Deep Learning Project

State driver detection

Made by :

- *EL Bouga Latifa*
- *Soulala Achraf*
- *Naji Oussama*

Supervised by:

Mousannif Hajar — *Professor, IT Department FSSM Marrakech*

Academic year: 2022/2023

Contents

State Driver Detection:

The Driver State Detection project is a computer vision and machine learning-based system that aims to address the problem of distracted driving. Distracted driving is a major cause of accidents on the road, and is often caused by the driver engaging in activities such as texting, talking on the phone, and reaching behind while driving. The goal of this project is to reduce the number of accidents caused by distracted driving by providing real-time feedback to drivers and potentially intervening in dangerous situations. This is achieved by using a dataset of video footage from in-vehicle cameras, which is analyzed using a machine learning model to classify the driver's state into different categories. The system is trained to recognize different driver behaviors, such as safe driving, texting, and drinking, and provide a prediction on the driver's state.

The system is designed to provide real-time feedback to the driver and potentially intervene in dangerous situations. It can be integrated with existing in-vehicle technologies, such as advanced driver assistance systems (ADAS) and infotainment systems, to provide real-time feedback to the driver. A flutter app can be designed to allow the user to predict the state of the driver by either importing an image or by providing real-time prediction using the device's camera. The app also has a history feature that allows the user to see the previous predictions and a user-friendly interface, with clear and intuitive controls, making it easy for anyone to use. The system is expected to have a significant impact on reducing the number of accidents caused by distracted driving, and provide valuable insights into driver behavior to inform future research and development in this area.

Figure 1: Some state of drivers

Where can we use it:

The Driver State Detection project can be used in a variety of applications to improve road safety and reduce the number of accidents caused by distracted driving. One of the main applications is in the automotive industry, where the system can be integrated with

existing in-vehicle technologies, such as advanced driver assistance systems (ADAS) and infotainment systems, to provide real-time feedback to the driver and potentially intervene in dangerous situations. The system can also be used in commercial vehicles, such as buses and trucks, to improve the safety of these modes of transportation. Additionally, the system can be used in research and development, to gather data on driver behavior and inform future research and development in this area. The system can also be used in the transportation industry, such as train and airplane, to improve the safety of these modes of transportation.

State Driver Detection Process:

The Driver State Detection project is a process that involves several steps to classify the driver's state into different categories. The process begins by collecting a dataset of video footage from in-vehicle cameras. This dataset is then preprocessed, which includes techniques such as normalization, image resizing and data cleaning. The preprocessed dataset is then used to train a deep learning model, such as a convolutional neural network (CNN), to classify the driver's state into different categories. The trained model is then integrated with a flutter app that allows the user to predict the state of the driver by either importing an image or by providing real-time prediction using the device's camera. The system is designed to provide real-time feedback to the driver and potentially intervene in dangerous situations.

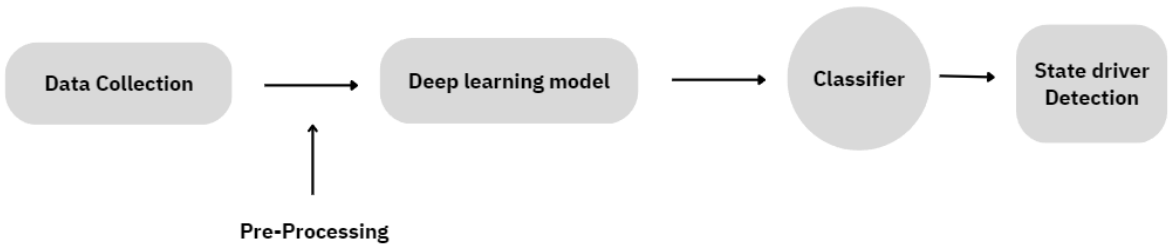


Figure 2: State Driver Detection Process

The steps we followed to carry out this project:

Step 1 – Dataset collection:

The first step in carrying out the Driver State Detection project was the collection and preparation of the dataset. The dataset consisted of video footage from in-vehicle

cameras that captured the driver's behavior. The dataset was collected from two main sources:

1. **State Farm Distracted Driver Detection from Kaggle:**

The first dataset used in the Driver State Detection project is the State Farm Distracted Driver Detection dataset from Kaggle. This dataset is provided by State Farm company and contains a large number of images and videos of drivers engaging in various activities such as safe driving, texting, and drinking.

The dataset is labeled with 10 different categories of driver behavior, including "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", and "talking to passenger".

Figure 3: Labels of driver behavior

The dataset contains around 22,424 images and 2,625 videos. Each image is labeled with one of the 10 different categories, which allows the model to learn the different characteristics of each category. The images were collected from various sources, including in-vehicle cameras and smartphones, and are diverse in terms of lighting conditions and driver's demographics.

The State Farm Distracted Driver Detection dataset is a valuable resource for training and evaluating the performance of the model. It provides a large and diverse dataset of driver behavior, which allows the model to learn the different characteristics of each category and improve its accuracy over time.

