College of Engineering and Technology

B.Sc. Computer Engineering

Final Report

Copilot Application for Traffic Safety

***Abstract*:**

A Copilot system is a network of several components that are used in order to conduct an observing system. This system monitors and notifies the user of any possible obstacle along the road. These obstacles include speeding limits, car accidents, and many other factors such as sudden increases or decreases in acceleration. The main aim of the project is to design a traffic copilot system that has the ability to conduct all the mentioned tasks alongside achieving several criteria including accuracy, high performance, low cost…etc.

I. TABLE OF CONTENTS

II. TABLE OF FIGUERES

III. Introduction

Driving carefully and following the rules will avoid many problems, especially a car accident. Avoiding excessive speed is the common denominator of all causes of accidents. Beware the mistake of others, and that is the only safe way, which is to create a sufficient distance as a safety area around the car to move within its range. These days not many people follow this system, for this reason, we find many accidents on the streets. Because a lot of people have bad habits on the roads such as speeding and wrong way driving also texting while driving, for this reason, we try to build a copilot application for traffic safety. The copilot application acts as an extra pair of eyes for the driver, monitoring their location on the road, warning them if there are any important traffic signs, and assisting them in safely driving within the speed limit. The main aim of the project is to design and implement a copilot application for traffic safety that can help drivers significantly improve the safety of their driving. The motivations behind developing this solution can be summarized in the following points.

1. Accidents

The phenomenon of traffic accidents has spread widely nowadays; either because of the driver's error, the lack of attention of pedestrians, or some defect in the roads and bridges, and perhaps the accident was preordainment without the presence of any human or material defect. The state and citizens must take into account the reasons that preserve the safety of the passenger and pedestrians at the same time, in order to avoid frequent accidents.

2. Traffic congestion

Traffic congestion has many negative effects, as this problem increases the rate of air pollution, which in turn affects the health of the individual and the climate, and the time that the driver spends stuck in the congestion and may cause some accidents.

3. Traffic safety

Traffic on roads is one of the most important serious problems that our contemporary world suffers from, and the countries of the world have noticed this problem and have put in place some laws to alleviate it as much as possible, but they did not reach the required level, as the number of people who die in the world annually is estimated to be more than million people, and the number of injured as a result of road accidents reaches.

IV. Design Development

In this chapter, a detailed description of the design and its different elements is given.

1. Proposed Design

The proposed copilot system will be able to detect the traffic labels from distance with high accuracy. It will also have an alarming system when a critical traffic label is observed on the road. Moreover, the system will have the ability to warn the driver when the speed limit is exceeded. The system will also detect nearby collisions and accidents and report them to the user in real-time. Additionally, the driver will be warned once a sudden change in acceleration occurs.

2. Detailed High-Level Specifications

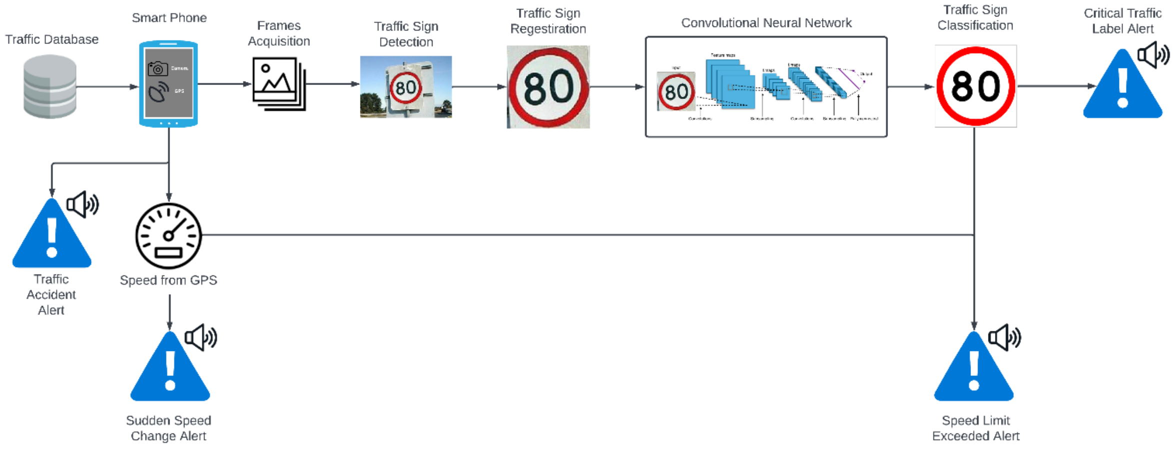


Figure 1. The high-level design of the system

First, we have a smartphone with GPS, camera, and Internet connection. The GPS of the smartphone is accessed to get information about the current location of the vehicle and the speed of the vehicle is calculated using GPS data. If the current speed of the vehicle changes abruptly, the smartphone sends an audio traffic accident alert. The smartphone is connected to the cloud database via the Internet, and the database holds traffic data such as locations of traffic jams, car accidents, etc. If the current location is within a certain radius from any of the traffic events recorded in the database, the smartphone sends an audio alert to warn the driver.

Furthermore, the smartphone camera sends a live feed of the road, where the accepted frames are sent through a pre-trained convolution neural network that classifies and recognizes the traffic signs from these frames. The speed limit from the given sign is updated to alert the driver when the current speed exceeds the speed limit. Moreover, when a critical traffic sign is detected, an audio alert is sent to the driver.

3. Detailed Low-Level Specifications

a) Interacting Components

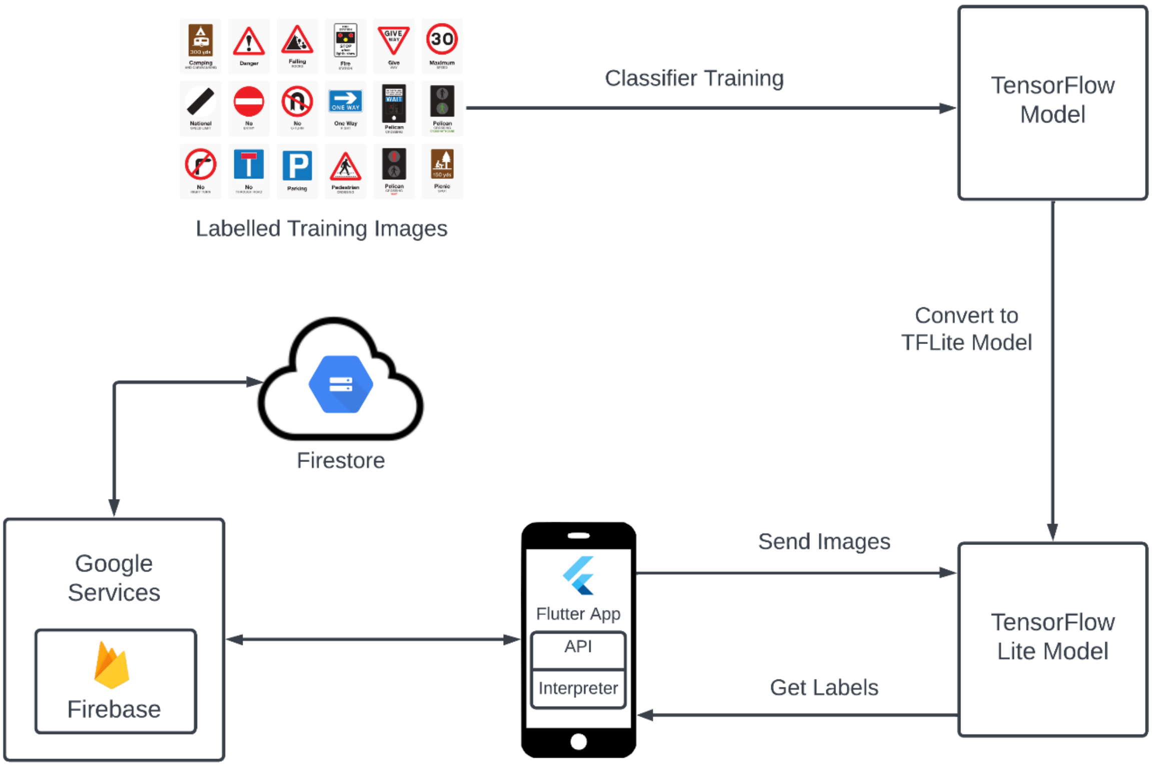


Figure 2. The interacting components of the system

i. TensorFlow

Google Brain Team is responsible for the creation of the TensorFlow open-source software library machine. It is created for machine learning and artificial intelligence applications. TensorFlow comes with several options, but most importantly it facilitates training and inference of deep neural networks. It was first released in 2015, and later it was updated to TensorFlow 2.0 in 2019. The software can be used in several programming languages. Some of these languages are Java, JavaScript, C++, and Python. This project uses TensorFlow on Python.

ii. TensorFlow Lite

The framework needed to run TensorFlow for Flutter applications is TensorFlow Lite. It is a framework that comes with software packages used in ML training locally on the hardware. It is primarily used for low-size and low-computational devices as it aids developers to run their models through such devices. In this project, the pre-trained TensorFlow model is converted into the TFLite format to be then integrated into the Flutter project.

iii. Flutter

Flutter is an open-source framework for developing native interfaces on iOS and Android. This UI framework is used to build applications from a single codebase. These applications can be used on the web, mobile, or desktop. Flutter also uses Dart for its several features such as Minix, isolates, and others. Dart can use Just-In-Time compilation, which allows Flutter to offer hot reloads through development without having to create a new build.

iv. Firebase

It is a real-time database used for developing applications. It is a newly founded back-end service and it is found on the Google Cloud Platform. This program is the reason users can access their data from the cloud across several different platforms. It provides its users with readily available data on their iOS or Android devices. Firebase Firestore is a NoSQL document database. It has several uses, some of these usages are automatic scaling, high performance, and application development. What makes Firestore a unique database is its flexibility and its description of relationships between objects. It still comes with the basic options present in other databases. It also syncs every user's data across several platforms. In this application, Firebase is integrated into the Flutter project and is used to store and retrieve the coordinates of accidents and traffic congestions.

b) Machine Learning Model

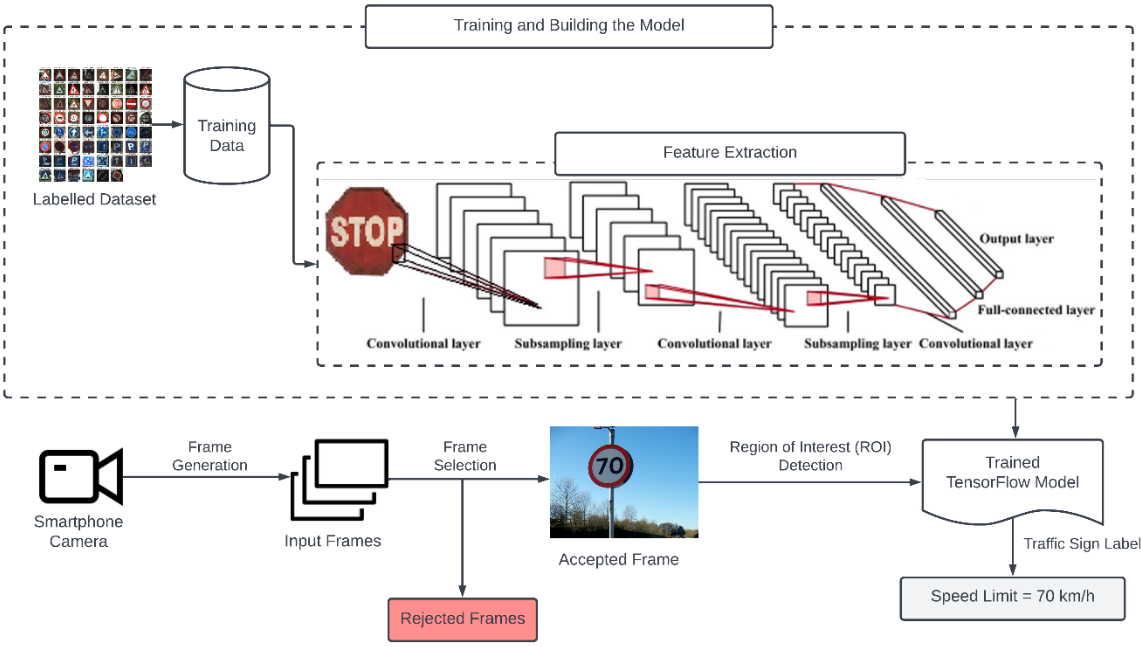


Figure 3. Detailed design of the TensorFlow model

This model is a machine learning one used to recognize images. The model works on labeling the image uploaded to it in a category. The categories these images fall under are previously taught to the model by the user through uploading labeled similar images. A training dataset is used to train the ML model. The training data is a set of data used to teach the model how to learn and deliver advanced results using technologies such as neural networks. It can be supplemented with additional datasets known as validation and testing sets. Feature extraction is the process of building values extracted from an initial set of data to aid users in learning, generalization, and interpretation. It facilitates the process of getting important and relevant information when there is a large data set with several resources.

Image classification is used to define the class of a certain object within an image, whilst object detection is used in computer vision to identify objects in images. The input of image classification is an image producing an output that is the label. The input of object detection is an image or more producing an output that is a bounding box or more and labeling of said boxes. Image classification's algorithm produces a list of categories from the inputs. Object detection's algorithm produces categories in the image along with its bounding box.

V. PROJECT REALIZATION AND PERFORMANCE OPTIMIZATION

1. Planned implementation and experiments. [PI-6.a]

a) Training Dataset

The TensorFlow model used in this project was trained on The German Traffic Sign Benchmark (GTSRB), which is a multi-class, single-image classification database introduced at the International Joint Conference on Neural Networks (IJCNN) in 2011. The database has the following properties: single-image, multi-class classification problem, more than 40 classes, more than 50,000 images in total, and large, lifelike database, reliable ground-truth data due to semi-automatic annotation, and physical traffic sign instances are unique within the dataset (i.e., each real-world traffic sign only occurs once).

The training set archive is structured as follows: one directory per class, each directory contains one CSV file with annotations ("GT-<ClassID>.csv") and the training images. Training images are grouped by tracks, and each track contains 30 images of one single physical traffic sign.

The images contain one traffic sign each. Images contain a border of 10 % around the actual traffic sign (at least 5 pixels) to allow for edge-based approaches. Images are stored in PPM format (Portable Pixmap, P6), and image sizes vary between 15x15 to 250x250 pixels. Images are not necessarily square, and the actual traffic sign is not necessarily centered within the image. This is true for images that were close to the image border in the full camera image. The bounding box of the traffic sign is part of the annotations.

