Introduction

In the modern day, technology can be utilized to perform vast and complex jobs.
Autonomous driving is one of the ways machine learning can be implemented to aid us in such tasks. With the potential to take a lot of the work out of what can be tedious for a human, it is important that large pools of data are studied and impending models rigorously trialled so that everything runs perfectly in practise.

The objective of this project is to correctly identify and classify road traffic sign images. In autonomous driving the recognition of an image is key as depending on what is 'seen', different instructions will be transmitted to the driver or vehicle. Communications of wrong images introduce a lot of risk, therefore studies such as this one is conducted in order to explore the different models and techniques that can be used to gain high numbers of the correct classification returns.

The German Traffic Sign Recognition Benchmark dataset will be used.

1. Literature Review

Automobiles have become an essential element of our daily lives as a result of rapid technical improvement. As a result, road traffic is becoming increasingly difficult, resulting in an increase in traffic accidents each year. Traffic signs can help to enhance traffic flow and keep drivers safe. Due to the widespread use of intelligent vehicles as driver assistance systems, traffic sign detection and recognition are frequently experimented. Several apps focusing on traffic signs have been created, including a potential danger alert system, navigation, and driving safety. Adaptive cruise control, lane departure warning, collision avoidance, night vision, traffic sign recognition, and other systems are examples of such systems.
In 1987, Akatsuka and Imai conducted the first traffic sign recognition research, attempting to create a very basic traffic sign recognition system. A system capable of recognising traffic signs on its own and providing aid to drivers by informing them of the presence of a certain restriction or danger, such as speeding. It may be used to detect and recognise specific traffic signs automatically.
The methodology of traffic sign detection can be separated into detection and classification. Traffic sign detection using single frame image has two major cons: 1. Difficult to detect signs when temporary blockage occurs like traffic 2. Preciseness of signs is difficult to verify. It is widely done using image processing technique which includes binarization which ensures the image is in good condition or not, ROI which locates the traffic sign so that a large portion of the image can be ignored (detection) and pixel matching which is a classification method (Radzak, M.Y., et al.,2015.). The other classification methods used in traffic sign detection are Histogram of oriented gradients (HOG) and UNet with an encoder decoder architecture. HOG breaks down an image into small squares cells, calculates a histogram of oriented gradients in each cell, normalises the output with a block-wise pattern, and returns a descriptor for each cell (keerimolel, A., et al.,2021.).

Traffic sign detection can also be done using shape based and machine learning methods. When colors are lacking or difficult to identify, shape-based methods are a useful option. Once the sign is converted to greyscale image, the edges are detected using detectors like Canny, Prewitt, Sobel or Hough transform techniques. The cons of these techniques are they are highly time consuming and thus, not appropriate for real time applications. Some literatures also suggests that traffic sign detection can be done by combining Haar wavelet and HOG features with SVM and adaboost classifiers. Meanwhile, many researchers recently shifted towards machine learning techniques like convolutional neural networks (CNN) and end to end deep learning for complex environments. The architecture is based on a multi-resolution feature fusion network. It also suggests a vertical spatial sequence attention (VSSA) module to obtain more context information for enhanced detection performance by framing traffic sign identification as a spatial sequence classification and regression challenge. Real-time single stage detectors include the well-known YOLO and SSD techniques. To boost performance, SSD makes use of multi-layer features for detecting. Many single shot framework-based solutions for traffic sign detecting systems have been proposed because single-shot detectors are more likely to be real-time (Ellahayani, A., et al.,2020.).
Once the image detection is done, authors of another literature include that a tracking phase is required to ensure the validity of the planned zone. Rather of detecting the image with just one frame, the algorithm would follow the proposed object for a set number of frames (usually four). This has been shown to considerably improve accuracy. The Kalman Filter is the most frequent object tracker. (Zaki, P.S., et al.,2003.). The literature also talks about using statistical based methods, support vector machine and principal component analysis for image classification. Faster recurrent convolutional neural network (F-RCNN) was used to achieve accurate results even in real life complex situations averaging around 96%. If a high-end GPU is available, accuracy can be improved by adding considerably more training data (at least 40k photos, with an average of 1,000 images for each class) and training the models for a longer time.

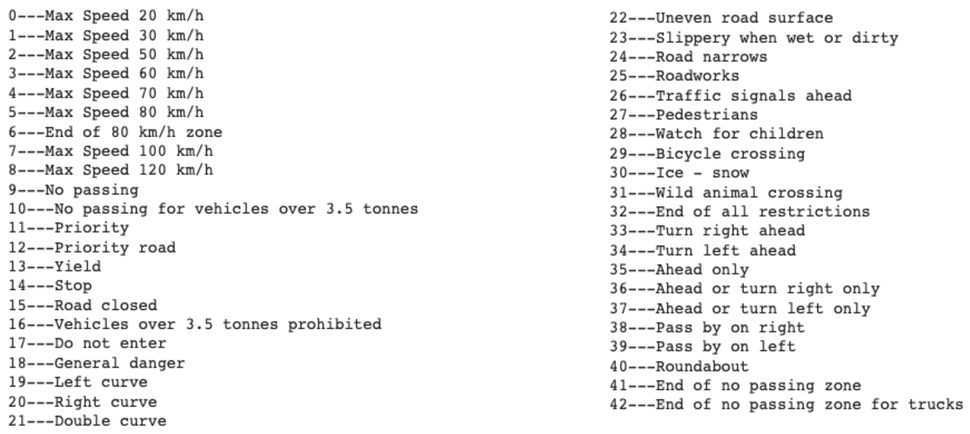
2. Data Exploration

To begin any machine learning we must first gain an understanding of the type of data we are dealing with. An investigation into the features of the data is essential, we must become more familiar with its structure to be able to easily manipulate it to serve our purpose. This is known as data exploration.

- **2.1 Number of Categories**

After obtaining the data, it was found that in each set (test and training) there were 43 subfolders, each corresponding to a single category and holding images from said category.

The category names were ordered as follows:



```
0---Max Speed 20 km/h
1---Max Speed 30 km/h
2---Max Speed 50 km/h
3---Max Speed 60 km/h
4---Max Speed 70 km/h
5---Max Speed 80 km/h
6---End of 80 km/h zone
7---Max Speed 100 km/h
8---Max Speed 120 km/h
9---No passing
10---No passing for vehicles over 3.5 tonnes
11---Priority
12---Priority road
13---Yield
14---Stop
15---Road closed
16---Vehicles over 3.5 tonnes prohibited
17---Do not enter
18---General danger
19---Left curve
20---Right curve
21---Double curve
22---Uneven road surface
23---Slippery when wet or dirty
24---Road narrows
25---Roadworks
26---Traffic signals ahead
27---Pedestrians
28---Watch for children
29---Bicycle crossing
30---Ice - snow
31---Wild animal crossing
32---End of all restrictions
33---Turn right ahead
34---Turn left ahead
35---Ahead only
36---Ahead or turn right only
37---Ahead or turn left only
38---Pass by on right
39---Pass by on left
40---Roundabout
41---End of no passing zone
42---End of no passing zone for trucks
```

Figure 1: List displaying the names and index of all image categories in the data.

- **2.2 Inspection of Random Images**

Firstly, random images from the dataset were visualised alongside labels dictating the image category. This allowed us to visually confirm the match between image and category while briefly checking for any incorrectly classified data.

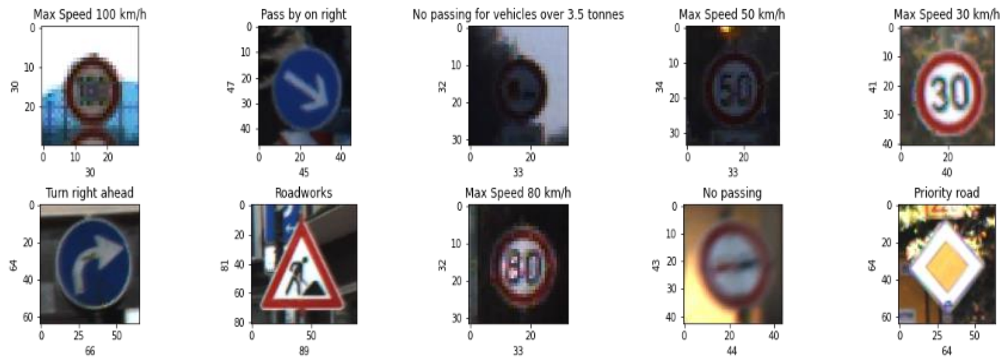


Figure 2: Picture displaying a series of images their respective categories.