2. **Scrapped data from the web:**

The second dataset used in the Driver State Detection project is the dataset scraped from the web. This dataset was scraped from Google using a Selenium script, and contains images of driver's behavior that were not included in the first dataset (State Farm Distracted Driver Detection dataset from Kaggle). The Selenium script was used to automatically navigate through different websites, search for images of driver's behavior and download them. This dataset contains 3461 images collected from the web, it's labeled as "unknown activity" to identify any driver behaviors that are not included in the existing classes, such as reading a book, playing a musical instrument or an empty car.

Figure 4: images represent unkown activities

This dataset provides additional data for the model to learn from, which increases the diversity of the training dataset and allows the model to generalize better. It also allows the system to recognize unknown activities, which makes the system more robust. The additional data from the web also allows the model to learn from a more diverse set of images, which can help improve its overall performance.

Step 2 – Data Pre-Processing:

1. **Resizing:**

Resizing the images to a fixed size is a common data processing technique in deep learning projects, and it can be especially beneficial for image classification tasks like the Driver State Detection project. By resizing the images to a fixed size, such as

128x128x3, it ensures that all the images have the same size. This can help the model to process the images more efficiently, as it can be designed to expect a specific input size.

```
def path_to_tensor(img_path):
    # loads RGB image as PIL.Image.Image type
    img = image.load_img(img_path, target_size=(128, 128))
    # convert PIL.Image.Image type to 3D tensor with shape (224, 224, 3)
    x = image.img_to_array(img)
    # convert 3D tensor to 4D tensor with shape (1, 224, 224, 3) and return 4D tensor
    return np.expand_dims(x, axis=0)

def paths_to_tensor(img_paths):
    list_of_tensors = [path_to_tensor(img_path) for img_path in tqdm(img_paths)]
    return np.vstack(list_of_tensors)
```

Additionally, resizing the images to a smaller size can help to reduce the computational costs and make the training process faster. This can be especially beneficial if the dataset is large. When the model is trained on large datasets, it requires more computational resources to process the images and can take longer to train. By reducing the size of the images, the model needs to process fewer pixels, which can significantly reduce computational costs and speed up the training process.

2. **Normalization:**

Normalization is an important step in data preprocessing that helps to standardize the input data, which can improve the performance of a deep learning model. In the context of the Driver State Detection project, normalization can be applied to the images by normalizing the pixel values to a range of 0-1. This is typically done by dividing each pixel value by the maximum possible value (255 for 8-bit images), which ensures that all pixel values are between 0 and 1.

```
from PIL import ImageFile
ImageFile.LOAD_TRUNCATED_IMAGES = True

# pre-process the data for Keras
train_tensors = paths_to_tensor(xtrain).astype('float32')/255 - 0.5
```

```
100%|██████████| 17939/17939 [01:46<00:00, 167.91it/s]
```

```
valid_tensors = paths_to_tensor(xtest).astype('float32')/255 - 0.5
```

```
100%|██████████| 4485/4485 [00:24<00:00, 180.18it/s]
```

Normalizing the pixel values can help the model to be less sensitive to variations in lighting conditions. For example, if an

image is taken in bright sunlight, the pixel values will be higher than if the same image is taken in dim light. Without normalization, the model may interpret these variations in pixel values as meaningful differences in the image, which could lead to poor performance.

3. **Distribution:**

After collecting and preparing the dataset, the next step is to analyze the distribution of the data. The distribution of the data refers to the number of images in each class of driver behavior. In the case of the Driver State Detection project, the dataset is labeled with 10 different classes.

The data distribution in our project is balanced since the number of images in each class is roughly the same. So, it ensures that the model is trained on a representative sample of the data, which helps to improve its overall performance and prevent overfitting.