Annotations are provided in CSV files. Fields are separated by ";" (semicolon). Annotations contain the following information:

- Filename: Filename of the corresponding image
- Width: Width of the image
- Height: Height of the image
- ROI.x1: X-coordinate of the top-left corner of the traffic sign bounding box
- ROI.y1: Y-coordinate of the top-left corner of the traffic sign bounding box
- ROI.x2: X-coordinate of the bottom-right corner of the traffic sign bounding box
- ROI.y2: Y-coordinate of the bottom-right corner of the traffic sign bounding box

The training data annotations additionally contain ClassId, which is the assigned class label. The distribution of the dataset classes is shown in the below figure.

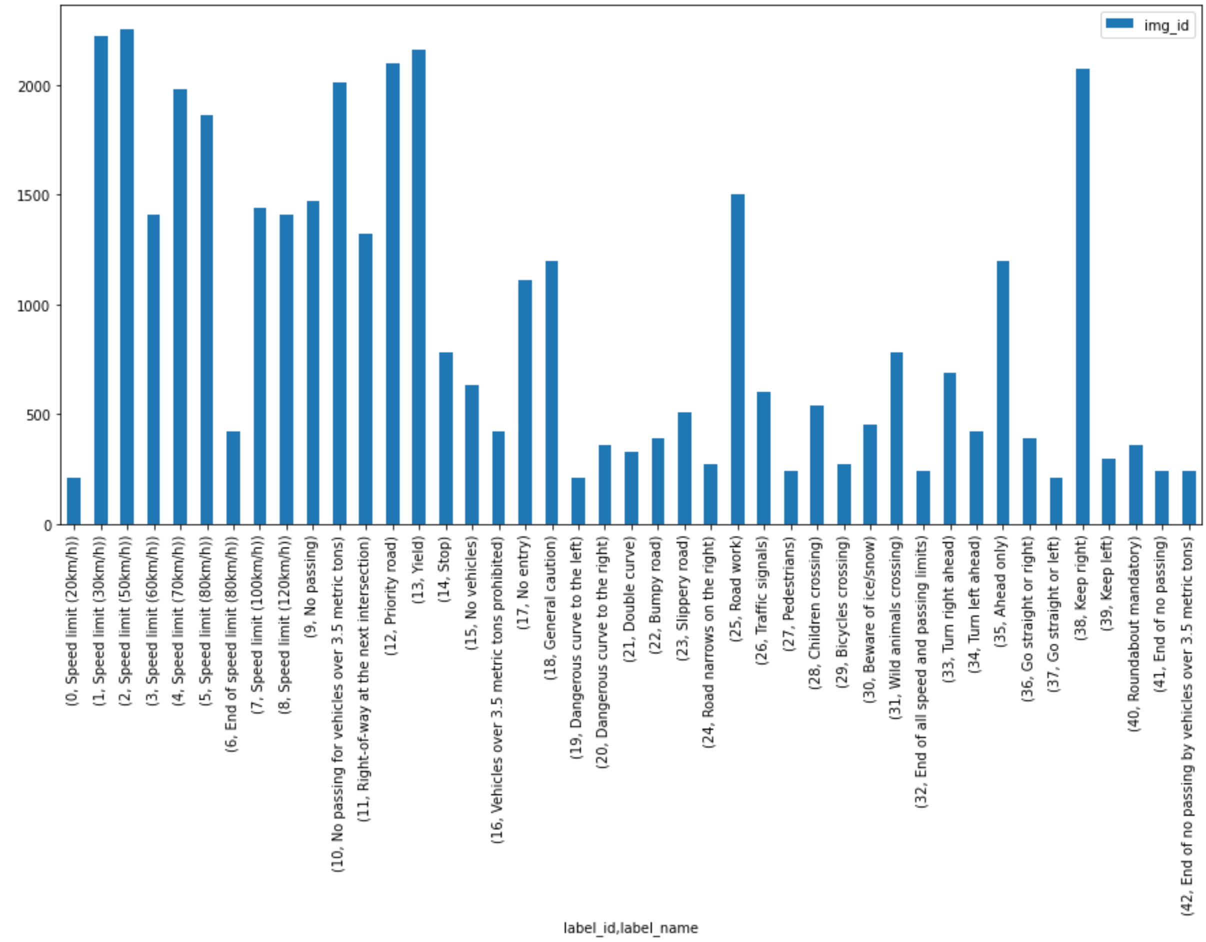


Figure 4. Distribution of the dataset images over classes

b) TensorFlow Model

A Convolutional Neural Network (CNN) is a machine learning unit that analyzes data using perceptron/computer graphs. The majority of the data is represented via photographs. A 3D vector dimension is processed using feature maps and then downsampled using the Pooling method. Two prominent pooling approaches for downsampling image feature maps are MaxPooling and MeanPooling. The Convolution Neural Network is a popular Deep Learning technique. CNN's main purpose is to shrink the size of the input shape. We'll utilize four-dimensional picture pixels in the example below, with a total of 50 photographs and 64 pixels of data. The 4 value 3 symbolizes a color image since a picture is made up of three colors, or RGB. Conv2D scales down the input size after receiving the input picture pixel.

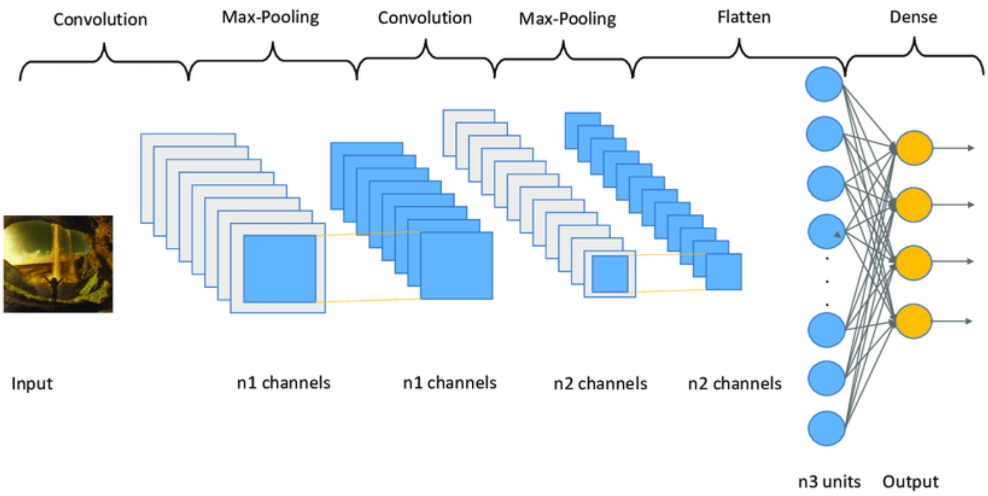


Figure 5. CNN model

Thus, the architecture of the TensorFlow model is chosen to be the following:

```
Compiling the model
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 32)        2432

 conv2d_1 (Conv2D)           (None, 24, 24, 32)        25632

 max_pooling2d (MaxPooling2D  (None, 12, 12, 32)       0
 )

 dropout (Dropout)           (None, 12, 12, 32)        0

 conv2d_2 (Conv2D)           (None, 10, 10, 64)        18496

 conv2d_3 (Conv2D)           (None, 8, 8, 64)          36928

 max_pooling2d_1 (MaxPooling  (None, 4, 4, 64)         0
 2D)

 dropout_1 (Dropout)         (None, 4, 4, 64)          0

 flatten (Flatten)           (None, 1024)              0

 dense (Dense)               (None, 256)               262400

 dropout_2 (Dropout)         (None, 256)               0

 dense_1 (Dense)             (None, 43)                11051

=================================================================
Total params: 356,939
Trainable params: 356,939
Non-trainable params: 0
_________________________________________________________________
```

Figure 6. TensorFlow model architecture

After cleaning the dataset and splitting it into training and validation subsets, we compile the TensorFlow model. Only 10 epochs are chosen to limit the chance of overfitting.

```
Compiling the model
Epoch 1/10
491/491 [==============================] - 4311s 9s/step - loss: 1.7064 - accuracy: 0.5191 - val_loss: 0.2724 - val_accuracy: 0.9235
Epoch 2/10
491/491 [==============================] - 59s 120ms/step - loss: 0.3169 - accuracy: 0.9036 - val_loss: 0.0735 - val_accuracy: 0.9806
Epoch 3/10
491/491 [==============================] - 58s 117ms/step - loss: 0.1704 - accuracy: 0.9468 - val_loss: 0.0383 - val_accuracy: 0.9903
Epoch 4/10
491/491 [==============================] - 60s 121ms/step - loss: 0.1281 - accuracy: 0.9606 - val_loss: 0.0510 - val_accuracy: 0.9871
Epoch 5/10
491/491 [==============================] - 60s 122ms/step - loss: 0.1116 - accuracy: 0.9653 - val_loss: 0.0361 - val_accuracy: 0.9897
Epoch 6/10
491/491 [==============================] - 59s 120ms/step - loss: 0.0857 - accuracy: 0.9734 - val_loss: 0.0217 - val_accuracy: 0.9945
Epoch 7/10
491/491 [==============================] - 60s 122ms/step - loss: 0.0738 - accuracy: 0.9775 - val_loss: 0.0208 - val_accuracy: 0.9952
Epoch 8/10
491/491 [==============================] - 60s 122ms/step - loss: 0.0655 - accuracy: 0.9789 - val_loss: 0.0172 - val_accuracy: 0.9955
Epoch 9/10
491/491 [==============================] - 59s 120ms/step - loss: 0.0594 - accuracy: 0.9818 - val_loss: 0.0216 - val_accuracy: 0.9948
Epoch 10/10
491/491 [==============================] - 59s 120ms/step - loss: 0.0554 - accuracy: 0.9823 - val_loss: 0.0178 - val_accuracy: 0.9953
```

Figure 7. Model training

The accuracy and loss curves for training and validation datasets are shown in the figure below.

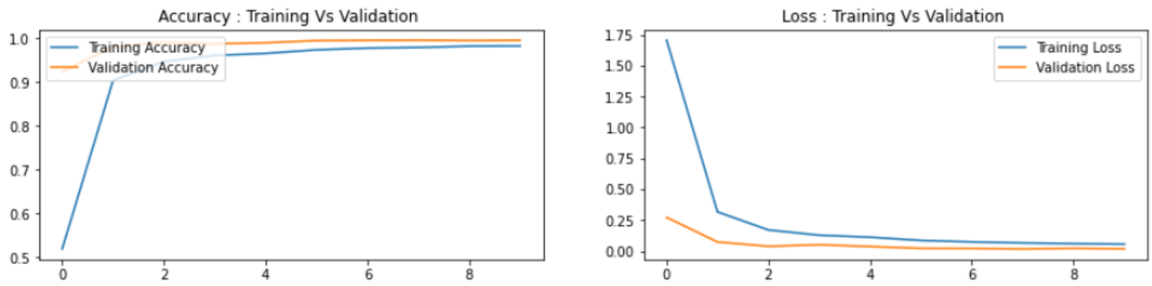
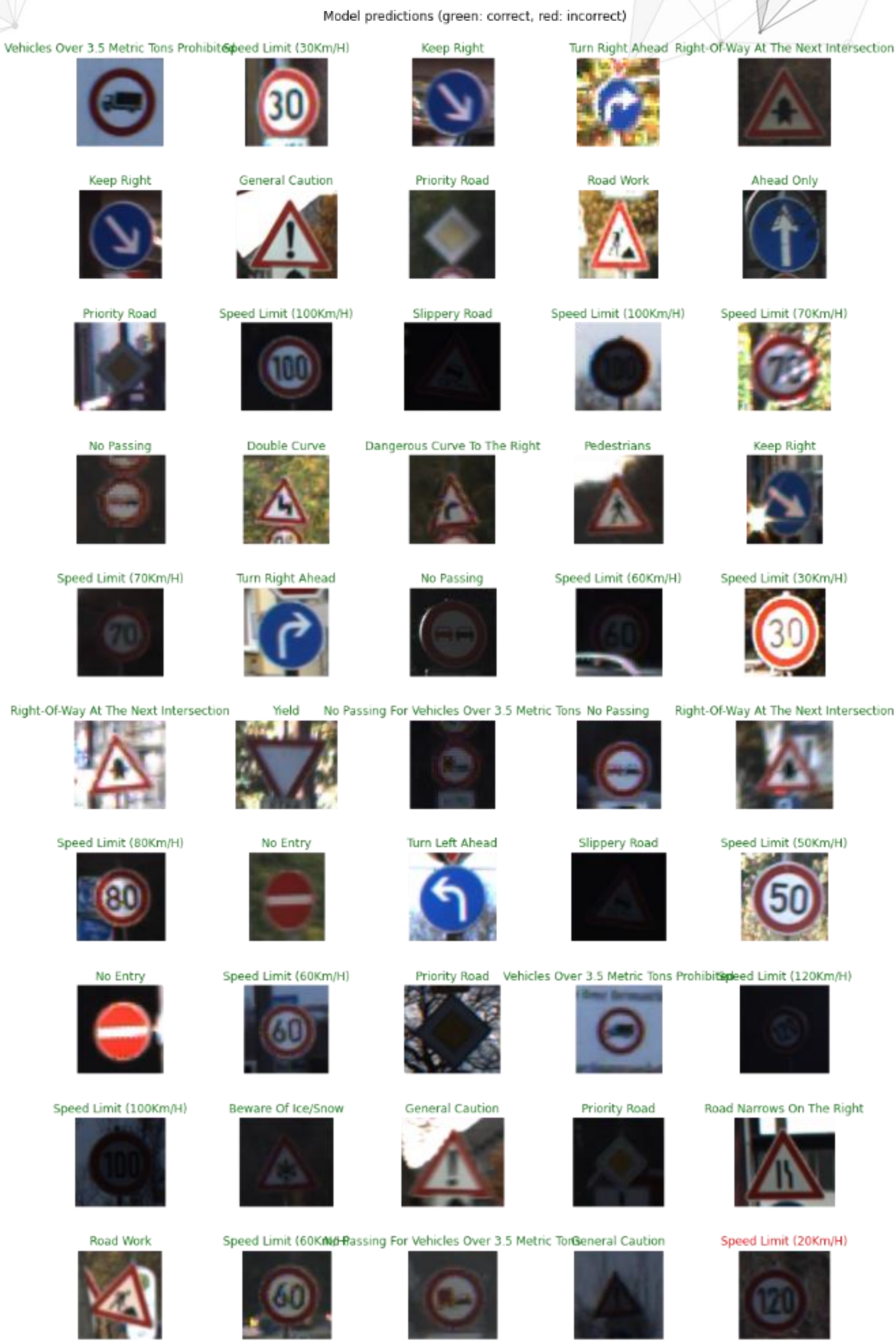


Figure 8. Accuracy and loss curves

Testing the model on the test data results in 97.82% accuracy as shown in the following figure.


