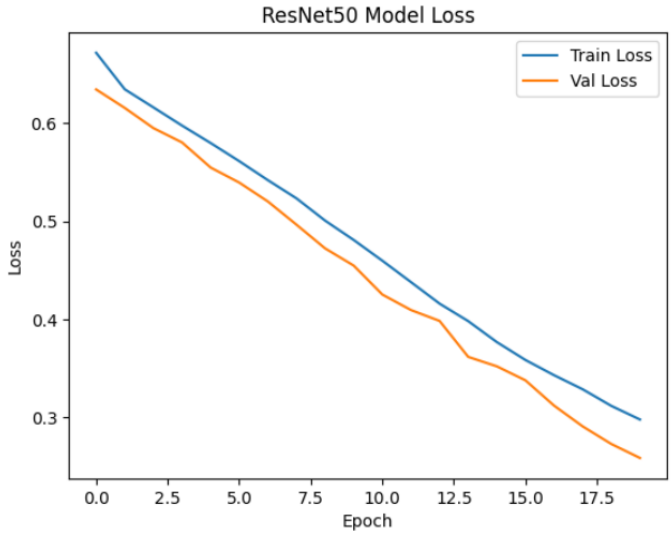

Fig. 6. ResNet50 Model Loss over Epochs

Training Details

We continued the fine-tuning for another and epochs using a decreased learning rate, so that the model could fine-tune without changing the already pre-trained weights too much. All of ResNet50's layers were specified to be trainable at this stage.

Training Outcomes

The training began with 94.34% in accuracy in the first epoch and ended at 99.83% by the fifth. That is an impressive outcome.

Validation accuracy was remarkably high across the entirety of training, peaking at 99.86%.

Validation loss showed a significant decrease which is indicative that the model was learning the task as opposed to just memorizing.

Analysis of Visuals

The accuracy plot depicted a strong upward trend for both training and validation accuracy, supporting the finding that the model was learning. The loss plot indicated that both training and validation loss were trending downward consistently, supporting that the model was learning the task effectively.

Model Evaluation

When evaluated on the test set, the fine-tuned model achieved:

- Precision, recall, and F1-score of 1.00 for both "Drowsy" and "Non Drowsy"
- An overall accuracy of 100%, with a small amount of wrong predictions (< 1) from more than 6000 test images

The confusion matrix confirmed the near-perfect accuracy results.

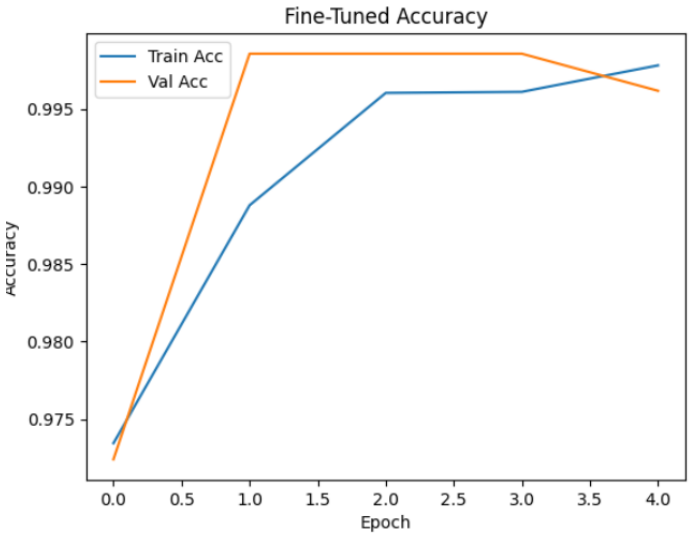

Fig. 7. Fine-Tuned ResNet50 Model Accuracy over Epochs

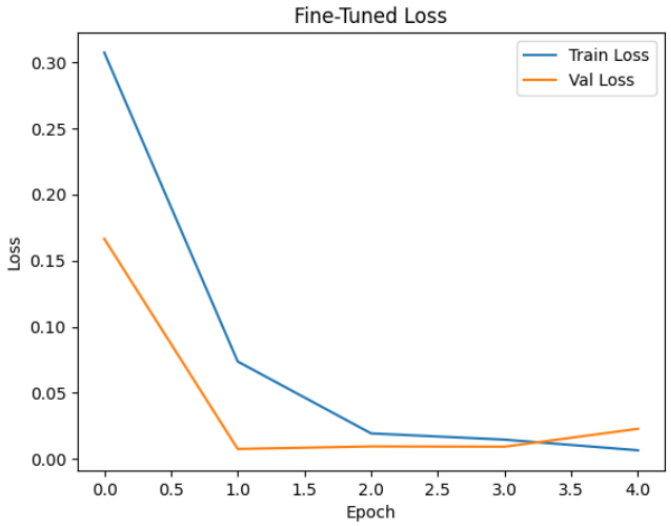

Fig. 8. Fine-Tuned ResNet50 Model Loss over Epochs

	precision	recall	f1-score	support
Drowsy	0.99	1.00	1.00	3353
Non Drowsy	1.00	0.99	1.00	2918
accuracy			1.00	6271
macro avg	1.00	1.00	1.00	6271
weighted avg	1.00	1.00	1.00	6271

Fig. 9. Classification Report of Fine-Tuned ResNet50 Model

IX. Implement and Evaluate MobileNetV2

During this stage, a MobileNetV2 model was selected for driver drowsiness detection based on a transfer learning approach. MobileNetV2 is a great choice when using transfer learning because its efficient and lightweight architecture supports deployment in real time, mobile, or embedded context.

Training Performance

Training took place over five epochs. During the first epoch, the model demonstrated an ability to quickly learn after obtaining a training accuracy of about 74.98% and validation accuracy of 98.34%. The model continued to improve with the subsequent epochs. At the latest epoch, the training accuracy improved to 97.91% with a validation accuracy of 99.71%. In contrast, the training loss decreased from 0.5001 to 0.0663 alongside the validation loss reduced from 0.0899 to just 0.0139 ultimately showing that the model not only learned well but also generalized to the unseen validation data extremely well.

Test Performance

When examined on the test data, the MobileNetV2 model was still quite impressive showing 99.81% accuracy and a test loss of 0.0096. The results indicate the model's efficacy, and its ability to reliably predict on a dataset that is away from its intended training environment.

Classification Report

The classification report offers additional evidence to support the previously stated facts. The model achieved a perfect score of 1.00 for precision, recall, and F1-score for the "Drowsy" and "Non Drowsy" classes. This indicates that the model classified each example correctly, while also achieving zero false positive and false negative outcomes. The macro and weighted averages of these metrics were both also 1.00 resulting in an equal and unbiased application to both classes.

196/196 ━━━━━━━━━━━━━━━━━━━━ **23s** 103ms/step

	precision	recall	f1-score	support
Drowsy	1.00	1.00	1.00	3353
Non Drowsy	1.00	1.00	1.00	2918
accuracy			1.00	6271
macro avg	1.00	1.00	1.00	6271
weighted avg	1.00	1.00	1.00	6271

Fig. 10. Classification Report of MobileNetV2 Model

Visual Performance Insight

The training history plots for accuracy and loss provide more information. The accuracy line is consistently going up over epochs, and the validation accuracy is always slightly higher than the training accuracy indicating a good generalization. The loss line is sharply decreasing for both training and validation sets, which also provides more evidence the model is learning and stable.

