A Project phase-II Report on

Traffic Sign Classification using CNN

Submitted to

The Department of Information Technology

In the partial fulfilment of the academic requirements of

Jawaharlal Nehru Technological University

For

The award of the degree of

Bachelor of Technology

In

Information Technology

(2018 – 2022)

By

B.R.ASHRITH	**18311A12C7**
MOVEENA NITESHINI	**18311A12D0**
ROHITH DABLA	**18311A12G3**

Under the guidance of

MS.K.Srilaxmi

Assistant Professor

Department of Information Technology

School of Computer Science and Informatics

Sreenidhi Institute of Science and Technology (An Autonomous Institution)

Yamnampet, Ghatkesar, R.R. District, Hyderabad -501301

Affiliated to

Jawaharlal Nehru Technological University Hyderabad

Hyderabad – 500085

Department of Information Technology

Sreenidhi Institute of Science and Technology

Certificate

This is to certify that this Group project report on **"TRAFFIC SIGN CLASSIFICATION USING CNN"**, submitted by B.R.Ashrith (18311A12C7), Moveena Niteshini (18311A12D0), Rohith Dabla (18311A12G3) in the year 2022 in partial fulfilment of the academic requirements of Jawaharlal Nehru Technological University for the award of the degree of Bachelor of Technology in Information Technology , is a bonafide work that has been carried out by them as a part of their Group Project during Fourth Year Second Semester, under our guidance. This report has not been submitted to any other institute or university for the award of any degree.

Internal guide	**Project Co-ordinator**	**Head of Department**
Ms. K. Srilaxmi	Dr. K. Kranthi Kumar	Dr. Sunil Bhutada
Assistant Professor	Associate Professor	Professor and HOD
Department of IT	Department of IT	Department of IT

External Examiner

Date:-

Declaration

We, **B.R.Ashrith (18311A12C7), Moveena Niteshini (18311A12D0) and Rohith Dabla (18311A12G3)**, students of **SREENIDHI INSTITUTE OF SCIENCE AND TECHNOLOGY, YAMNAMPET, GHATKESAR**, studying IVth year IInd Semester, **INFORMATION TECHNOLOGY** solemnly declare that the Group Project work, titled **"TRAFFIC SIGN CLASSIFICATION USING CNN"** is submitted **to SREENIDHI INSTITUTE OF SCIENCE AND TECHNOLOGY** for partial fulfilment for the Award of the degree of Bachelor of Technology in **INFORMATION TECHNOLOGY.**

It is declared to the best of our knowledge that the work reported does not form part of any dissertation submitted to any other University or Institute for an award of any degree.

ACKNOWLEDGEMENT

I would like to express my gratitude to all the people behind the screen who helped me to transform an idea into a real application.

I would like to express my heartfelt gratitude to my parents, without whom I would not have been privileged to achieve and fulfill my dreams. I am grateful to our CEO, **Mr.K.Abhijit Rao**, Director, **Prof.C.V.Tomy**, Principal, **Dr. T. CH. SIVA REDDY,** who most ably runs the institution and has had a major hand in enabling me to do my project.

I profoundly thank **Dr. Sunil Bhutada**, Head of the Department of Information Technology, who has been an excellent guide and also a great source of inspiration to my work.

I would like to thank my internal guide **Ms.K Srilaxmi,** for her technical guidance, constant encouragement, and support in carrying out my project at college.

I would like to thank my coordinator **Dr.K.Kranthi Kumar, Associate professor,** for his technical guidance, constant encouragement and support in carrying out my project at college.

The satisfaction and euphoria that accompany the successful completion of the task would be great but incomplete without the mention of the people who made it possible with their constant guidance and encouragement crowns all the efforts with success. In this context, I would like to thank all the other staff members, both teaching and non-teaching, who have extended their timely help and eased my task.

INDEX

ABSTRACT

This project presents an effective solution to detecting traffic signs on road by first classifying the traffic sign images using Convolutional Neural Network (CNN) on the German Traffic Sign Recognition Benchmark (GTSRB)and then detecting the images of Indian Traffic Signs using the Indian Dataset which will be used as testing dataset while building classification model. Therefore this system helps electric cars or self driving cars to recognize the traffic signs efficiently and correctly. The system involves two parts, detection of traffic signs from the environment and classification based on CNN thereby recognizing the traffic sign. The classification involves building a CNN model of different filters of dimensions 3 × 3, 5 × 5, 9 × 9, 13 × 13, 15 × 15,19 × 19, 23 × 23, 25 × 25 and 31 ×31 from which the most efficient filter is chosen for further classifying the image detected.

LIST OF FIGURES

1. INTRODUCTION

Humans are becoming more dependent on technology during this age of Artificial Intelligence. due to technological breakthroughs, international companies reminiscent of Google, Tesla, Uber, Ford, Audi, Toyota, Mercedes-Benz, et al are functioning on automating autos. They're working on creating more precise autonomous or self-driving cars. you'll have detected about self-driving vehicles, that perform on the road while not the requirement for human involvement and act sort of a driver. It' cheap to rely on the security implications—the chance of major mechanical malfunctions. Humans, on the opposite hand, are more actual than any machine. Researchers are testing a range of algorithms to ensure total road safety and accuracy. you'll face many traffic signs once driving on the road, reminiscent of traffic signals, flip left or right, speed limitations, no passing of enormous vehicles, no entering, kids crossing, so on, that you need to respect so as to drive safely. Autonomous vehicles must additionally analyse these indications and create choices in order to realize accuracy. the method of deciding which category a traffic sign belongs to is thought as traffic sign classification. during this Deep Learning project, we'll develop a model for categorising traffic signs in a picture into multiple classes using a convolutional neural network (CNN) and therefore the Keras framework.

1.1 Scope

The ultimate objective is to develop a system that would allow traffic signs to be catalogued. This technology can assist local or national authorities with the process of maintaining and improving their road and traffic signs by automatically detecting and categorising one or more traffic signs from a complex image collected by a vehicle's camera. The trick is to find the ideal colour combination in the scene, where one colour is contained within the convex hull of another colour, and then combine it with the proper shape. If a candidate is found, the algorithm uses the rim pictogram combination to try to identify the object and then delivers the categorization result. The objectives are as follows: For the identification challenge, to have a deeper grasp of the characteristics of road and traffic signs and how they effect picture processing. To get a thorough understanding of colour, colour spaces, and colour space conversion. To develop dependable colour segmentation algorithms that can be used in a wide range of situations. Create a recognizer that is invariant to in-plane transformations including translation, rotation, and scaling using invariant shape measurements. To find the most efficient way to extract characteristics from traffic signs. To develop an appropriate traffic sign classification algorithm. To test the aforementioned techniques' durability in a variety of weather, lighting geometry, and sign circumstances.

1.2 Existing System

Most earlier efforts employ hand-crafted feature extraction approaches like Scale-Invariant Feature Transform (SIFT) and Histogram of Oriented Gradient to tackle the target recognition problem (HOG). Traditional feature extraction approaches, which are overly reliant on the designer's knowledge, encounter several limitations in feature expression. The CNN-based deep network approach, on the other hand, is more powerful in terms of feature expression. Weight sharing may be achieved through local receptive fields using the CNN approach, which

has rotation, translation, and scaling invariance. Image classification, facial recognition, and pedestrian detection are all examples of target recognition sub-areas where it's commonly employed.

1.3 Proposed System

We will contribute two things to this project: one, we will provide a new dataset for traffic and road signs, and the other, we will create and construct a deep CNN architecture for traffic sign identification. The gathered data set is put into the specified CNN architecture for training, validation, and testing. The next slide has a full explanation of the CNN architecture. Once the CNN has been trained, it may be used to categorise fresh pictures that were not included in the dataset. For traffic sign identification, a Deep CNN architecture is also proposed. Between the input and output layers, CNNs often include numerous hidden layers. Python is used to implement the suggested CNN's design.

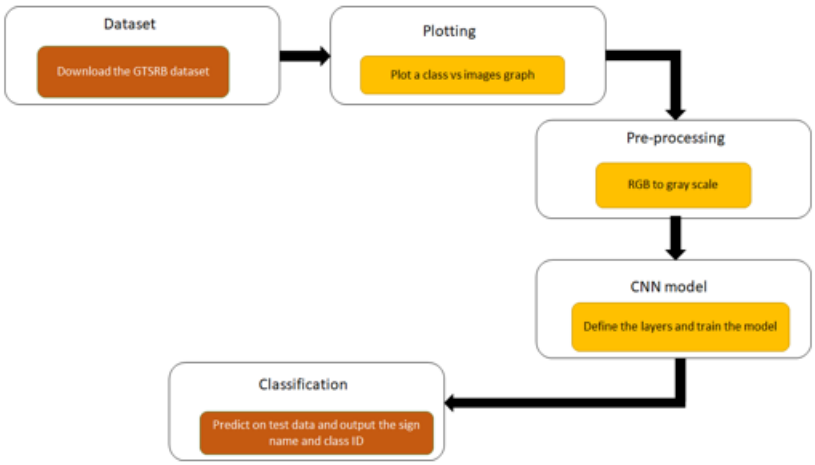


Fig 1.1 Flow chart of Proposed System

1.4 Traffic Signs

Fig 1.2 Traffic Signs

2. System Analysis

This System Study is strongly connected to the analysis of requirements. It's also "an explicit formal inquiry done to aid someone (referred to as the decision-maker) in selecting a better course of action and arriving at a better conclusion than he would have reached on his own." This stage includes breaking down the system into discrete pieces to assess the situation, reviewing project goals, breaking down what needs to be constructed, and attempting to involve users in order to identify unambiguous requirements.

2.1 Functional Requirement Specification

Following rigorous examination, the System has been determined to contain the following modules.

System Module: This module is in charge of carrying out our project's image processing responsibilities. We're doing a live categorization to double-check traffic signs, therefore we need the camera's input.

2.2 Performance Requirements

The output of the software is utilised to assess its effectiveness. The specification of requirements is crucial in the analysis of a system. It is only feasible to develop a system that will fit into the proper environment if the main requirements are written correctly. Because they will be the ones to utilise the system in the end, the existing system's users are primarily responsible for providing the required needs. This is because early in the project, needs must be identified in order for the system to be created to meet them. It's extremely difficult to update a system once it's been developed, and much more difficult to build a system that doesn't adapt to the user's demands.