Model predictions (green: correct, red: incorrect)
Vehicles Over 3.5 Metric Tons Prohibit
Speed Limit (30Km/H)
Keep Right
Turn Right Ahead
Right-Of-Way At The Next Intersection
Keep Right
General Caution
Priority Road
Road Work
Ahead Only
Priority Road
Speed Limit (100Km/H)
Slippery Road
Speed Limit (100Km/H)
Speed Limit (70Km/H)
No Passing
Double Curve
Dangerous Curve To The Right
Pedestrians
Keep Right
Speed Limit (70Km/H)
Turn Right Ahead
No Passing
Speed Limit (60Km/H)
Speed Limit (30Km/H)
Right-Of-Way At The Next Intersection
Yield
No Passing For Vehicles Over 3.5 Metric Tons
No Passing
Right-Of-Way At The Next Intersection
Speed Limit (80Km/H)
No Entry
Turn Left Ahead
Slippery Road
Speed Limit (50Km/H)
No Entry
Speed Limit (60Km/H)
Priority Road
Vehicles Over 3.5 Metric Tons Prohibit
Speed Limit (120Km/H)
Speed Limit (100Km/H)
Beware Of Ice/Snow
General Caution
Priority Road
Road Narrows On The Right
Road Work
Speed Limit (60Km/H)
No Passing For Vehicles Over 3.5 Metric To
General Caution
Speed Limit (20Km/H)


```
Accuracy of the shown eval batch: 0.9782264449722882
```

Figure 9. Model accuracy and predictions

The following is the confusion matrix of the trained model. As it is shown in the following figure, the model is performing well since the highest predicted count label in each class is the true label of that class.

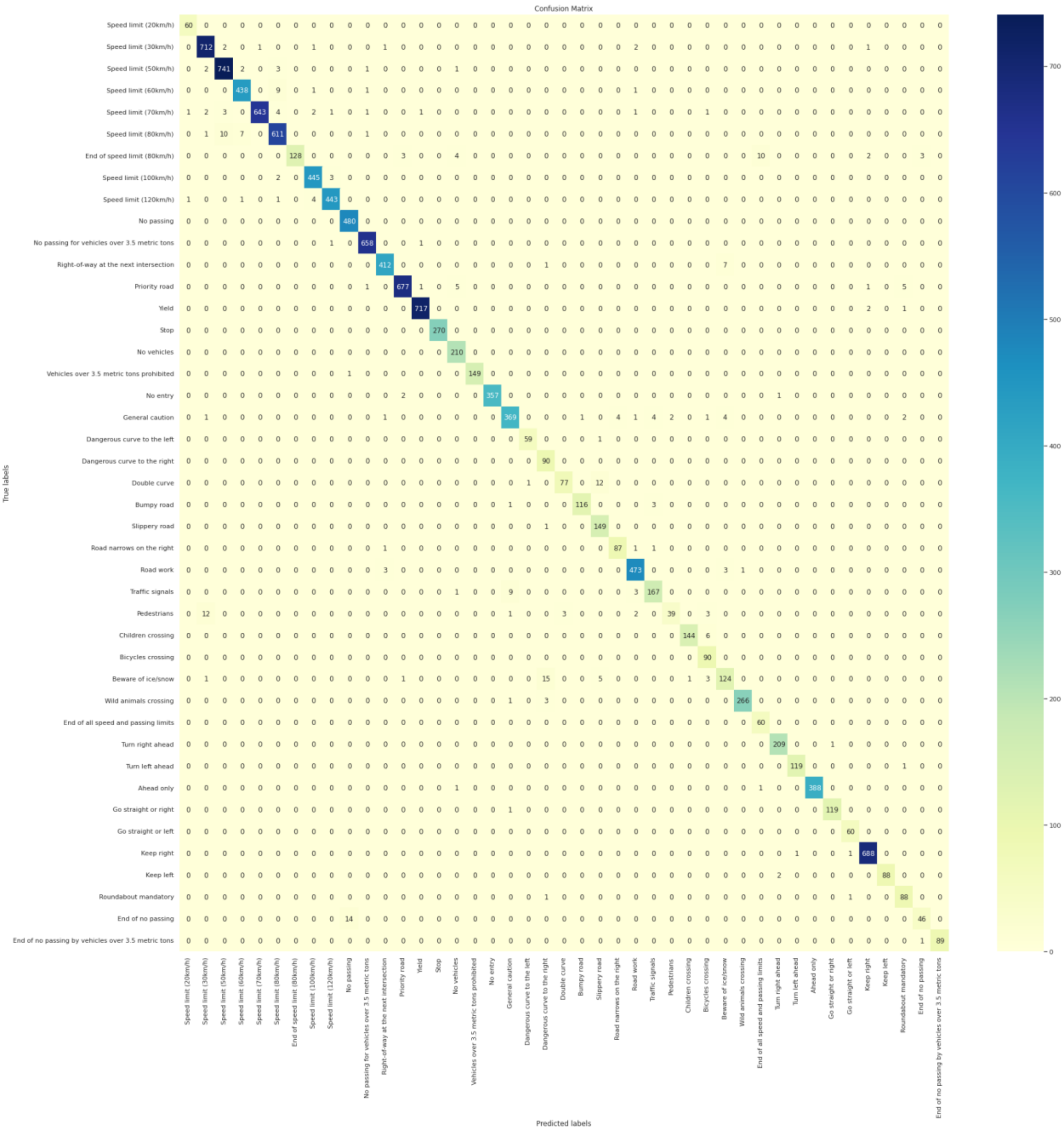
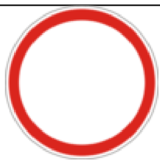
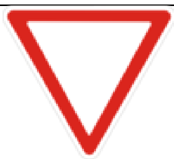
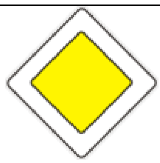
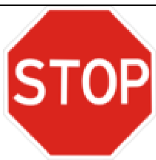


Figure 10. Confusion matrix

c) *Application*

To limit the possible confusion in the detection of traffic signs from the smartphone camera feed, we limited the number of classes used in the application to the following 20 labels:

Label	Sign	Label	Sign
Children Crossing Road		Speed Limit 10	
Crosswalk		Speed Limit 20	
Don’t Enter		Speed Limit 30	
No Vehicles		Speed Limit 40	
Don’t Stop		Speed Limit 50	
Give Road		Speed Limit 60	
Main Road		Speed Limit 70	
No Overtaking		Speed Limit 80	
No Parking		Speed Limit 90	
Stop		Speed Limit 100	

Next, we recorded the warning voice notifications to be used in the application.

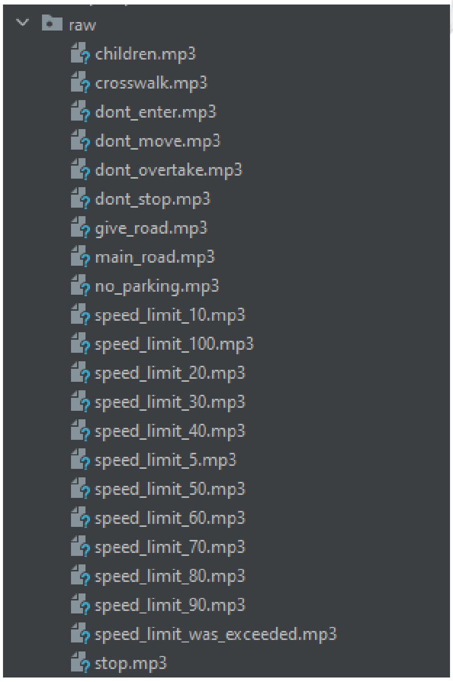
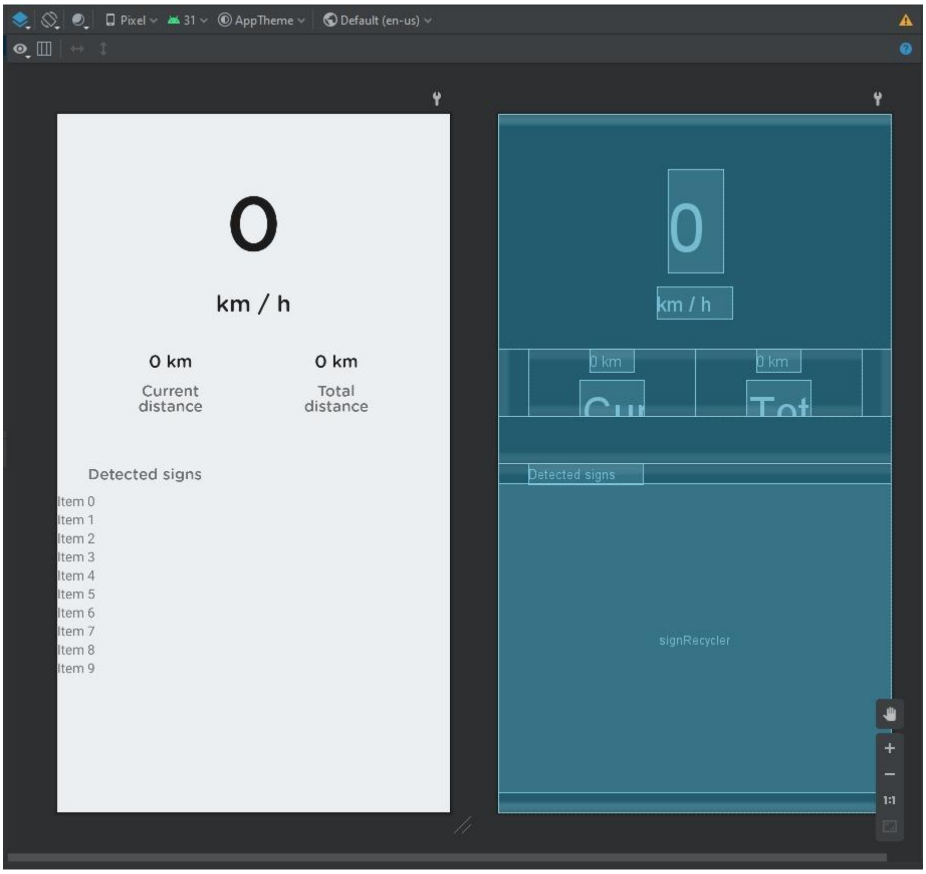
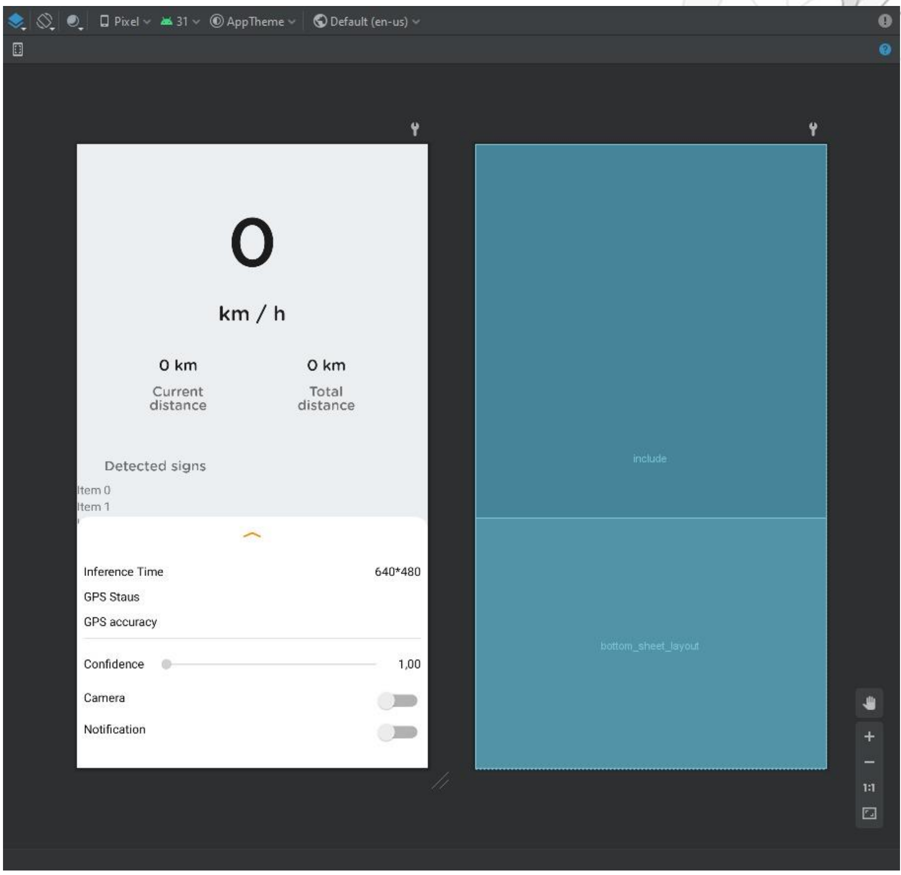


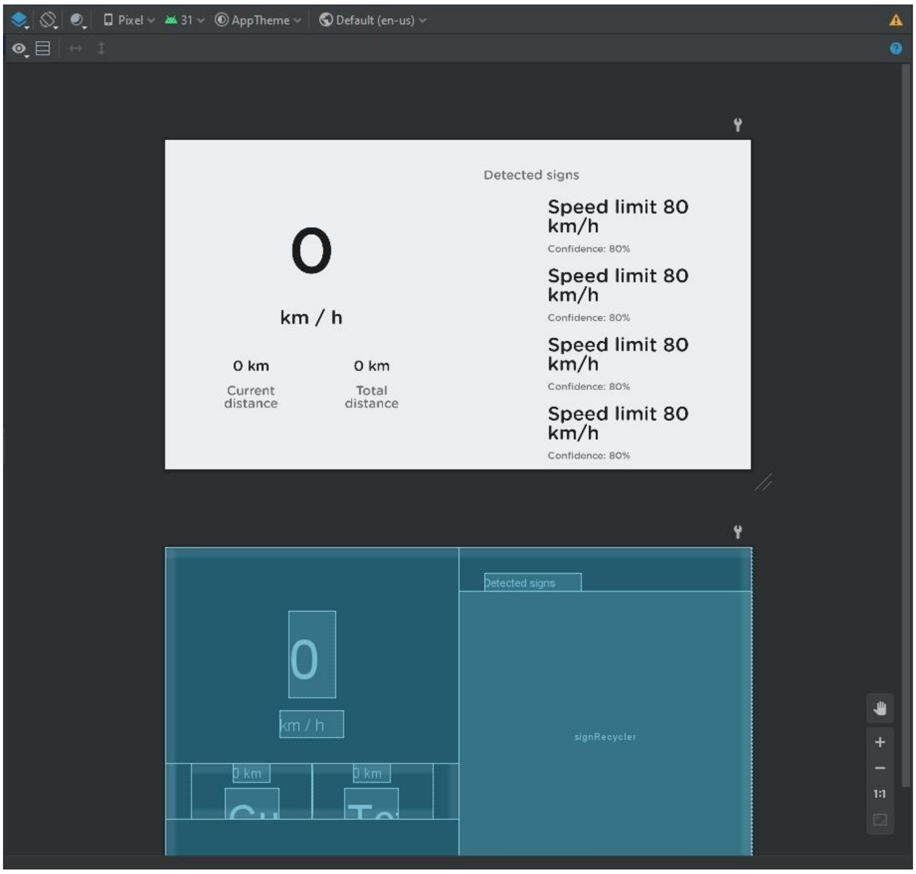
Figure 11. Warning voice recordings

After coding the application in Flutter, we build and export the Android project. Then, we used Android Studio to design the interface of the Android application.