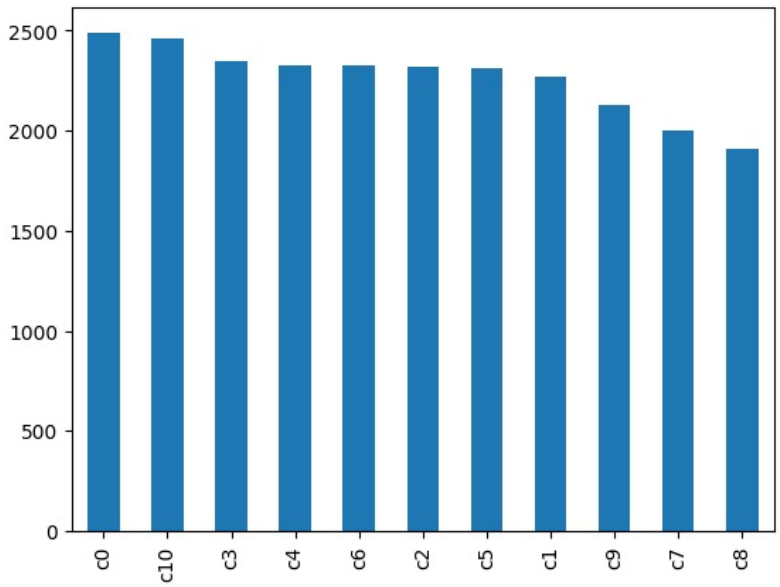


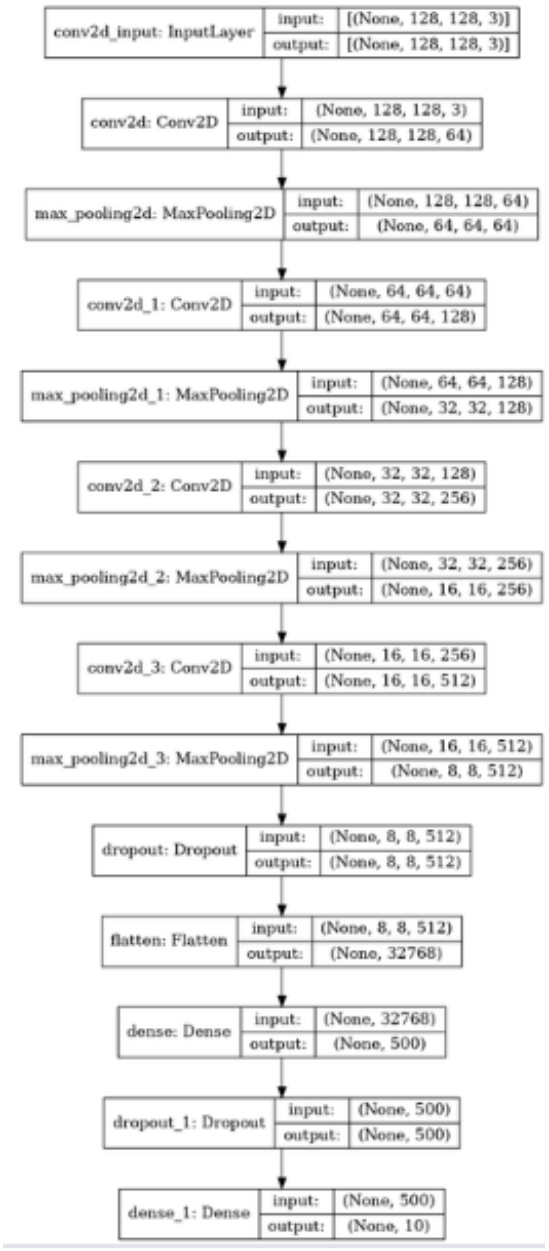
Figure 5: Distribution of the classes

Step 3 – Modeling:

1. **CNN:**

The model starts with a convolutional layer that has 64 filters, a kernel size of 2, and uses the 'same' padding. The activation function used in this layer is ReLU. The input shape of the model is specified as (128,128,3) which means that it expects a 128x128 image with 3 color channels (RGB).

The model then has several other convolutional layers, each with a number of filters that doubles in size compared to the previous layer (64, 128, 256, 512). They also have a kernel size of 2, 'same' padding, and ReLU activation. Each of these layers is followed by a max pooling layer with a pool size of 2. This is a common pattern in convolutional neural networks that helps to reduce the spatial dimension of the data while increasing the number of filters.



Between the convolutional layers, there are Dropout layers. Dropout is a regularization technique that randomly "drops out" a certain percentage of neurons during training, which can help to prevent overfitting. In this case, the dropout rate is set to 0.5, which means that half of the neurons will be dropped out during training.

After the convolutional layers, the model has a Flatten layer, which flattens the output from the previous layers into a 1D array. This is followed by two dense layers, each with 500 and 10 units respectively, and ReLU and softmax activations. The softmax activation function is commonly used in the output layer of a classification model, as it produces a probability distribution over the different classes.