Any system's requirement statement may be summarised as follows:

- The system should be able to communicate with other systems.
- The system must be precise.
- The system should be superior to the current system.
- The current system is fully reliant on the user to execute all tasks.

2.3 Software Requirements

- **Operating System:** Windows 10
- **Technology:** Deep Learning, Image Processing, Machine Learning.
- **Programming Language**: Python
- **IDE**: Jupyter Notebook, PyCharm.
- **Software libraries:** Numpy, mathplotlib, scikitlearn, keras, Tensorflow, and OpenCV.

2.4 Hardware Requirements

- Processor: Intel Core i5 or higher.
- RAM: 8 GB and higher.
- Graphics Card: NVIDIA GeForce 1050 Ti (or higher)
- Hard disk: 128GB or higher

3.SYSTEM DESIGN

Systems design is the process of defining a system's architecture, components, modules, interfaces, and data in order to satisfy particular requirements. It may be thought of as a systems theory application in product development. Object-oriented analysis and design approaches are rapidly gaining popularity as a means of designing computer systems.

3.1 Architectural Design

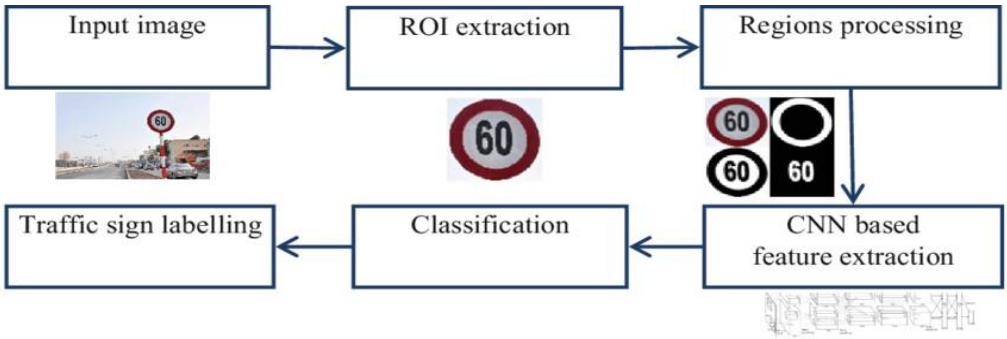


Fig 3.1 Architectural Design

The first is to create a new traffic and road sign database, and the second is to create and construct a deep CNN architecture for traffic sign identification. The high-level perspective of the system is shown in Fig. 3.1. For training, validation, and testing, the acquired data set is fed into the suggested CNN architecture. In the next part, we'll go through the CNN architecture in depth. Once the CNN has been trained, it may be used to categorise fresh pictures that were not included in the dataset. This is the first time a comprehensive database on Traffic has been created.For traffic sign identification, a Deep CNN architecture is also proposed. Between the input and output layers, CNNs often include numerous hidden layers. Python is used to implement the suggested CNN's design.

3.2 Modules

System Module:

The module is responsible for classifying image of traffic signs. It is also responsible for pre-processing the live image stream into a format that is suitable for the model to perform prediction. Some of these tasks involve converting an image to grayscale and resizing the image.

3.3 UML Diagrams

UML Diagrams for our application are as follows:

3.3.1 Use Case Diagrams

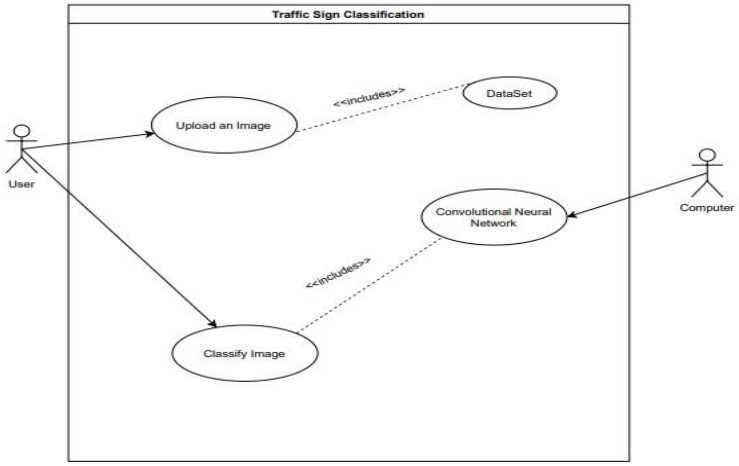


Fig 3.2 Use case Diagram for System Module

The functioning of the main system is depicted in this Use Case Diagram. This module's key features are the ability to upload and categorise images. It also uses categorization to classify the photos. It collaborates with CNN to assess whether the traffic sign is valid. Depending on which result appears on the user's screen.

3.3.2 Class Diagrams

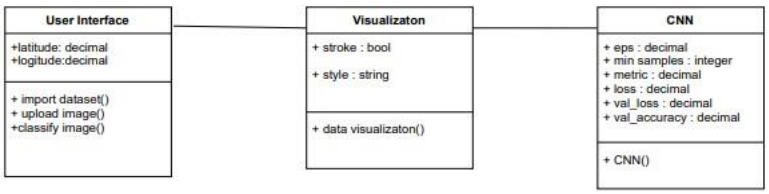


Fig 3.3 Class Diagram for Traffic Sign Classification

We discovered three primary classes in traffic sign classification: User Interface, Visualisation, and CNN model. We've also specified each of these classes' characteristics and methods, as well as the links between them. As needed, dependence and association

connections are used to explicitly characterise the nature of the relationship. The above class diagram depicts these classes, along with their associated characteristics and actions.

3.3.3 Sequence Diagrams

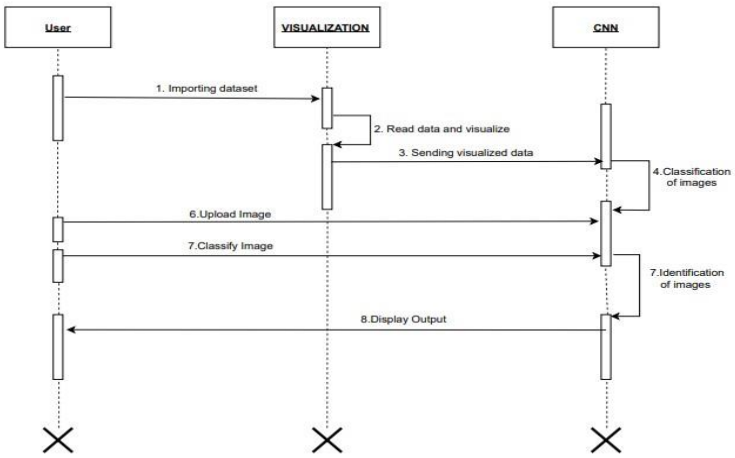


Fig 3.4 Sequence Diagram for Traffic Sign Classification

The above sequence diagram for Traffic Sign Classification shows the user uploading a picture, which allows the System to collect the image and do different preprocessing before applying a traffic sign classifier. Finally, we load the model we've trained and run a prediction, with the system displaying the outcome.

3.3.4 Collaboration Diagrams

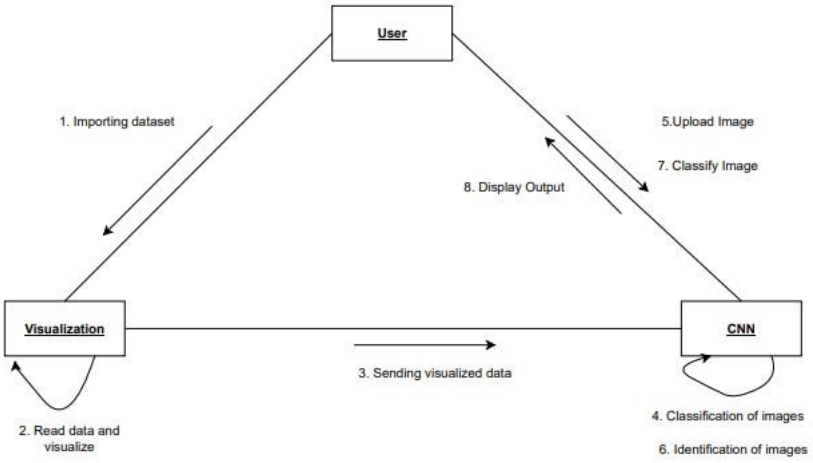


Fig 3.5 Collaboration Diagram for Traffic Sign Classification

The Collaboration Diagram for Traffic Sign Classification is illustrated above, where the user must input a picture in order for the System to collect it and do different preprocessing before

applying a traffic sign classifier. Finally, we load the model we've trained and run a forecast, after which the system displays the results.

3.3.5 Activity Diagrams

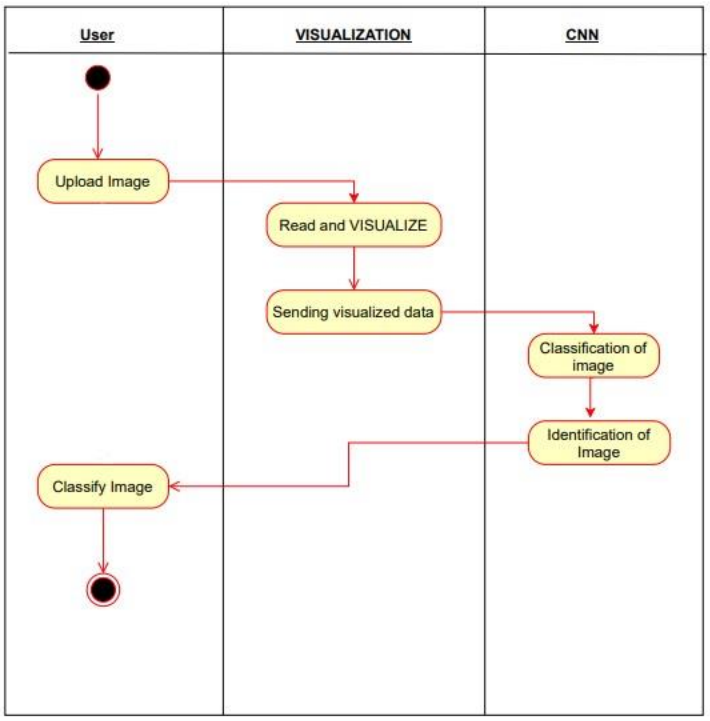


Fig 3.6 Activity Diagram for Traffic Sign Classification

Activity diagram depicts the flow of the events in the whole system as shown above.