Pixel
31
AppTheme
Default (en-us)
0
km / h
0 km
Current distance
0 km
Total distance
Detected signs
Item 0
Item 1
Inference Time
640*480
GPS Staus
GPS accuracy
Confidence
1,00
Camera
Notification
include
bottom_sheet_layout



Pixel
31
AppTheme
Default (en-us)
Detected signs
Speed limit 80 km/h
Confidence: 80%
Speed limit 80 km/h
Confidence: 80%
Speed limit 80 km/h
Confidence: 80%
Speed limit 80 km/h
Confidence: 80%
0
km / h
0 km
Current distance
0 km
Total distance
signRecycler

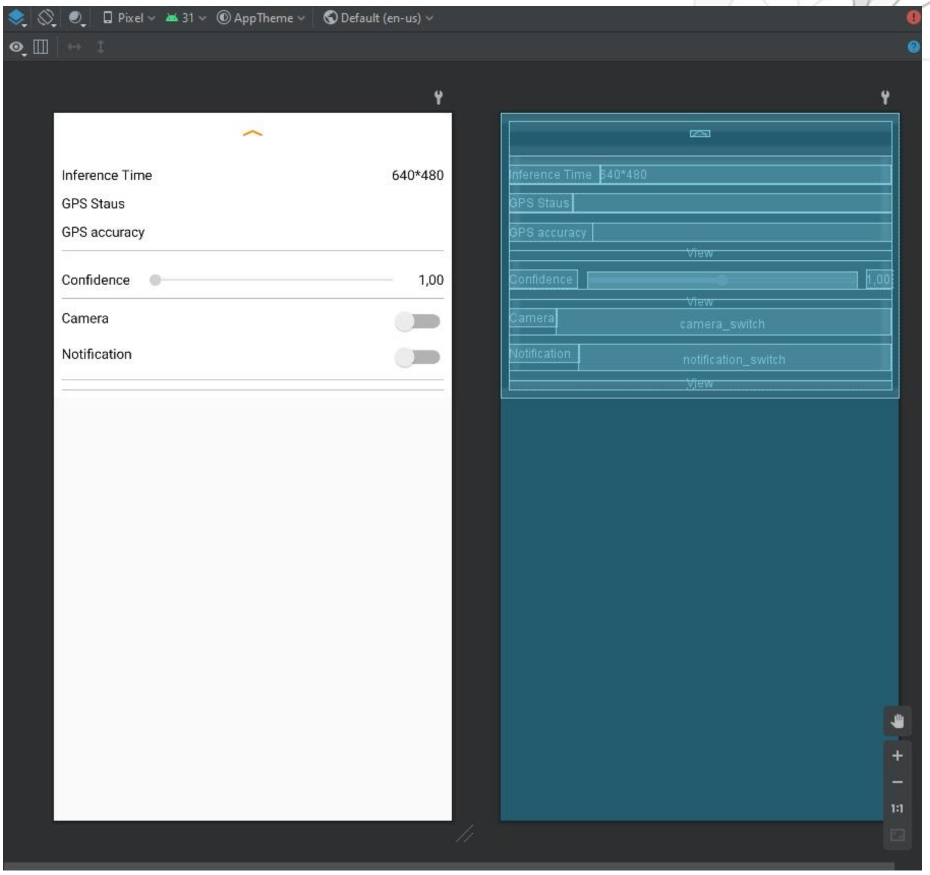
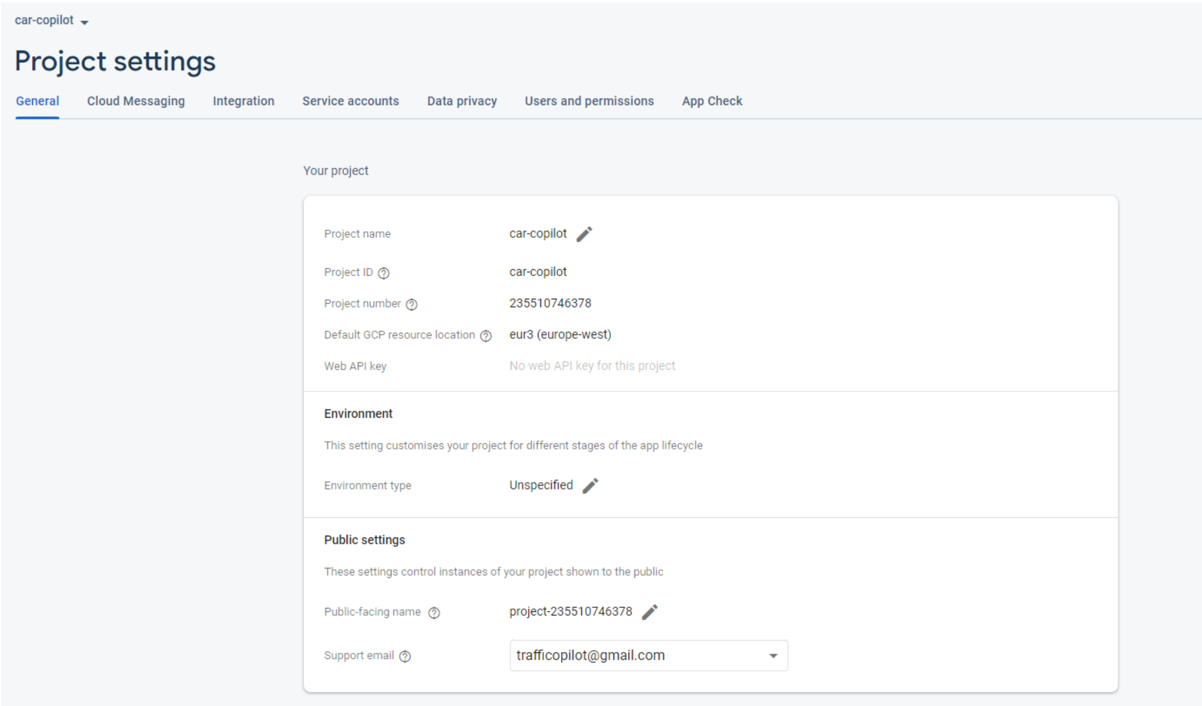


Figure 12. Application UI

The Firebase plugins were also integrated into the application and the Android app was registered into the application's project settings.



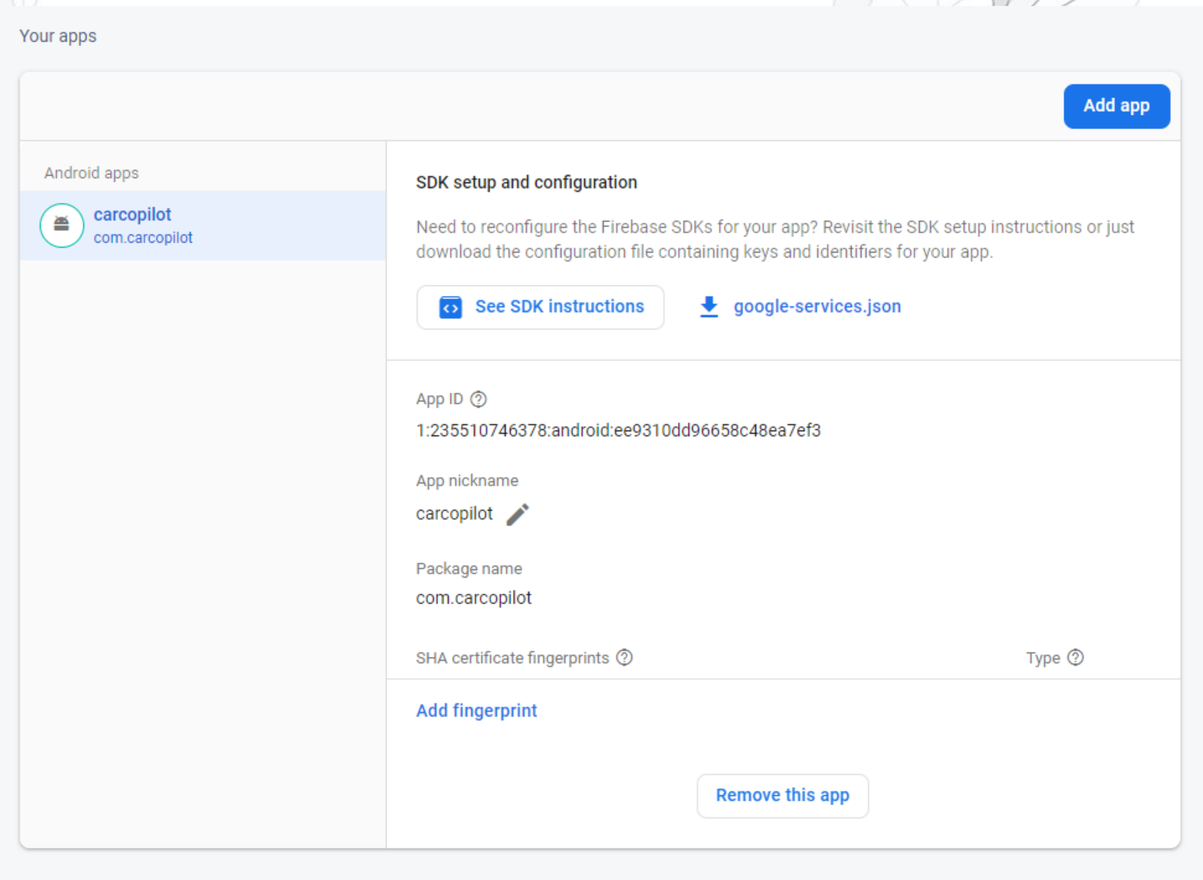


Figure 13. Firebase integration

Finally, the app icon was chosen to be the following icon.

Figure 14. App icon

2. Design Analysis and Feedback [PI-6.b]

The experiments needed to test the major functionalities of the app are as follows:

1) Installing and launching the application
2) Accurately calculating the speed of the moving vehicle
3) Detecting a speed limit traffic sign with high accuracy
4) Detecting a close traffic sign
5) Detecting a traffic sign from afar
6) Exceeding the speed limit warning
7) Detecting a car accident or a collision nearby

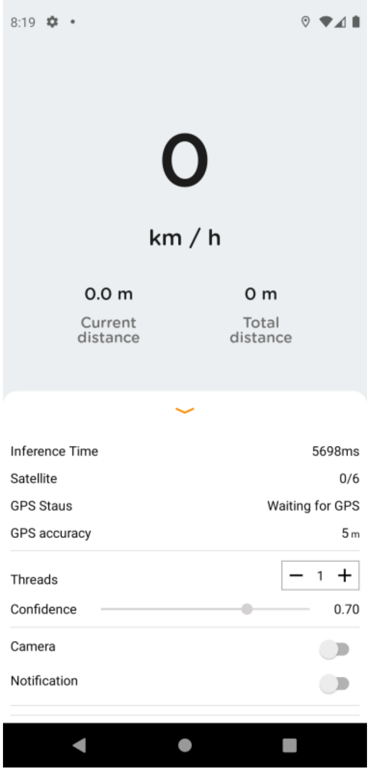
The tasks were distributed as follows:

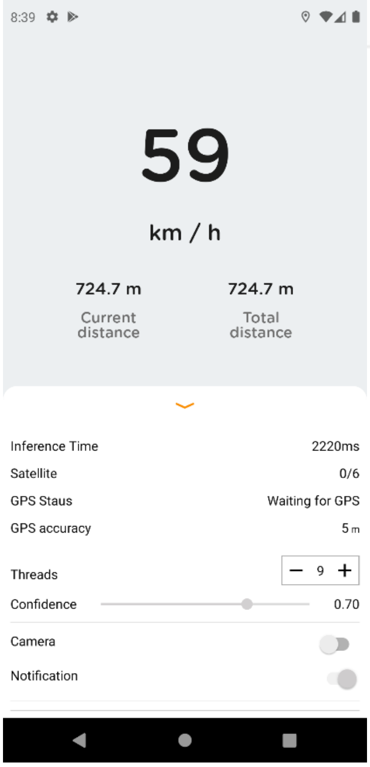
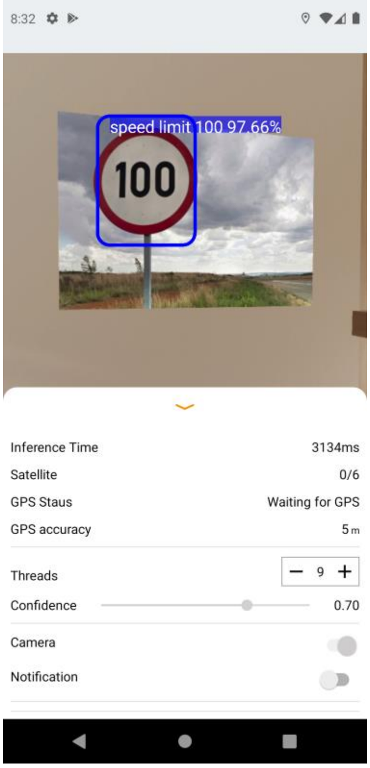
Shahad Alaradi	database and writing the app code
Shaikha Almutairi	database and writing the app code
Manar Fzaie	check the labels and write the app code
Raghad Alshammari	building the TensorFlow model
Maha Alkhars	training the model and printing the accuracy metrics