- **2.3 Distribution of Length, Height and Width**

Next, for the images contained in each category the distribution of length and width was plotted.

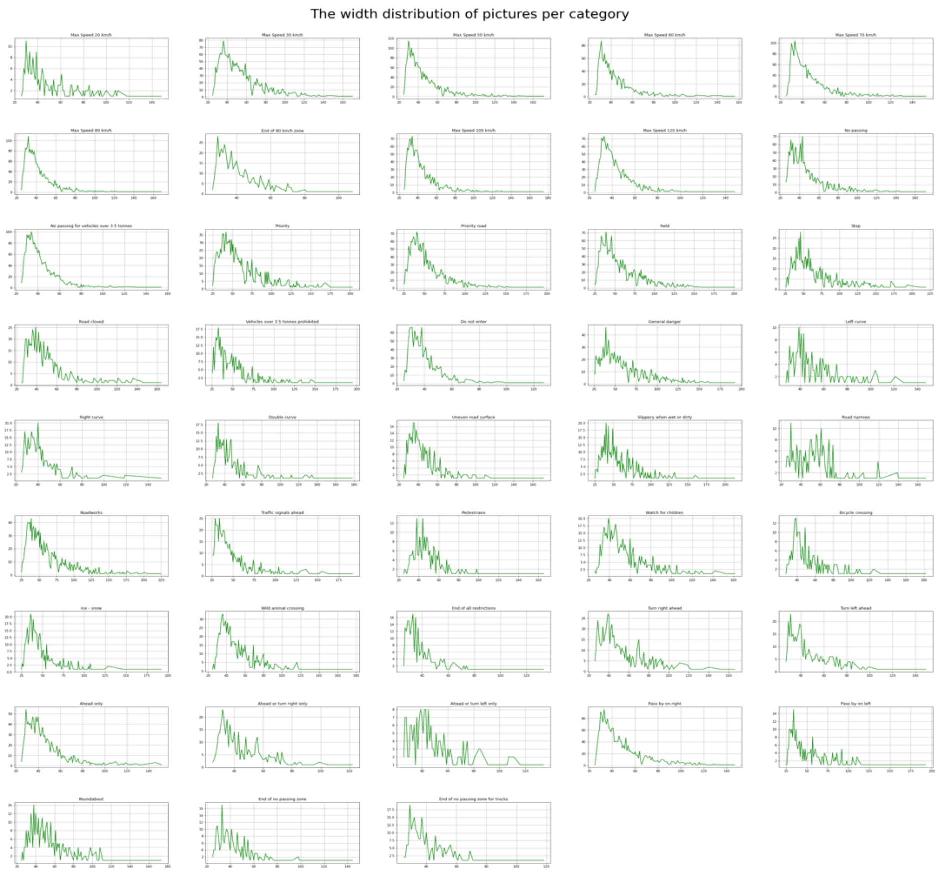


Figure 3: Height and width distribution plots for individual categories.

To gain a more general sense of the data dimensions, data from Figure 2 was compiled, forming height and width distribution plots covering the whole range of data.

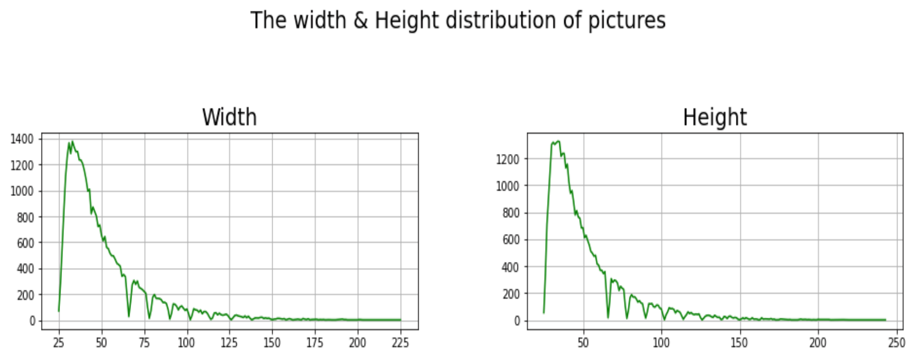


Figure 4: Height and width distribution plots for whole dataset.

The range of pixels that many of the images fall between can be found by reading these plots. It was concluded that 30-35pixels in width and 30-35 pixels in height would be the ideal values for image resizing.

- **2.4 Exploration of number of pictures in the Test and Training Set**

The number of pictures in each category was explored to learn more about the balance of the data.

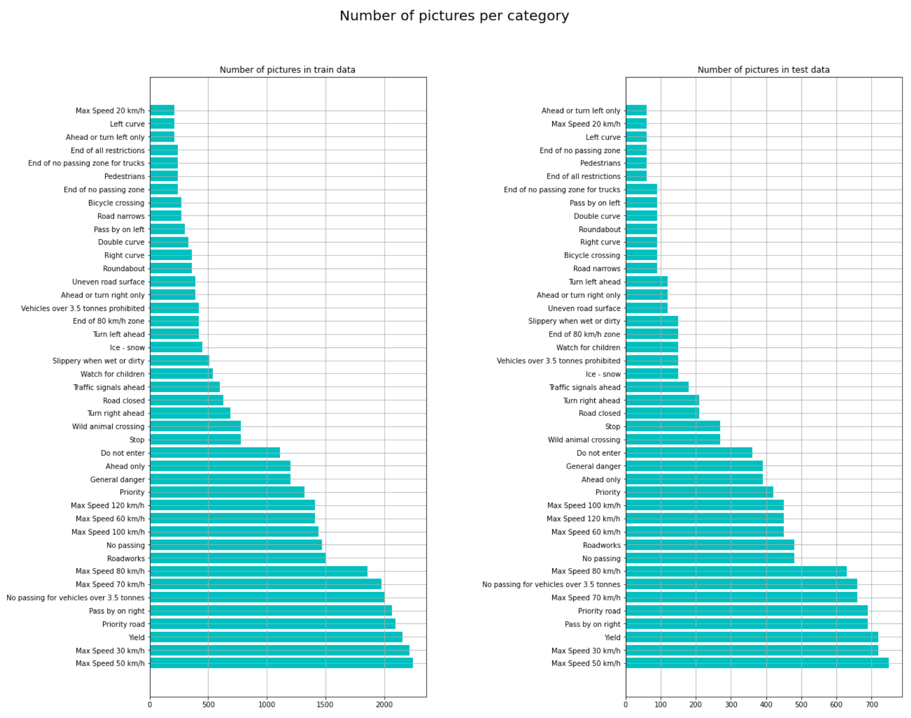


Figure 5: Histogram displaying number of images per class for training (left) and test (right) data.

As the categories in the histogram are displayed in ascending order of magnitude, we can see that the class with the maximum number 'Max speed 50 km/h' has ~2250 points, while the class with the minimum number 'Max speed 20 km/h' is ~210. This is a large difference and the type of discrepancy that must be taken into consideration during the pre-processing stage.

- **2.5 Average Intensity**

The average intensity for each class of image was also investigated.

