School of Engineering and Applied Sciences

Robotics and Vision (MECT 424)

Robust Object Detection and Tracking System for safe Autonomous Navigation

Submitted by:

Bibek Gupta

Submitted to:

Dr. Mohamed A Wahby Shalaby

12th of April 2023 – Spring 2023

Table of Contents

1. Introduction

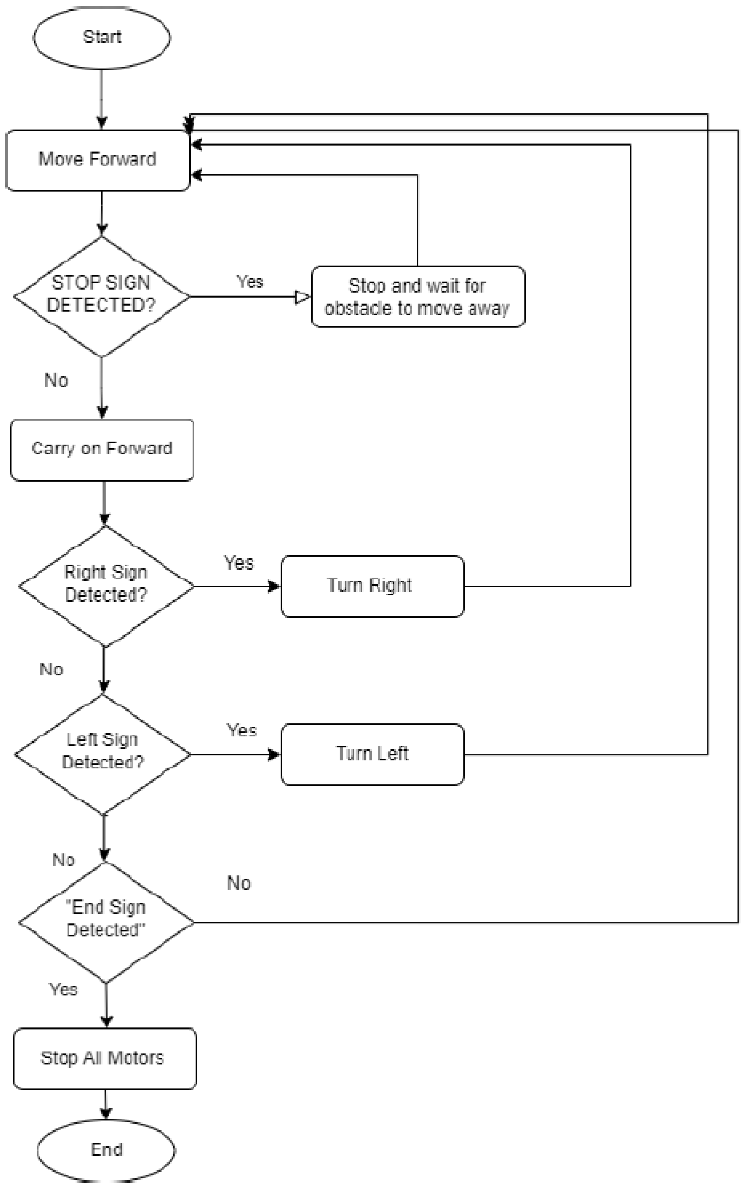
Recent evolution in the technology have led to huge contribution to the mobile robots industry by strong technological advancements such as artificial intelligence, network communication and computer vision. But first off, how exactly can one define mobile robots? Mobile robots are machines (or computers) which use sensors to identify its surroundings in its environment and move using wheels, tracks or legs. They are a combination of locomotion, perception, cognition and navigation.

Throughout this report, we will first introduce and identify our proposed mobile robot and showcase its importance, input and benefits. We will then start discussing in depth the technical side of our mobile robot; we will compare between its proposed features and the features of the current mobile robots in the market. Finally, we will showcase the implementationof the computer vision part of the project through ----.

According to the different classifications of mobile robots available, we could introduce our mobile robot as a land, autonomous robot. Our robot would be able to analyze the surrounding environment, and more specially to focus on the “signs” throughout the road. The signs shall guide the robot through the right/needed path towards the destination. Prior to the robot’s launch, it will not know its destination or its path yet, it will rather be guided throughout the signs on the road, whether to stop, turn right, turn left or go straightforward. The robot will keep following the signs and it will start a timer once it starts its journey and at the end it will display the time taken to complete the journey, bearing in mind that the objects present in its path could affect this time greatly.