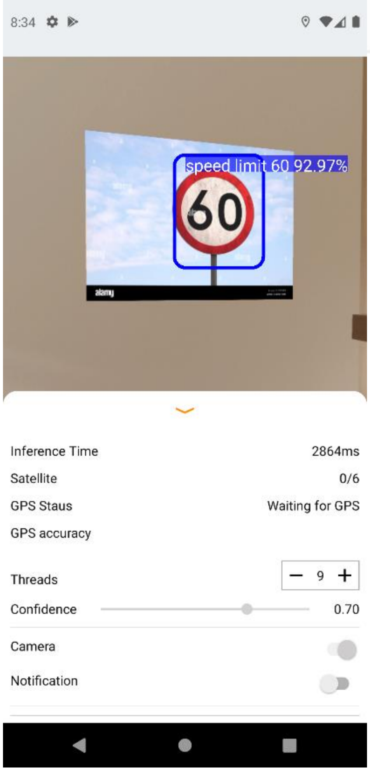
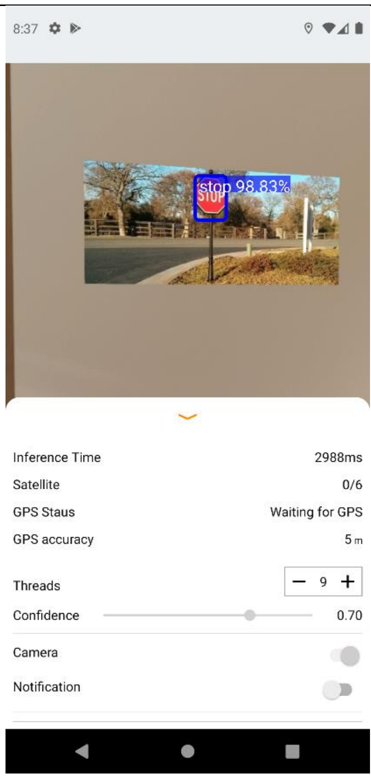
3. Design Optimization and Improvements [PI-6.c]

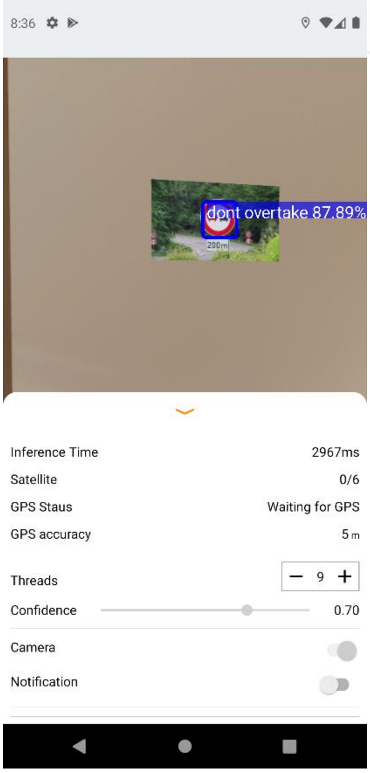
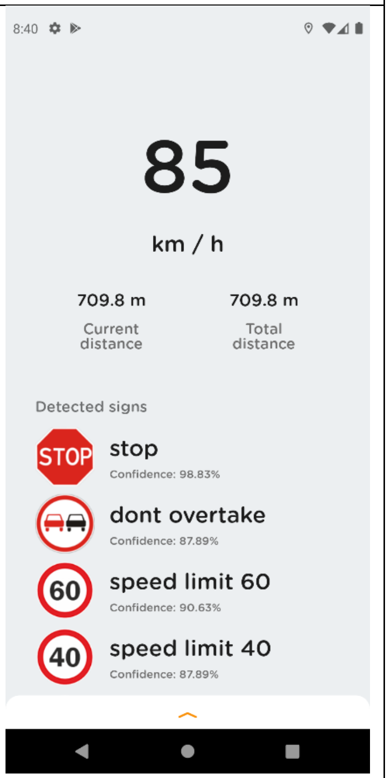
Upon testing the application, the application worked as predicted. The proposed copilot application was able to detect the traffic labels from distance with high accuracy. The alarming application sent a verbal notification when a critical traffic label was detected. Moreover, the application warns the driver when the speed limit is exceeded. However, due to the limitations of an existing dataset of nearby collisions and accidents, the application could not report them to the user in real time. The following are screenshots of the copilot application.

The experiments were conducted during the daytime and proved to be successful, and the results are shown below:

Experiment	Outcome	Screenshot/Details
Installing and launching the application	Success	

Calculating the speed of the moving vehicle	Success	
Detecting a 100 km/hr speed limit traffic sign with high accuracy	Success	

Detecting a 60 km/hr speed limit traffic sign with high accuracy	Success	
Detecting a stop sign	Success	

Detecting a traffic sign from afar	Success	
Exceeding the speed limit warning	Success	
Detecting a car accident or a collision nearby	Failure	Due to the limitations of no existing dataset of nearby collisions and accidents, the application could not report them in real-time

However, the traffic copilot app performs a computationally expensive task. As a result, there are several limitations:

- Due to phone camera limitations, traffic sign identification may fail at night and in poor lighting situations.
- The software is trained on German road signs. However, because the signs in many nations are extremely similar, the software should function there as well. Unfortunately, there is still little support for speed restrictions in Kuwait.
- There is no support for city/place signs that may reduce the speed limit in accordance with local traffic laws.
- Due to the camera's limited view angle, traffic signs in sharp turns are occasionally missed.
- Due to the computational power needed for the traffic detection to work correctly, low-performance phones may cause road signs to be missed or their detection would be false.

VI. General Discussion

1. Final Cost Analysis and Discussion

The development and building of the application did not result in extra costs due to the pre-existing availability of the smartphone used to test the application and the laptop used to code, develop, and simulate the application. The sensors and camera used are also available in the smartphone and hence no external camera or sensors were used in the development of this solution. A similar existing solution is an iOS app called Radarbot which is a GPS navigator that specializes in speed cameras. The combines real-time warnings with a radar detection alert system available offline. Radarbot is a strong program that combines radar alerts, real-time traffic alerts, and particular speed restriction warnings for various vehicles (cars, motorcycles, trucks, and commercial vehicles). However, the app costs 20 KWD to buy from the App Store to use all its features. Moreover, it does not detect real-time traffic signs using the phone camera and relies on a dataset that stores and retrieves all the info related to speed limits.

2. Commercializing the Project and Relevance to Region (Social, Cultural and Political issues)

Traffic apps have built quite a following to them over the years. This following has been on the increase year by year due to several benefits that have been shown not only individually but also on a societal scale. Traffic apps operate in a way it gives their users real-time updates using certain variables such as geographic information, cell phone data, and municipal sensors to enable them to reach their destination quicker and faster. With citizens managing their time and their car rides, the presence of cars in the streets will be lesser causing lower pollution. It is sound pollution from car honking or air pollution from harmful gasses being transmitted from vehicles. Furthermore, the air pollution caused by traffic congestion is the result of the increase in carbon monoxide emitted from said vehicles, contributing to the increase in ozone concentration and amplification of global warming. By lessening the problem of mere traffic, the smaller picture, we manage a more complicated one of pollution i.e., the bigger picture.

VII. Project Management

1. Encountered Problems and Proposed Solutions

Some of the encountered problems during our project are listed in the below table.

Encountered Problem	Proposed Solution
Due to phone camera limitations, traffic sign identification may fail at night and in poor lighting situations.	Advise users to use the application during the daytime. Another solution that may allow the phone camera to have a night vision can be further researched and developed.
There is no support for city/place signs that may reduce the speed limit in accordance with local traffic laws.	Collect a new training dataset of Kuwaiti traffic signs and train the model using them.
Due to the camera's limited view angle, traffic signs in sharp turns are occasionally missed.	Develop a solution that can incorporate the new flagship phone camera's wide-vision lenses.
Due to the computational power needed for the traffic detection to work correctly, low-performance phones may cause road signs to be missed or their detection would be false.	Develop the application to use a smaller/less complex model and to use less computational power so that it can run on older phones efficiently.
The software is trained on German road signs. However, because the signs in many nations are extremely similar, the software should function there as well.	Collect a new training dataset of Kuwaiti traffic signs and train the model using them.
Due to the limitations of no existing dataset of nearby collisions and accidents, the application could not report them in real-time	Ask the local Traffic Authority for permission to access the national datasets of accidents and traffic jams.

VIII. Conclusion and Future works

In conclusion, accidents are one of the most reasons behind death. Having an assisting system that helps and warns the driver about possible obstacles along the road might reduce the risks of car accidents. The project will be able to conduct a design that works as a copilot that notifies the driver of possible sources of risk. This design has gone through several engineering design steps that started with defining the problem, searching for possible solutions, evaluating them, and choosing one. The choosing procedure was conducted using a decision matrix that determined the solution based on weighted criteria based on their importance to decide the most suitable solution. Moreover, the design was conducted following several requirements and criteria. It was also designed considering several constraints. In the future, an accident warning system that can determine the location of the vehicle and notify the rescue department of the occurrence of an accident can be added and will be available to all people to use it.

IX. References

Fernandes, B. (2015). Mobile Application for Automatic Accident Detection and multimodal alert. *IEEE*, 1-5.

Godsmark, P. (2014). Autonomous Vehicles: Are we ready? *Focus on the future*.

KIM, J., Kim, K., Yoon, D., Koo, Y., & Han, W. (2016). Fusion of Driver-information Based Driver Status Recognition for Co-pilot System. *IV*, 19-22.

Levinson, J., Askeland, J., Becker, J., Dolson, J., Held, D., & Kammel, S. (2011). Towards Fully Autonomous Driving: Systems and Algorithms. *IV*, 5-9.

Michalke, T., & Kastner, R. (2011). The Attentive Co-Pilot. *IEEE*, 1-18.

Noh, S. (2015). Co-pilot Agent for Vehicle Cooperative and Autonomous Driving. *ETRI Journal*, 1-12.

Noh, S., An, K., & Han, W. (2015). Situation Assessment and Behavior Decision for Vehicle/Driver Cooperative Driving in Highway Environments. *IEEE*, 1-8.

Rayle, Shaheen, S., Chan, N., Dai, D., & Cervero, R. (2014). App-Based, On-Demand Ride Services. *UCTC*.

Thrun, S., Laugier, C., & Yoder, D. (2012). Autonomous Driving. *handbook of intell*, 12171-1310.

Urmson, C. (2008). Autonomous driving in urban environments. *Field Robot*, 425-466.

X. Appendices

1. TensorFlow Model (JuPyter Notebook)

Importing necessary tools

```
import os
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
import matplotlib.pyplot as mpimg

import tensorflow as tf
print("TF version: ", tf.__version__)
```

```
TF version:  2.8.0
```

Getting the dataset ready

```
from google.colab import drive
drive.mount('/content/drive',force_remount=True)
```

```
train_df = pd.read_csv('/content/drive/MyDrive/gtsrb-german-traffic-sign/Train.
train_df.head()
```

```
train_df.describe()
```

```
train_df = train_df.drop(['Width', 'Height', 'Roi.X1', 'Roi.Y1', 'Roi.X2', 'Roi
train_df.head()
```

Getting images and their labels

```
# Load sign names file
sign_names = pd.read_csv("/content/drive/MyDrive/gtsrb-german-traffic-sign/sign
sign_names.set_index("ClassId")

sign_names.head(n=10)
```

```
# Create pathnames from image Id's
filenames = ['/content/drive/MyDrive/gtsrb-german-traffic-sign/' + fname for fn
filenames[:10]
```

```
from google.colab import drive
drive.mount('/content/drive')
```

```
len(filenames)
```

```
labels = train_df['ClassId'].to_numpy()
labels.shape[0]
```

```
unique_signs = np.unique(labels)
len(unique_signs)
```

```
def group_img_id_to_lbl(lbs_ids, lbs_names):
    """
    Utility function to group images by label
    """
    arr_map = []
    for i in range(0, lbs_ids.shape[0]):
        label_id = lbs_ids[i]
        label_name = lbs_names[lbs_names["ClassId"] == label_id]["SignName"].va
        arr_map.append({"img_id": i, "label_id": label_id, "label_name": label_

    return pd.DataFrame(arr_map)
```

```
ids_to_signnames = group_img_id_to_lbl(labels, sign_names)
ids_to_signnames
```

	img_id	label_id	label_name
0	0	20	Dangerous curve to the right
1	1	20	Dangerous curve to the right
2	2	20	Dangerous curve to the right
3	3	20	Dangerous curve to the right
4	4	20	Dangerous curve to the right
...	...	...	...
39204	39204	42	End of no passing by vehicles over 3.5 metric ...
39205	39205	42	End of no passing by vehicles over 3.5 metric ...
39206	39206	42	End of no passing by vehicles over 3.5 metric ...
39207	39207	42	End of no passing by vehicles over 3.5 metric ...
39208	39208	42	End of no passing by vehicles over 3.5 metric ...