4. System Implementation

The implementation stage of any project is a realistic portrayal of the critical moments that influence whether or not a project succeeds. During the implementation stage, the system or system modifications are installed and made operational in a production environment. The phase begins once the system has been tested and accepted by the user. This stage will persist until the system is in production and meets the set of user requirements.

4.1 Dataset

The dataset was taken from The German Traffic Sign Recognition Benchmark (GTSRB) and was initially presented at the International Joint Conference on Neural Networks' single-image classification challenge (IJCNN). It was made using around 10 hours of footage taken while driving on various highways in Germany during the day. About 40.000 authentic vivid photographs of traffic signs make up the data. The photos have a.ppm extension and are 15x15 to 250x250 pixels in size..

4.2 Libraries

For our Project, we need the following libraries: some standard ones as NumPy, OS, and Matplotlib; cv2, a powerful library developed for solving computer vision tasks; sklearn.model_selection.train_test_split for splitting the dataset into train and test subsets; some components from tf.keras.models and tf.keras.layers for building the model.

4.3 Reading and Pre-processing image files

We start with reading images from the dataset. The images are distributed over 43 folders representing 43 classes. We loop through folders and images, open them, resize to 32x32 pixels, convert from RGB to gray, and save them as np arrays. We normalize the data now: we divide images by 255 to get pixel values between 0.0 and 1.0. Finally, we have a total amount of 39.209 images assigned to 43 classes. When working with images, it is always worth looking at some samples. Let’s take 25 random images from the dataset and show them with their labels. In this set of images, we can see some possible misclassification problems. For example, on image 7 it is hard to recognize the number, images 13 and 14 are too dark. Image 8 (“Traffic signals”) could be probably misclassified as “General caution”.

4.4 Building The Model

Before creating the model, we have to split our dataset into train and test subsets. For the test subset, we take out a standard 20% of the dataset. In addition, it is important to make sure that our data has np.float32 format. For our classification task, we will use an implementation of LeNet-5 Convolutional Neural Network. LeNet-5 was designed by Yann LeCun et al. and

others [3] in 1998 and was one of the earliest convolutional neural networks. Its architecture is extremely simple but very efficient. There are three Convolutional Layers based on 5x5 filters and followed by average pooling with 2×2 patches. We use the ReLU function for activation as it leads to faster training. Then we add Dropout Layer with a factor of 0.2 to deal with overfitting. It means that 20% of the input will be randomly nullified to prevent strong dependencies between layers. We end up with Flattening and two Dense Layers. In the last Dense Layer, we have to assign the number of neurons equal to the number of classes, and the Softmax activation function to get probabilities between 0.0 and 1.0. The resulting number of parameters in this network is 70,415. Now with the model**.**compile method, we configure the model. The Adam learning rate optimization algorithm is an extension to Stochastic Gradient Descent and is a good option in terms of training speed. The Categorical Cross-Entropy loss function fits here as it delivers us multiclass probability distribution for our classification problem. For performance evaluation of the model, we will take accuracy metrics.