The robot will also be able to detect objects on its track at which point one of 2 decisions will be taken, it will either eliminate/remove the object from its path and carry on its path, or it will turn right/left depending on the nearest sign available. The decisions will be taken based upon the signs which means that it will be based on the computer vision subsystem installed in our robot. The system will be trained according to a large dataset of labeled images in order for the robot to be able to recognize and identify different signs and different fonts. The robot will mainly rely on the raspberry pi as its main control system. The navigation of the robot will be through the omni wheels installed on the robot. The robot will also include a mix between proprioceptive and exteroceptive sensors. The exteroceptive will be the ultrasonic sensor installed on the front of the mobile robot to avoid any crashes with any objects. As for the proprioceptive sensors; it will include the voltage and wheel(motor) speed.

Below in figure 1 is the flowchart of how the robot will operate and navigate put simply:



The mobile robotics field has been an incredibly evolving field in the past decade. It has been used extensively by large corporates in almost every industry. This includes industrial, medical, distribution, service robotics and delivery. The market capital of 3 of the large robotics company on their own exceeds a value of 100 billion dollars; this is a huge value in comparison to other recently evolving technologies and fields. One of the important applications of mobile robotics is the exploration and navigation. What makes this field different than other fields, is that this field can’t be done by humans at all when it comes to dangerous places. For example, delivery could be done by both, humans and robots, distribution and medical also can be done by both; humans and robots. Although it would be much wiser to use robots in some applications as they would be faster and more efficient in certain applications, yet these applications can still be done by both; humans and robots. However, on the other hand, the exploration as an application can’t be done by humans in certain areas as it would be too dangerous and it would directly subject humans’ lives to hazard. This is the main reason which led us to choosing this application as our project. We chose the application which humans can’t do, not the application which robots can do better.

Our selected application could be crucial in the sense that in case humans are looking for an object or looking for anything in general, as an advanced example; a sign of life on another planet. Certain areas couldn’t be navigated by humans. So this would leave the options to only two options; either autonomous mobile robots or non-autonomous mobile robots. As a matter of fact, both options would do the required objective. However, it would be smart to use non-autonomous robots, because in case of deploying more than 1 robot to look for something or explore a certain area; communication will be needed between the robots as to avoid certain paths or not re-enter a path which leads nowhere. So in case of non-autonomous robots, the robots will be controlled by humans and humans will find out such information (as to a path is blocked for example) and then the human will inform other humans controlling other robots to not explore that path. On the other hand, in case of autonomous robots, the communication will take place directly between robots which would be much faster and much more intelligent and efficient and hence less time consuming. This gives a lot of value to our application as it puts the first step on this huge project or view and it allows for further continuation on our project where more robots of the same type could be deployed and communication would take place between them and thus it’s not a limited or bounded project.

2. Comparison of proposed robot according to the market survey

According to the available resources with us, these are H/W components which would be used in our robot:

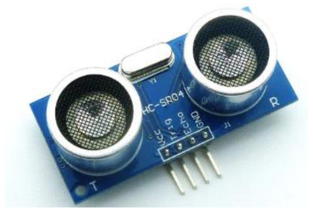
1. **A four wheeled mecanum directional robot chassis (The Trade Name: Omni4WD).**
   (Price: 624$)

   It includes:

   - 4 x 100mm Aluminum Mecanum wheels
   - 3 x 12V DC coreless motors
   - Arduino 328 controller
   - Arduino IO expansion V1.1
   - 12V Ni-Mh battery
   - 12V charge

2. **A raspberry pi as the main control system.**
   (Average Price: 100$)

3. **An ultrasonic sensor.**
   (Average Price: 1.5$)



4. **Raspberry pi camera REV1.3.**

   (Average Price: 25$)

5. **A voltage and motor speed sensor.**

We can summarize the specifications of our robot based on the previous data as following:

- **Size:** it is a medium sized robot, and it is better than the large and small robots because the large one requires more cost for its chassis and the small one requires smaller sensors and components which are more complex and expensive.
- **Cost:** its estimated cost is (from 750 to 800) $ and this cost is limited by our available resources, and it can be decreased by using cheaper components or chassis.