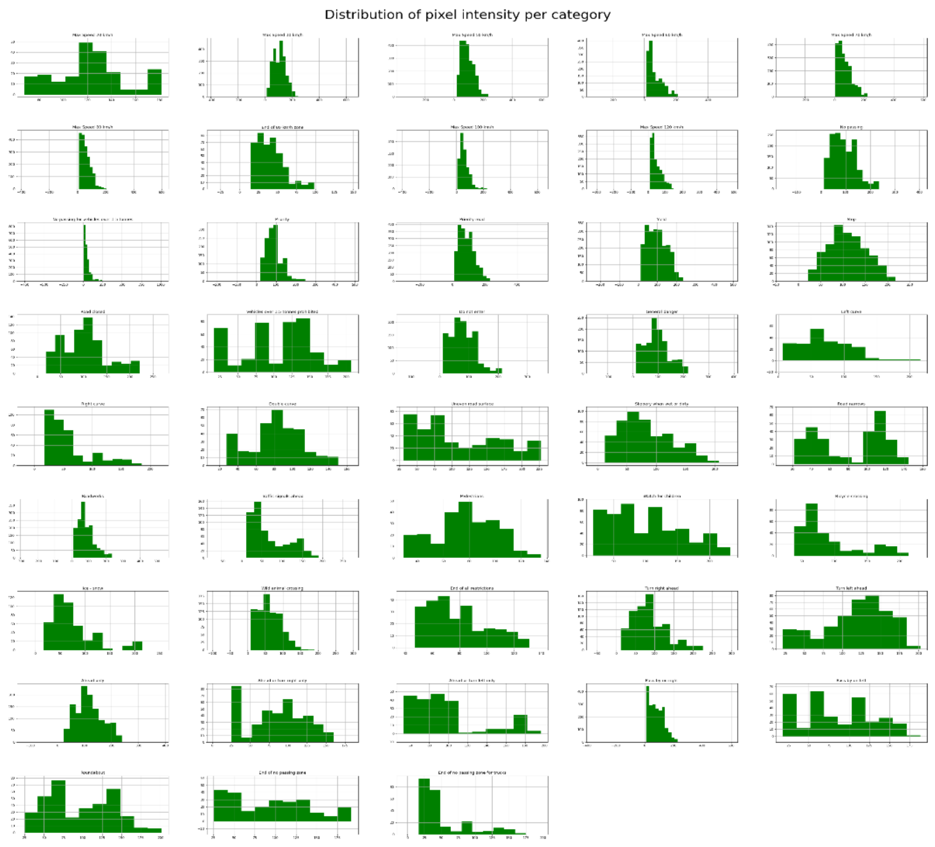


Figure 6: Pixel intensities mapped to histograms for each class of data.

Through the plot from Figure 5 it was found that the pixel intensities displayed more variance in value than the height and width plots. This meant that it would probably be more effective to tailor the image arrays on an individual basis to compensate, rather than adjusting them all to the same general intensity.

3. <u>Data Pre-Processing</u>

Before developing a Deep Learning model, data pre-processing is required. We may not always have access to clean, well-formatted data when working on a deep learning project. As a result, the data must be cleaned and formatted before any operation can be performed on it. Data pre-processing is the process of preparing raw data for use in a machine or deep learning model, and it is also the first and most crucial step in the model construction process.

Various pre-processing techniques have been used to convert our raw training data into a form suitable for feeding into the models. It is worth noting that not all these methods are to be applied to the test data, however this will be covered more in the individual pre-processing sections. The first image from the first 3 categories (20mph, 30mph and 50mph) of the dataset will be shown at each stage of the pre-processing steps to illustrate the changes made to the data.

- **3.1 Grayscale**

As colour is not an essential feature for the categorisation of the images in the traffic sign detection, we can strip the images of their RGB values. Another use of grayscale is that an RGB image is 3 times of greyscale image. So, by doing this we reduce the size of image as well. While RGB image is a 3D pixel value, converting it into a greyscale gives to 1D pixel value thereby simply reducing the complexity.

This reduces dimensionality as the depth/channel of an image array matrix is changed from being composed of 3 RGB values ranging from 0 to 255 pixels to a single scale where 0 is black and 255 is white.

Less parameters to be fed into the network means that the training and test processes are less computationally intensive, take less memory space and are considerably faster.

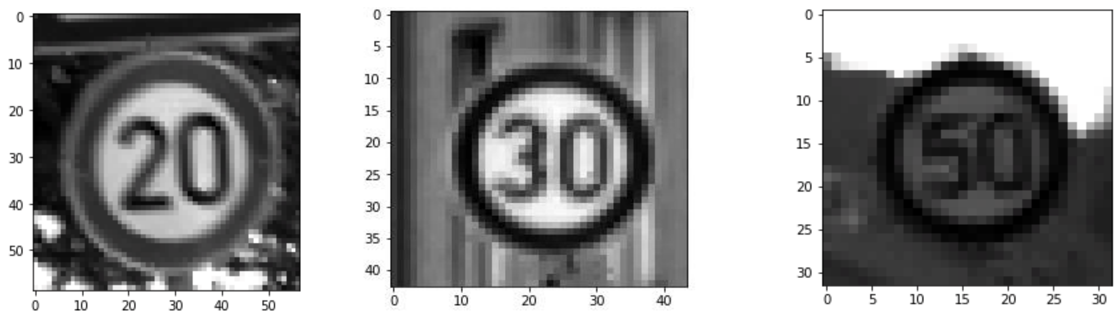


Figure 7: Visualisation of greyscale application to data. First images from 20mph, 30mph and 50mph categories used respectively.

- **3.2 Data Scaling**

As discussed in the data exploration section the raw images from the dataset appear in a variety of sizes, meaning that the images need to be resized to the same measurements.

It was concluded that pixel dimensions of 35x35 would be used as the height and width distribution plots indicated a concentration of data points at these numbers, suggesting it would be the most optimal choice for resizing.

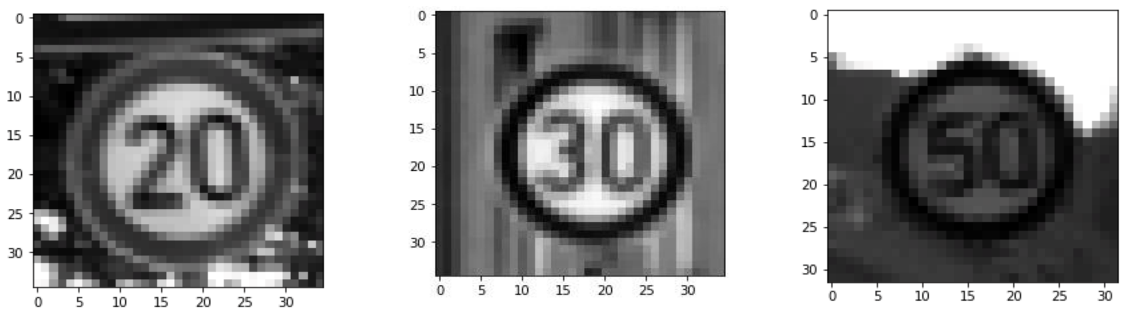


Figure 8: Visualisation displaying the resizing of data. First images from 20mph, 30mph and 50mph categories used respectively

- **3.3 Brightness/Contrast**

To improve the pixel intensity in the images, the histogram equalisation technique was applied to the data. This process enhances an images contrast by identifying the most frequent pixel intensity values and then spreading them out, thus allowing regions of an image with low contrast to increase in contrast.

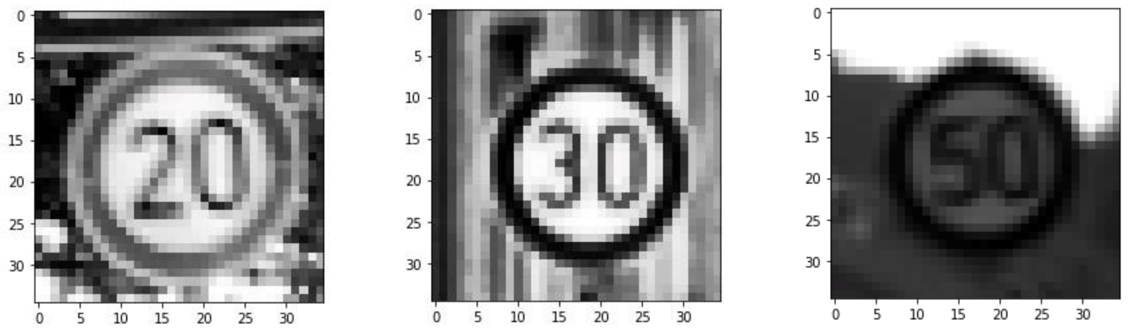
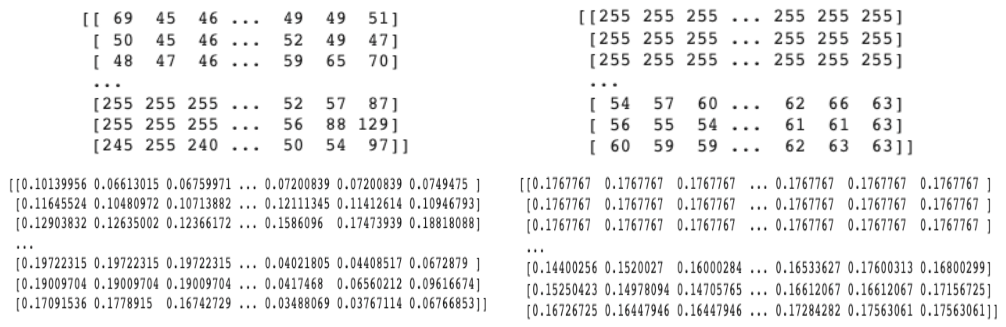


Figure 9: Images displaying the effects of applying histogram equalisation to the data. First images from 20mph, 30mph and 50mph categories used respectively.