4.5 Parameters obtained:

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 28, 28, 60)        1560
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 24, 24, 60)        90060
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 12, 12, 60)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 10, 10, 30)        16230
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 8, 8, 30)          8130
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 4, 4, 30)          0
_________________________________________________________________
dropout (Dropout)            (None, 4, 4, 30)          0
_________________________________________________________________
flatten (Flatten)            (None, 480)               0
_________________________________________________________________
dense (Dense)                (None, 500)               240500
_________________________________________________________________
dropout_1 (Dropout)          (None, 500)               0
_________________________________________________________________
dense_1 (Dense)              (None, 43)                21543
=================================================================
Total params: 378,023
Trainable params: 378,023
Non-trainable params: 0
_________________________________________________________________
None
```

Fig 4.1 Trainable Parameters

Methods and arguments used

Sequential:

It groups a linear stack of layers into a tf.keras. Model.

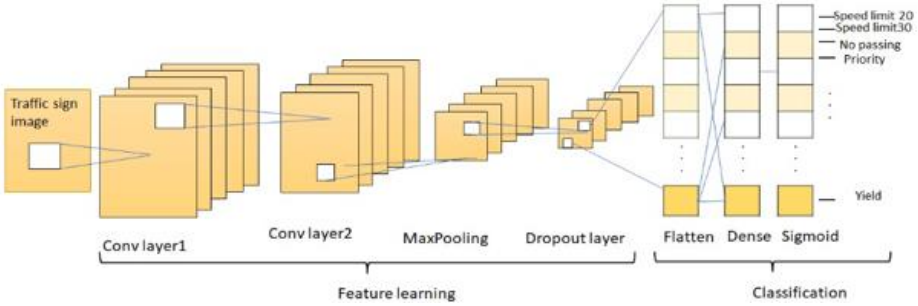


Fig 4.2 Convolution Neural Network

Conv2d:

By convolving the layer input with a convolution kernel, this layer yields a tensor of outputs. If utilise bias is True, a bias vector is created and attached to the outputs. Finally, if activation is set to None, the outputs will be altered.

When utilising this layer as the first layer in a model, use the keyword parameter input shape (tuple of integers or None, does not contain the sample axis), for example, input shape= (128, 128, 3) for 128x128 RGB photos in data format="channels last." When the size of a dimension may be changed, you can use None.

Activation Function:

An Activation layer or the activation argument, which is supported by all forward layers, can be used to use activations.

The activation function is a node in a Neural Network that is positioned at the end or in the centre. They help to determine whether a neuron will fire or not.

ReLU:

The rectified linear unit activation function is used.

This returns the typical ReLU activation with default values: max (x, 0), the element-wise maximum of 0, and the input tensor.

You can utilise non-zero thresholds, adjust the activation's max value, and use a non-zero multiple of the input for values below the threshold by changing the default settings.

The activation function in a neural network is responsible for converting the node's summed weighted input into the node's activation or output for that input.

The rectified linear activation function, or ReLU, is a piecewise linear function that, if the input is positive, will output it directly. Otherwise, it will return a value of zero.

- Due of the vanishing gradient problem, the sigmoid and hyperbolic tangent activation functions cannot be employed in networks with numerous layers.
- The vanishing gradient problem is solved with the rectified linear activation function, allowing models to train quicker and perform better.
- When creating multilayer Perceptron and convolutional neural networks, the rectified linear activation is the default activation.

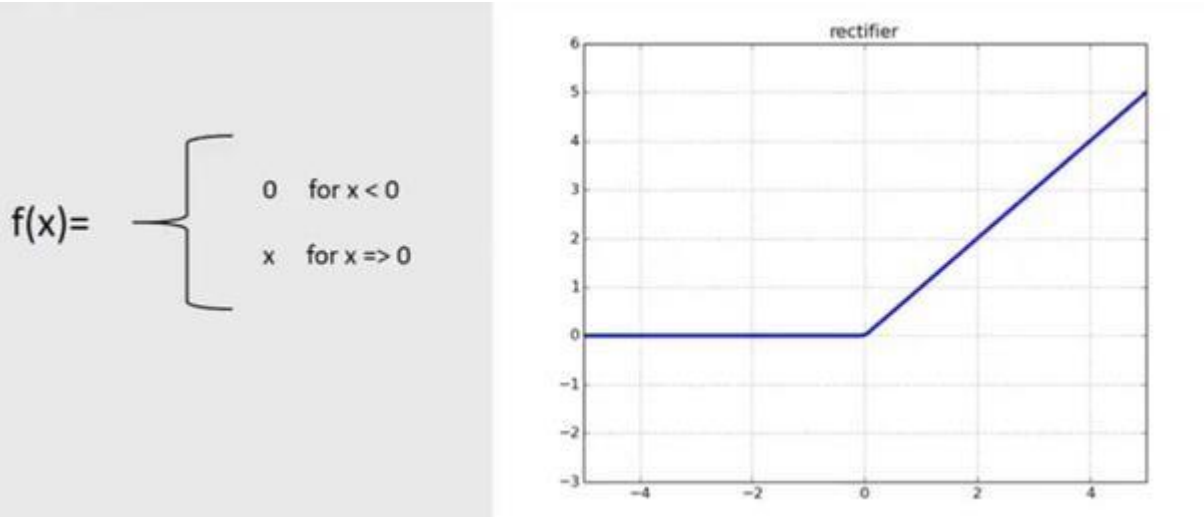


Fig 4.3 ReLu Activation Function

Softmax:

- Softmax creates a probability distribution from a set of values.
- The output vector's members are in the range (0, 1) and add up to 1.
- Each vector is dealt with separately. The axis parameter specifies the axis of the input to apply the function upon.
- Because the outcome may be understood as a probability distribution, Softmax is frequently employed as the activation for the final layer of a classification network.
- Each vector's softmax is calculated as exp(x) / tf.reduce sum(exp(x)).
- The log-odds of the derived probability are the input values.

Formula:

$$\sigma(\vec{z})_i = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

E.g.,

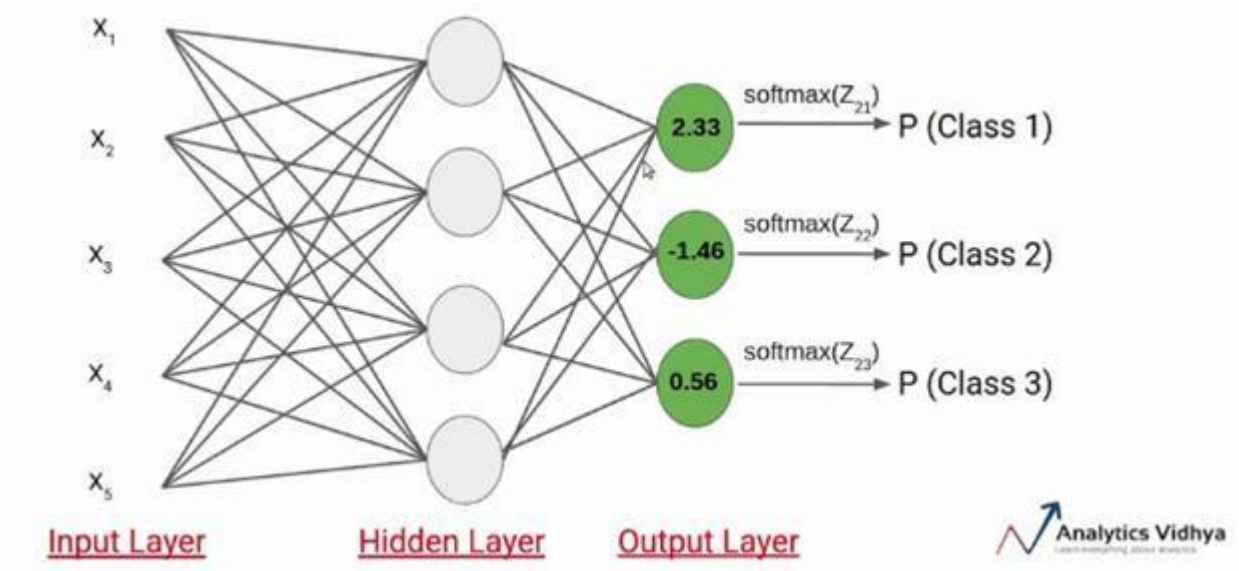


Fig 4.4 Softmax Activation Function

MaxPooling2D:

For 2D spatial data, the max pooling operation is used.

Downsamples the input along its spatial dimensions (height and width) by obtaining the maximum value for each channel of the input over an input window (of size determined by pool size). Each dimension of the window is adjusted by strides.

When utilising the "valid" padding option, the final output has the following spatial form (number of rows or columns): output shape = math.floor((input shape - pool size) / strides) + 1 (when input shape >= pool size)

When using the "same" padding option, the output shape is: output shape = math.floor((input shape - 1) / strides) + 1

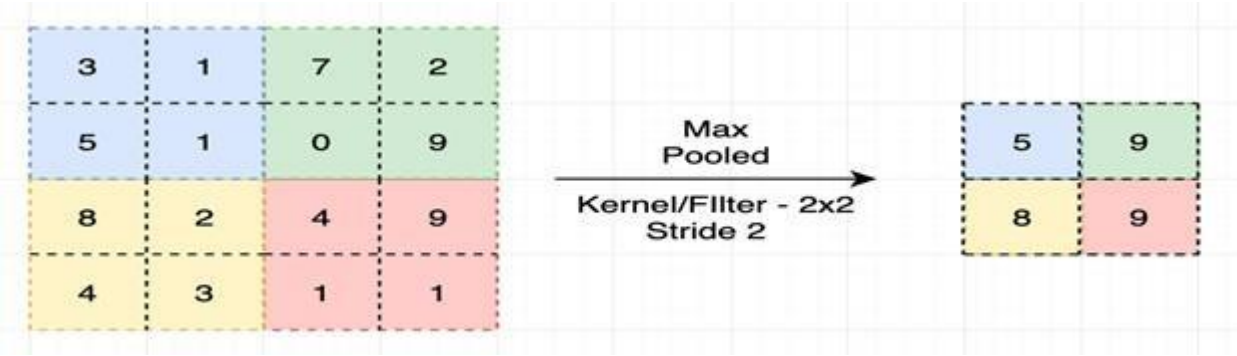


Figure 4.5 MaxPooling2d

Example Flatten:

Flattening is converting the data into a 1-dimensional array for inputting it to the next layer.

We **flatten** the **output** of the **convolutional** layers to create a single long feature vector.

Figure 4.6 Flattening Example

Dropout Layer:

The input is subjected to dropout. It accepts the following as a parameter: To decrease a percentage of the input units, float between 0 and 1.

The Dropout layer reduces overfitting by changing input units to 0 at random with a rate frequency at each step during training. Inputs that aren't set to 0 are scaled up by 1/(1 - rate) to maintain the entire amount.

It's worth mentioning that the Dropout layer only works if the training option is set to True, implying that no data is lost during the inference process. When using a model.fit, training is set to True by default, but you can also explicitly set the kwarg to True when utilising the layer in other circumstances.

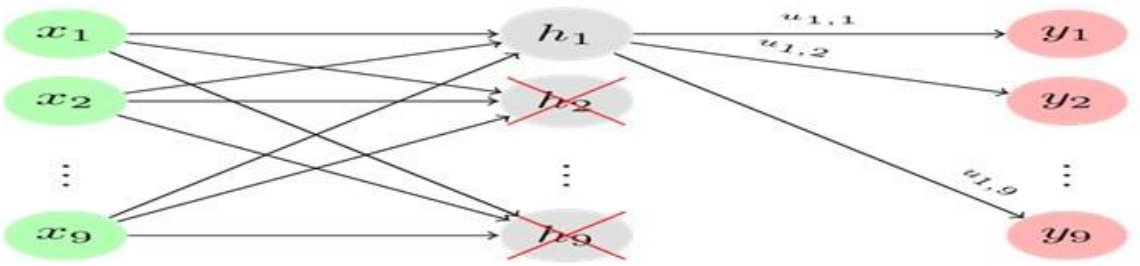


Figure 4.7 Dropout Layer

Dense Layer:

Dense implements the operation output = activation (dot (input, kernel) + bias), where activation is the element-wise activation function supplied as the activation parameter, a kernel is the layer's weights matrix, and bias is the layer's bias vector (only applicable if use bias is True). All of these traits are present in dense.

If the input to the layer has a rank greater than 2, Dense computes the dot product between the inputs and the kernel along the final axis of the inputs and axis 0 of the kernel (using tf.tensordot). If an input has dimensions (batch size, d0, d1), we create a kernel with shape (d1, units) that acts on every sub-tensor of shape (1, 1, d1) along axis 2 of the input (there are batch size * d0 such sub-tensors). The output in this case will be shaped (batch size, d0, units).

In addition, once a layer is called, its characteristics cannot be modified (except the trainable attribute). If a common kwarg input shape is specified, Keras will construct an input layer to insert before the current layer. This is the equivalent of explicitly declaring an InputLayer.

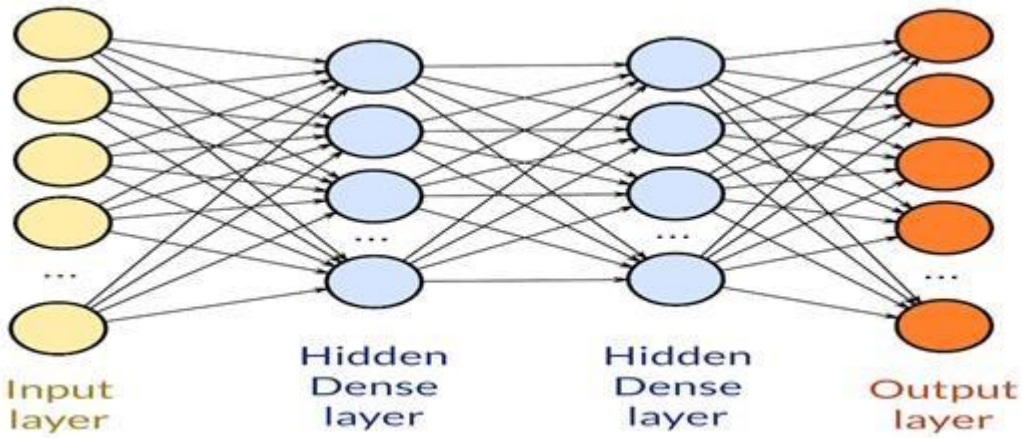


Fig 4.8 Dense Layer

Libraries or modules used:

1.Os:

You may access operating system-dependent functionalities on the fly using this module. If you only want to read or write a file, use open(); if you want to change paths, use os.pathmodule; and if you want to read all the lines in all the files on the command line, use fileinput. For creating temporary files and directories, see the tempfile module, and for managing high-level files and directories, see the shuttle module.

2.numpy:

NumPy is a Python library that allows you to interact with arrays. It also provides functions for working with matrices, fourier transforms, and linear algebra.

keras.models.load_model():

It is a method in keras model library to load the weights of the saved model.

5.OUTPUT SCREENS

Output Screens of various functionalities in our application are shown over here, along with the description.

```
Epoch 1/15
696/696 [==============================] - 122s 176ms/step - loss: 1.4509 - accuracy: 0.5940 - val_loss: 0.3094 - val_accuracy: 0.9139
Epoch 2/15
696/696 [==============================] - 123s 176ms/step - loss: 0.4240 - accuracy: 0.8696 - val_loss: 0.1284 - val_accuracy: 0.9616
Epoch 3/15
696/696 [==============================] - 121s 174ms/step - loss: 0.2894 - accuracy: 0.9120 - val_loss: 0.1081 - val_accuracy: 0.9670
Epoch 4/15
696/696 [==============================] - 117s 169ms/step - loss: 0.2205 - accuracy: 0.9312 - val_loss: 0.0825 - val_accuracy: 0.9749
Epoch 5/15
696/696 [==============================] - 122s 175ms/step - loss: 0.1905 - accuracy: 0.9384 - val_loss: 0.0536 - val_accuracy: 0.9859
Epoch 6/15
696/696 [==============================] - 121s 174ms/step - loss: 0.1657 - accuracy: 0.9489 - val_loss: 0.0675 - val_accuracy: 0.9845
Epoch 7/15
696/696 [==============================] - 119s 171ms/step - loss: 0.1581 - accuracy: 0.9509 - val_loss: 0.0559 - val_accuracy: 0.9842
Epoch 8/15
696/696 [==============================] - 119s 171ms/step - loss: 0.1367 - accuracy: 0.9564 - val_loss: 0.0457 - val_accuracy: 0.9865
Epoch 9/15
696/696 [==============================] - 119s 171ms/step - loss: 0.1152 - accuracy: 0.9626 - val_loss: 0.0392 - val_accuracy: 0.9894
Epoch 10/15
696/696 [==============================] - 117s 168ms/step - loss: 0.1172 - accuracy: 0.9635 - val_loss: 0.0359 - val_accuracy: 0.9904
Epoch 11/15
696/696 [==============================] - 119s 171ms/step - loss: 0.1056 - accuracy: 0.9657 - val_loss: 0.0308 - val_accuracy: 0.9907
Epoch 12/15
696/696 [==============================] - 117s 169ms/step - loss: 0.0968 - accuracy: 0.9692 - val_loss: 0.0348 - val_accuracy: 0.9908
Epoch 13/15
696/696 [==============================] - 118s 169ms/step - loss: 0.0916 - accuracy: 0.9726 - val_loss: 0.0399 - val_accuracy: 0.9907
Epoch 14/15
696/696 [==============================] - 120s 172ms/step - loss: 0.0860 - accuracy: 0.9726 - val_loss: 0.0300 - val_accuracy: 0.9925
Epoch 15/15
696/696 [==============================] - 122s 175ms/step - loss: 0.0848 - accuracy: 0.9743 - val_loss: 0.0329 - val_accuracy: 0.9914
```

Fig 5.1 Accuracy and Validation accuracy for our model

Our model's accuracy and validation accuracy are depicted in the diagram above. An epoch is a unit of time used to train a neural network using all of the training data for a single cycle. We use all of the data exactly once in an era. A forward and backward pass are combined to make one pass: An epoch is made up of one or more batches in which we train the neural network using a portion of the dataset. Our model has a 97.43 percent accuracy and a 99.14 percent validation accuracy.

PLOT THE TRAINING ACCURACY AND VALIDATION ACCURACY