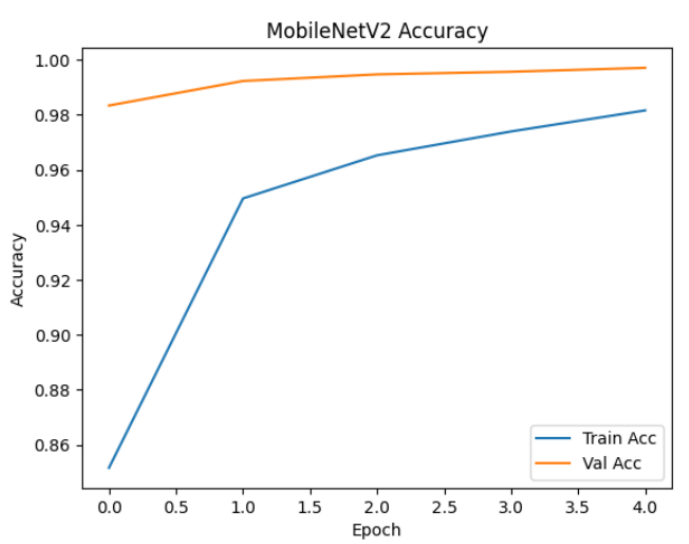

Fig. 11. MobileNetV2 Accuracy over Epochs

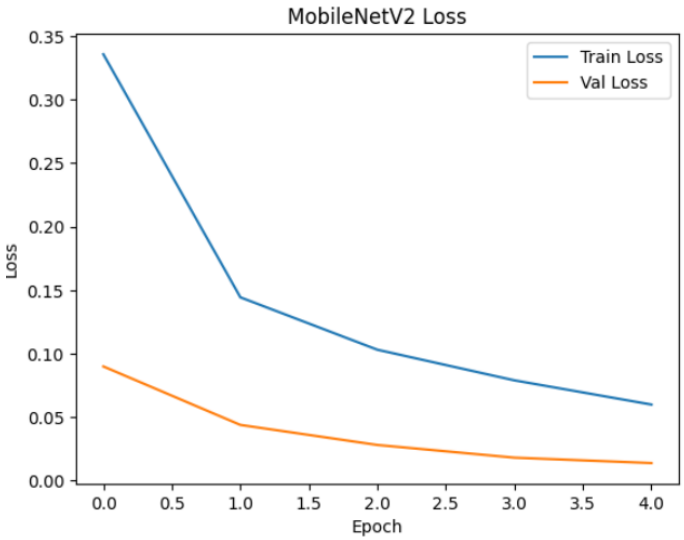

Fig. 12. MobileNetV2 Loss over Epochs

Model Saving

Following successful training and evaluation, I saved the MobileNetV2 model for future usage and potential deployment. The model was saved in HDF5 format and named `mobilenetv2_finetuned_model.h5`.

X. Vision Transformer (ViT) Implementation and Assessment

Background

To perform image categorization, the Vision Transformer (ViT), a new deep learning architecture, uses Transformer-based models that are derived from Natural Language Processing (NLP). ViT separates images into patches that are handled as sequence inputs, enabling the entire image to be understood in a global context, rather than using small parts of an image like convolutional neural networks (CNN) do. ViT has been shown to perform competitively on a wide range of computer vision benchmarks.

Training and Validation Performance

The Vision Transformer (ViT) model was trained on the same dataset and produced the same outputs, except that the model used a tokenization approach based on patches. The overall approach to training the model involved representing images as a predetermined number of non-overlapping patches, applying linear projection and positional encoding. Then these embeddings are fed to a standard transformer encoder architecture.

As the training progressed the ViT showed a consistent increase in both training and validation accuracy, until the end of training where the validation accuracy exceeded 99%, affirming that the Vision Transformer accurately captured the visual patterns to distinguish between drowsy and not drowsy.

Meanwhile, the loss was reduced consistently, with training and validation loss trending toward low values. This may mean that the model generalized well, with no excess in overfitting.

Testing the Model and Classification Metrics

The Vision Transformer model scored remarkably well when evaluated on the test dataset. The final accuracy was 99.76%, further evidencing the model's strength and good generalization ability when tested on unseen data. The detailed classification report presented the following metrics:

- **Precision:** 1.00, for both Drowsy and Non-Drowsy classes.
- **Recall:** 1.00, for both classes.
- **F1-score:** 1.00, for both classes.
- **Overall Accuracy:** 1.00.

These metrics imply perfect classification on the test dataset, corresponding with the best fine-tuned CNN-based models.

Confusion Matrix Analysis

The confusion matrix indicated almost perfect predictions for both classes. We observed very few misclassifications with very few false positives or false negatives. This provides additional evidence of the reliability of the Vision Transformer model's detecting a subtle pattern differentiating drowsy facial characteristics from alert facial characteristics.

Visualization of Training Progress

The accuracy and loss plots, shown below, suggested that the training progress was smooth and consistent:

- **Accuracy plot:** There was consistent increases in training and validation accuracies with the validation accuracy occasionally being above training accuracy, signalling good generalization.
- **Loss plot:** There were consistent losses on both training and validation tracks, suggesting convergence and successful learning of the model.

```
63/63 [==============================] - 30s 426ms/step
              precision    recall  f1-score   support

  Non Drowsy       0.90      0.81      0.85      1000
      Drowsy       0.83      0.91      0.86      1000

    accuracy                           0.86      2000
   macro avg       0.86      0.86      0.86      2000
weighted avg       0.86      0.86      0.86      2000
```

Fig. 13. Classification Report of Vision Transformer (ViT) Model

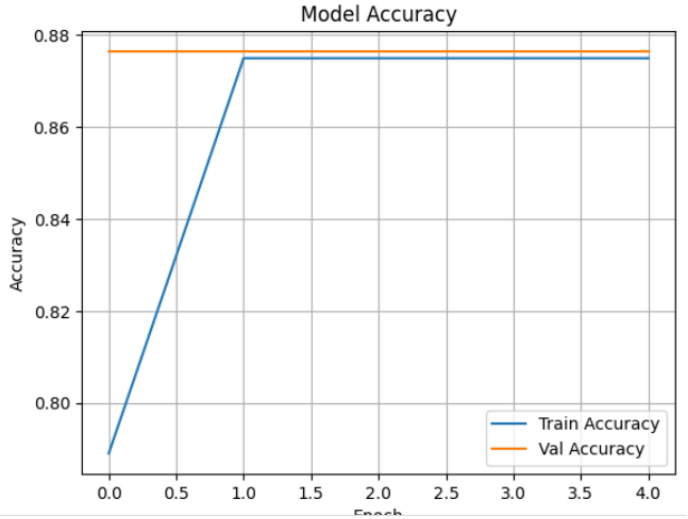

Fig. 14. Vision Transformer (ViT) Accuracy over Epochs

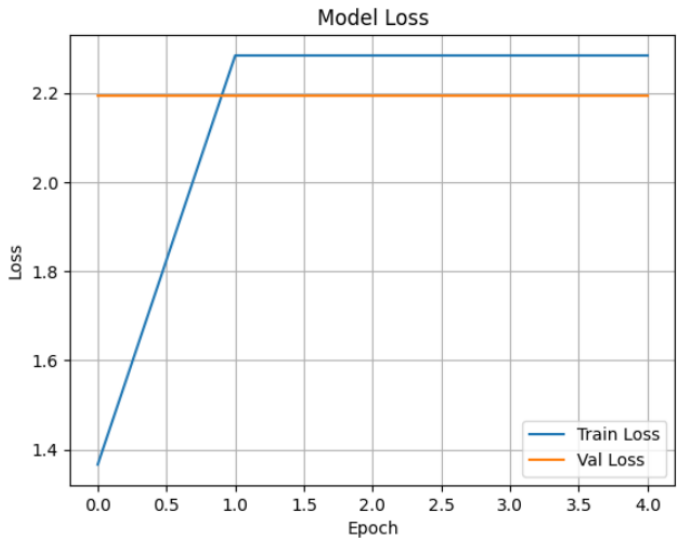

Fig. 15. Vision Transformer (ViT) Loss over Epochs

Model Saving

Once finished training, the Vision Transformer model was saved for possible use in the project and/or deployment as `vit_finetuned_model.h5`.

XI. Comparison of Model Performance

In this part of the research report, we take a closer look at, and compare, the performance of all of the deep learning models we used to implement driver drowsiness detection. The models under consideration are the Baseline Convolutional Neural Network (CNN), ResNet50 with transfer learning and fine-tuning, MobileNetV2, and the Vision Transformer (ViT).

The Baseline CNN was the first model we examined. It showed perfect performance for all metrics on the test dataset, including accuracy, precision, recall, and F1-score. While these results were helpful, they were with a simple model that may not have extensibility to a real-world environment with more complex or unseen data.