Optimizer: *rmsprop*

Loss function: *Categorical crossentropy*

learning rate: *1e-3*

the evaluation metric: *accuracy*

Number of epochs: *25*

Batch size: *40*

2. VGG16:

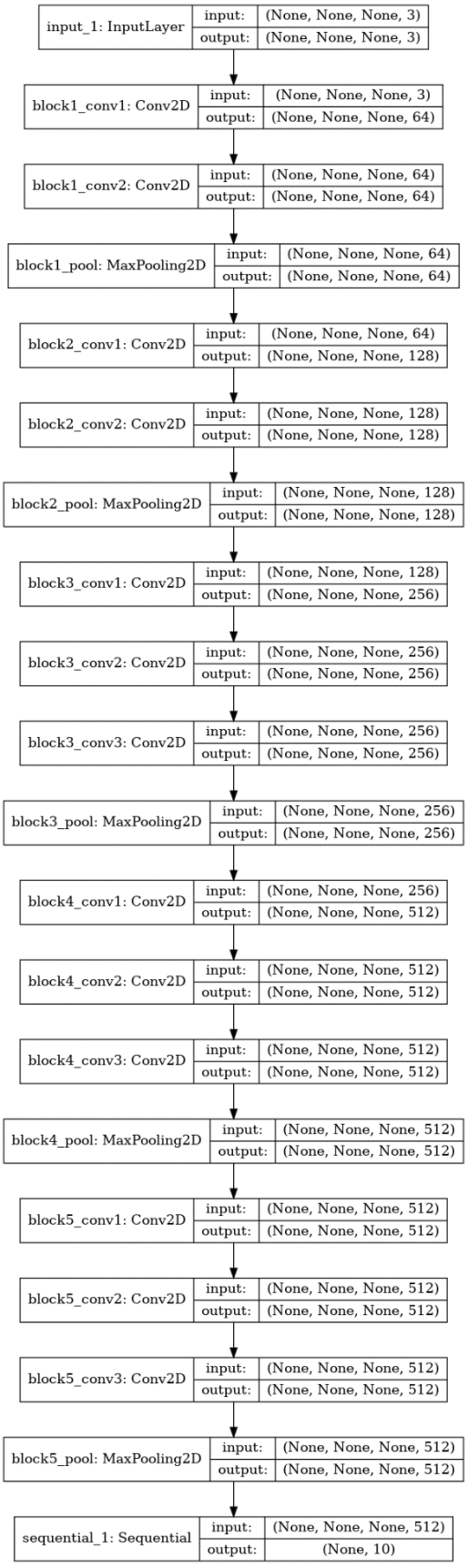
The VGG16 model is a pre-trained convolutional neural network (CNN) that has been trained on a large dataset of images, and it has proven to be effective in several image classification tasks.

The features are the output of a specific layer of the VGG16 model and they represent a condensed version of the input image, capturing the most important information. These features can then be used as input to a new model to train it for a specific task like in the driver state detection project.

The fine-tuning process involves taking the pre-trained VGG16 model, and modifying the last layers to adapt it to a new task. In this case, the last layers of the VGG16 model are replaced with a new sequential model.

The new model starts with a GlobalAveragePooling2D layer, which applies global average pooling to the output of the previous layers. This is a technique that reduces the dimensionality of the data by taking the average of each feature map, rather than keeping the entire feature map.

The new model then has a Dense layer with 10 units and a softmax activation function. The softmax activation function is commonly used in the output layer of a classification model, as it produces a probability distribution over the different classes. The kernel_initializer is set to 'glorot_normal' which is a specific initialization method.



Optimizer: *Stochastic Gradient Descent (SGD)*

Loss function: *Categorical crossentropy*

learning rate: *1e-4*

momentum: *0.9*

the evaluation metric: *accuracy*

Number of epochs: *25*

Batch size: *16*

Step 4 – Results:

1. CNN:

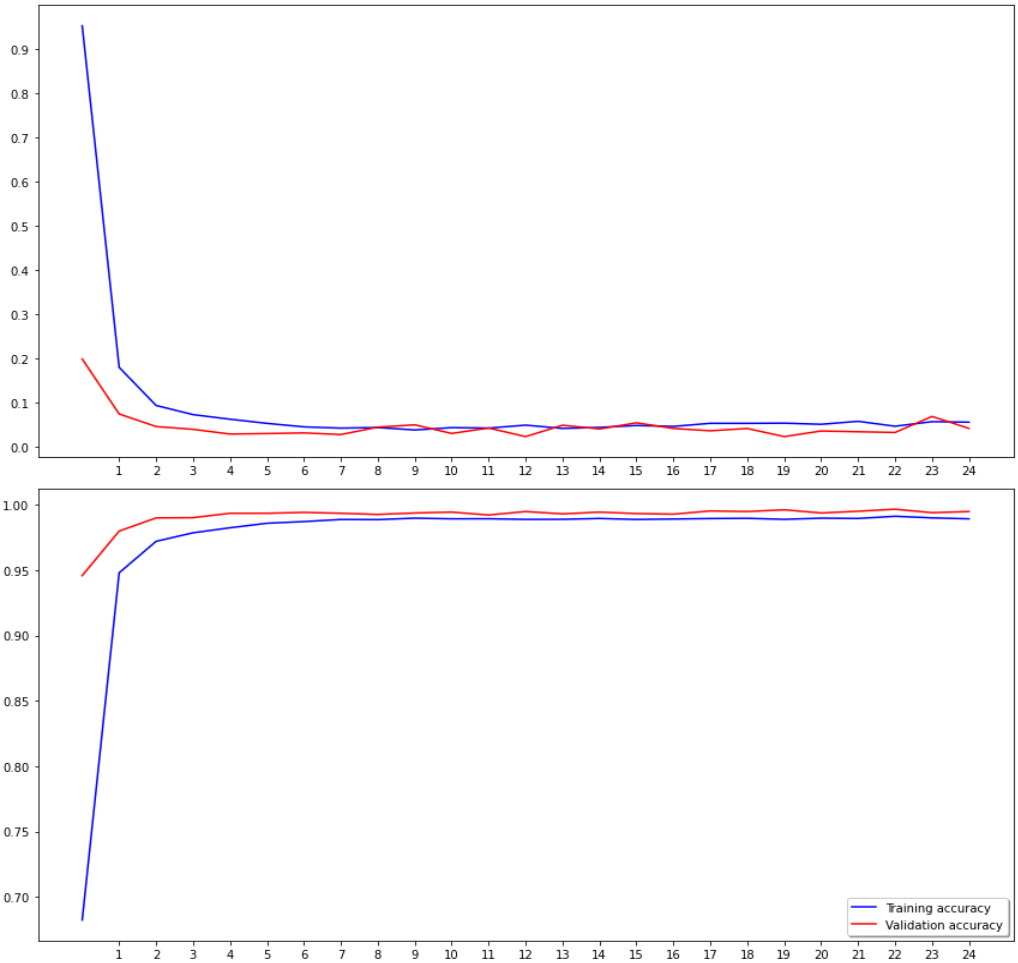


Figure 6: CNN loss and accuracy

The plot shows good results of the CNN, the model would typically show a clear upward trend in both the train accuracy and test accuracy, indicating that the model is learning to recognize the patterns and features in the training data, and is generalizing well to

new unseen data. The train accuracy line would typically start at a lower value and increase over time, while the test accuracy line would also typically start at a lower value and increase over time.

Additionally, the plot would show a clear downward trend in the loss function, indicating that the model is improving and finding a better solution as it is trained on the input data. The loss function line would typically start at a higher value and decrease over time.