The data was normalised to a range from 0 to 1 due to varying scale of the numbers featured in the image arrays. This again, can reduce the time it takes to train a model.



Raw data

```
[[ 69  45  46 ...  49  49  51]
 [ 50  45  46 ...  52  49  47]
 [ 48  47  46 ...  59  65  70]
 ...
 [255 255 255 ...  52  57  87]
 [255 255 255 ...  56  88 129]
 [245 255 240 ...  50  54  97]]
```

```
[[255 255 255 ... 255 255 255]
 [255 255 255 ... 255 255 255]
 [255 255 255 ... 255 255 255]
 ...
 [ 54  57  60 ...  62  66  63]
 [ 56  55  54 ...  61  61  63]
 [ 60  59  59 ...  62  63  63]]
```

Normalised data

```
[[0.10139956 0.06613015 0.06759971 ... 0.07200839 0.07200839 0.0749475 ]
 [0.11645524 0.10480972 0.10713882 ... 0.12111345 0.11412614 0.10946793]
 [0.12903832 0.12635002 0.12366172 ... 0.1586096  0.17473939 0.18818088]
 ...
 [0.19722315 0.19722315 0.19722315 ... 0.04021805 0.04408517 0.0672879 ]
 [0.19009704 0.19009704 0.19009704 ... 0.0417468  0.06560212 0.09616674]
 [0.17091536 0.1778915  0.16742729 ... 0.03488069 0.03767114 0.06766853]]
```

```
[[0.1767767  0.1767767  0.1767767  ... 0.1767767  0.1767767  0.1767767 ]
 [0.1767767  0.1767767  0.1767767  ... 0.1767767  0.1767767  0.1767767 ]
 [0.1767767  0.1767767  0.1767767  ... 0.1767767  0.1767767  0.1767767 ]
 ...
 [0.14400256 0.1520027  0.16000284 ... 0.16533627 0.17600313 0.16800299]
 [0.15250423 0.14978094 0.14705765 ... 0.16612067 0.16612067 0.17156725]
 [0.16726725 0.16447946 0.16447946 ... 0.17284282 0.17563061 0.17563061]]
```

Figure 10: Printed image arrays for raw and normalised data. 20mph and 50mph stop signs used respectively.

From these images it is clear to see how the initial image arrays have been normalised to a range between 0 and 1.

- **3.4 Data Shuffle**

A minor pre-processing technique but worth mentioning, this process shuffles the form of the outputted array and thus the order of the data.

This helps to reduce bias in a model, as well as preventing it from learning the order/format of the input data and drawing its conclusions from this pattern.

- **3.5 Categorical Data**

By indexing each folder from which the data was obtained, each image category was allocated a number in the range of 1 to 43. This allowed us to form tuples of the image data and category the image belonged to.

Returning images from the dataset new provides us with the pre-processed images in a random order (a result of the data shuffling).

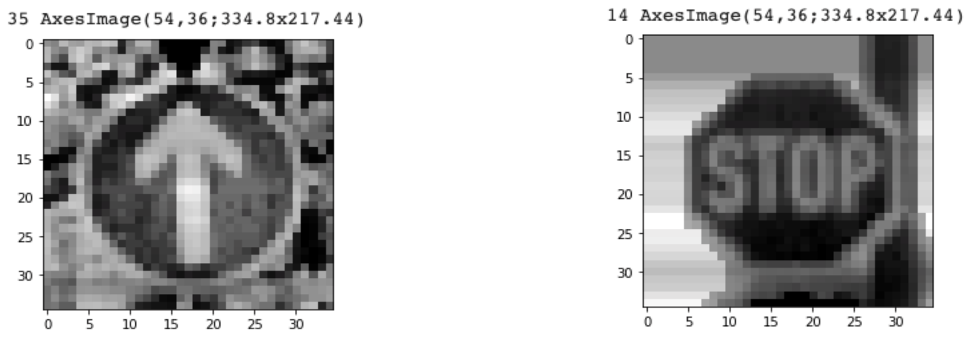
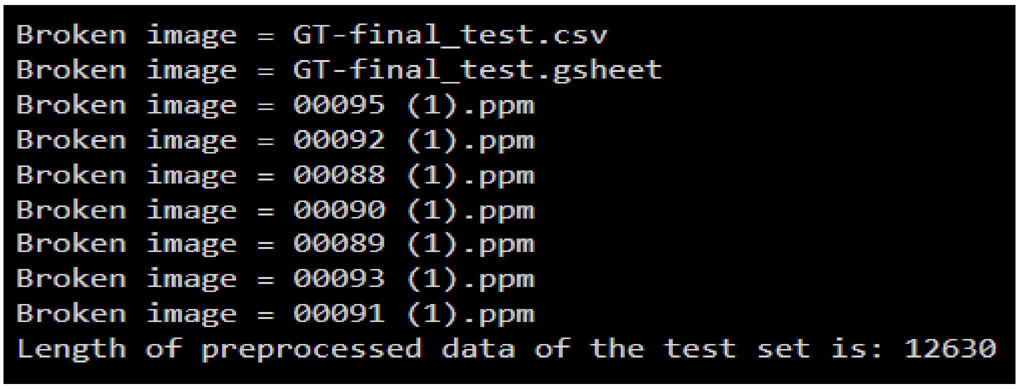

35 AxesImage(54,36;334.8x217.44)
14 AxesImage(54,36;334.8x217.44)

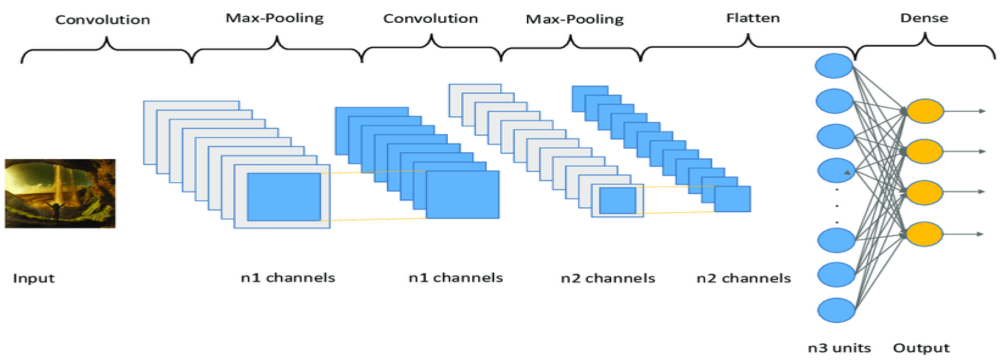
Figure 11: Images from dataset with category index attached. 35 indicates the category 'Ahead only' and 14 indicates 'Stop'.



```
Broken image = GT-final_test.csv
Broken image = GT-final_test.gsheet
Broken image = 00095 (1).ppm
Broken image = 00092 (1).ppm
Broken image = 00088 (1).ppm
Broken image = 00090 (1).ppm
Broken image = 00089 (1).ppm
Broken image = 00093 (1).ppm
Broken image = 00091 (1).ppm
Length of preprocessed data of the test set is: 12630
```

Figure 12: List of broken images present in the test dataset and the total number of images in the processed testing set

4. <u>Convolution Neural Network</u>



A neural network typically consists of an input layer, hidden layers, and an output layer. The architecture of the brain inspired CNNs. Artificial neurons or nodes in CNNs accept input, process it, and send the result as output, similar to how a neuron in the brain processes and transmits information throughout the body. The image is used as input. As input, the input layer takes picture pixels in the form of arrays. In CNNs, there could be a slew of hidden layers that calculate and extract features from images. Such approaches include convolution, pooling, rectified linear units, and fully connected layers. The first layer, convolution, extracts feature from an input image. The object is classified by the fully linked layer.