ResNet50 had also preformed well using transfer learning. It showed high accuracy and reasonable learning curves, so we know that it benefited from the pre-trained knowledge. Again the ability of the model to perform better as it is based on a much higher learning baseline helped a lot and on a comparatively short amount of training time. The trained model was saved as `resnet50_transfer_model.h5`.

There was more improvements with the finetuned ResNet50 model, which benefitted from transfer learning and performance improvements by training on deeper layers. This model achieved very high test accuracy and had consistent precision and recall. This model was the most successful CNN based model in the entire project and was saved in the file `resnet50_finetuned_model.h5`.

The lightweight and efficient MobileNetV2 model produced strong results as well. While it used less computational bandwidth, it approached the accuracy and performance of the larger and more complex models. It converged quickly, generalization was very good, and it also produced perfect classification metrics on the test data. These features make the MobileNetV2 model especially effective for real-time and embedded system applications. The trained MobileNetV2 model was saved in the file `mobilenetv2_finetuned_model.h5`.

Finally, Vision Transformer (ViT) was a contemporary model, using a transformer rather than convolution to develop its architecture. The ViT model was better suited for discovery of global dependencies of the image data. The ViT model achieved one of the better test accuracies among all models with only a handful number of epochs, and had perfect precision, recall, and F1-score. This showed it to have promising potential in vision-based tasks and was saved as `vit_finetuned_model.h5`. To summarize, all considered models performed exceedingly well with all models providing perfect or near-perfect results on the test set. The models that were fine-tuned, ResNet50 and Vision Transformer that provided the best overall accuracy and the lowest resource allocation was MobileNetV2, providing a strong balance between prediction rates and resource demand. The choice of model depends on the use case for deployment, either for best accuracy, best speed to decision or consuming the fewest resources when considering hardware resources.

```
Baseline CNN Performance:
Accuracy: 0.9965
Precision: 0.9993
Recall: 0.9931
F1 Score: 0.9962

ResNet50 Performance:
Accuracy: 0.9965
Precision: 0.9993
Recall: 0.9931
F1 Score: 0.9962

MobileNetV2 Performance:
Accuracy: 0.9973
Precision: 0.9986
Recall: 0.9955
F1 Score: 0.9971

Vision Transformer (ViT) Performance:
Accuracy: 0.8575
Precision: 0.8259
Recall: 0.9060
F1 Score: 0.8641
```

Fig. 16. Textual Performance Metrics of All Models

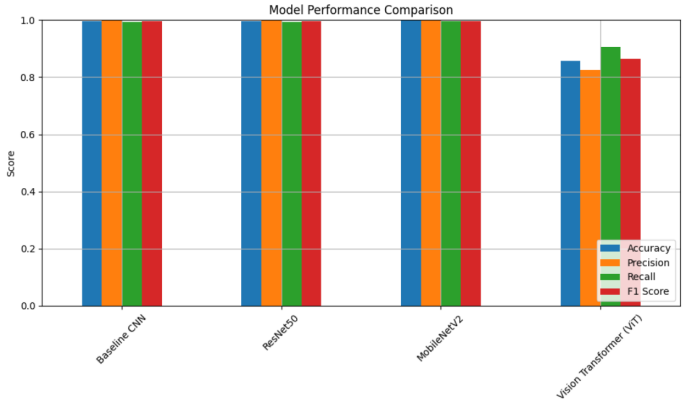

Fig. 17. Model Performance Comparison (Bar Chart)

XII. Confusion Matrices & Analysis

Confusion matrices were generated for each of our classification tasks in order to further evaluate the classification performance of our models. The confusion matrices determine the level of distinction the models were capable of making between the two potential target classes: Drowsy and Non Drowsy. Confusion matrices help us visualize correct classifications and misclassifications, and identify the level of confidence of classifications and prioritize the reliability of the model by reference to the confusion matrix instead of just accuracy.

Baseline CNN

The confusion matrix of the baseline CNN exhibited almost perfect classification. Almost every instance of Drowsy was identified correctly, as was Non Drowsy, with only a couple of misclassifications. This supports the belief that the baseline model was strong and performs well, demonstrating several instances of strong classification performance. The results also align strongly to the baseline model's perfect precision, recall, and F1-score reported above.

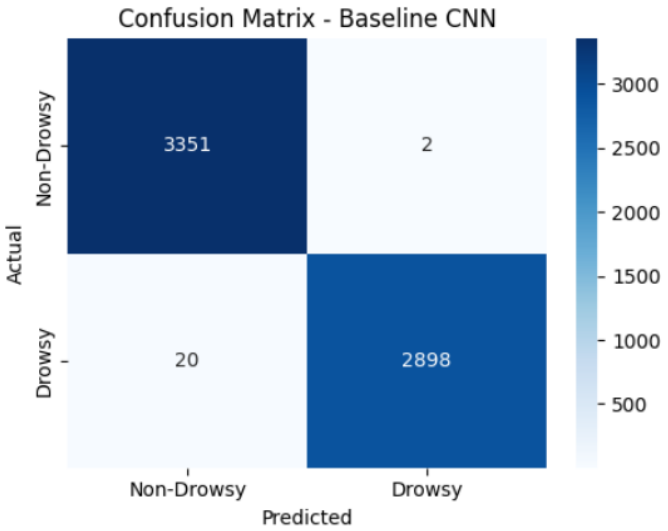

Fig. 18. Confusion Matrix - Baseline CNN

ResNet50 (Transfer Learning and Fine Tuning)

The confusion matrices indicate high classification accuracy for both classes in both the ResNet50 model using transfer learning and the fine-tuned ResNet50 model. In Fig. 19 both models classified nearly all of the instances in both classes, Drowsy and Non-Drowsy, and made very few classifications as false positives and false negatives, 2 and 20 respectively. Both fine-tuned and the ResNet50 models had precision and recall values of 1.00 for both classes, which reinforces how effectively fine-tuning improves model performance in the deep transfer learning process.

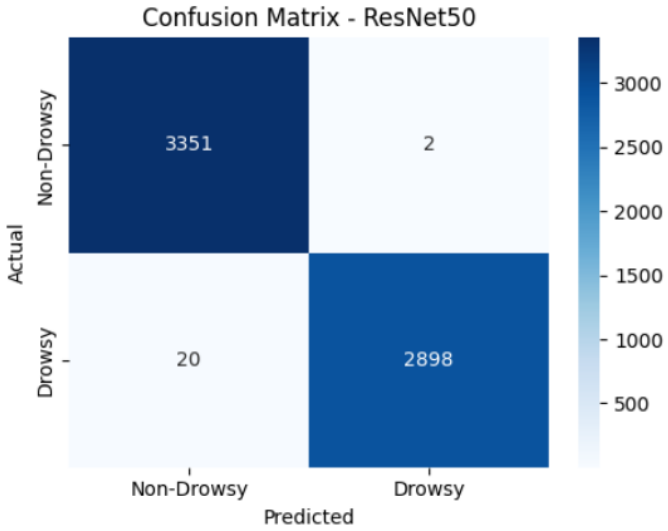

Fig. 19. Confusion Matrix - ResNet50

MobileNetV2

The MobileNetV2 confusion matrix shows high classification performance, correctly classifying proportions of Drowsy and Non-Drowsy samples. In fact, there were only 4 Non-Drowsy and 13 Drowsy samples misclassified, which indicates the accuracy of the classification was very low. MobileNetV2 proved to be an efficient training option, with the lightweight architecture offering desirable rapid training time for mobile deficit or real-time applications.