39209 rows × 3 columns

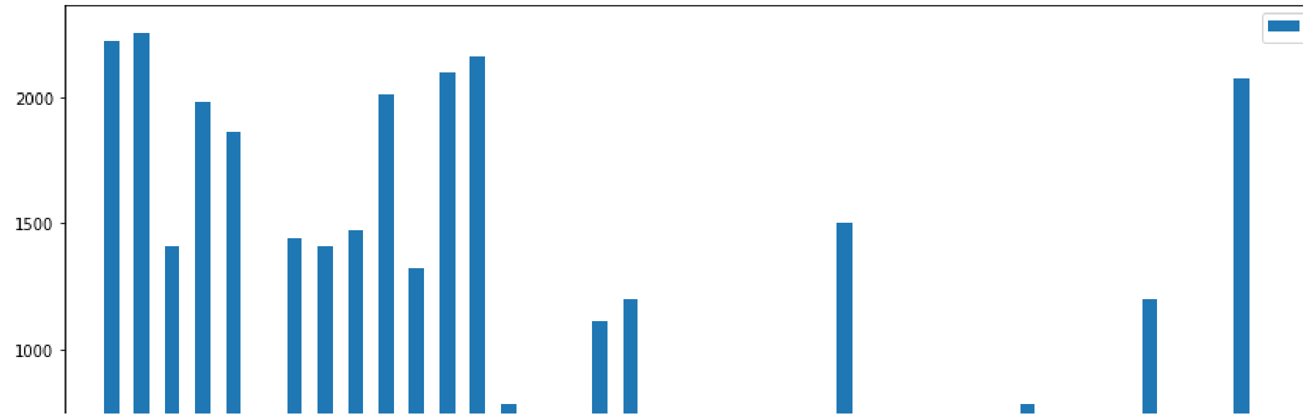
```
labels_numpy = ids_to_signnames.to_numpy()
```

```
count_of_each_sign = pd.pivot_table(ids_to_signnames,index=["label_id","label_n
count_of_each_sign
```

		img_id
label_id	label_name	
0	**Speed limit (20km/h)**	210
1	**Speed limit (30km/h)**	2220
2	**Speed limit (50km/h)**	2250
3	**Speed limit (60km/h)**	1410
4	**Speed limit (70km/h)**	1980
5	**Speed limit (80km/h)**	1860
6	**End of speed limit (80km/h)**	420
7	**Speed limit (100km/h)**	1440
8	**Speed limit (120km/h)**	1410
9	**No passing**	1470
10	**No passing for vehicles over 3.5 metric tons**	2010
11	**Right-of-way at the next intersection**	1320
12	**Priority road**	2100
13	**Yield**	2160

```
count_of_each_sign.plot(kind='bar', figsize=(15, 7))
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fb42ec88910>
```



Visualizing the dataset

```
labels_numpy
```

```
array([[0, 20, 'Dangerous curve to the right'],
       [1, 20, 'Dangerous curve to the right'],
       [2, 20, 'Dangerous curve to the right'],
       ...,
       [39206, 42, 'End of no passing by vehicles over 3.5 metric tons'],
       [39207, 42, 'End of no passing by vehicles over 3.5 metric tons'],
       [39208, 42, 'End of no passing by vehicles over 3.5 metric tons']],
      dtype=object)
```

```
for n in range(5):
  plt.figure()
  i = np.random.randint(0, high=len(filenames), dtype='int')
  plt.imshow(mpimg.imread(filenames[i]))
  plt.title(labels_numpy[i][2])
  plt.axis('off')
```

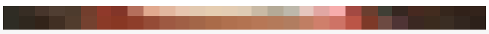
Yield

No vehicles

Speed limit (30km/h)

Double curve

italicised text## One-hot encoding



```
labels = tf.keras.utils.to_categorical(labels, 43)
labels
```

```
array([[0., 0., 0., ..., 0., 0., 0.],
       [0., 0., 0., ..., 0., 0., 0.],
```

```
        [0., 0., 0., ..., 0., 0., 0.],
        ...,
        [0., 0., 0., ..., 0., 0., 1.],
        [0., 0., 0., ..., 0., 0., 1.],
        [0., 0., 0., ..., 0., 0., 1.]], dtype=float32)
```

```
labels[0]
```

```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0.], dtype=float32)
```

```
len(labels)
```

```
39209
```

Splitting our data into train and validation sets

```
# Create X & y variables
X = filenames
y = labels

X_train, X_val, y_train, y_val = train_test_split(X, y, test_size = 0.2, random
len(X_train), len(y_train), len(X_val), len(y_val)
```

```
(31367, 31367, 7842, 7842)
```

Processing image into Tensors

> Indented block

```
# importing necessary tools
import datetime
import os
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Activation, Dropout, Flatten, Dense, Conv2D
from tensorflow.math import confusion_matrix
import matplotlib.image as mpimg

import tensorflow as tf
print("TF version: ", tf.__version__)
```

```
TF version:  2.8.0
```

```
device_name = tf.test.gpu_device_name()
if "GPU" not in device_name:
    print("No")
else:
    print(device_name)
```

```
/device:GPU:0
```

```
print(tf.test.is_gpu_available())
```

```
WARNING:tensorflow:From <ipython-input-3-ae932be897c3>:1: is_gpu_available (from tens
Instructions for updating:
Use `tf.config.list_physical_devices('GPU')` instead.
True
```

Getting our data ready

```
from google.colab import drive
drive.mount('/content/drive',force_remount=True)
```

```
Mounted at /content/drive
```

```
train_df = pd.read_csv('/content/drive/MyDrive/gtsrb-german-traffic-sign/Train.
train_df.head()
```

	Width	Height	Roi.X1	Roi.Y1	Roi.X2	Roi.Y2	ClassId	Pa
0	27	26	5	5	22	20	20	Train/20/00020_00000_00000.p
1	28	27	5	6	23	22	20	Train/20/00020_00000_00001.p
2	29	26	6	5	24	21	20	Train/20/00020_00000_00002.p
3	28	27	5	6	23	22	20	Train/20/00020_00000_00003.p

```
train_df.describe()
```

	Width	Height	Roi.X1	Roi.Y1	Roi.X2	Roi
count	39209.000000	39209.000000	39209.000000	39209.000000	39209.000000	39209.000
mean	50.835880	50.328930	5.999515	5.962381	45.197302	44.728
std	24.306933	23.115423	1.475493	1.385440	23.060157	21.971
min	25.000000	25.000000	0.000000	5.000000	20.000000	20.000
25%	35.000000	35.000000	5.000000	5.000000	29.000000	30.000
50%	43.000000	43.000000	6.000000	6.000000	38.000000	38.000
75%	58.000000	58.000000	6.000000	6.000000	53.000000	52.000
max	243.000000	225.000000	20.000000	20.000000	223.000000	205.000

```
train_df = train_df.drop(['Width', 'Height', 'Roi.X1', 'Roi.Y1', 'Roi.X2', 'Roi
train_df.head()
```

	ClassId	Path
0	20	Train/20/00020_00000_00000.png
1	20	Train/20/00020_00000_00001.png
2	20	Train/20/00020_00000_00002.png
3	20	Train/20/00020_00000_00003.png
4	20	Train/20/00020_00000_00004.png

Getting images and their labels

```
# Load sign names file
sign_names = pd.read_csv("/content/drive/MyDrive/gtsrb-german-traffic-sign/sign
sign_names.set_index("ClassId")

sign_names.head(n=10)
```

	ClassId	SignName
0	0	Speed limit (20km/h)
1	1	Speed limit (30km/h)
2	2	Speed limit (50km/h)
3	3	Speed limit (60km/h)
4	4	Speed limit (70km/h)
5	5	Speed limit (80km/h)
6	6	End of speed limit (80km/h)
7	7	Speed limit (100km/h)
8	8	Speed limit (120km/h)

```
label_map = sign_names['SignName'].to_dict()
```

```
# Create pathnames from image Id's
filenames = ['/content/drive/MyDrive/gtsrb-german-traffic-sign/' + fname for fn
filenames[:10]
```

```
['/content/drive/MyDrive/gtsrb-german-traffic-sign/Train/20/00020_00000_00000.png',
 '/content/drive/MyDrive/gtsrb-german-traffic-sign/Train/20/00020_00000_00001.png',
 '/content/drive/MyDrive/gtsrb-german-traffic-sign/Train/20/00020_00000_00002.png',
 '/content/drive/MyDrive/gtsrb-german-traffic-sign/Train/20/00020_00000_00003.png',
 '/content/drive/MyDrive/gtsrb-german-traffic-sign/Train/20/00020_00000_00004.png',
 '/content/drive/MyDrive/gtsrb-german-traffic-sign/Train/20/00020_00000_00005.png',
 '/content/drive/MyDrive/gtsrb-german-traffic-sign/Train/20/00020_00000_00006.png',
 '/content/drive/MyDrive/gtsrb-german-traffic-sign/Train/20/00020_00000_00007.png',
 '/content/drive/MyDrive/gtsrb-german-traffic-sign/Train/20/00020_00000_00008.png',
 '/content/drive/MyDrive/gtsrb-german-traffic-sign/Train/20/00020_00000_00009.png']
```

```
len(filenames)
```

```
39209
```

```
labels = train_df['ClassId'].to_numpy()
labels.shape[0]
```

```
39209
```

```
unique_signs = np.unique(labels)
len(unique_signs)
```

```
43
```

```
def group_img_id_to_lbl(lbs_ids, lbs_names):
    """
    Utility function to group images by label
```

```
    """
    arr_map = []
    for i in range(0, lbs_ids.shape[0]):
        label_id = lbs_ids[i]
        label_name = lbs_names[lbs_names["ClassId"] == label_id]["SignName"].va
        arr_map.append({"img_id": i, "label_id": label_id, "label_name": label_

    return pd.DataFrame(arr_map)
```

```
ids_to_signnames = group_img_id_to_lbl(labels, sign_names)
ids_to_signnames
```

	img_id	label_id	label_name
0	0	20	Dangerous curve to the right
1	1	20	Dangerous curve to the right
2	2	20	Dangerous curve to the right
3	3	20	Dangerous curve to the right
4	4	20	Dangerous curve to the right
...	...	...	...
39204	39204	42	End of no passing by vehicles over 3.5 metric ...
39205	39205	42	End of no passing by vehicles over 3.5 metric ...
39206	39206	42	End of no passing by vehicles over 3.5 metric ...
39207	39207	42	End of no passing by vehicles over 3.5 metric ...
39208	39208	42	End of no passing by vehicles over 3.5 metric ...

39209 rows × 3 columns

```
labels_numpy = ids_to_signnames.to_numpy()
```

```
count_of_each_sign = pd.pivot_table(ids_to_signnames,index=["label_id","label_n
count_of_each_sign
```

		img_id
label_id	label_name	
0	Speed limit (20km/h)	210
1	Speed limit (30km/h)	2220
2	Speed limit (50km/h)	2250
3	Speed limit (60km/h)	1410
4	Speed limit (70km/h)	1980
5	Speed limit (80km/h)	1860
6	End of speed limit (80km/h)	420
7	Speed limit (100km/h)	1440
8	Speed limit (120km/h)	1410
9	No passing	1470
10	No passing for vehicles over 3.5 metric tons	2010
11	Right-of-way at the next intersection	1320
12	Priority road	2100
13	Yield	2160
14	Stop	780
15	No vehicles	630
16	Vehicles over 3.5 metric tons prohibited	420
17	No entry	1110
18	General caution	1200
19	Dangerous curve to the left	210
20	Dangerous curve to the right	360
21	Double curve	330
22	Bumpy road	390

Visualizing the dataset

25 Road work 1500

```
labels_numpy
```

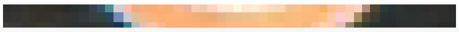
```
array([[0, 20, 'Dangerous curve to the right'],
       [1, 20, 'Dangerous curve to the right'],
       [2, 20, 'Dangerous curve to the right'],
       ...,
       [39206, 42, 'End of no passing by vehicles over 3.5 metric tons'],
       [39207, 42, 'End of no passing by vehicles over 3.5 metric tons'],
```

```
       [39208, 42, 'End of no passing by vehicles over 3.5 metric tons']],
```

```
for n in range(5):
  plt.figure()
  i = np.random.randint(0, high=len(filenames), dtype='int')
  plt.imshow(mpimg.imread(filenames[i]))
  plt.title(labels_numpy[i][2])
  plt.axis('off')
```

Speed limit (30km/h)

One-hot encoding



```
# Converting the labels into one hot encoding
labels = tf.keras.utils.to_categorical(labels, 43)
labels[0]
```

```
array([0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0.], dtype=float32)
```

```
len(labels)
```

```
39209
```

Creating Validation set

```
# Create X & y variables
X = filenames
y = labels

# Splitting our data into train and validation sets
X_train, X_val, y_train, y_val = train_test_split(X, y, test_size = 0.2, random
len(X_train), len(y_train), len(X_val), len(y_val)
```

```
(31367, 31367, 7842, 7842)
```

Processing image and turning into Tensors

```
IMG_SIZE = 32

def process_image(image_path):
    """
    Takes an image file path and turns the image into a Tensor.
    """
    # Read in an image file
    image = tf.io.read_file(image_path)
    # Turn the jpeg image into numerical Tensor with 3 colour channels (Red, Gr
```

```
    image = tf.image.decode_png(image, channels=3)
    # Convert the colour channel values from 0-255 to 0-1 values
    image = tf.image.convert_image_dtype(image, tf.float32)
    # Resize the image to our desired value (32, 32)
    image = tf.image.resize(image, size=[IMG_SIZE, IMG_SIZE])
    return image
```

Turning data into batches

```
# Create a simple function to return tuple
def get_image_label (image_path, label):
    """
    Takes an image file path name and the assosciated label,
    processes the image and reutrns a typle of (image, label).
    """
    image = process_image(image_path)
    return image, label
```

```
# Define batch size
BATCH_SIZE = 64

# Create a function to turn data into batches
def create_data_batches (X, y=None, batch_size=BATCH_SIZE, valid_data=False, te
    """
    Creates batches of data out of image (X) and label (y) pairs.
    Shuffles the data if it's training data but doesn't shuffle if it's validat
    a.
    Also accepts test data as input (no labels).
    """
    # If the data is a test dataset, we probably don't have have labels
    if test_data:
        print("Creating test data batches...")
        data = tf.data.Dataset.from_tensor_slices((tf.constant(X)))
        data_batch = data.map(process_image).batch(BATCH_SIZE)
    # If the data is a valid dataset, we don't need to shuffle it
    elif valid_data:
        print("Creating validation dataset batches...")
        data = tf.data.Dataset.from_tensor_slices((tf.constant(X), tf.constant(
        # Create (image, label) tuples (this also turns the iamge path into a p
        data_batch = data.map(get_image_label).batch(BATCH_SIZE)
    else:
        print("Creating training dataset batches...")
        # Turn filepaths and labels into Tensors
        data = tf.data.Dataset.from_tensor_slices((tf.constant(X), tf.constant(
        # Shuffling pathnames and labels before mapping image processor functio
        data = data.shuffle(buffer_size=len(X))
        # Create (image, label) tuples (this also turns the iamge path into a p
```

```
        data_batch = data.map(get_image_label).batch(BATCH_SIZE)
    return data batch

# Creating training and validation batches
train_data = create_data_batches(X_train, y_train)
val_data = create_data_batches(X_val, y_val, valid_data=True)
```

```
Creating training dataset batches...
Creating validation dataset batches...
```