Our robot is also compared with some other robots which mentioned in our market survey:

Robot	Functionality	Size	Cost ($)	Locomotion based on	Reconfigurable Sensors	Embedded PC	Max. Speed (cm/s)
Pioneer P3AT	Research	Large	6000	Wheels	Yes	Optional	70
Khepera III	Research & Education	Small	3000	Wheels	No	No	50
TurtleBot 2	Research & Education	Medium	2000	Wheels	No	Yes	65
Lego EV3	Education	Medium	349	Varies	Yes	No	Varies
Kilobot	Research	Small	100	Vibration	No	No	NI
R-one	Research & Education	Small	NI	Wheels	No	No	25
ExaBot	Research & Education	Medium	250	Caterpillars	Yes	Yes	50
Our robot	Research & Education	Medium	750:800	Wheels	Yes	Yes	60

The S/W technique used in our robot is: Convolution Neural Network (CNN) architecture.
The image is presented to CNN for feature extraction; then it will be passed to the fully connected network that used a regression method to determine steering commands.
The advantage of this technique is that it is fast in making control decisions.

3. Referenced research paper Overview

Paper Title : Real time Traffic Sign detection and recognition for autonomous vehicle (Volume 8 Issue 3 – 2022) (Ahmed J. Abougarair, Mohammed Elmaryul, Mohamed KI Aburakhis)

The main highlights of the paper are presented in the below bulleted points:

Introduction:

- Method Used for Image processing: Convolutional Neural Networks (modified LeNet architecture.)
- The paper depicts an experiment to build a CNN model which can classify traffic signs in real-time effectively using OpenCV.
- A CNN model based on a modified LeNet architecture with four convolutional layers, two max-pooling layers and two dense layers.
- The model was trained and tested with the German Traffic Sign Recognition Benchmark (GTSRB) dataset.
- Parameters were tuned with different combinations of learning rate and epochs to improve the model's performance.

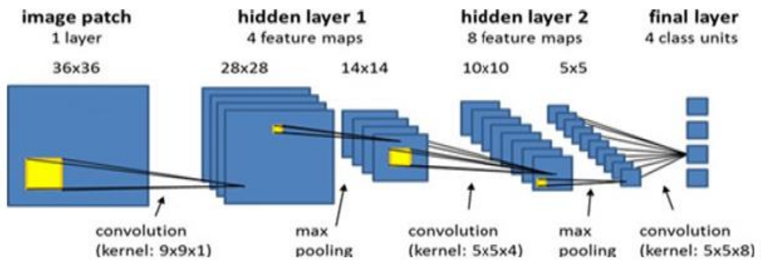
The traffic sign recognition systems include two phases: the first part deals with applying image processing to detect the traffic sign and the second phase entails applying an artificial neural network model for classification.

First Phase:

- The image is extracted using OpenCV and converted to an array of pixels, then resized to 32*32 D-array.
- The transformed array will be preprocessed with the grayscale and histogram equalization method before being supplied to the trained and tested CNN model.
- Every frame of the live video is processed to predict the likelihood of the processed image.

Second Phase

- They used Adaptive Moment Estimation (Adam), a gradient-based approach that accumulates past squared gradients and stores the decaying mean of the past gradients. This optimization method is simple to implement and takes less memory.
- The parameter adjustment was made after evaluating the model, which included a systematic test with numerous combinations of epochs and learning rate.



4. Objective of our CNN model:

In the first phase, the main objective for our current CNN model is to predict two traffic signs i.e. Stop sign and Speed limit 40 sign.

5. Methodology for the training the CNN Model

- Setting up working environment and Libraries
- Dataset Generation
- Preprocessing the images
- Splitting the Datasets
- Building a Convolutional Neural Network
- Training the model
- Plotting the Performance parameters
- Evaluation
- Testing the model

5.1 Setting up working environment and libraries

We developed our code in SPYDER IDE python platform. Different libraries were used throughout the program. The used libraries are:

cv2: OpenCV libraries to load and process the images

numpy: python library to convert the images into arrays

tensorflow: open end machine learning platform to model and train CNN models

os: to load the datasets from the directory

matplotlib: for plotting graphs and show stacked or single images

tensorflow.keras.models: to train the built CNN model using different parameters and algorithms

tensorflow.keras.layers: to create the different models required to create CNN architecture