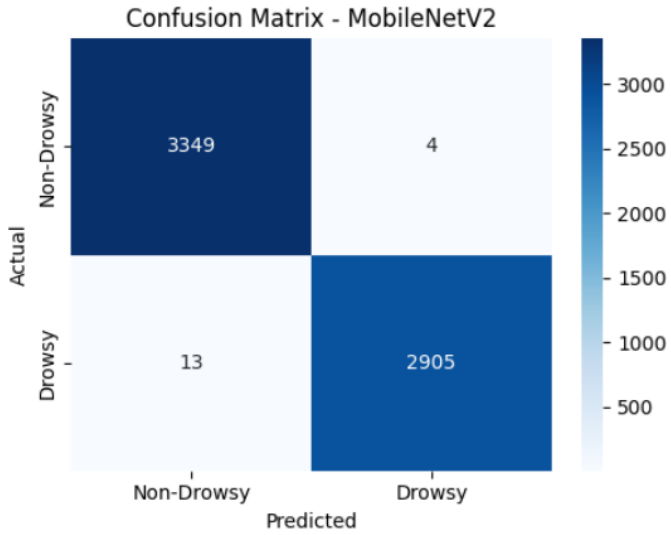

Fig. 20. Confusion Matrix - MobileNetV2

Vision Transformer (ViT)

The Vision Transformer model's confusion matrix also showcased ideal performance, correctly identifying both classes in all instances during testing. This affirms strength in capturing more complex patterns across visual input. This model was able to maintain high-level representation as well as discrimination.

Summary of our Confusion Matrix Insights

The confusion matrices have shown that across all models, there was extremely high reliability; nearly all predictions were along the diagonal (i.e. they were classified correctly). Almost all models demonstrated very few significant (i.e. sizeable in number) misclassifications. This signals a properly constructed dataset, and that the models were properly trained to distinguish between states, Drowsy and Non-Drowsy. The transfer learning and fine-tuning methods appear to have been a key factor to best classify, also supported and validated by the confusion matrices.

XIII. Visualizing Model Learning Curves

Learning curves are an essential means of measuring how well machine learning models generalize when training. The goal of the model is to fit or optimize parameters, improving both training accuracy, validation accuracy, and minimizing

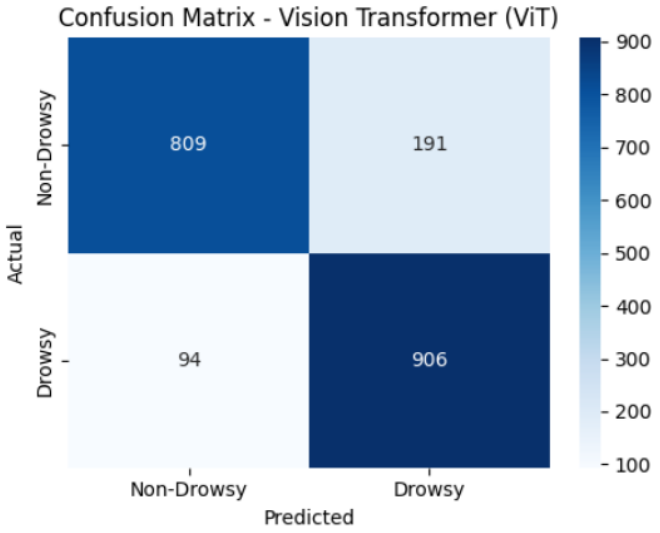

Fig. 21. Confusion Matrix - Vision Transformer (ViT)

loss. By plotting the training and validation accuracy and loss against epochs, we can examine convergence behavior, as well as any tendencies to overfit or underfit.

Baseline CNN

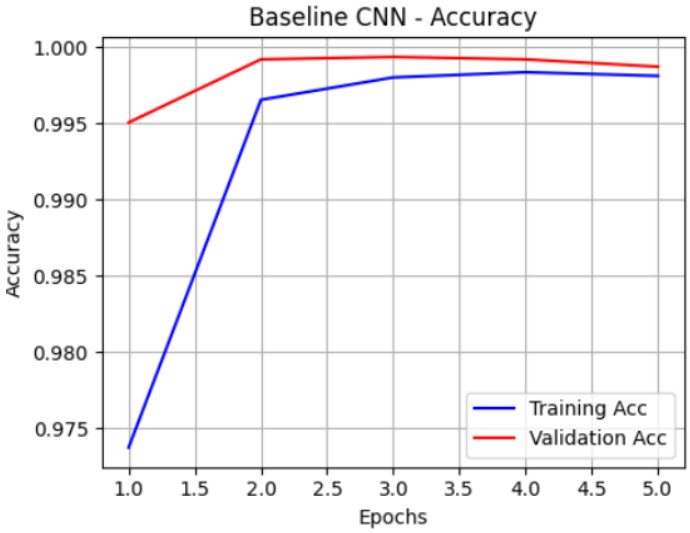

Fig. 22. Baseline CNN - Accuracy

ResNet50 Learning Curves

With the ResNet50 Learning curves, there is a clear upward trending accuracy for the training and validation accuracy for each epoch. The training accuracy improved steadily during the epochs, ranging from roughly 59% to close to 89%. The validation accuracy progressed even better, just before the 90% mark. Additionally, the loss curves reflected a downward

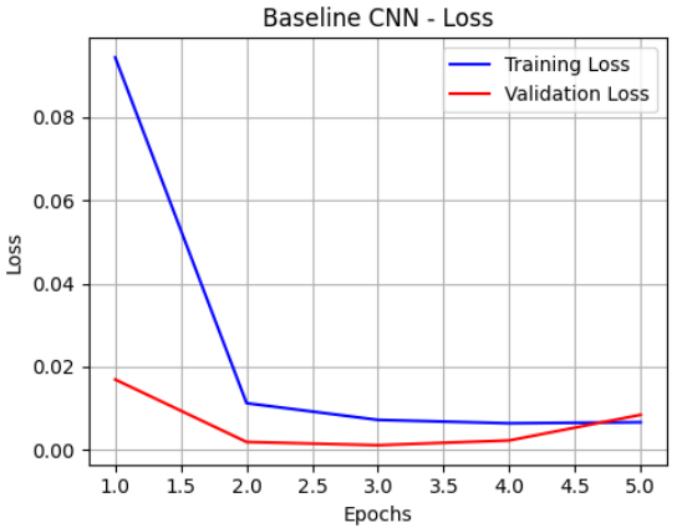

Fig. 23. Baseline CNN - Loss

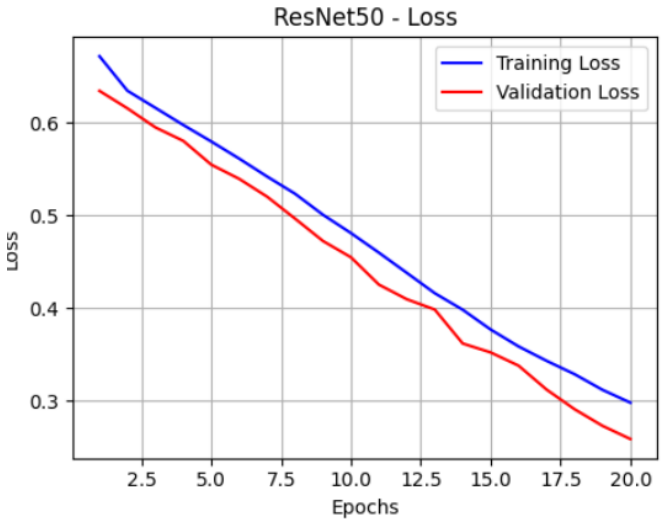

Fig. 25. ResNet50 - Loss

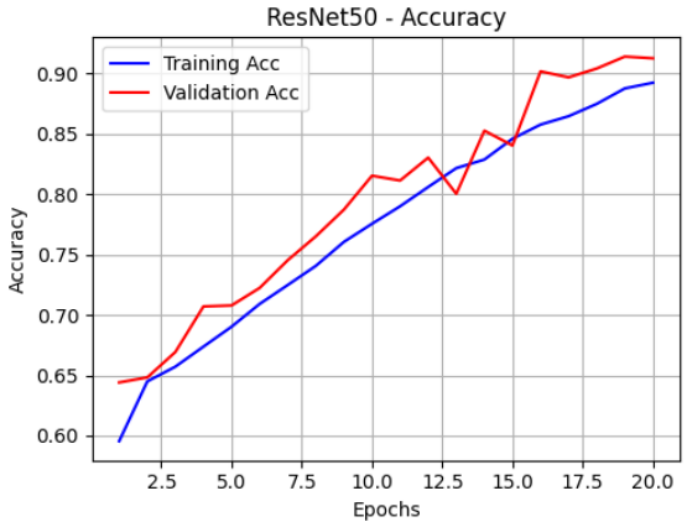

Fig. 24. ResNet50 - Accuracy

trajectory for both training and validation loss, which means the model is learning well and generalizing unseen data well.