- **4.1 BENEFITS OF CNN**

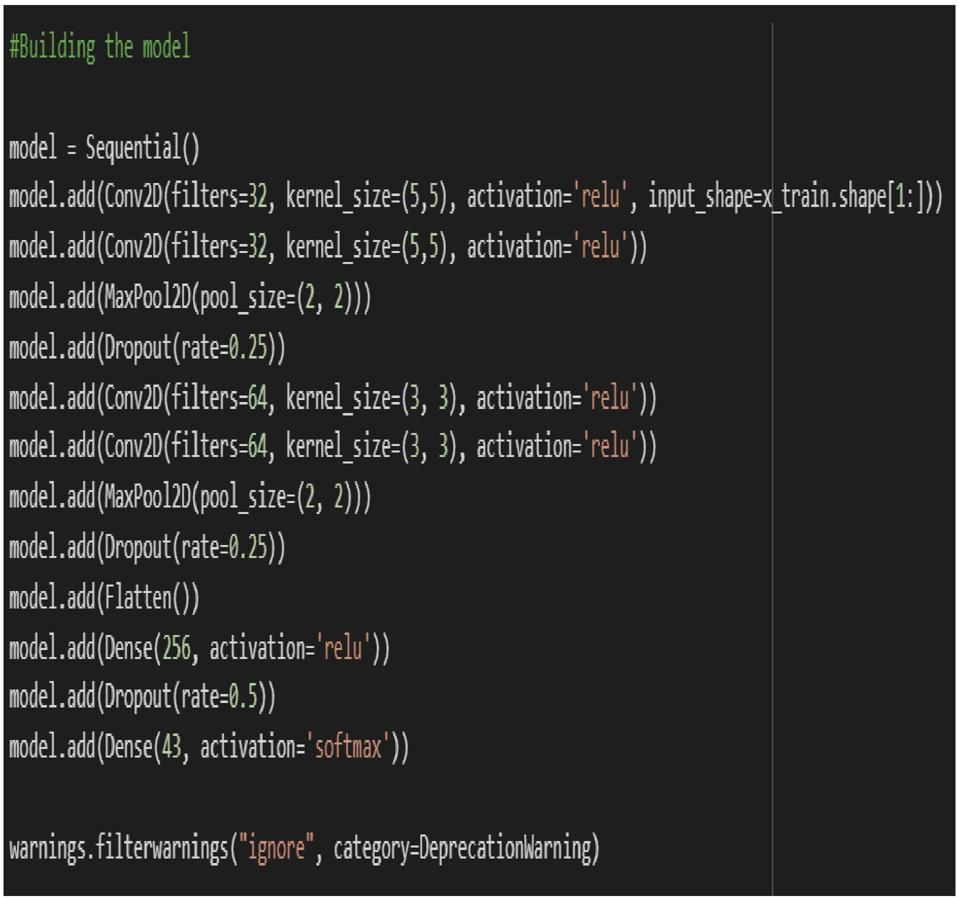
1. Because CNNs can detect and analyse patterns, they have fundamentally transformed our approach to image recognition. Because of the high accuracy of their outputs, they are considered the most effective architecture for image classification, retrieval, and detection applications.
2. They have a wide range of applications in real-world tests, where they provide high-quality results and can locate and identify objects in images, such as people, cars, and birds. As a result, they've become the go-to method for making predictions using any image as an input.
3. A significant attribute of CNNs is their ability to achieve spatial invariance, which means they can be trained to recognise and extract visual elements wherever in the image. Manual extraction is unnecessary since CNNs learn features from images/data and extract them from photographs quickly. Hence, CNNs are an effective method for obtaining accurate Deep Learning results.
4. The pooling layer reduces the number of parameters needed to process the image, resulting in faster processing and lower memory and compute expenses.
5. While CNNs are most commonly used for image analysis, they can also be used to handle other data analysis and classification problems. As a result, they may be utilised to obtain precise results in a variety of industries, including face recognition, video classification, street/traffic sign recognition, galaxy classification, and medical image interpretation and diagnosis/analysis, to name a few.

- **4.2 BUILDING CNN**

For building we will use sequential model from keras library. Then we will add the layers to make convolutional neural network. In the first 2 Conv2D layers we have used 32 filters and the kernel size is (5,5).

In the MaxPool2D layer we have kept pool size (2,2) which means it will select the maximum value of every 2 x 2 area of the image. By doing this, the dimensions of the image will reduce by factor of 2. In dropout layer we have kept dropout rate = 0.5 that means 50% of neurons are removed randomly.

We apply these 3 layers again with some change in parameters. Then we apply flatten layer to convert 2-D data to 1-D vector. This layer is followed by dense layer, dropout layer and dense layer again. The last dense layer outputs 43 nodes as the traffic signs are divided into 43 categories in our dataset. This layer uses the softmax activation function which gives probability value and predicts which of the 43 options has the highest probability.

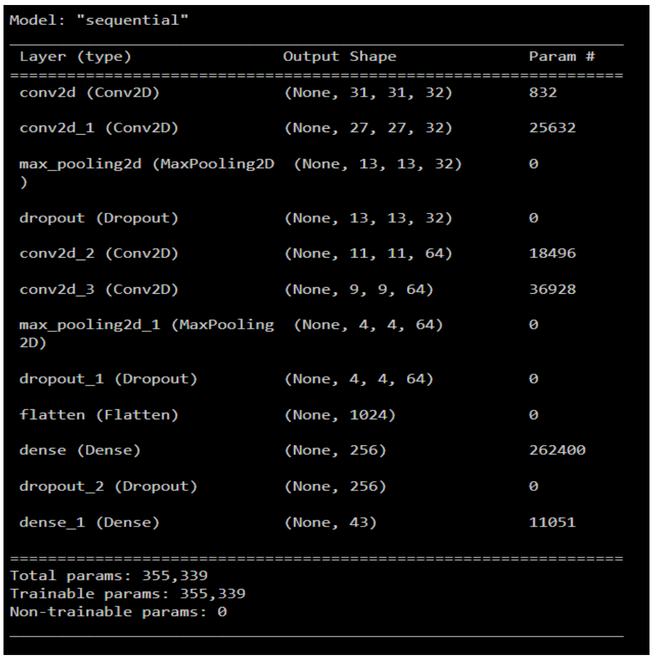


```
#Building the model

model = Sequential()
model.add(Conv2D(filters=32, kernel_size=(5,5), activation='relu', input_shape=x_train.shape[1:]))
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'))

warnings.filterwarnings("ignore", category=DeprecationWarning)
```

Figure 13: Iteratively building the model

- **4.3 Compiling the Model**

  Compiling the CNN for final training of the model on the given dataset.

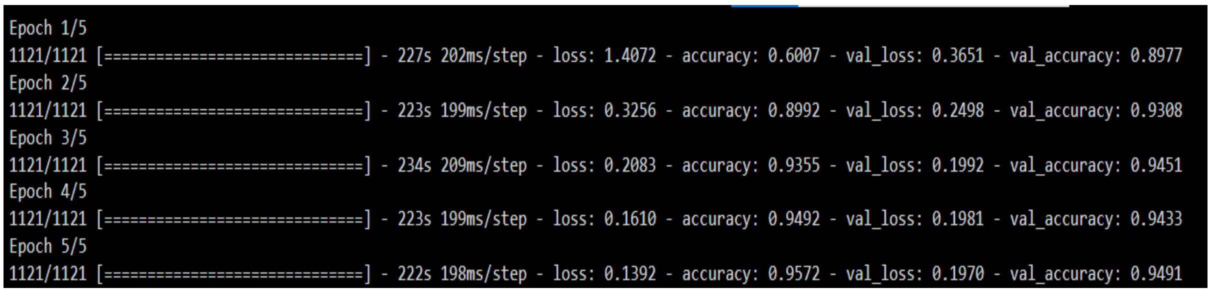


```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d (Conv2D)             (None, 31, 31, 32)        832

 conv2d_1 (Conv2D)           (None, 27, 27, 32)        25632

 max_pooling2d (MaxPooling2D  (None, 13, 13, 32)       0
 )

 dropout (Dropout)           (None, 13, 13, 32)        0

 conv2d_2 (Conv2D)           (None, 11, 11, 64)        18496

 conv2d_3 (Conv2D)           (None, 9, 9, 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: 355,339
Trainable params: 355,339
Non-trainable params: 0
_________________________________________________________________
```