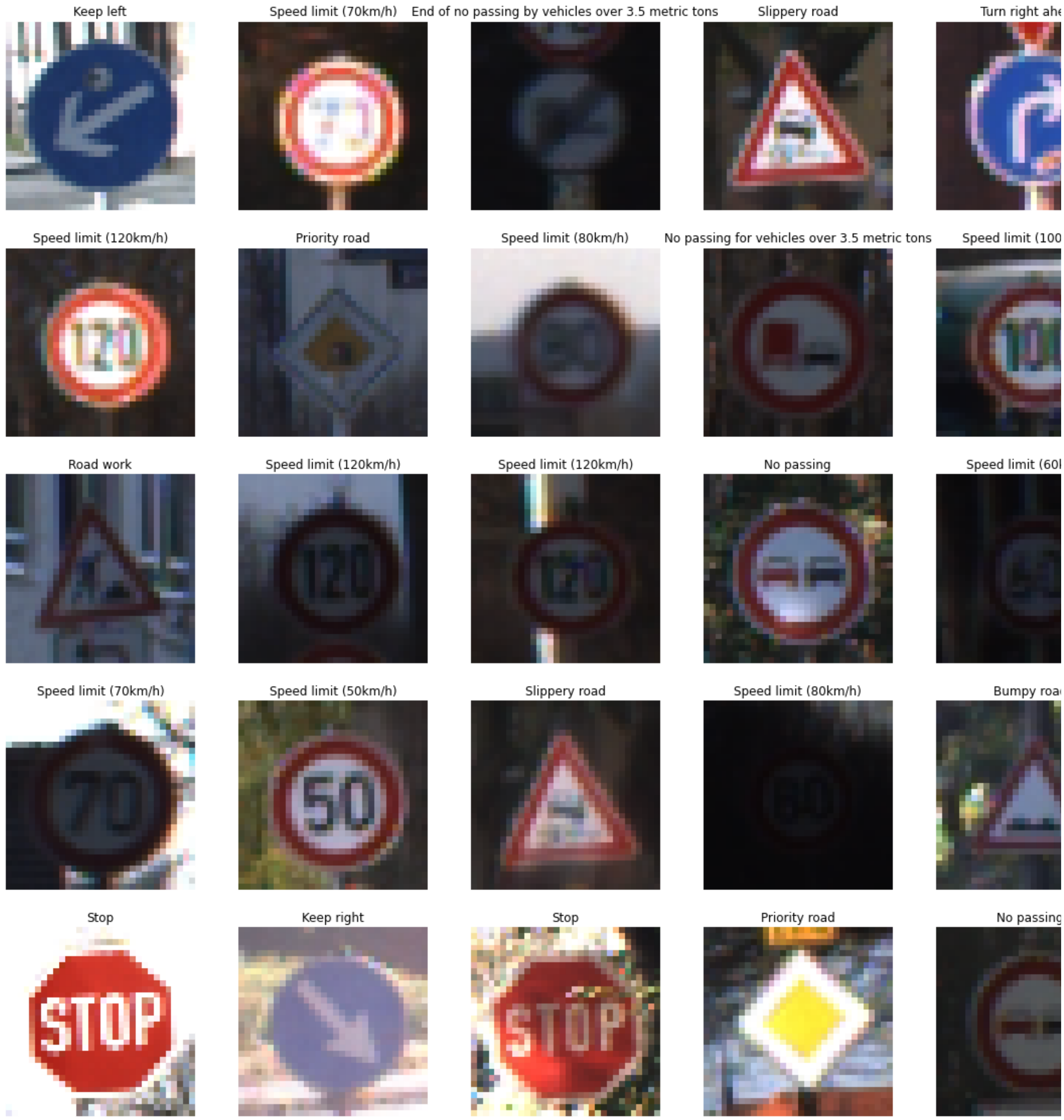
```
# Check out the different attributes of our data batches
train_data.element_spec, val_data.element_spec
```

```
((TensorSpec(shape=(None, 32, 32, 3), dtype=tf.float32, name=None),
  TensorSpec(shape=(None, 43), dtype=tf.float32, name=None)),
 (TensorSpec(shape=(None, 32, 32, 3), dtype=tf.float32, name=None),
  TensorSpec(shape=(None, 43), dtype=tf.float32, name=None)))
```

Visualizing Data Batches

```
# Create a function for viewing images in a data batch
def show_25_images (images, labels):
    """
    Displays a plot of 25 images and their labels from a data batch.
    """
    plt.figure(figsize=(20,20))
    for i in range(25):
        ax = plt.subplot(5, 5, i+1)
        plt.imshow(images[i])
        plt.title(label_map[unique_signs[labels[i].argmax()]])
        plt.axis("off")
```

```
# Visualizing traing batch
train_images, train_labels = next(train_data.as_numpy_iterator())
show_25_images(train_images, train_labels)
```



Building the model

```
# Setup input shape to the model
INPUT_SHAPE = [IMG_SIZE, IMG_SIZE, 3]

# Setup the output shape
OUTPUT_SHAPE = len(unique_signs)
```

```
# Creating CNN Model
def traffic_sign_net(input_shape):
    model = Sequential()
    model.add(Conv2D(filters=32, kernel_size=(5, 5), activation='relu', input_s
```

```
    model.add(Conv2D(filters=32, kernel_size=(5, 5), activation='relu'))
    model.add(MaxPool2D(pool_size=(2, 2)))
    model.add(Dropout(rate=0.25))
    model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
    model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
    model.add(MaxPool2D(pool_size=(2, 2)))
    model.add(Dropout(rate=0.25))
    model.add(Flatten())
    model.add(Dense(256, activation='relu'))
    model.add(Dropout(rate=0.5))
    model.add(Dense(43, activation='softmax'))
    return model
```

```
# Create a function that creates model
def create_model(input_shape=INPUT_SHAPE, output_shape=OUTPUT_SHAPE):
    # Setup the model layers
    model = traffic_sign_net(input_shape=input_shape)
    # Compile the model
    print("Compiling the model")
    model.compile(
      optimizer=tf.keras.optimizers.Adam(),
      loss='categorical_crossentropy',
      metrics=['accuracy']
    )
    return model
```

```
model = create_model()
model.summary()
```

```
Compiling the model
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d (Conv2D)             (None, 28, 28, 32)        2432      
                                                                 
 conv2d_1 (Conv2D)           (None, 24, 24, 32)        25632     
                                                                 
 max_pooling2d (MaxPooling2D  (None, 12, 12, 32)       0         
 )                                                               
                                                                 
 dropout (Dropout)           (None, 12, 12, 32)        0         
                                                                 
 conv2d_2 (Conv2D)           (None, 10, 10, 64)        18496     
                                                                 
 conv2d_3 (Conv2D)           (None, 8, 8, 64)          36928     
                                                                 
 max_pooling2d_1 (MaxPooling  (None, 4, 4, 64)         0         
 2D)                                                             
                                                                 
 dropout_1 (Dropout)         (None, 4, 4, 64)          0         
```

```
 flatten (Flatten)           (None, 1024)              0

 dense (Dense)               (None, 256)               262400

 dropout_2 (Dropout)         (None, 256)               0

 dense_1 (Dense)             (None, 43)                11051

=================================================================
Total params: 356,939
Trainable params: 356,939
Non-trainable params: 0
_________________________________________________________________
```

Training our model

```
NUM_EPOCHS = 10
```

```
# Build a fn to train and return a trained model
def train_model():
    """
    Trains a given model and returns the trained version.
    """
    # Create a model
    model = create_model()

    # Fit the model to the data passing it the callbacks we created
    model.fit(x=train_data,
        epochs=NUM_EPOCHS,
        validation_data=val_data,
        validation_freq=1,
             )
    return model
```

```
# Fit the model to data
model = train_model()
```

```
Compiling the model
Epoch 1/10
491/491 [==============================] - 4311s 9s/step - loss: 1.7064 - accuracy: 0
Epoch 2/10
491/491 [==============================] - 59s 120ms/step - loss: 0.3169 - accuracy:
Epoch 3/10
491/491 [==============================] - 58s 117ms/step - loss: 0.1704 - accuracy:
Epoch 4/10
491/491 [==============================] - 60s 121ms/step - loss: 0.1281 - accuracy:
Epoch 5/10
491/491 [==============================] - 60s 122ms/step - loss: 0.1116 - accuracy:
Epoch 6/10
491/491 [==============================] - 59s 120ms/step - loss: 0.0857 - accuracy:
Epoch 7/10
```

```
491/491 [==============================] - 60s 122ms/step - loss: 0.0738 - accuracy:
Epoch 8/10
491/491 [==============================] - 60s 122ms/step - loss: 0.0655 - accuracy:
Epoch 9/10
491/491 [==============================] - 59s 120ms/step - loss: 0.0594 - accuracy:
Epoch 10/10
491/491 [==============================] - 59s 120ms/step - loss: 0.0554 - accuracy:
```

```
# Save the entire model as a SavedModel.
!mkdir -p saved_model
model.save('/drive/MyDrive/saved_model/my_model')
new_model = tf.keras.models.load_model('/drive/MyDrive/saved_model/my_model')

# Check its architecture
new_model.summary()
```

```
INFO:tensorflow:Assets written to: /drive/MyDrive/saved_model/my_model/assets
Model: "sequential_2"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 conv2d_8 (Conv2D)           (None, 28, 28, 32)        2432      
                                                                 
 conv2d_9 (Conv2D)           (None, 24, 24, 32)        25632     
                                                                 
 max_pooling2d_4 (MaxPooling  (None, 12, 12, 32)       0         
 2D)                                                             
                                                                 
 dropout_6 (Dropout)         (None, 12, 12, 32)        0         
                                                                 
 conv2d_10 (Conv2D)          (None, 10, 10, 64)        18496     
                                                                 
 conv2d_11 (Conv2D)          (None, 8, 8, 64)          36928     
                                                                 
 max_pooling2d_5 (MaxPooling  (None, 4, 4, 64)         0         
 2D)                                                             
                                                                 
 dropout_7 (Dropout)         (None, 4, 4, 64)          0         
                                                                 
 flatten_2 (Flatten)         (None, 1024)              0         
                                                                 
 dense_4 (Dense)             (None, 256)               262400    
                                                                 
 dropout_8 (Dropout)         (None, 256)               0         
                                                                 
 dense_5 (Dense)             (None, 43)                11051     
                                                                 
=================================================================
Total params: 356,939
Trainable params: 356,939
Non-trainable params: 0
_________________________________________________________________
```

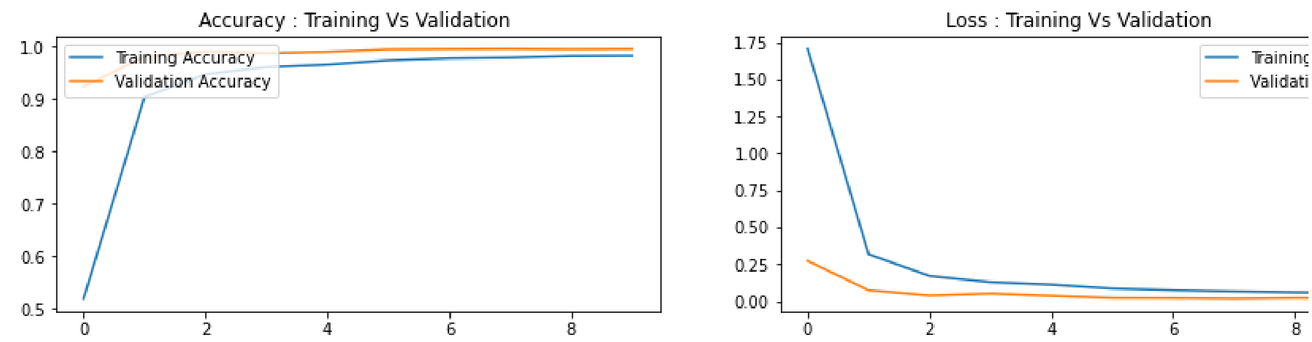
```
accuracy = model.history.history['accuracy']
loss = model.history.history['loss']
```

```
validation_loss = model.history.history['val_loss']
validation_accuracy = model.history.history['val_accuracy']


plt.figure(figsize=(15, 7))
plt.subplot(2, 2, 1)
plt.plot(range(NUM_EPOCHS), accuracy, label='Training Accuracy')
plt.plot(range(NUM_EPOCHS), validation_accuracy, label='Validation Accuracy')
plt.legend(loc='upper left')
plt.title('Accuracy : Training Vs Validation ')



plt.subplot(2, 2, 2)
plt.plot(range(NUM_EPOCHS), loss, label='Training Loss')
plt.plot(range(NUM_EPOCHS), validation_loss, label='Validation Loss')
plt.title('Loss : Training Vs Validation ')
plt.legend(loc='upper right')
plt.show()
```



Creating test dataset batches

```
test_df = pd.read_csv('/content/drive/MyDrive/gtsrb-german-traffic-sign/Test.cs
test_df = test_df.drop(['Width', 'Height', 'Roi.X1', 'Roi.Y1', 'Roi.X2', 'Roi.Y
test_df.head()
```

	ClassId	Path
0	16	Test/00000.png
1	1	Test/00001.png
2	38	Test/00002.png
3	33	Test/00003.png
4	11	Test/00004.png

```
test_img_paths = ['/content/drive/MyDrive/gtsrb-german-traffic-sign/' + path fo
test_img_paths[:10]
```

```
['/content/drive/MyDrive/gtsrb-german-traffic-sign/Test/00000.png',
 '/content/drive/MyDrive/gtsrb-german-traffic-sign/Test/00001.png',
 '/content/drive/MyDrive/gtsrb-german-traffic-sign/Test/00002.png',
 '/content/drive/MyDrive/gtsrb-german-traffic-sign/Test/00003.png',
 '/content/drive/MyDrive/gtsrb-german-traffic-sign/Test/00004.png',
 '/content/drive/MyDrive/gtsrb-german-traffic-sign/Test/00005.png',
 '/content/drive/MyDrive/gtsrb-german-traffic-sign/Test/00006.png',
 '/content/drive/MyDrive/gtsrb-german-traffic-sign/Test/00007.png',
 '/content/drive/MyDrive/gtsrb-german-traffic-sign/Test/00008.png',
 '/content/drive/MyDrive/gtsrb-german-traffic-sign/Test/00009.png']
```

```
X_test = create_data_batches(test_img_paths, test_data=True)
y_test = list(test_df['ClassId'])
y_test[:10]
```

```
Creating test data batches...
[16, 1, 38, 33, 11, 38, 18, 12, 25, 35]
```