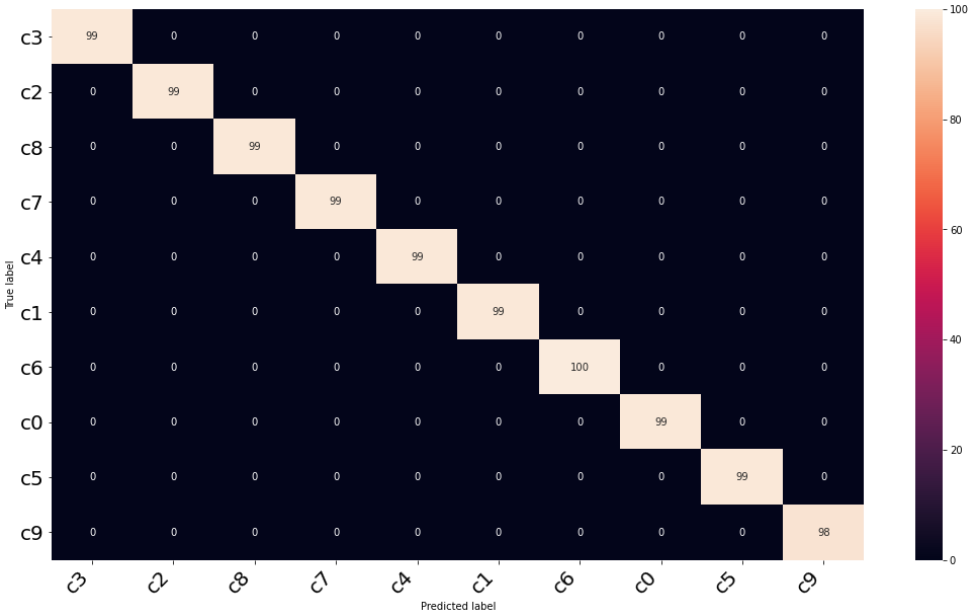


Figure 7: Confusion matrix of CNN

2. VGG16:

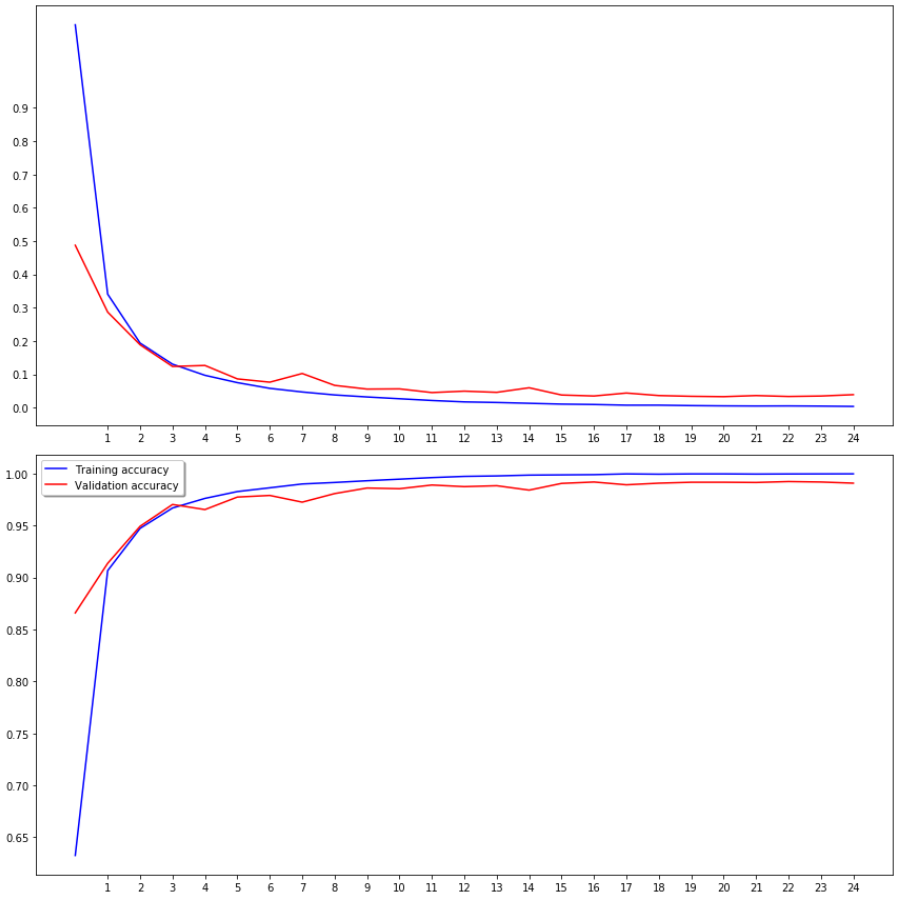


Figure 8: VGG loss and accuracy

In the other hand, the plot shows good results of the VGG16, the model would typically show a clear upward trend in both the train accuracy and test accuracy, indicating that the model is learning to recognize the patterns and features in the training data, and is generalizing well to new unseen data. The train accuracy line would typically start at a lower value and increase over time, while the test accuracy line would also typically start at a lower value and increase over time. The plot would show a clear downward trend in the loss function, indicating that the model is improving and finding a better solution as it is trained on the input data. The loss function line would typically start at a higher value and decrease over time.

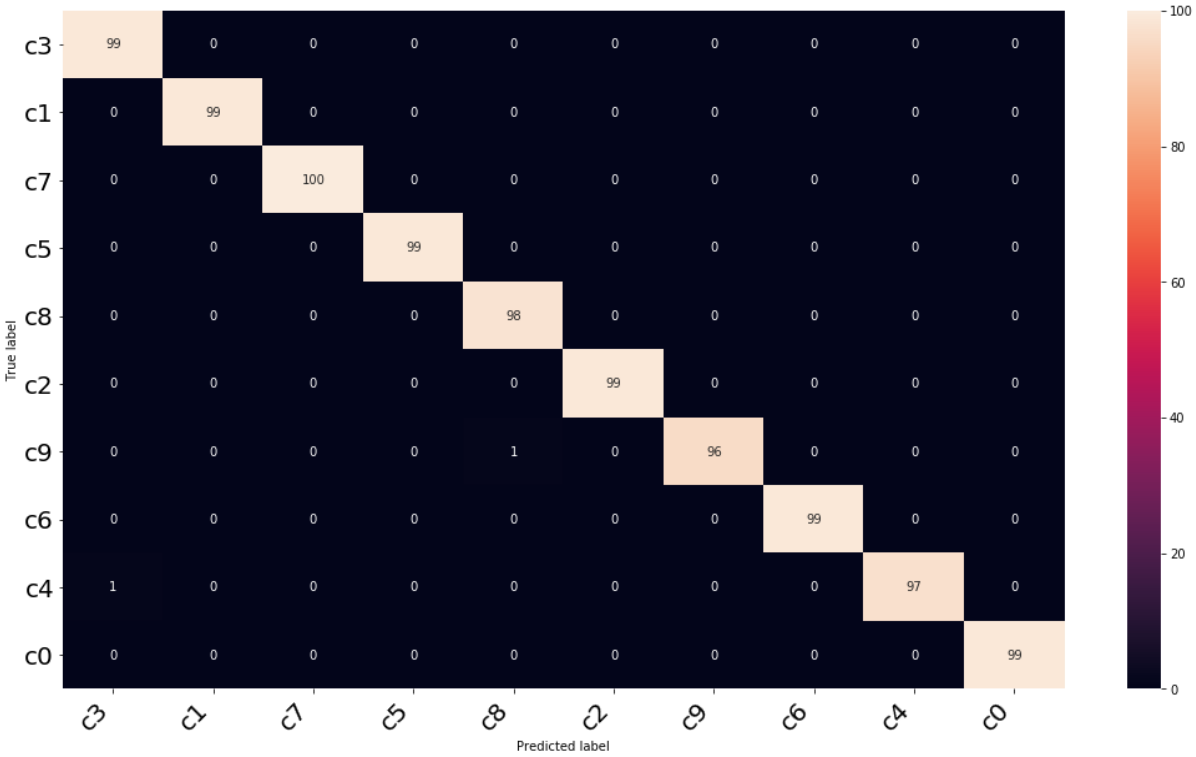


Figure 9: VGG16 confusion matrix

3. Comparison:

The results of the CNN model and the VGG16 model are quite similar in terms of accuracy, precision, recall, and F1-score. The CNN model has an accuracy of 0.994872, precision of 0.994911, recall of 0.994872 and an F1 score of 0.994876. The VGG16 model has an accuracy of 0.990858, precision of 0.990961, recall of 0.990858 and an F1 score of 0.990856.