Figure 14: Model summary

- **4.4 Training the Model**

One Epoch denotes one cycle of training the model with all of the data. We have 5 epochs here since each epoch improves the model's accuracy and yields better results. This makes it easier to train a model using data without overfitting or underfitting it.



```
Epoch 1/5
1121/1121 [==============================] - 227s 202ms/step - loss: 1.4072 - accuracy: 0.6007 - val_loss: 0.3651 - val_accuracy: 0.8977
Epoch 2/5
1121/1121 [==============================] - 223s 199ms/step - loss: 0.3256 - accuracy: 0.8992 - val_loss: 0.2498 - val_accuracy: 0.9308
Epoch 3/5
1121/1121 [==============================] - 234s 209ms/step - loss: 0.2083 - accuracy: 0.9355 - val_loss: 0.1992 - val_accuracy: 0.9451
Epoch 4/5
1121/1121 [==============================] - 223s 199ms/step - loss: 0.1610 - accuracy: 0.9492 - val_loss: 0.1981 - val_accuracy: 0.9433
Epoch 5/5
1121/1121 [==============================] - 222s 198ms/step - loss: 0.1392 - accuracy: 0.9572 - val_loss: 0.1970 - val_accuracy: 0.9491
```

Figure 15: Epoch cycles to train the model

- **4.5 Graph for Accuracy vs Epochs:**

The Accuracy vs. Epochs graph depicts the increase in accuracy with each epoch.
Each cycle of data training for the model aids in fitting the data.
The line graph is used for the graphs because it is simple to understand and serves its goal.
The accuracy vs. epochs graph is shown below.

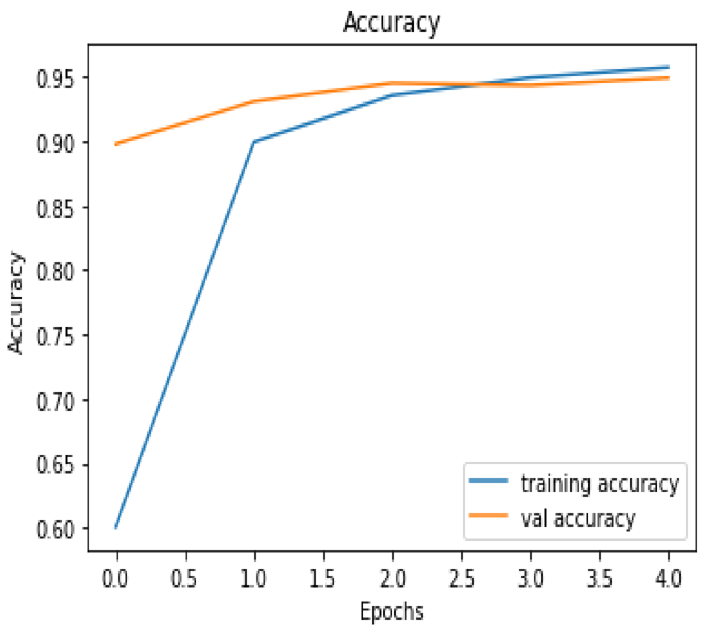


Figure 16: Epoch vs training and validation accuracy

- **Graph for the Loss vs Epochs:**

The Loss vs. Epochs graph depicts the loss of validation and training after each epoch.
The graph is plotted using a line graph.
The loss of validation and the loss of training start to decrease after each epoch, as shown in this graph.
This program has an epoch for each period that will assist you adequately to train your model.

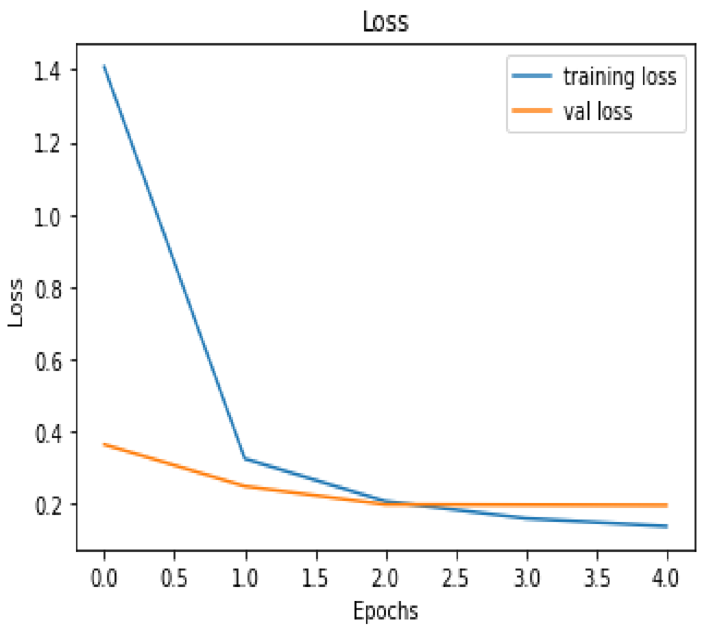


Figure 17: Epoch vs Validation and training loss

- **Training Accuracy and Validation Accuracy vs Epochs:**

The model's training accuracy is the accuracy achieved during training, whereas the model's validation accuracy is the model's final accuracy after the epochs.
The graph depicts how, after each epoch, both training and valuation accuracy begin to improve, while training and valuation losses begin to decrease. This demonstrates the model's effectiveness.

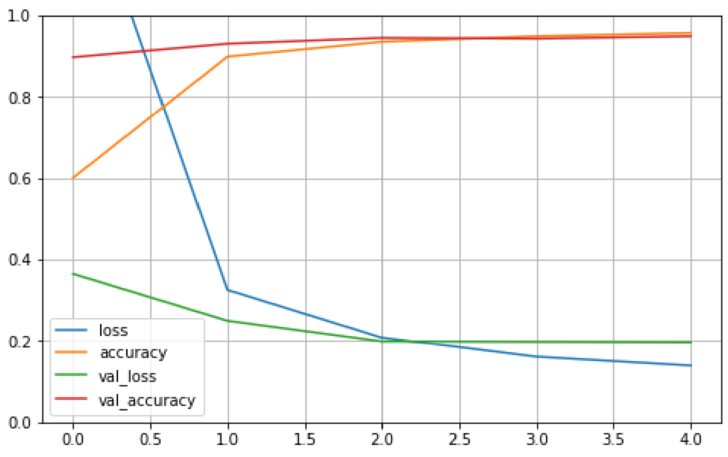
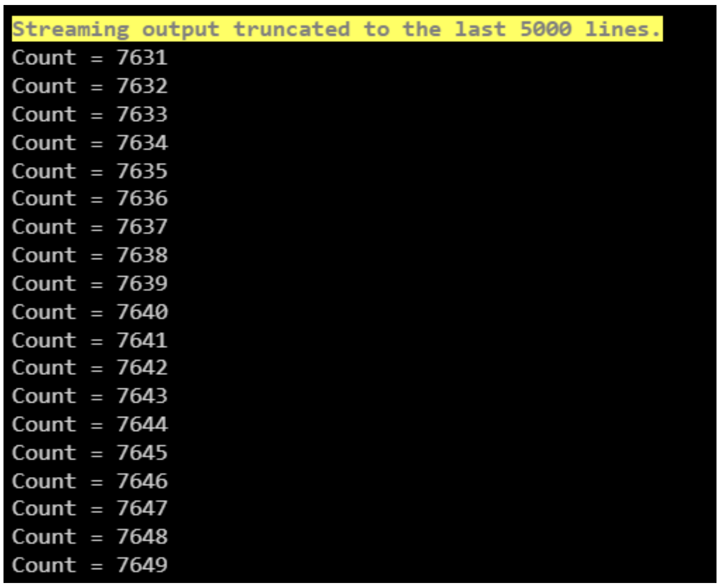


Figure 18: Accuracy and loss with respect to epoch cycles

- **4.8 Making Predictions on Test Data**

We'll concentrate on making predictions based on the test results in this section. The photographs in the test directory are first converted to grayscale, then shrunk and altered to accomplish this. This is because the model was trained on a 4D array and making predictions on grayscale photos is significantly easier for the model than making predictions on coloured images. The output is stored in a "predictions" directory which will be used later to measure the accuracy of the model.