Making and Evaluating predictions using a trained model on test data

```
predictions = model.predict(X_test, verbose=1)
```

```
198/198 [==============================] - 1370s 7s/step
```

```
# Function to convert probabilities to labels
def get_pred_label(prediction_probabilities):
    """
    Turns an array of prediction probabilities into a label.
    """
    return unique_signs[np.argmax(prediction_probabilities)]
```

```
# Turning probabilities to labels
pred_labels = []
for i in predictions:
    pred_labels.append(get_pred_label(i))
pred_labels[:10]
```

```
[16, 1, 38, 33, 11, 38, 18, 12, 25, 35]
```

```
# Getting the accuracy of the model on test data
acc = accuracy_score(y_test, pred_labels)
acc
```

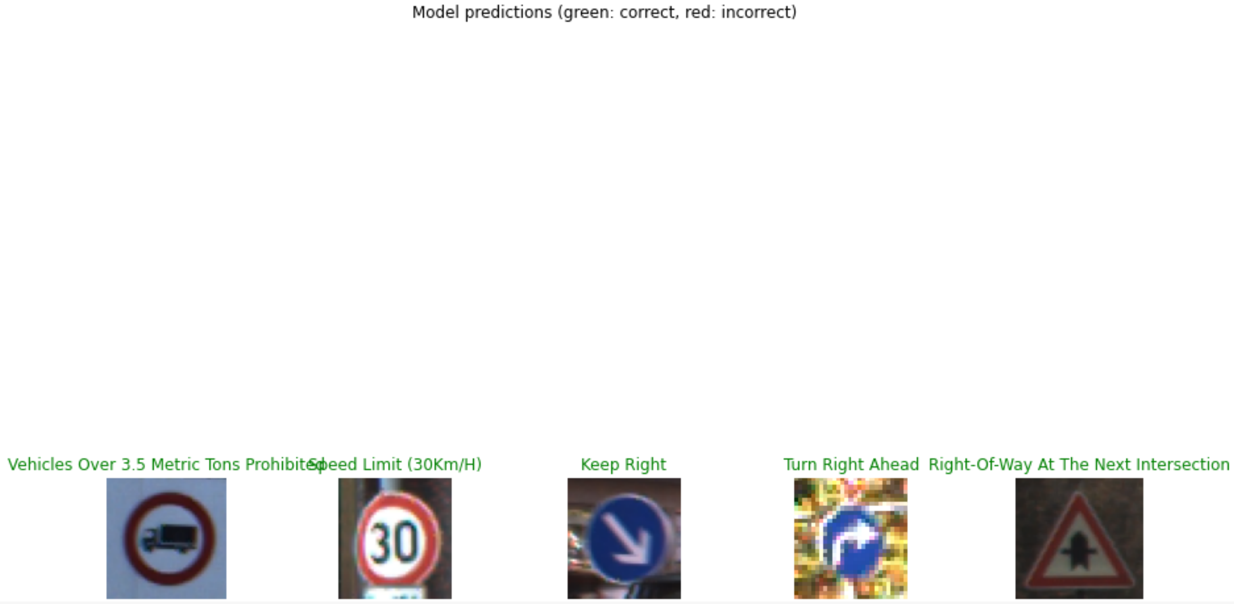
```
0.9782264449722882
```

```
batch_size = 100
num_plot_column = 5
num_plot_row = batch_size // num_plot_column + (batch_size % num_plot_column >

plt.figure(figsize=(15,50))
plt.subplots_adjust(hspace=0.5)
for n in range(batch_size):
  plt.subplot(num_plot_row,num_plot_column,n+1)
  plt.imshow(mpimg.imread(test_img_paths[n]))
  color = "green" if pred_labels[n] == test_df['ClassId'][n] else "red"
  plt.title(label_map[pred_labels[n]].title(), color=color)
  plt.axis('off')
_ = plt.suptitle("Model predictions (green: correct, red: incorrect)")

print("Accuracy of the shown eval batch: " + str(accuracy_score(y_test, pred_la
```

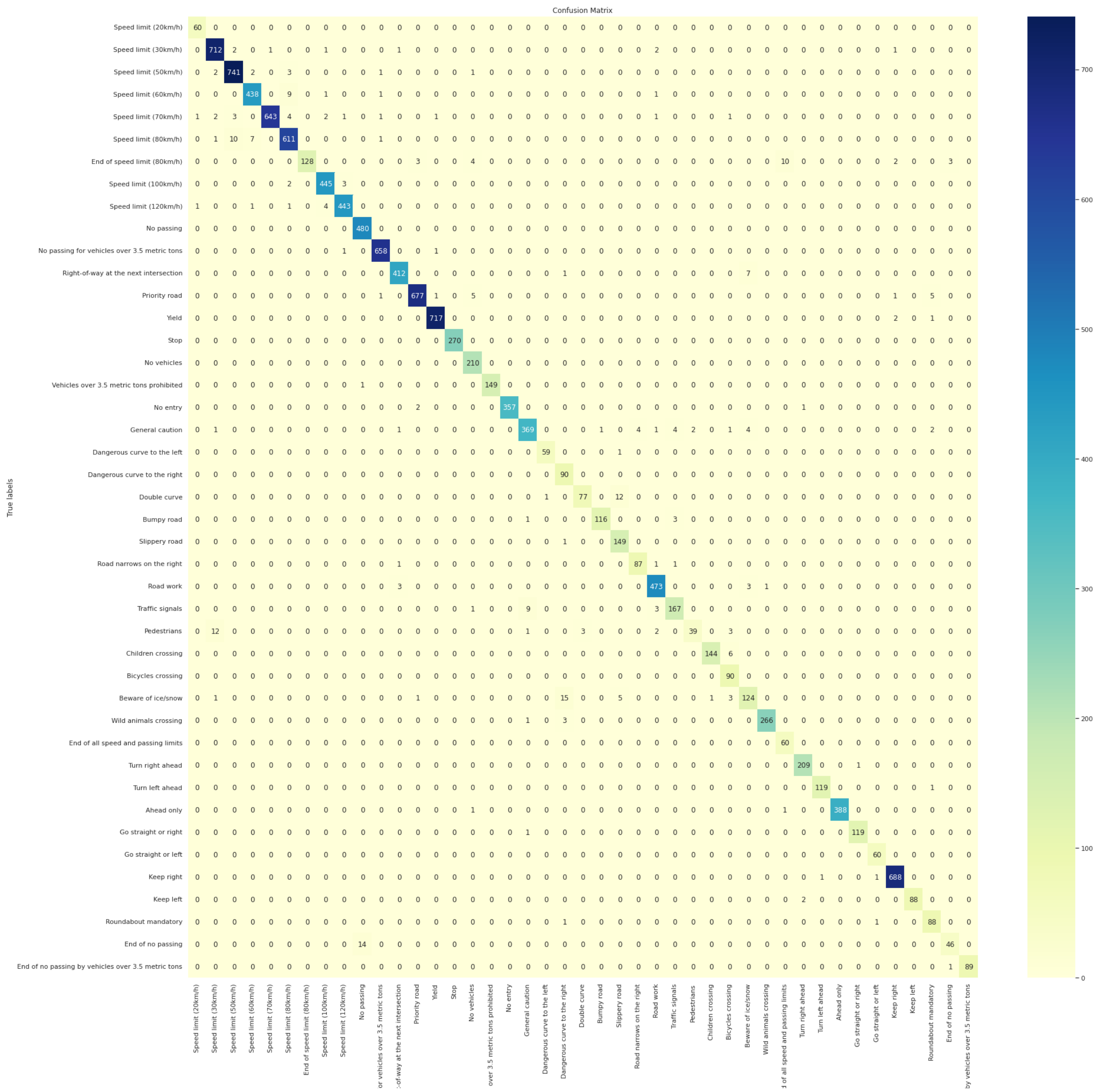
```
Accuracy of the shown eval batch: 0.9782264449722882
```




```
import seaborn as sns
```

```
test_labels = []
preds_labels = []
for n in range(len(y_test)):
    test_labels.append(label_map[y_test[n]])
    preds_labels.append(label_map[pred_labels[n]])
```

```
cm = confusion_matrix(y_test, pred_labels)
ax = plt.subplot()
sns.set(rc = {'figure.figsize':(50,50)})
axis_labels = sign_names['SignName'].to_numpy()
sns.heatmap(cm, annot=True, fmt='g', ax=ax, cmap="YlGnBu", xticklabels=axis_lab
# labels, title and ticks
ax.set_xlabel('Predicted labels');ax.set_ylabel('True labels'); ax.set_title('C
```



Converting to tflite model

```
import time
t = time.time()

export_path = "/content/drive/MyDrive/output/model{}".format(int(t))
model.save(export_path, save_format='tf')

export_path
```

```
INFO:tensorflow:Assets written to: /content/drive/MyDrive/output/model1649106813/asse
INFO:tensorflow:Assets written to: /content/drive/MyDrive/output/model1649106813/asse
'/content/drive/MyDrive/output/model1649106813'
```

```
OUTPUT_TFLITE_MODEL = "/content/drive/MyDrive/output/saved_model.tflite"
```

```
# Convert the model
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()

# Save the TF Lite model.
with tf.io.gfile.GFile(OUTPUT_TFLITE_MODEL,'wb') as f:
  f.write(tflite_model)
```

```
INFO:tensorflow:Assets written to: /tmp/tmpuqvusw4b/assets
INFO:tensorflow:Assets written to: /tmp/tmpuqvusw4b/assets
WARNING:absl:Buffer deduplication procedure will be skipped when flatbuffer library i
```

Testing the tflite model

```
OUTPUT_TFLITE_MODEL = "/content/drive/MyDrive/output/saved_model.tflite"
```

```
for image_val_batch, label_val_batch in val_data:
  print("Image batch shape: ", image_val_batch.shape)
  print("Label batch shape: ", label_val_batch.shape)
  break
```

```
Image batch shape:  (64, 32, 32, 3)
Label batch shape:  (64, 43)
```

```
# Load the TFLite model and allocate tensors.
interpreter = tf.lite.Interpreter(model_path=OUTPUT_TFLITE_MODEL)
interpreter.allocate_tensors()

input_details = interpreter.get_input_details()
output_details = interpreter.get_output_details()

batch_size = image_val_batch.shape[0]
predicted_id = np.zeros(batch_size)

for i, image in enumerate(np.split(image_val_batch, batch_size)):
  interpreter.set_tensor(input_details[0]['index'], image)
  interpreter.invoke()
  output_data = interpreter.get_tensor(output_details[0]['index'])
  predicted_id[i] = np.argmax(output_data)

label_id = np.argmax(label_val_batch, axis=-1)

num_plot_column = 5
num_plot_row = batch_size // num_plot_column + (batch_size % num_plot_column >

plt.figure(figsize=(20,50))
```

```
plt.subplots_adjust(hspace=0.5)
for n in range(batch_size):
  plt.subplot(num_plot_row,num_plot_column,n+1)
  plt.imshow(image_val_batch[n])
  color = "green" if predicted_id[n] == label_id[n] else "red"
  plt.title(label_map[predicted_id[n]].title(), color=color)
  plt.axis('off')
_ = plt.suptitle("Model predictions (green: correct, red: incorrect)")


print("Accuracy of the shown eval batch, with the TensorFlow Lite model:")
accuracy_score(label_id, predicted_id)
```

```
Accuracy of the shown eval batch, with the TensorFlow Lite model:
1.0
```

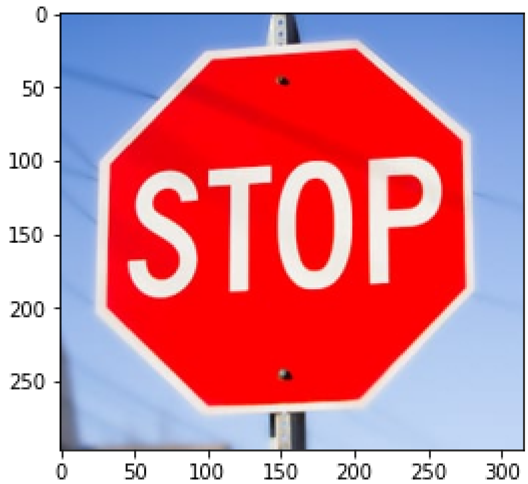
Model predictions (green: correct, red: incorrect)

Testing a single image

```
test_me_path = '/content/drive/MyDrive/test_inputs/5.jpg'
```

```
plt.imshow(mpimg.imread(test_me_path))
```

```
<matplotlib.image.AxesImage at 0x7fd445e7c410>
```




```
input_data = process_image(test_me_path)
```

```
input_data = tf.expand_dims(input_data, axis=0)
```

```
# Load the TFLite model and allocate tensors.
interpreter = tf.lite.Interpreter(model_path="/content/drive/MyDrive/output/sav
interpreter.allocate_tensors()

# Get input and output tensors.
input_details = interpreter.get_input_details()
```

```
output_details = interpreter.get_output_details()

# Test the model on random input data.
input_shape = input_details[0]['shape']
interpreter.set_tensor(input_details[0]['index'], input_data)

interpreter.invoke()

# The function `get_tensor()` returns a copy of the tensor data.
# Use `tensor()` in order to get a pointer to the tensor.
output_data = interpreter.get_tensor(output_details[0]['index'])
print(output_data)
```

```
[[2.44010025e-17 2.64569996e-13 6.75505333e-13 4.88851711e-12
  6.93470617e-15 5.08395340e-11 6.52302587e-17 1.81415804e-18
  4.38870375e-16 1.54678752e-11 3.07766546e-09 4.82838161e-17
  7.73513903e-13 1.50435247e-13 1.00000000e+00 1.10711557e-13
  3.16539488e-17 2.80879782e-08 1.89214506e-14 8.34889936e-18
  5.27447967e-15 6.12526540e-18 4.64495792e-14 3.92996055e-15
  1.82171602e-19 2.47348772e-13 7.02011005e-10 3.01682902e-22
  1.38862661e-16 2.63118628e-13 2.05917142e-18 1.81798597e-14
  2.01926637e-18 1.42198468e-17 1.76539833e-19 3.00462527e-19
  3.78759476e-20 9.40122617e-24 2.07759738e-16 5.36607605e-20
  2.69131419e-20 1.86145644e-18 1.66279754e-18]]
```

```
unique_signs[np.argmax(output_data)]
```

```
14
```

```
label_map[unique_signs[np.argmax(output_data)]]
```

```
'Stop'
```