```
In [21]: plt.plot(history.history['accuracy'],'r-*',label='training accuracy')
         plt.plot(history.history['val_accuracy'],label='validation accuracy')
         plt.xlabel('epochs')
         plt.ylabel('loss')
         plt.legend()
         plt.show()
```

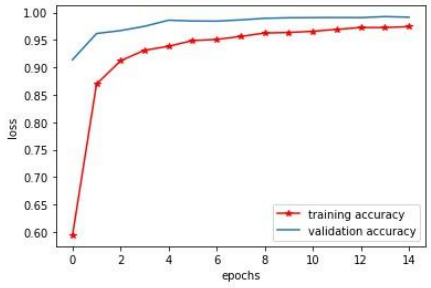


Fig 5.2 Plot of training and validation accuracy

From the curves of training and validation accuracy, we can see that the model performs well on the training data compared to the validation data. With this, we can proceed with the project.

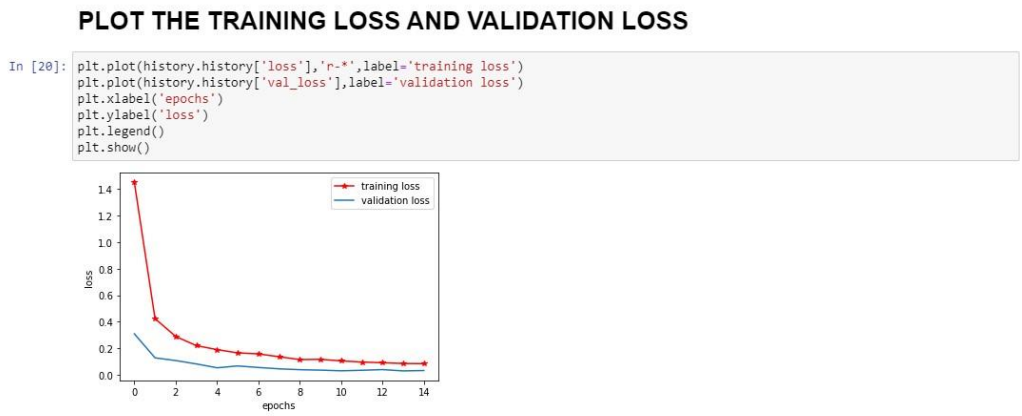


Fig 5.3 Plot of Training and Validation loss

From the curves of training and validation loss, we can see that validation loss is greater than the training loss.