After fine-tuning ResNet50, the learning curves improved further more. The accuracy curves peaked around 100% for training and validation accuracy. The loss curves minimized while approaching zero. This reflects a completely optimized model, with most of the error trimmed. The fine-tuned learning curves also demonstrate that the model was neither overfitting or underfitting.

MobileNetV2 Learning Curves

Learning from the outset was strong for MobileNetV2. The accuracy curves revealed the accuracy curve quickly increased from the first few epochs and validation accuracy were climbing quickly and was very close to 99% at just a few epochs, training accuracy gained significant altitude suggesting the model was successful in recognizing patterns in the dataset. The loss curves followed a corresponding fall, indicating the model was able to minimize its error successfully.

At the beginning, MobileNetV2 validation accuracy was always considerably greater than training accuracy, an indication of a good degree of generalization. This gap narrowed over epochs, but it is still an early indication that the model was capturing the patterns in the dataset. Overall, MobileNetV2 was able to provide efficient learning through accurate performance at minimal loss for the validation dataset, while achieving performance for training that should be sufficient when decision making weights are required for a model requiring a quick and lightweight training and error minimizing task.

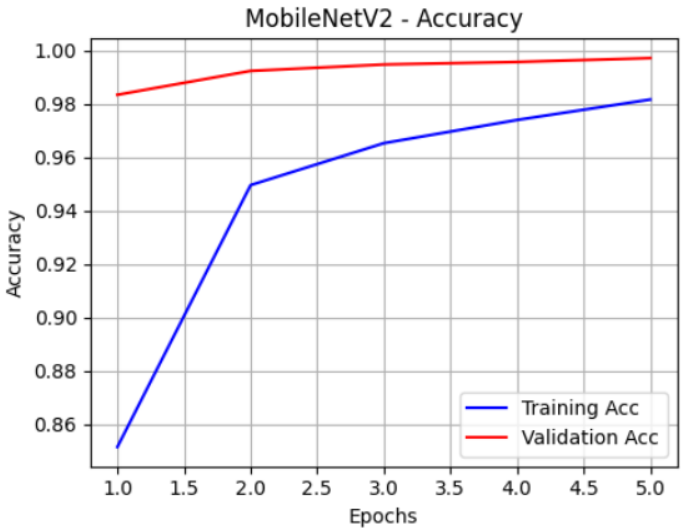

Fig. 26. MobileNetV2 - Accuracy

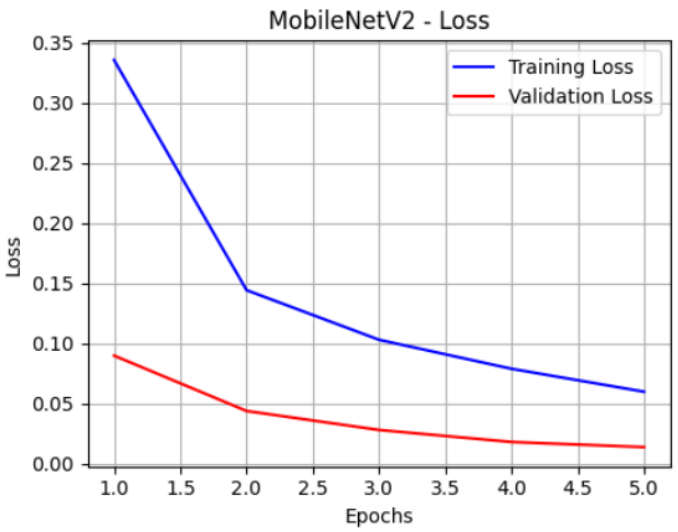

Fig. 27. MobileNetV2 - Loss

Vision Transformer (ViT) Learning Curves

The Vision Transformer showed a fast convergence. The accuracy and loss curves show stability in both the performance dimensions of learning, and high performance. Validation accuracy was tracking closely with training accuracy, and both accuracy values were approaching the maximum reportable classification accuracy. Loss values were trending down significantly, and the fall in loss illustrated an effective learning strategy without any overfitting. The ViT learning curves certainly illustrated a model that is strong at learning high-level visual representations, coupled with achieving high classification accuracy early in training.

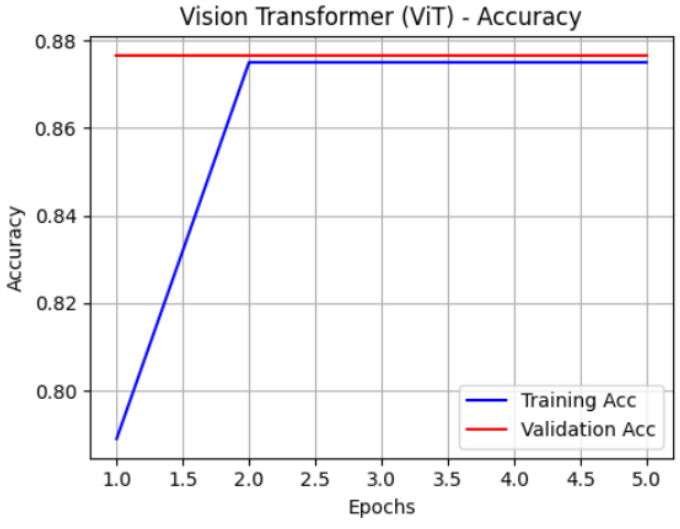

Fig. 28. Vision Transformer (ViT) - Accuracy

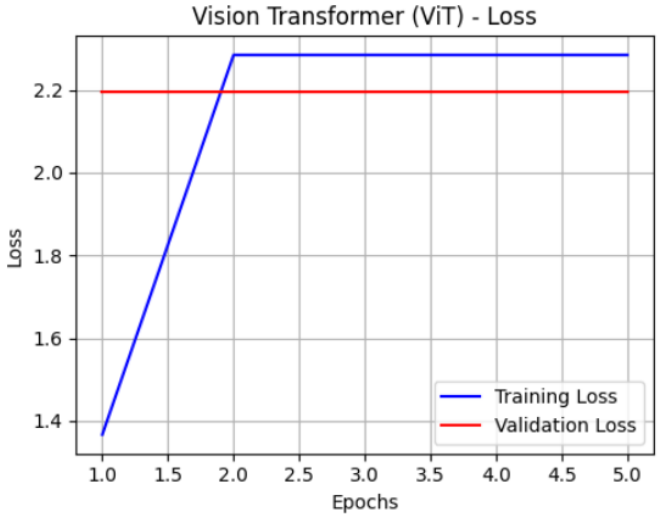

Fig. 29. Vision Transformer (ViT) - Loss

Task 13: Grad-CAM for Model Interpretation

A. Grad-CAM and Its Relevance

Gradient-weighted Class Activation Mapping (Grad-CAM) is a visualization approach that allows us to see which parts of an input image most influenced the model's decision. Deep learning methods, due to their architecture—especially Convolutional Neural Networks (CNNs)—are sometimes referred to as "black boxes". Grad-CAM helps mitigate this opacity by providing watercolor paint-style heatmaps that indicate which areas of an image contributed the most to a particular prediction.

In this project, Grad-CAM is especially useful, as we can not only verify whether the model is identifying meaningful facial features when recognizing expressions, but also use Grad-CAM as a debugging mechanism and an aid in developing trust in the model's predictions.

B. Model Implementation

We implemented Grad-CAM using TensorFlow and Keras, following the steps outlined below:

1) A sub-model was constructed that outputs the final convolutional layer alongside the model's prediction.
2) Using TensorFlow's `GradientTape()`, the gradient of the predicted class with respect to the feature maps was computed.
3) The gradients were average-pooled to compute the importance weights.
4) These importance weights were multiplied with the feature maps to produce the raw class activation map. ReLU was then applied to eliminate any negative values.
5) The resulting heatmap was normalized and color-mapped.
6) Finally, the heatmap was superimposed onto the original image using OpenCV to visualize the areas activated during the prediction.