The accuracy of the CNN model is slightly higher than that of the VGG16 model, indicating that the CNN model is able to classify the input data with a slightly higher degree of accuracy. The precision, recall and F1-score of the CNN model are also slightly higher than that of the VGG16 model, which indicates that the CNN model is able to identify the positive class with a slightly higher degree of accuracy.

Metrics	CNN	VGG16
Accuracy	0.994872	0.990858
Precision	0.994911	0.990961
Recall	0.994872	0.990858
F1	0.994876	0.990856

The precision, recall and F1-score are all similar for both models, which indicates that the model is able to identify the positive class with a similar degree of accuracy, and the number of false positives and false negatives is similar

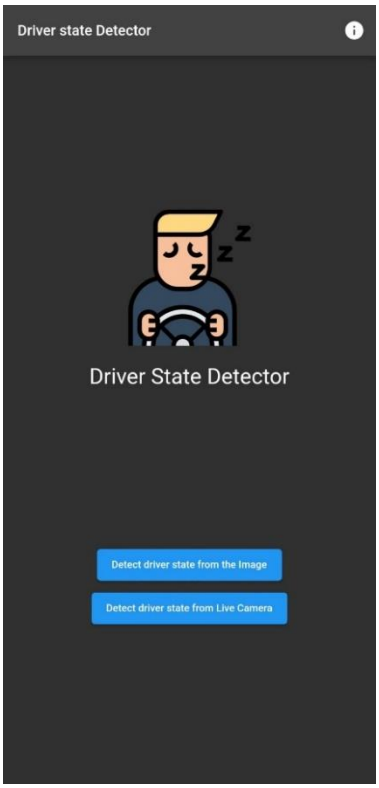
It's worth noting that while the difference in accuracy, precision, recall and F1-score between the CNN and VGG16 models is not significant, the difference in performance may be more pronounced depending on the specific dataset and task. For example, if the dataset has a large number of similar classes, the CNN model may perform better due to its ability to learn more complex features. On the other hand, if the dataset is large and computationally intensive, the VGG16 model may perform better due to its ability to extract features more efficiently.

Another factor to consider is the amount of labeled data available for training. The CNN model can be trained from scratch, which can be beneficial if the dataset is small. However, if the dataset is large, the VGG16 model may perform better as it can leverage the pre-trained weights to learn more effectively.

In conclusion, both the CNN model and the VGG16 model can be suitable choices for a driver state detection project, depending on the specific dataset and task. The choice of model might depend on other factors such as the amount of labeled data available, the computational resources, and the desired trade-off between performance and interpretability.

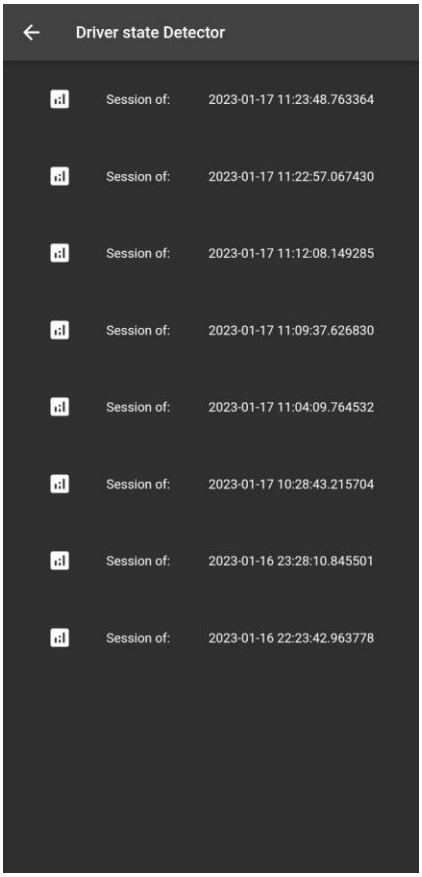
Step 5 – Deployment :

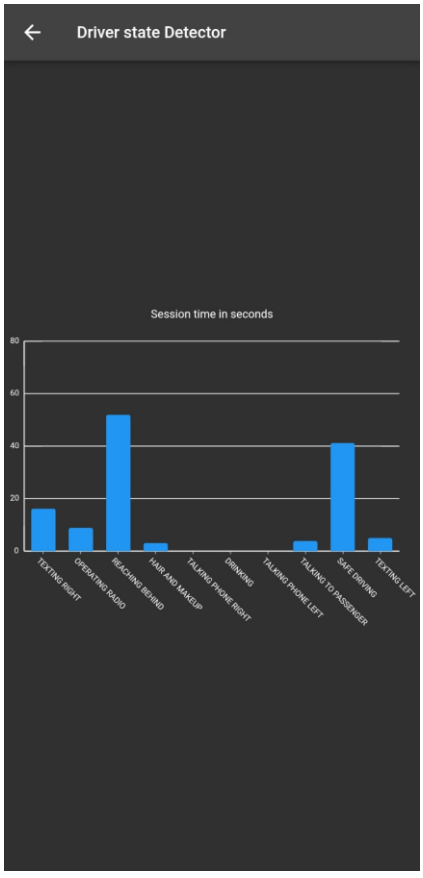
The deployment of the CNN model in a flutter application is an important step in making the model accessible and useful to the end user. The flutter application provides a user-friendly interface that allows for easy and convenient use of the model.