From the previous two plots, we can say that the system is trained on the training data and tested on the validation data, which it has not seen during the training; it's new for the model and hence the reason for this behaviour.

With these results, we can proceed with using the model in our application. If the results were not satisfactory, we would have to perform hyperparameter tuning on our model. In this case, it is not required.

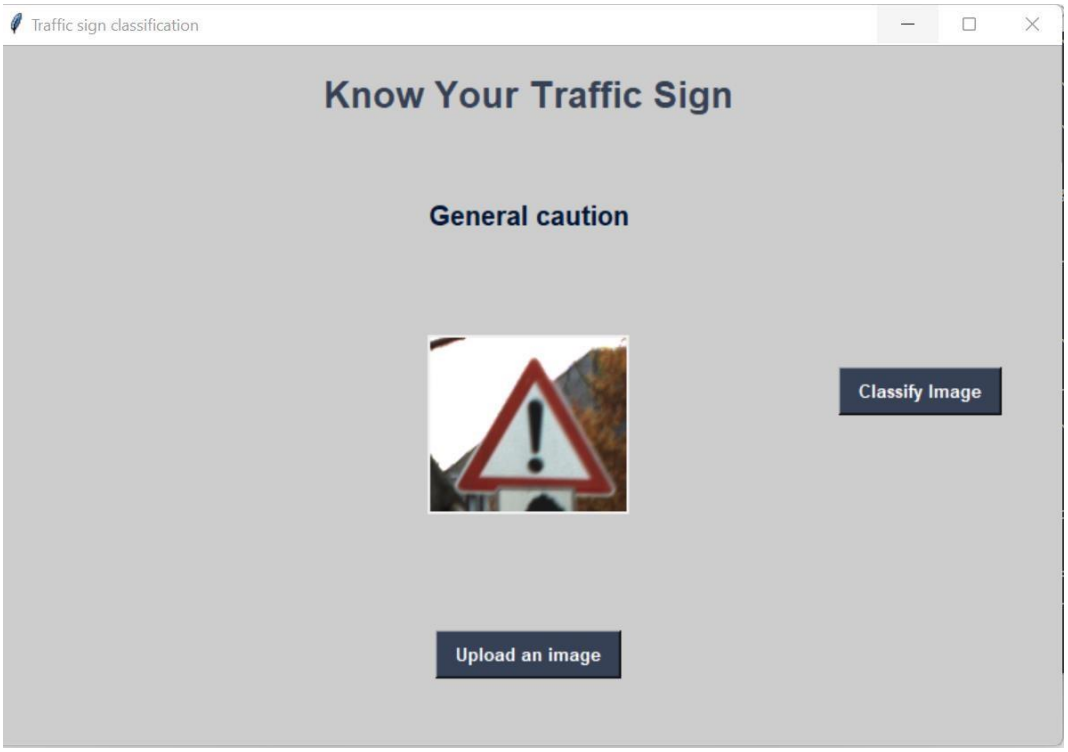


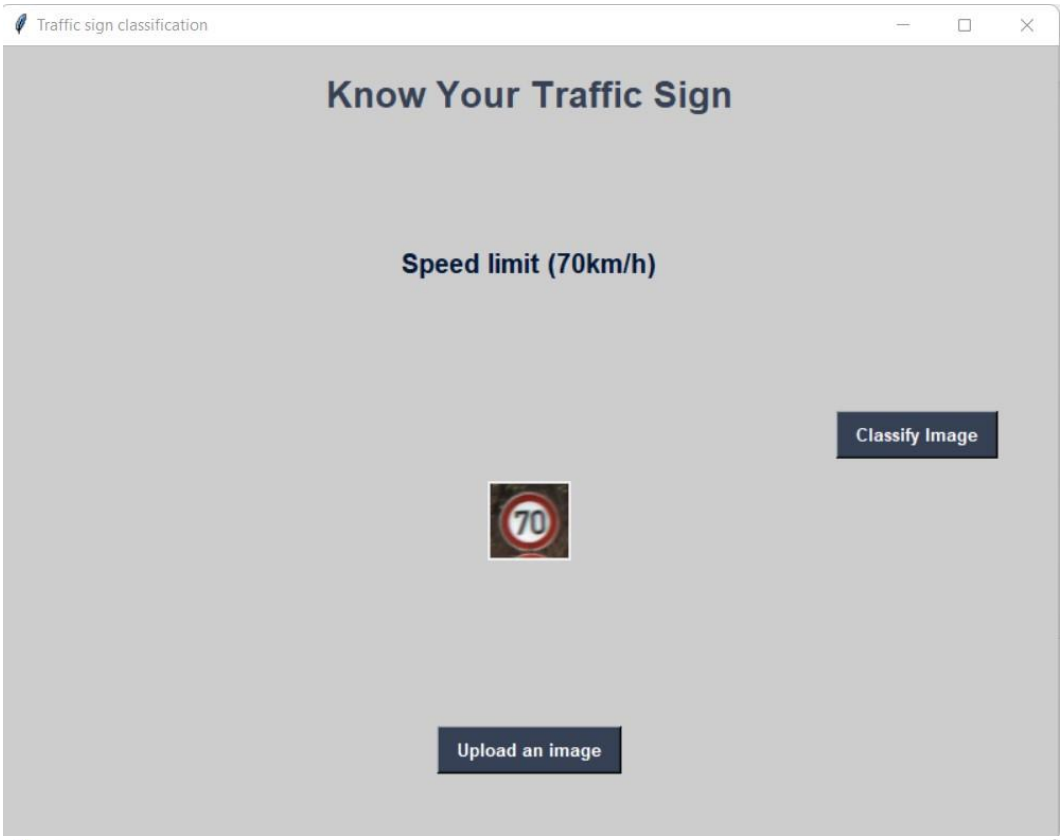
Fig 5.4 Traffic Sign Classification Using GUI

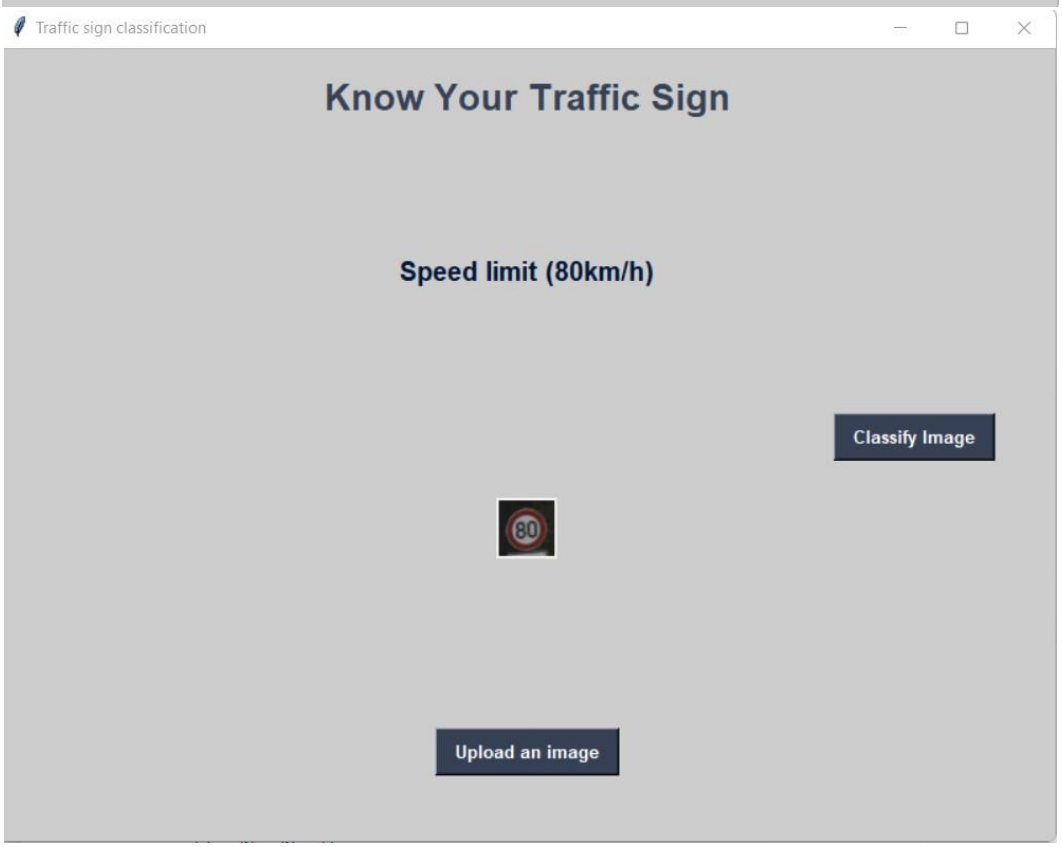
In this case, the user gives the application permission to upload an image and classify image. The image is preprocessed and given as input to the model we have created. The model classifies the traffic sign. Based on the classification, the appropriate result is displayed on the screen. For example: If we upload an image and classify image. It displayed “General Caution” on the screen.