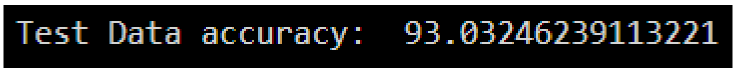
```
Streaming output truncated to the last 5000 lines.
Count = 7631
Count = 7632
Count = 7633
Count = 7634
Count = 7635
Count = 7636
Count = 7637
Count = 7638
Count = 7639
Count = 7640
Count = 7641
Count = 7642
Count = 7643
Count = 7644
Count = 7645
Count = 7646
Count = 7647
Count = 7648
Count = 7649
```

Figure 19: Counter for image prediction

- **4.9 Accuracy with Test Data**

Accuracy generally describes how the model performs across all classes. It is useful when all classes are of equal importance. It is calculated as the ratio between the number of correct predictions to the total number of predictions.

The accuracy of the model after training is 93.032



```
Test Data accuracy:  93.03246239113221
```

Figure 20: Accuracy of the model

- **4.10 Confusion Matrix**

A Confusion matrix is a N x N matrix that is used to assess the effectiveness of a classification model, where N is the number of target classes. The matrix compares the actual goal values to the machine learning model's predictions. This provides us with a comprehensive picture of how well our classification model is working and the kind of errors it is making.

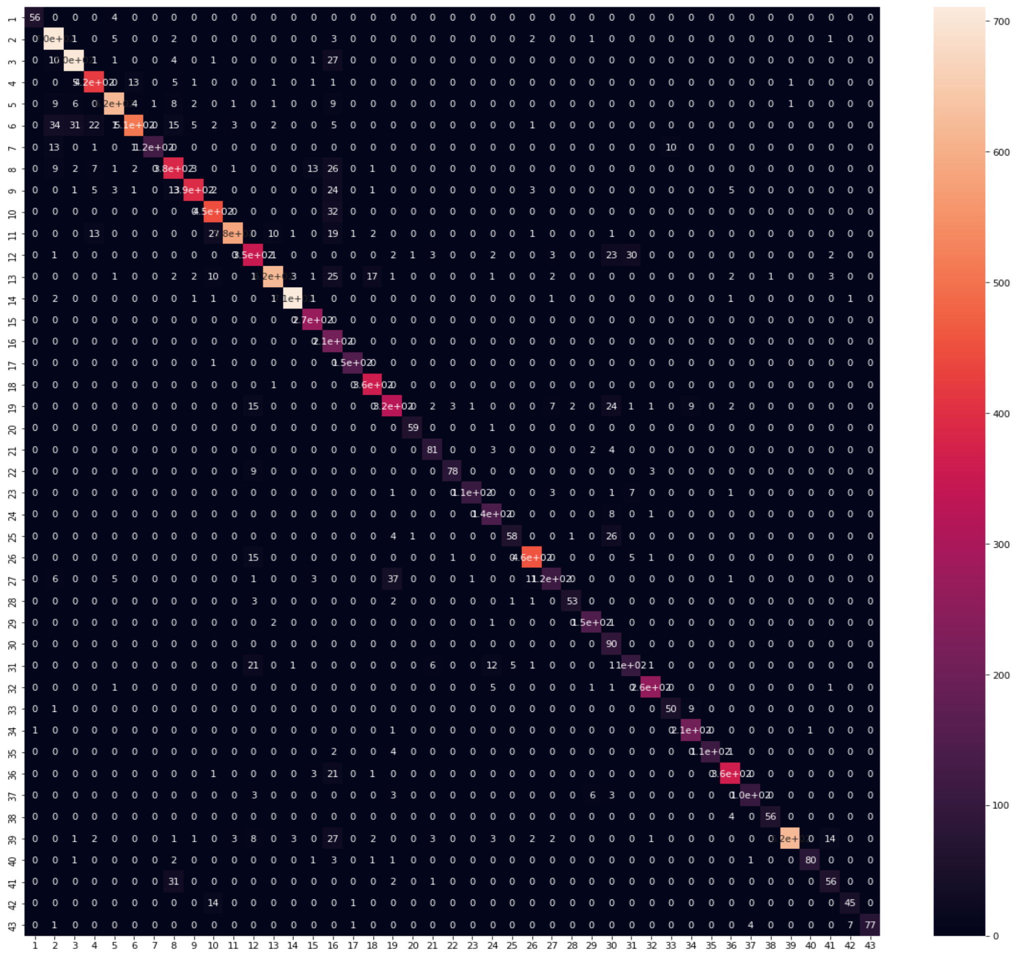
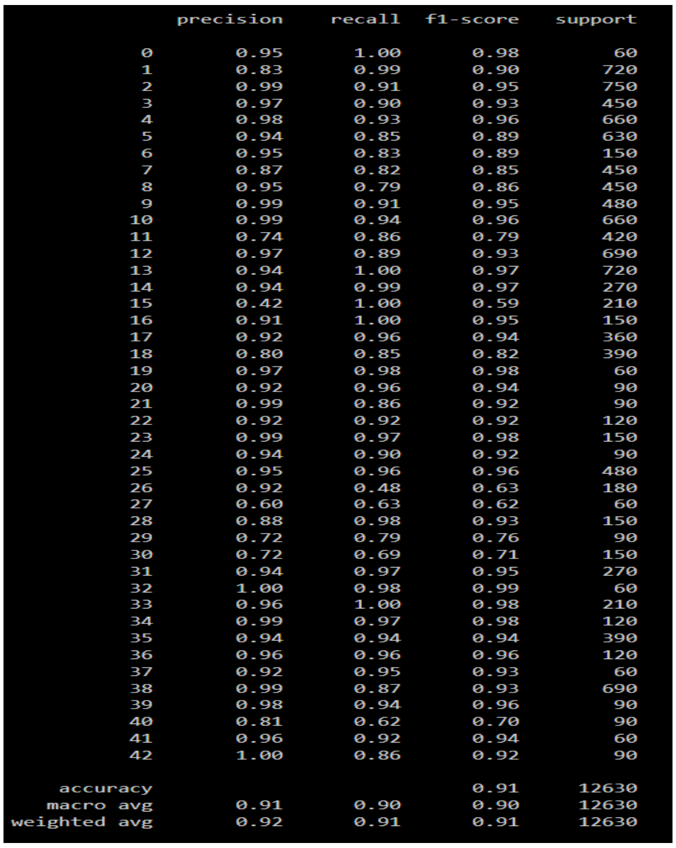


Figure 21: Confusion matrix

• 4.11 Classification Report

It is one of the metrics used to evaluate the performance of a classification-based machine learning model. It indicates the precision, recall, F1 score, and support of your model. It allows us to gain a better understanding of our trained model's overall performance.

1. **Precision**:
   The ratio of genuine positives to the sum of true and false positives is defined as precision.
2. **Recall**:
   The ratio of true positives to the sum of true positives and false negatives is known as recall.
3. **F1 Score:**
   The weighted harmonic mean of precision and recall is the F1. The closer the F1 score number is to 1.0, the higher the model's projected performance.
4. **Support**:
   The number of actual instances of the class in the dataset is referred to as support. It does not differ between models; it simply diagnoses the process of performance evaluation.



```
              precision    recall  f1-score   support

           0       0.95      1.00      0.98        60
           1       0.83      0.99      0.90       720
           2       0.99      0.91      0.95       750
           3       0.97      0.90      0.93       450
           4       0.98      0.93      0.96       660
           5       0.94      0.85      0.89       630
           6       0.95      0.83      0.89       150
           7       0.87      0.82      0.85       450
           8       0.95      0.79      0.86       450
           9       0.99      0.91      0.95       480
          10       0.99      0.94      0.96       660
          11       0.74      0.86      0.79       420
          12       0.97      0.89      0.93       690
          13       0.94      1.00      0.97       720
          14       0.94      0.99      0.97       270
          15       0.42      1.00      0.59       210
          16       0.91      1.00      0.95       150
          17       0.92      0.96      0.94       360
          18       0.80      0.85      0.82       390
          19       0.97      0.98      0.98        60
          20       0.92      0.96      0.94        90
          21       0.99      0.86      0.92        90
          22       0.92      0.92      0.92       120
          23       0.99      0.97      0.98       150
          24       0.94      0.90      0.92        90
          25       0.95      0.96      0.96       480
          26       0.92      0.48      0.63       180
          27       0.60      0.63      0.62        60
          28       0.88      0.98      0.93       150
          29       0.72      0.79      0.76        90
          30       0.72      0.69      0.71       150
          31       0.94      0.97      0.95       270
          32       1.00      0.98      0.99        60
          33       0.96      1.00      0.98       210
          34       0.99      0.97      0.98       120
          35       0.94      0.94      0.94       390
          36       0.96      0.96      0.96       120
          37       0.92      0.95      0.93        60
          38       0.99      0.87      0.93       690
          39       0.98      0.94      0.96        90
          40       0.81      0.62      0.70        90
          41       0.96      0.92      0.94        60
          42       1.00      0.86      0.92        90

    accuracy                           0.91     12630
   macro avg       0.91      0.90      0.90     12630
weighted avg       0.92      0.91      0.91     12630
```

Figure 22: Classification report

- **4.12 Results**

We built a function where the photos and their real and expected labels are displayed after evaluating the model and verifying its correctness, demonstrating where the images lack and where it succeeds in making the proper prediction.