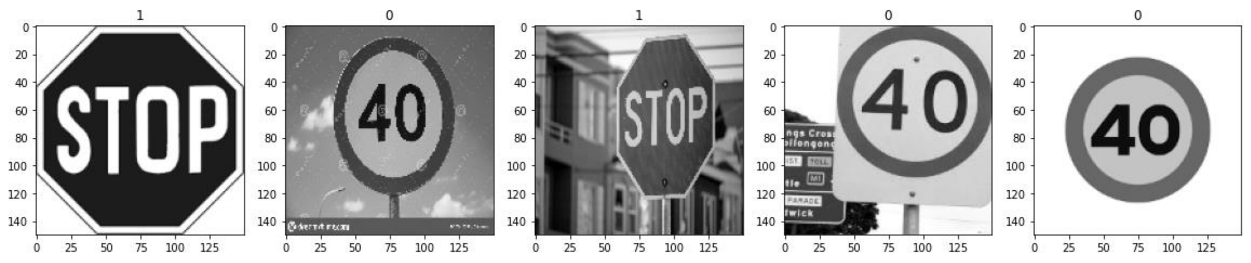
tensorflow.keras.metrics: to analyze the performance metrics and store the history data during training

5.2 Dataset Generation

In this phase we import the different datasets into our IDE environment. We used the in-built data library (tf.keras.utils.image_dataset_from_directory) to load our datasets. The images were imported as grayscale, with image size of (150,150) and batch size was defined as 5. The total images were 95 images divided in two classes. One of the classes has 50 images of speed limit 40 signs and the other has 45 images of stop signs. First class of speed signs was labelled as 0 and second class of stop signs was labelled as 1.

So in total there were total 19 batches created with 5 shuffled images from each class.

As we can see in the figure below for a single batch, we have different images from two classes with the labels 0 and 1.



5.3 Dataset Preprocessing

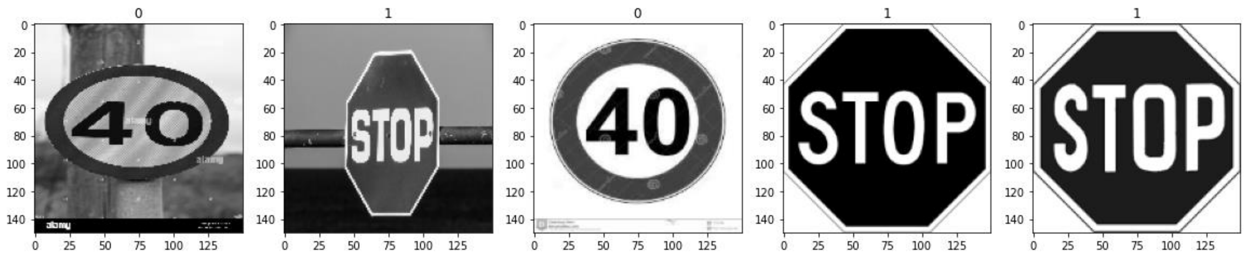
Dataset images were stored under batch array. This array stores the image as two elements of a tuple. The first element dictates the images batch and the second element denotes the class labels for the images.

The images are converted into numpy array using (data.as_numpy_iterator()) function.

We access the different batches of images using data_iterator.next() function. Here we perform the normalization of the entire dataset i.e convert the matrices values from [0 255] to [0 1].

We check the a sample batch of images to verify the preprocessing process which shall be fed into training the model.

The figure below shows a different batch of images with labels which are normalized and ready to be fed into the CNN model.



5.4 Splitting the Datasets

Datasets were split into 3 parts according to the batches. The distribution was defined as follows:

Total no. of batches = 19

Training Datasets= 70% of batches = 13 batches

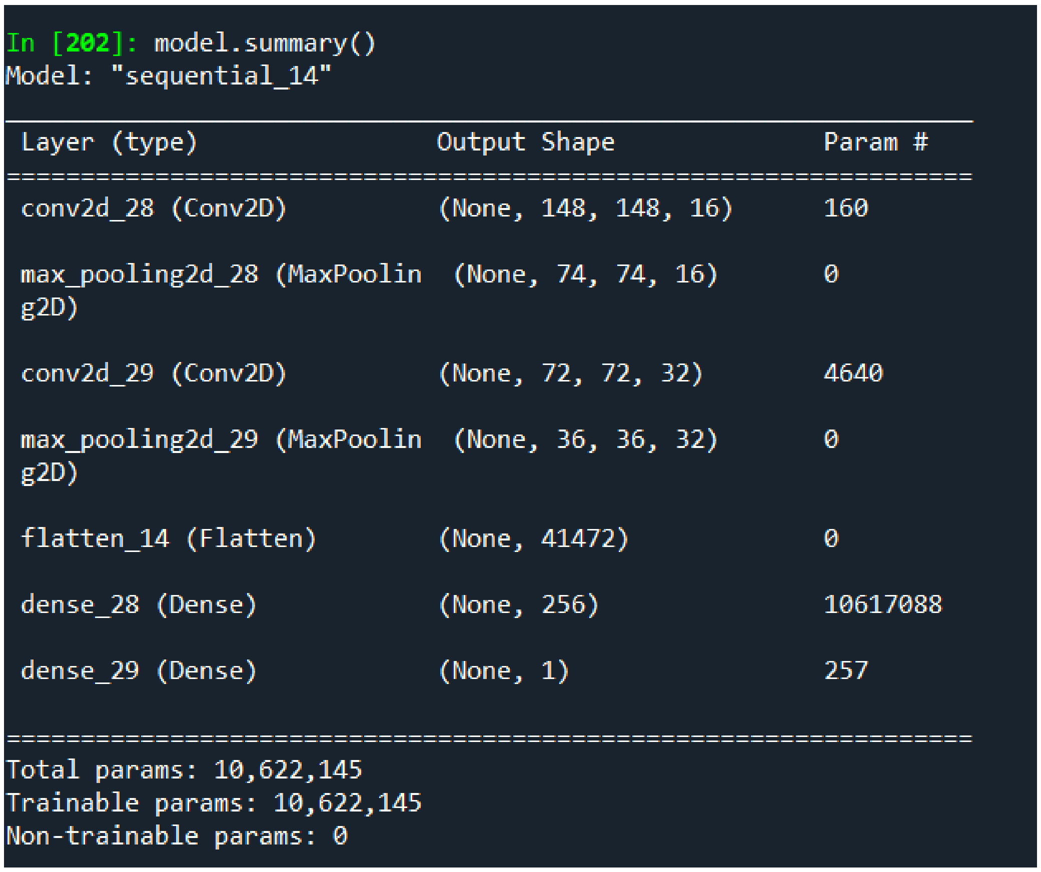
Validation Datasets= 20% of batches = 4 batches

Testing Datasets=10% of batches = 2 batches

We utilized take() and skip() data library functions to select the different batches on random basis and categorize into 3 datasets. These functions randomly take set of batches and assign them to the different datasets, so that we have unique batch of images in each training, validation and testing datasets.

5.5 Building a CNN model

We employed two Conv2D layers , two Max Pooling layers, 1 Flatten layer, 2 Dense layer
Adam (Adaptive Moment Estimation) (a gradient –based approach that accumulates the past gradients) was used to compile the model as this method is simple to implement and takes lower computing time. Since the model is a binary classifier the loss Function was used as losses.BinaryCrossentropy().
The first Convolution has 16 filters with a kernel size of (3,3) and unity stride. The next layer is a Maxpooling layer with default kernel size of (2, 2). The next convolution layer has 32 filters with a kernel size of (3,3) and unity stride. Next a Flatten layer is added to convert into a 1D shape. This is a fully connected layer which is fed to the Dense layer with Relu activation function. Finally, for binary classifier output we employ sigmoid function in the Dense layer.
The detailed model shapes and layers are shown below using model.summary() function.