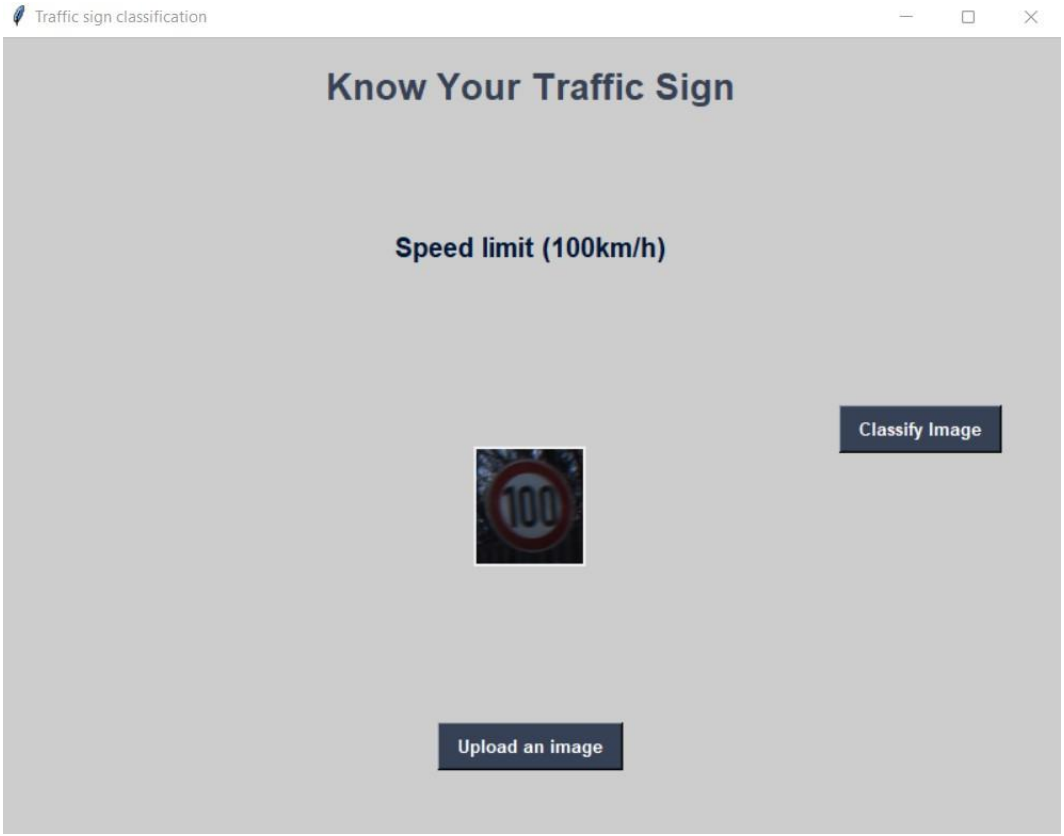
Sample Output Screens

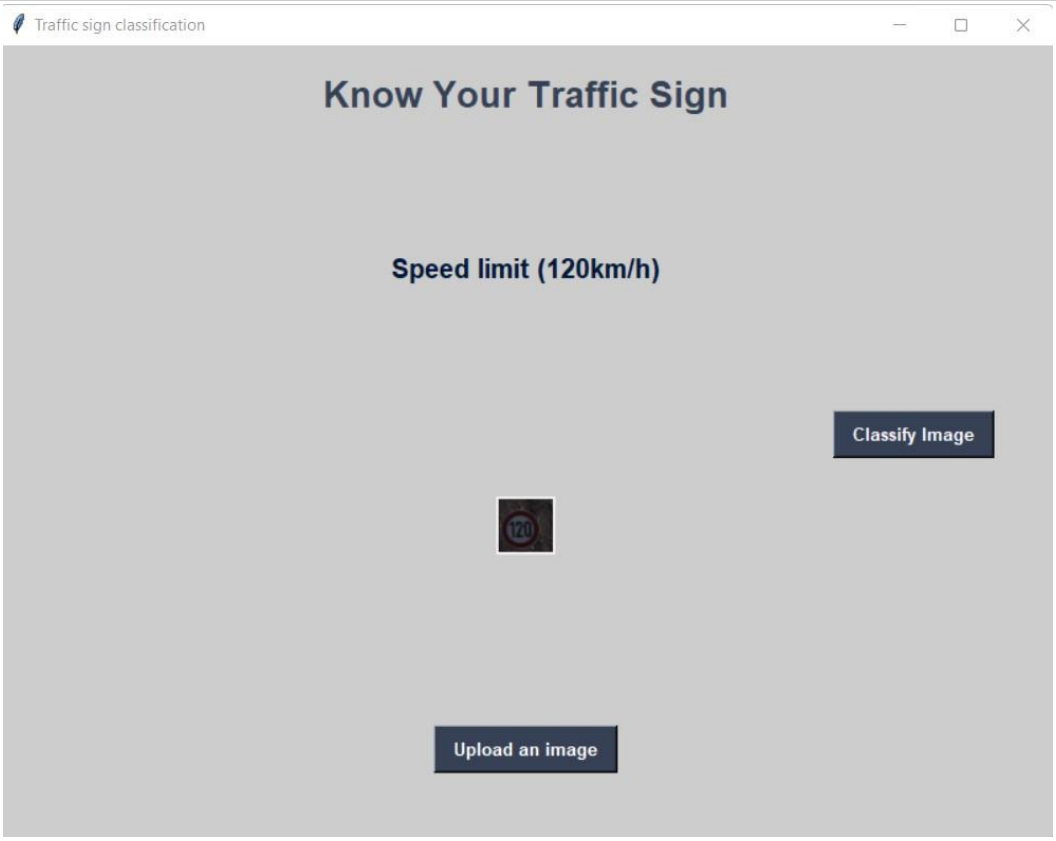

Traffic sign classification
Know Your Traffic Sign
Speed limit (50km/h)
Classify Image
Upload an image

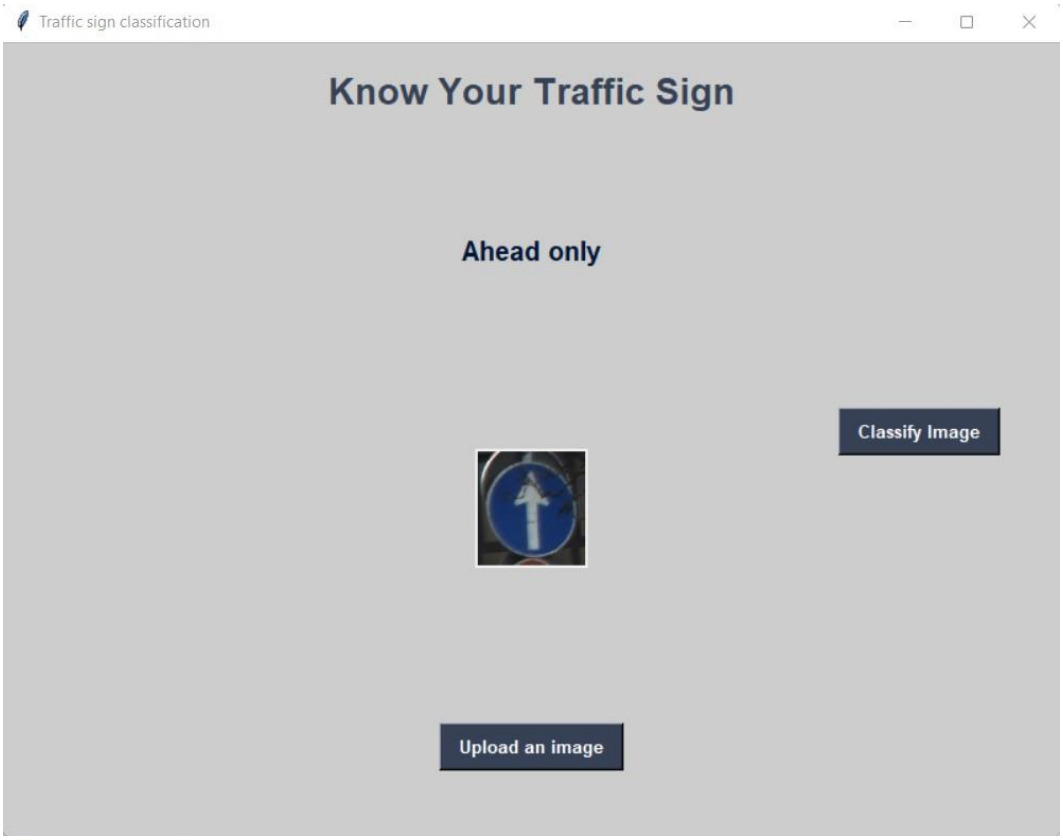

Traffic sign classification
Know Your Traffic Sign
Speed limit (60km/h)
Classify Image
Upload an image


Traffic sign classification
Know Your Traffic Sign
Speed limit (70km/h)
Classify Image
Upload an image

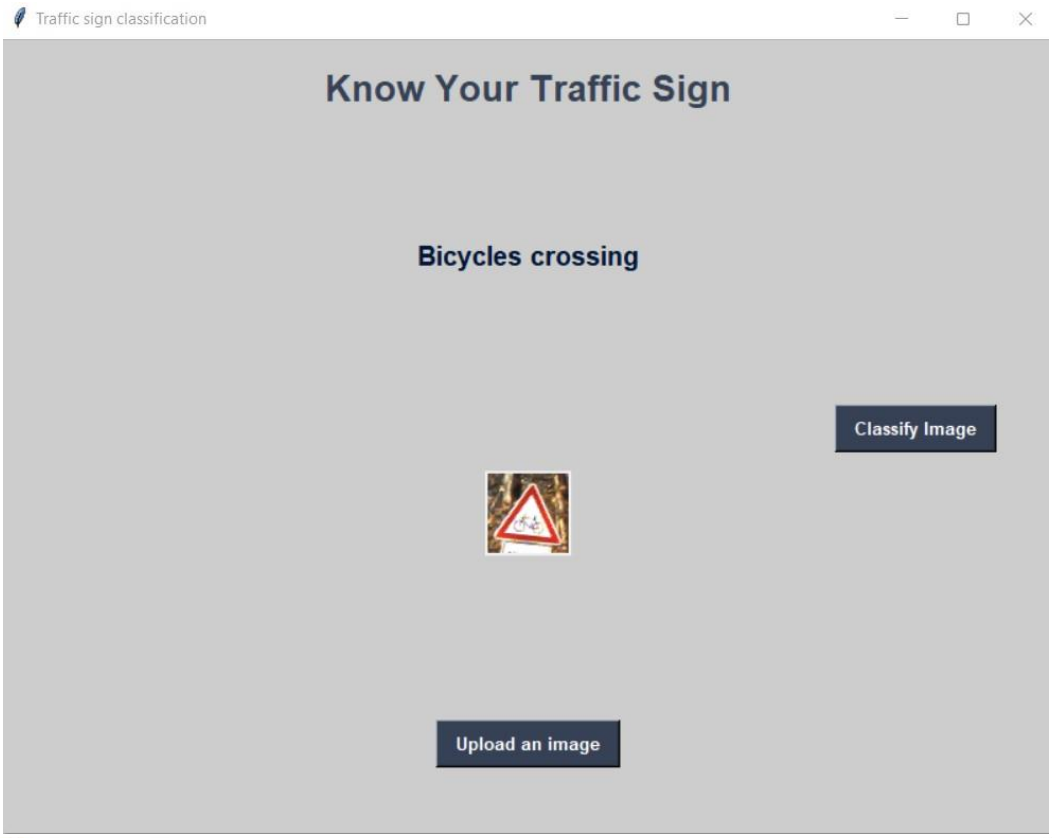

Traffic sign classification
Know Your Traffic Sign
Speed limit (80km/h)
Classify Image
Upload an image

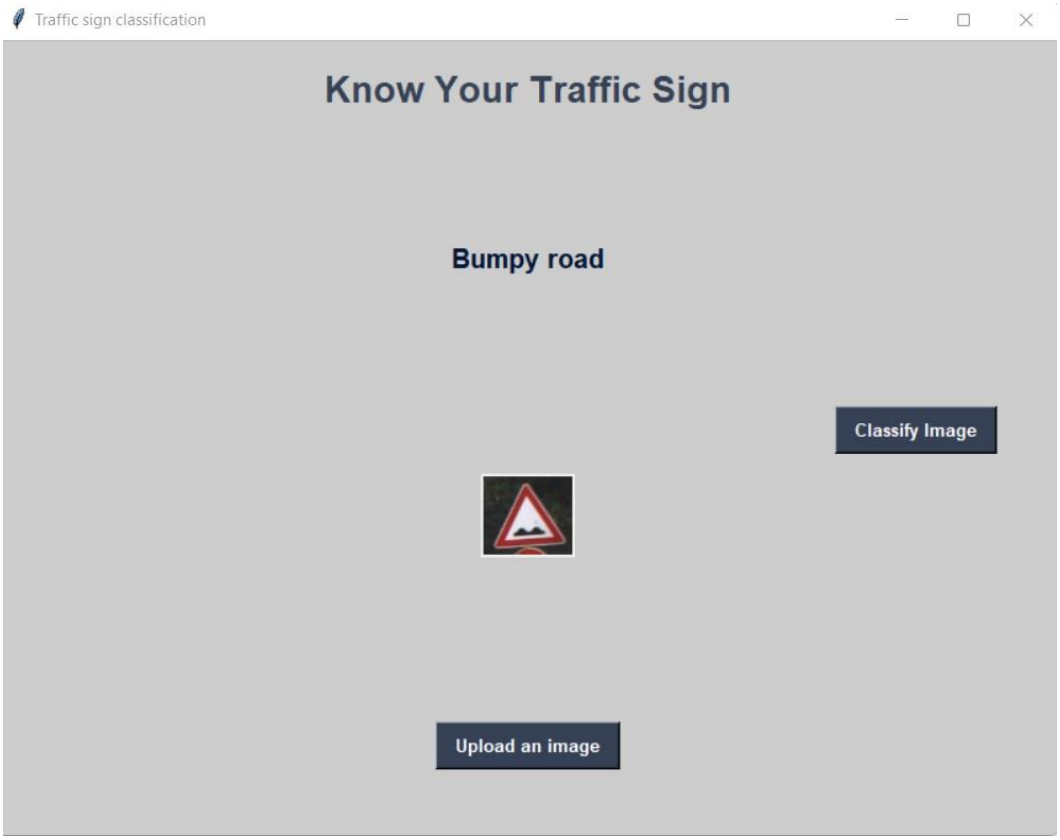

Traffic sign classification
Know Your Traffic Sign
Speed limit (100km/h)
Classify Image
Upload an image

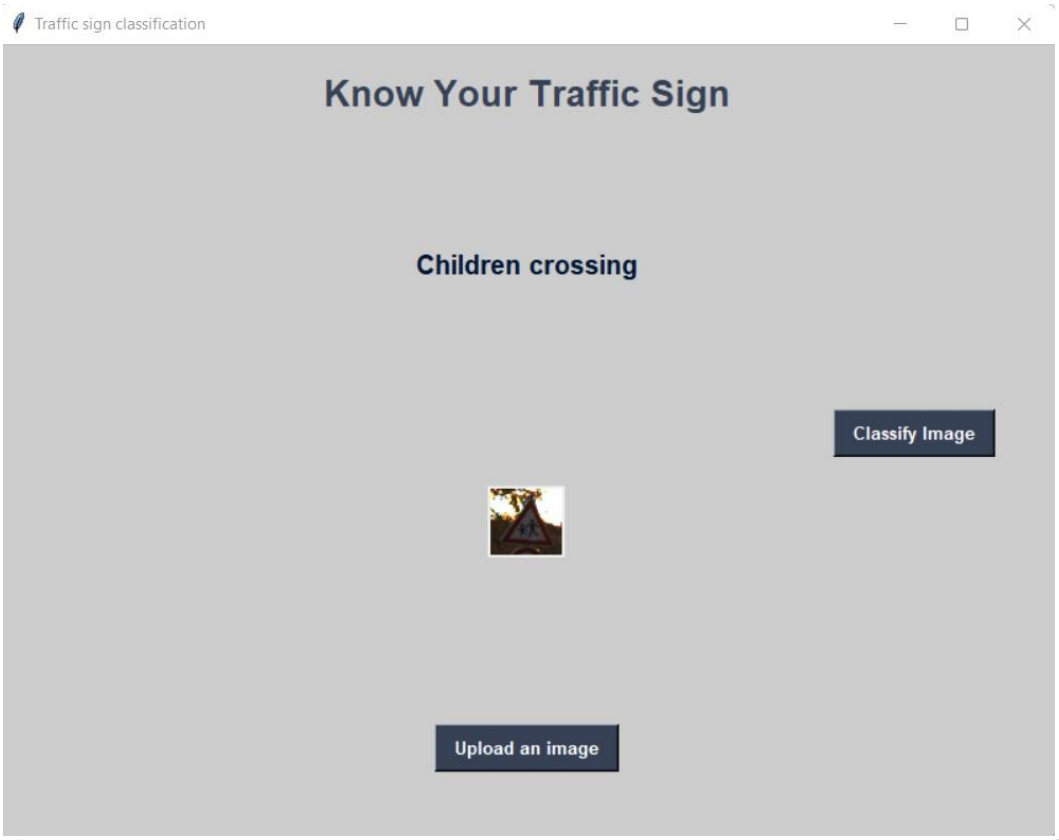

Traffic sign classification
Know Your Traffic Sign