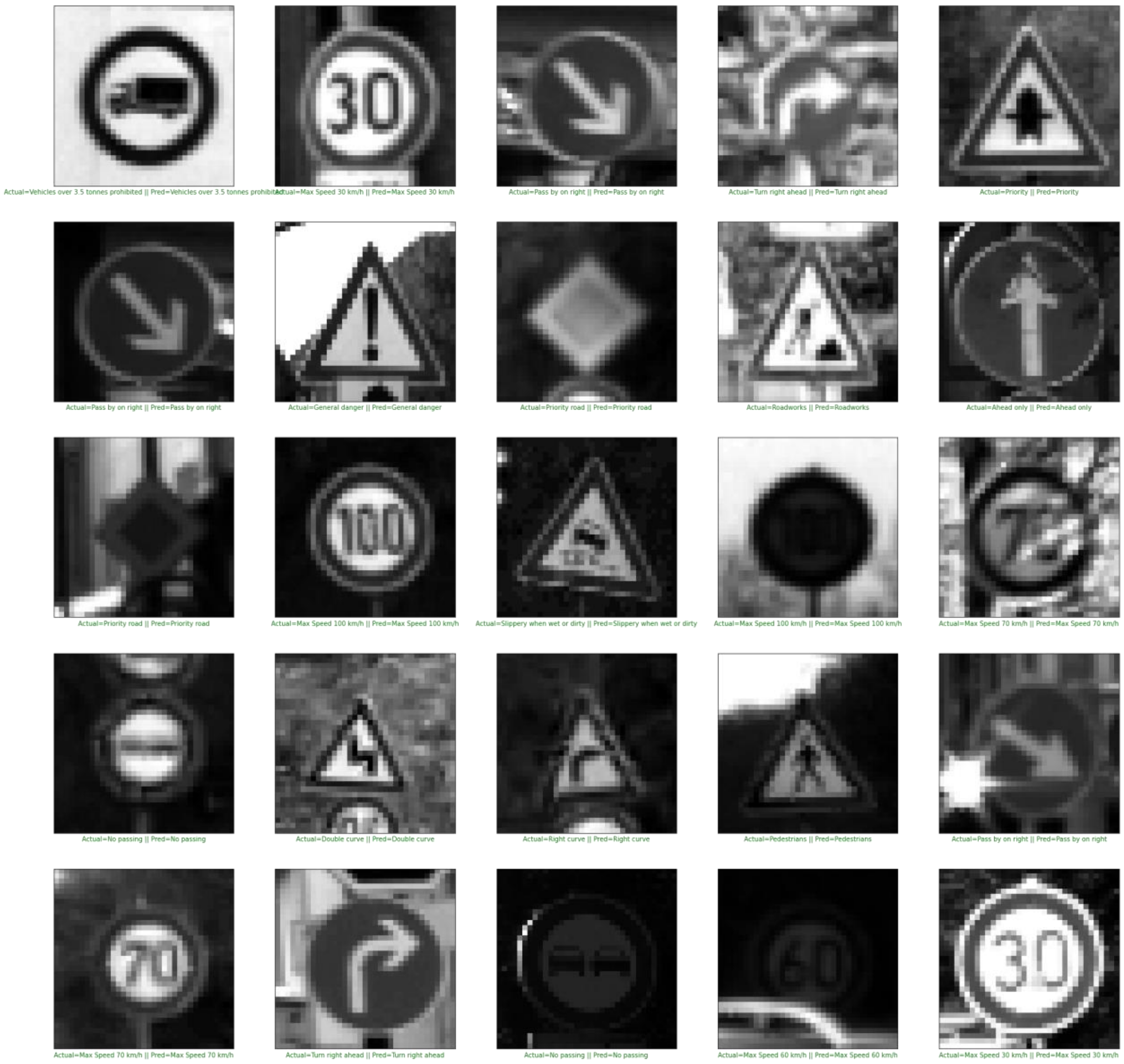

Actual=Vehicles over 3.5 tonnes prohibited || Pred=Vehicles over 3.5 tonnes prohibited
Actual=Max Speed 30 km/h || Pred=Max Speed 30 km/h
Actual=Pass by on right || Pred=Pass by on right
Actual=Turn right ahead || Pred=Turn right ahead
Actual=Priority || Pred=Priority
Actual=Pass by on right || Pred=Pass by on right
Actual=General danger || Pred=General danger
Actual=Priority road || Pred=Priority road
Actual=Roadworks || Pred=Roadworks
Actual=Ahead only || Pred=Ahead only
Actual=Priority road || Pred=Priority road
Actual=Max Speed 100 km/h || Pred=Max Speed 100 km/h
Actual=Slippery when wet or dirty || Pred=Slippery when wet or dirty
Actual=Max Speed 100 km/h || Pred=Max Speed 100 km/h
Actual=Max Speed 70 km/h || Pred=Max Speed 70 km/h
Actual=No passing || Pred=No passing
Actual=Double curve || Pred=Double curve
Actual=Right curve || Pred=Right curve
Actual=Pedestrians || Pred=Pedestrians
Actual=Pass by on right || Pred=Pass by on right
Actual=Max Speed 70 km/h || Pred=Max Speed 70 km/h
Actual=Turn right ahead || Pred=Turn right ahead
Actual=No passing || Pred=No passing
Actual=Max Speed 60 km/h || Pred=Max Speed 60 km/h
Actual=Max Speed 30 km/h || Pred=Max Speed 30 km/h

Figure 23: Actual vs Predicted output

5. <u>Conclusion</u>

In this work, we employed a CNN model to identify traffic signs and classify them with 93.032 %. We have gone over how to utilise deep learning to accurately categorise traffic signs using a number of pre-processing and visualisation techniques. To successfully interpret road traffic signals, we created a simple, easy-to-understand CNN model. On the test set, our model achieved above 90% accuracy, which is impressive given the computational resource constraints and the simplistic design. There is still a lot of work to be done, especially with newer Deep Learning systems that use more recent and intricate designs like Google Net or ResNet. On the other side, this certainly comes at a higher computational cost.

6. Improvements

Parameter sweeping was not done due to a lack of processing resources. It's possible that this model may perform better if the hyper parameters were changed. Additionally, the model architecture can be improved.
The goal of this project is to test the utmost level of accuracy without modifying the data. However, it is believed that augmentation would improve the model even further.
Apart from that, the proposed model could be improved with the use of YOLO algorithm for real-time traffic sign detection.

7. References

[1]: Ellahyani, A., Jaafari, I.E., Charfi, S., 2021. Traffic Sign Detection for Intelligent Transportation Systems: A Survey, E3S Web of Conferences 229, ICCSRE.

[2]: Keerimolel, A., Galsulkar, S., Gowray, B., 2021, April. A SURVEY ON TRAFFIC SIGN RECOGNITION AND DETECTION, International Journal of Trendy Research in Engineering and Technology, Volume 7.

[3]: Radzak, M.Y., Alias, M.F., Arof, S., Ahmad, M.R., Muniandy, I., Study on Traffic Sign Recognition, 2015, June. International Journal of Research Studies in Computer Science and Engineering, Volume 2, (PP 33-39).

[4]: Zaki, P.S., William, M.M., Soliman, B.K., Alexsan, K.G., Mansour, M., El-Moursy, M., Khalil, K., Traffic Signs Detection and Recognition System using Deep Learning, 2003.

[5]: https://www.geeksforgeeks.org/keras-conv2d-class/

[6]: https://www.geeksforgeeks.org/python-tensorflow-tf-keras-layers-conv2d-function/