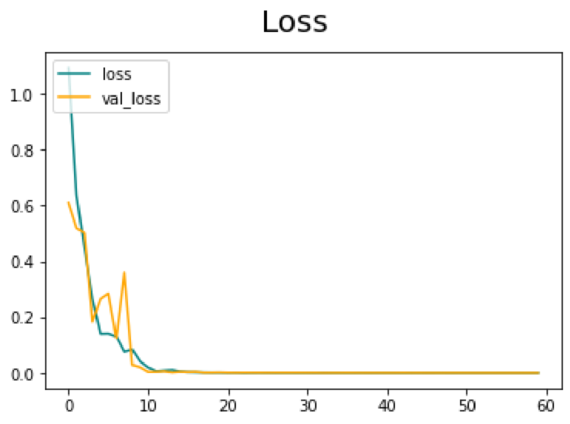


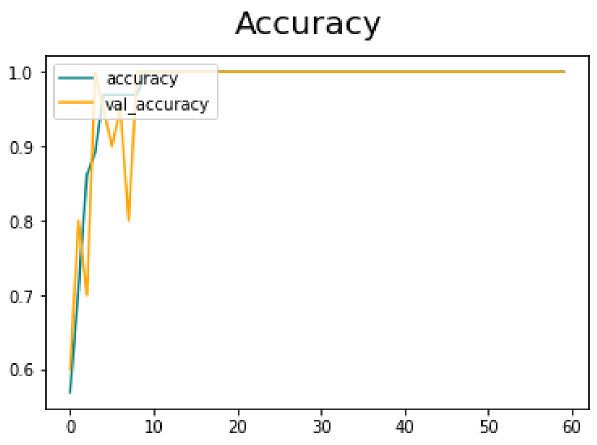
```
In [202]: model.summary()
Model: "sequential_14"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_28 (Conv2D)          (None, 148, 148, 16)      160

 max_pooling2d_28 (MaxPoolin  (None, 74, 74, 16)       0
 g2D)

 conv2d_29 (Conv2D)          (None, 72, 72, 32)        4640

 max_pooling2d_29 (MaxPoolin  (None, 36, 36, 32)       0
 g2D)

 flatten_14 (Flatten)        (None, 41472)             0

 dense_28 (Dense)            (None, 256)               10617088

 dense_29 (Dense)            (None, 1)                 257

=================================================================
Total params: 10,622,145
Trainable params: 10,622,145
Non-trainable params: 0
```

5.6 Training the model

To train the model the default learning rate as 0.001 was unchanged. The model was trained for 60 epochs. The performance metrics were loss and accuracy. The data logged during the model training were used to graph the training and validation datasets losses and accuracy using plt.plot() library function. The graphs for both the history data are depicted below.

Here we see the data loss and accuracy for the dataset are maxed out for accuracy at 100% and Loss minimized at 0% at around 20 epochs. This could be the result of low amount of datasets.





5.7 Evaluating the model

To evaluate the model we compute the indices from the inbuilt Precision() and BinaryAccuracy() functions. The test datasets are used fed into the model to check the precision and binary accuract. The results were 1.0 for both the indices. After the evaluation the model is saved to the working directory.

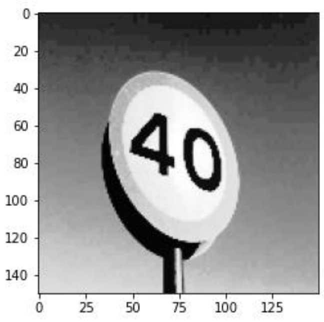
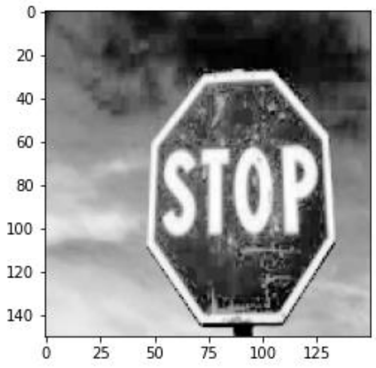
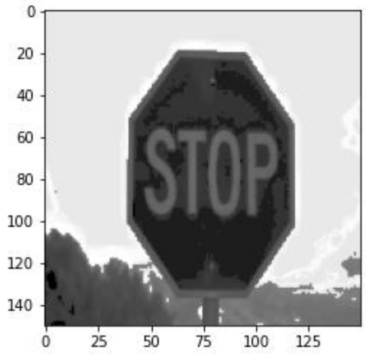
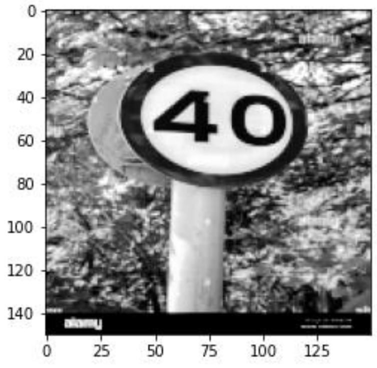
5.8 Testing the model

A sample image was taken outside of the data sets. Before we feed the image to the trained model it needs preprocessing. Firstly, a Image is loaded from directory for e.g. (275,183,3), then resized to the defined dimensions (150,150,3), Next it is converted to Grayscale (150,150,1). After it Histogram Equalization & Normalization was performed.