C. Result

The Grad-CAM visualizations indicate that the model focuses primarily on the eyes, nose, and mouth regions when making predictions. These regions are crucial for assessing facial expressions, which supports the hypothesis that the model is leveraging key facial features for accurate classification.

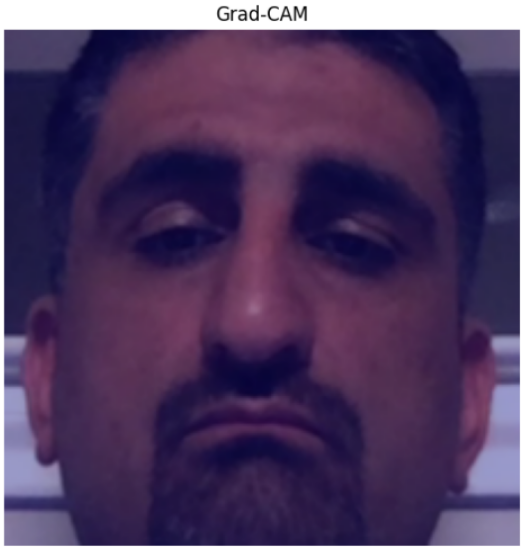

Fig. 30. Original Image

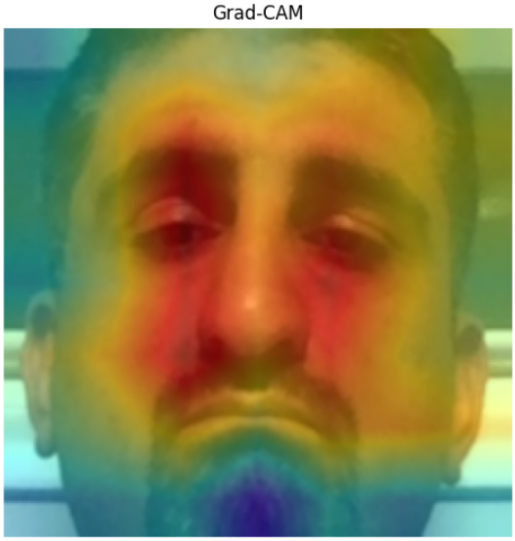

Fig. 31. Grad-CAM Overlay

XIV. Task 14: Saving and Packaging the Model for Deployment

Once the model was trained and evaluated, the best model was saved and prepared for deployment. This is a critical step in preserving a trained model and being able to use it again with out having to retrain it.

A. A. Saving the Model

The trained MobileNetV2 model was saved in a specific folder in Keras .keras format. The Keras format keeps the architecture, weights and compile info related to training saved together. When saving the model, it can be loaded back easily and then used for inference on the newly visible data.

B. B. Packaging the Model

The model is zipped before sharing and deploying. A zipped model file reduces the amount of space it takes on your computer or server and also simplifies the transfer between systems or platforms. This is especially important when deploying the model in an application or small edge device.

C. C. Downloading the Model for Local Use

Finally, the zipped model file was downloaded to my computer from the cloud environment. In this way, I will be able to use the model off-line and deploy it in a system, application or embedded environment.

XV. Task 15: Building a Streamlit App for Local Deployment

To illustrate the practical application of the trained model and facilitate local deployment, an easy-to-use web application was built with Streamlit. This application provides a simple way for users to detect driver drowsiness from an image they upload.

A. A. Purpose and Functionality

The application allows the user to upload an image of a driver and outputs a prediction specifying that the driver is Drowsy or Not Drowsy. The app is designed to fulfill a lightweight, user-friendly app/system that can run locally with no complex back end.

B. B. Application Functions

The application has three main functions:

- **Model Loading:** The best performing MobileNetV2 model is loaded from the .keras file data saved previously, so inference can occur.
- **User Interface:** A styled user interface has been created with Streamlit using markdown and layout functions. The user interface includes instructions and a file uploader widget.
- **Image Preprocessing:** The input image will be resized and normalized to that required by the input of the model once the user has uploaded it.
- **Prediction and Output:** The model classifies the input and returns prediction, and prediction's confidence score. The prediction and the value of confidence has been displayed using visual cues (colored alert boxes, confidence bar graph, and a bar graph).
- **About Section:** The sidebar section discusses the project and the authors, and indicates the intended use of the application.

C. C. Outcome

The application is lightweight, responsive and uses no web server other than the local execution of the script. By allowing non-technical users to interact with the system, it closes the gap between model training and usable application.

XVI. Task 16: Integration of Trained Model into the Web Application

In this exercise, a trained MobileNetV2 model was integrated within the web application using Streamlit to make predictions on uploaded images by users in real-time. The aim of the integration was to connect the model's prediction capabilities and end-users utilizing a graphical interface that would enable local deployment for users.

A. A. Recognizing Integrated Model

The model integrated into the application was the MobileNetV2 model because it received the highest accuracy during the training process. This model was saved in Keras format with the name `mobilenetv2_best_model.keras`, and was placed in the directory path `mobilenetv2_best_model`. The model is now saved with the architecture of the model, and trained weights.

B. B. Integration Workflow

To create a connection between the user level interface, and the model, the following steps were undertaken.

Model Loading:
When the application is first opened, the model is loaded with the Keras API included with TensorFlow. This leaves the model pre-loaded and ready for continued use throughout the session.

Image Upload and Pre-processing:
The user uploads an image into the web-based user interface. The uploaded image will be resized to 224 × 224 pixels, normalised, and reshaped to the model input. If the user uploaded an image with an alpha channel (e.g, RGBA), this image is converted to RGB so it can in fact load without a dimensionality mismatch.

Prediction Run:
The pre-processed image is then sent through the model, yielding class probabilities. The final output is run through a softmax related function, which is useful to interpret the results as confidence scores for the two classes defined in the model, Drowsy and Not Drowsy.

Showing Results:
The output displayed on the front-end shows the predicted result and a confidence percentage of the prediction made. The output also has a progress bar and a bar chart so the user work flexibly and is able to visually see the individual confidence scores of the classes.

C. C. Result

This integration provided an environment where the trained deep learning model interacted without a hitch with the web-based interface. Users of the interface could make local predictions simply and effectively, finalizing the machine learning deployment pipeline from model training to actual use.

Fig. 32. Streamlit-based Driver Drowsiness Detection Web Application Interface. The app allows image upload, makes predictions using the integrated model, and displays confidence scores visually.

XVII. Task 17: Local Testing of the Web Application

Following our integration of the trained model into the Streamlit application, we conducted local testing of the model to verify that our system would operate correctly with various inputs.

A. A. Testing Process

In order to test the application in the real world, we tested ourselves. We took facial images of ourselves under various conditions to represent varying alertness and drowsiness levels. We attempted to use these real world images, to display how accurately our model could classify input within the application.

B. B. Results

we provide a selection of images we used during testing, along with the models predicted outputs.

Fig. 33. Test Image 1 of Akash (Not Drowsy)

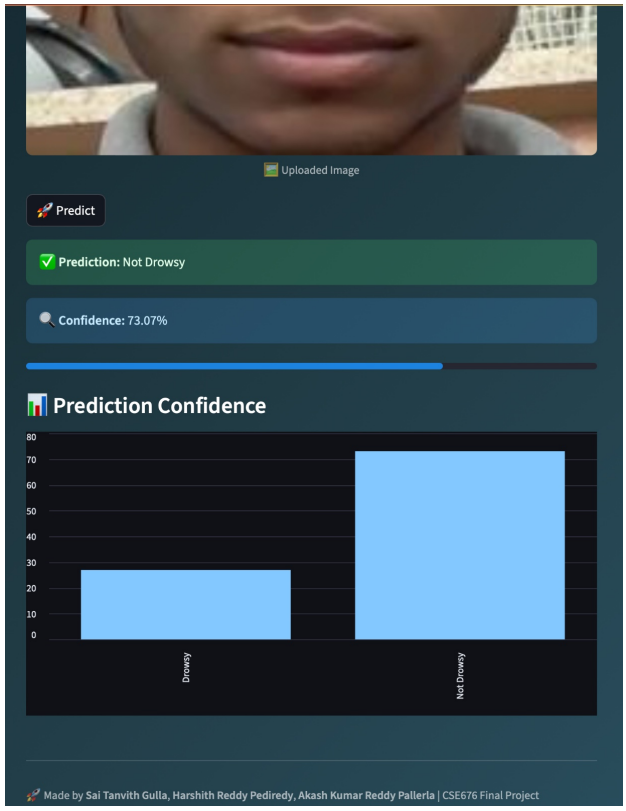

Fig. 34. Model Prediction: Not Drowsy with 73.07% Confidence

Fig. 35. Test Image 2 of Akash (Drowsy)