The application provides two options for making predictions, either by importing an image or by performing real-time predictions using a camera. This provides flexibility and convenience for the user to make predictions in different scenarios.

The interface of the application is designed to be easy to use, with clear and concise display of the prediction results. The application also saves the sessions of the real-time predictions in a Firebase NoSQL database. This allows for easy storage and retrieval of the predictions for further analysis or review. The stored data can be used to track the driver's behavior over time, allowing for a better understanding of the driver's state during the drive.





The application also provides a bar plot that visualizes the data and time spent by the driver in each class. The classes used in this project are: c0: safe driving, c1: texting - right, c2: talking on the phone - right, c3: texting - left, c4: talking on the phone - left, c5: operating the radio, c6: drinking, c7: reaching behind, c8: hair and makeup, c9: talking to passenger. This provides a clear and concise way to understand the predictions made by the model and allows for easy identification of the driver's state while driving. This can be useful for identifying patterns in the driver's behavior and can be used to take proactive measures to improve safety on the road.

There are several benefits of the CNN model-based flutter application for driver state detection. Some of the most notable benefits include:

- Increased safety on the road: The application is able to detect and alert the driver of dangerous behaviors such as texting or talking on the phone while driving. This can help to reduce the number of accidents caused by distracted driving.
- Improved driver behavior: The application provides a clear and concise visualization of the driver's behavior over time. This can help the driver to identify patterns in their behavior and take steps to improve their driving habits.
- Easy to use: The application is designed to be user-friendly and easy to use. The interface is clear and intuitive, making it easy for the user to make predictions and understand the results.
- Flexible and convenient: The application allows for predictions to be made in different scenarios, whether by importing an image or by performing real-time predictions using a camera.

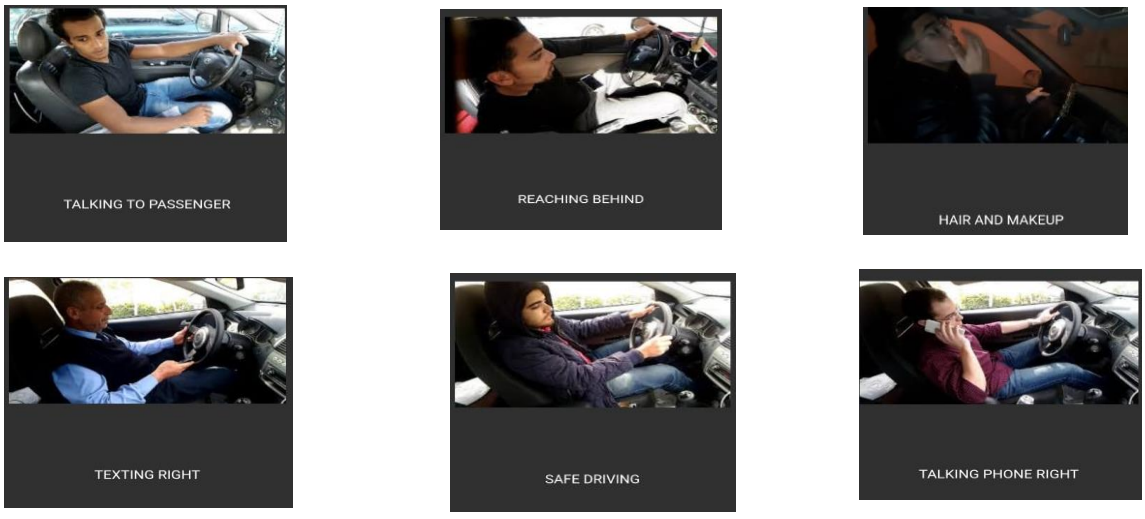


Figure 10: Predictions on new data

- Data storage and visualization: The application stores the predictions in a Firebase NoSQL database and provides a bar plot to visualize the data and time spent by the driver in each class. This allows for easy storage, retrieval, and analysis of the predictions.
- Cost-effective: By using a flutter, the development cost and time required to build the application can be reduced, this can be beneficial for the company that is trying to deploy this model.

Conclusion:

The results of the model have shown high accuracy, precision, recall and F1-score, this indicates that the model is able to classify the input data with a high degree of accuracy and identify the positive class with a high degree of accuracy. The CNN model was slightly better than the VGG16 model in terms of accuracy, precision, recall and F1-score but the difference is not significant.

The deployment of the CNN model in a flutter application is an important step in making the model accessible and useful to the end user. The application provides several benefits such as increased safety on the road, improved driver behavior, easy-to-use, flexible, and convenient way for users to make predictions and understand the results, data storage, visualization, and cost-effectiveness. This makes it a valuable tool for any organization that aims to improve the road safety.