We need re-shaped the dimensions of the image, since the model accepts image as batches. Basically, here we create copies of the image in a single batch.

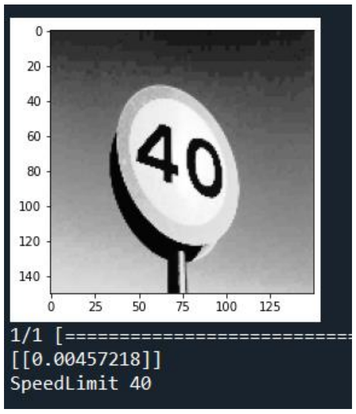
The threshold was testing phase was set at 50%. That implies if the results were >50 % it would print “Stop” with the prediction value and if the prediction value was <50% the result would print “Speed 40”.

Below are some of the sample images and preprocessed images which are fed into the trained CNN model.

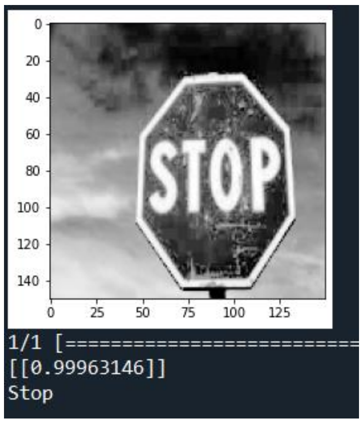





5.9 Results

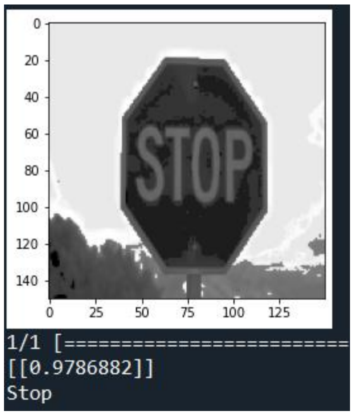
After the above images are sent to the model, we have the output as shown below. Here we see the images are graded between 0 and 1. For example, the prediction value close to 0 signifies the image belongs to class 0 which is the Speed limit 40 class label and vice versa for the stop sign.



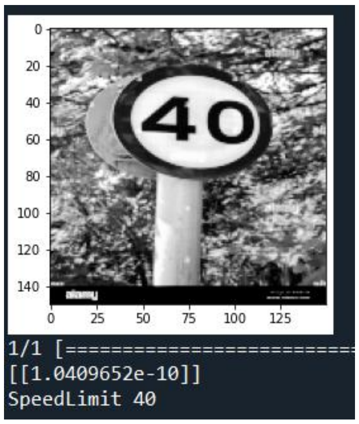
```
1/1 [==========================
[[0.00457218]]
SpeedLimit 40
```



```
1/1 [==========================
[[0.99963146]]
Stop
```



```
1/1 [==========================
[[0.9786882]]
Stop
```



```
1/1 [==========================
[[1.0409652e-10]]
SpeedLimit 40
```

6. Future work plan

The work plan ahead shall be to train and test the model for more number of classes with increased batch sizes as per required by the designed path. We need to compile and test the code for the robot movement and image capture using raspberry-Pi module as well as design a path for the robot to follow during navigation.

Finally, we shall combine the model with the robot to follow navigating the signs placed in front of it.

7. References

- https://companiesmarketcap.com/robotics/largest-companies-by-market-cap/
- https://pyimagesearch.com/2019/07/08/keras-imagedatagenerator-and-data-augmentation/
- https://images.cv/dataset/stop-sign-image-classification-dataset
- https://www.roadtrafficsigns.com/40-mph-signs
- https://anaconda.org/conda-forge/tensorflow
- https://pixabay.com/images/search/stop%20sign/
- https://scikit-learn.org/stable/modules/generated/sklearn.metrics.log_loss.html
- https://keras.io/api/
- https://pyimagesearch.com/
- https://d2l.ai/chapter_linear-classification/index.html