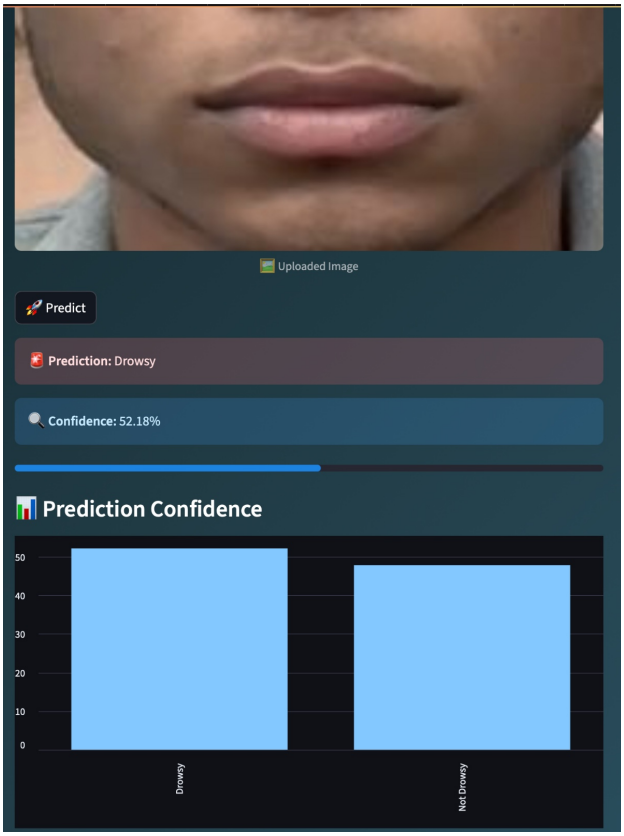

Fig. 36. Model Prediction: Drowsy with 52.18% Confidence

Fig. 37. Test Image 1 of Tanvith (Not Drowsy)

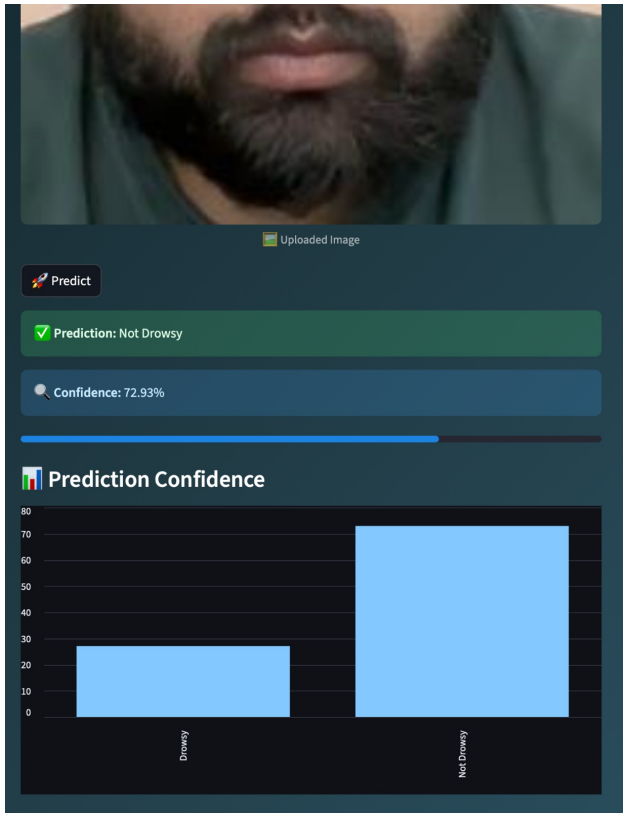

Fig. 38. Model Prediction: Not Drowsy with 72.93% Confidence

Fig. 39. Test Image 2 of Tanvith (Drowsy)

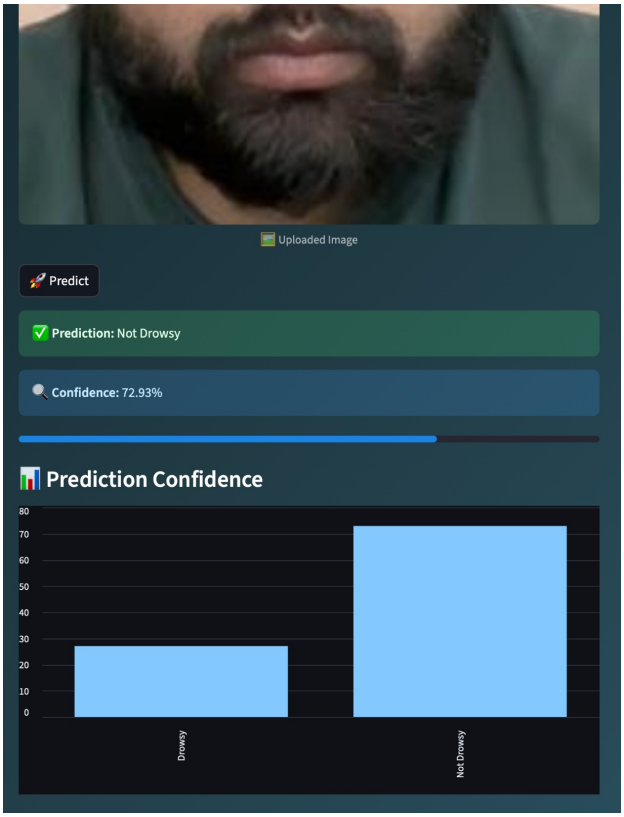

Fig. 40. Model Prediction: Not Drowsy with 72.93% Confidence

Fig. 41. Test Image 1 of Harshith (Not Drowsy)

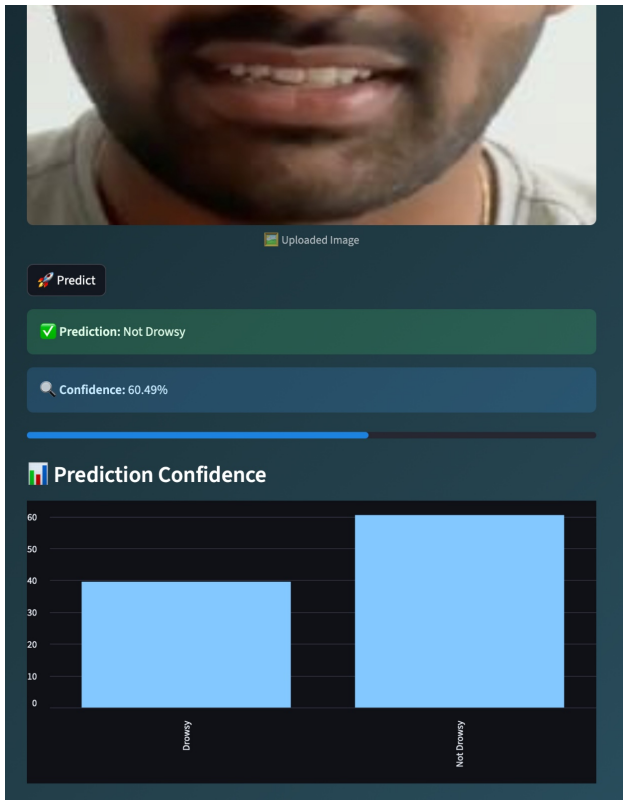

Fig. 42. Model Prediction: Not Drowsy with 60.49% Confidence

Fig. 43. Test Image 2 of Harshith (Drowsy)

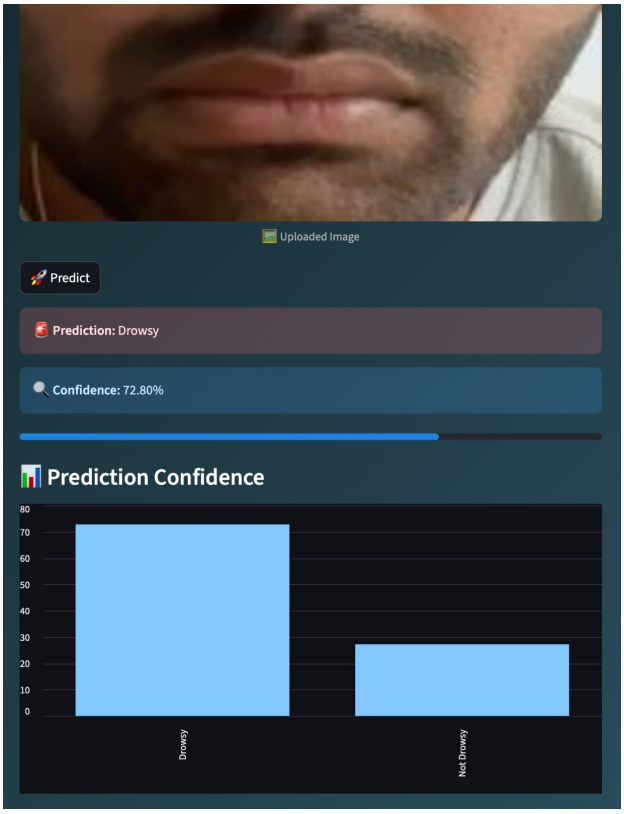

Fig. 44. Model Prediction: Drowsy with 72.80% Confidence

XVIII. Task 18: User Interface Design

In designing the application, we kept in mind usability and accessibility, emphasizing simplicity and functionality. Streamlit was our choice for developing our User interface.

The User interface, shown in the figure below, has a clean layout with a dark theme and clearly defined sections. Users are presented with some clear information in the welcome area and step-by-step instructions for uploading images and acquiring predictions. The file uploader, featuring drag-and-drop functionality, is clearly displayed in the middle of the page.

The left sidebar also details information about the app (what it can do) and the project's contributors. Overall, the design allows for a seamless experience for users in terms of both clarity and honest interaction.

Fig. 45. Streamlit-based User Interface for Driver Drowsiness Detection

XIX. Task 19: Producing a Demonstration Video of the Application

In order to demonstrate the full functionality and usability of our deployed application, we recorded and produced a demonstration video with a full display and explanation of how the system is used.

The video covers the entire user interaction loop including: launching the web interface; uploading driver images; and analyzing model predictions. It shows how the trained MobileNetV2 model identifies and categorizes input images as either Drowsy or Not Drowsy, along with the associated confidence scores and other visual feedback.

Our video serves as a record of completed model deployment, and a clear depiction of the operation of the system for assessment and demonstration.

The demo video link is presented below:

Demo Video

XX. Conclusion

Our project has created a driver drowsiness detection system using deep learning models applied to facial images. The goal of our project was to design a robust and interpretable model to classify a driver as drowsy vs alert.

To begin, we trained and evaluated several different models, including a custom Baseline CNN, MobileNetV2, ResNet, and Vision Transformer (ViT). All four models were compared to each other using performance metrics (accuracy, loss) for classification, and MobileNetV2 was ultimately chosen as the best model of all of them because of its accuracy, efficiency, and inference speed.

After determining the best model, we interpreted the model by utilizing Grad-CAM to illustrate and visualize what facial features the models referenced while classifying the images as drowsy or alert. The images Grad-CAM produced showed that the trained model utilized relatively meaningful facial regions.

The best model (MobileNetV2) was saved and deployed using the Streamlit platform, which allowed us to build a user-friendly web application to upload an image, and provide real-time predictions along with the results that would help clarify confidence scores and visual feedback. The deployed application was tested by us using our own sample images to validate the performance of the web application in the real-world scenario.

To make the application easier to use, a clean and modern user interface was wireframed and designed. We also recorded a demonstration video in the last stage, documenting the entire workflow and showing the functionality of the page launch with all applicable features.

In summary, this is a fully completed machine learning pipeline—from training and evaluating the model to deploying it in real-time—showcasing both technical expertise and real-life application.

Contribution Table

Team Member	Project Part	Contribution (%)
saitanvi	Task-1, Task-4, Task-7, Task-10, Task-13, Task-19, Task-17, Task-20	33.3%
hpeddire	Task-2, Task-5, Task-8, Task-11, Task-14, Task-16, Task-17, Task-20	33.3%
apallerl	Task-3, Task-6, Task-9, Task-12, Task-15, Task-18, Task-17, Task-20	33.3%
Total		**100%**

TABLE I
Individual contributions of team members across various tasks.

References

[1] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. [Online]. Available: https://www.deeplearningbook.org

[2] F. Chollet, *Deep Learning with Python*, 2nd ed. Manning Publications, 2021. [Online]. Available: https://www.manning.com/books/deep-learning-with-python-second-edition

[3] C. M. Bishop, *Pattern Recognition and Machine Learning*. Springer, 2006. [Online]. Available: https://www.springer.com/gp/book/9780387310732

[4] A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*, 2nd ed. O'Reilly Media, 2019. [Online]. Available: https://www.oreilly.com/library/view/hands-on-machine-learning/9781492032632/

[5] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proc. CVPR*, 2001, pp. 511–518. [Online]. Available: https://ieeexplore.ieee.org/document/990517

[6] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. CVPR*, 2016, pp. 770–778. [Online]. Available: https://doi.org/10.1109/CVPR.2016.90

[7] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. CVPR*, 2018, pp. 4510–4520. [Online]. Available: https://doi.org/10.1109/CVPR.2018.00474

[8] A. Dosovitskiy et al., "An image is worth 16x16 words: Transformers for image recognition at scale," in *Proc. ICLR*, 2021. [Online]. Available: https://arxiv.org/abs/2010.11929

[9] M. Abhiram, R. N. Kumar, and V. Kumaravel, "Driver drowsiness detection using deep learning," *Materials Today: Proceedings*, vol. 45, pp. 3184–3189, 2021. [Online]. Available: https://doi.org/10.1016/j.matpr.2020.11.808

[10] R. R. Selvaraju et al., "Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization," in *Proc. ICCV*, 2017, pp. 618–626. [Online]. Available: https://doi.org/10.1109/ICCV.2017.74

[11] S. J. Pan and Q. Yang, "A Survey on Transfer Learning," *IEEE Trans. on Knowledge and Data Engineering*, vol. 22, no. 10, pp. 1345–1359, 2010. [Online]. Available: https://doi.org/10.1109/TKDE.2009.191

[12] T. Baltrusaitis, C. Ahuja, and L. P. Morency, "Multimodal Machine Learning: A Survey and Taxonomy," *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 41, no. 2, pp. 423–443, 2019. [Online]. Available: https://doi.org/10.1109/TPAMI.2018.2798607

[13] B. Zhou, A. Khosla, A. Lapedriza, A. Oliva, and A. Torralba, "Learning Deep Features for Discriminative Localization," in *Proc. CVPR*, 2016. [Online]. Available: https://doi.org/10.1109/CVPR.2016.319

[14] TensorFlow Developers, "TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems," 2015. [Online]. Available: https://www.tensorflow.org

[15] A. Vaswani et al., "Attention Is All You Need," in *Proc. NeurIPS*, 2017. [Online]. Available: https://arxiv.org/abs/1706.03762

[16] Streamlit Inc., "Streamlit Documentation." [Online]. Available: https://docs.streamlit.io/

[17] S. Suresh, P. N. Awasthi, and M. Bansal, "A deep learning approach for real-time driver drowsiness detection," *Journal of Ambient Intelligence and Humanized Computing*, 2022. [Online]. Available: https://doi.org/10.1007/s12652-021-03046-5

[18] S. Ghosh, T. Das, and S. Das, "Driver drowsiness detection using eye state classification," in *Proc. ICCCNT*, 2020. [Online]. Available: https://doi.org/10.1109/ICCCNT49239.2020.9225626