Speed limit (120km/h)
Classify Image
Upload an image

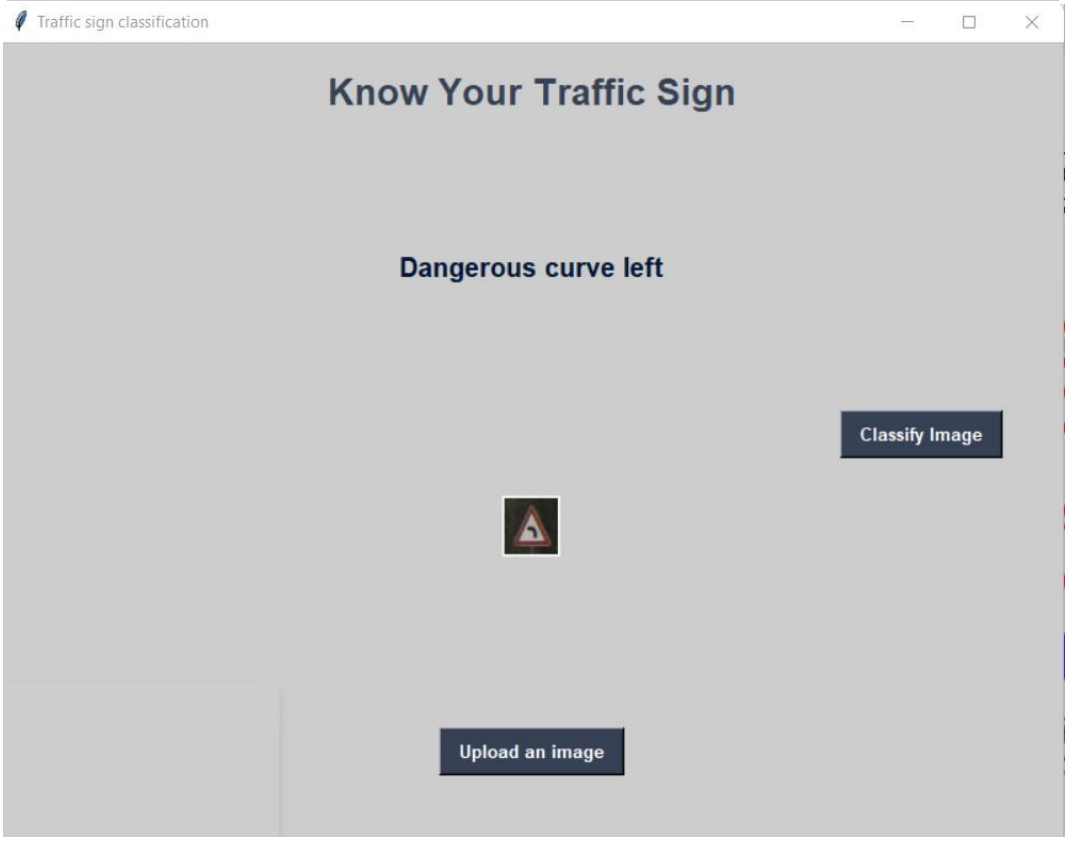

Traffic sign classification
Know Your Traffic Sign
Ahead only
Classify Image
Upload an image

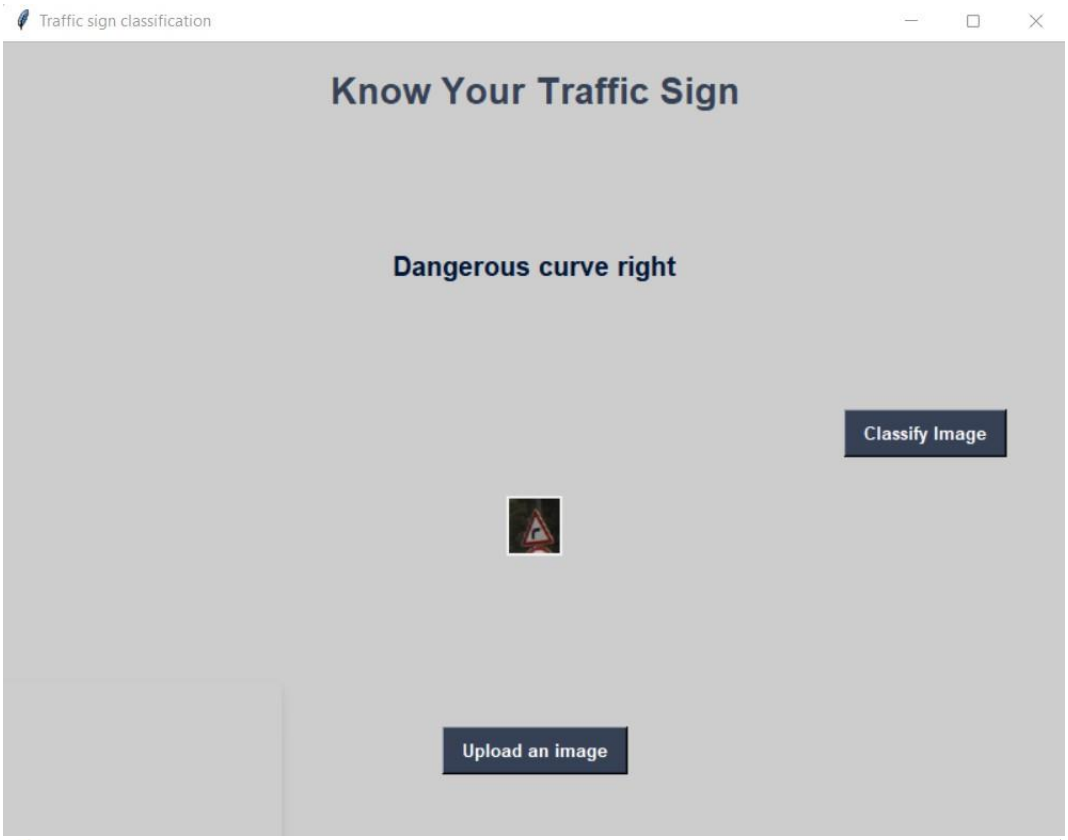

Traffic sign classification
Know Your Traffic Sign
Beware of ice/snow
Classify Image
Upload an image

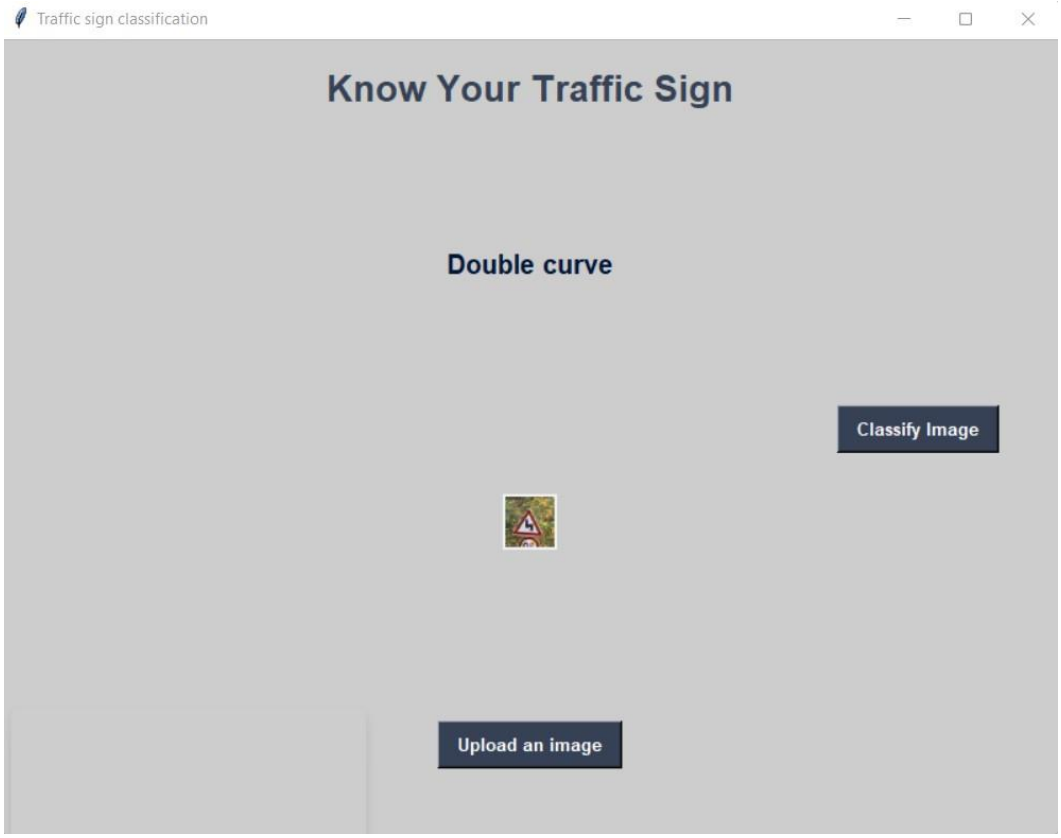

Traffic sign classification
Know Your Traffic Sign
Bicycles crossing
Classify Image
Upload an image

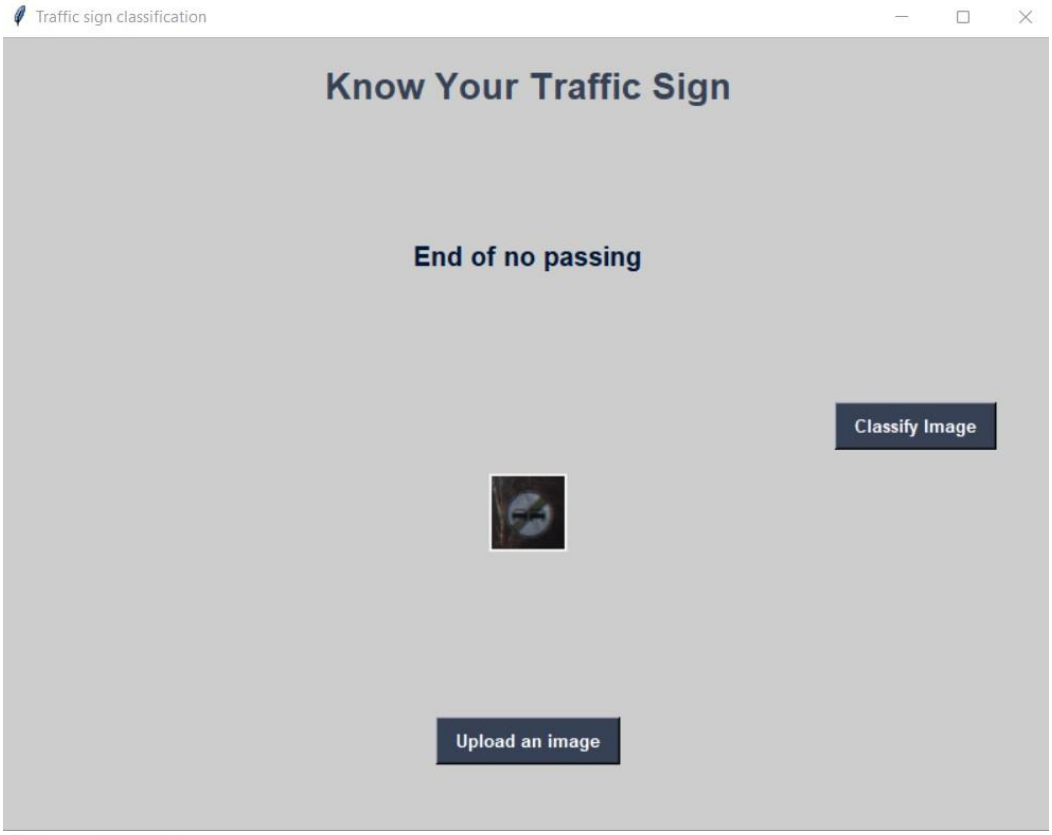

Traffic sign classification
Know Your Traffic Sign
Bumpy road
Classify Image
Upload an image


Traffic sign classification
Know Your Traffic Sign
Children crossing
Classify Image
Upload an image


Traffic sign classification
Know Your Traffic Sign
Dangerous curve left
Classify Image
Upload an image


Traffic sign classification
Know Your Traffic Sign
Dangerous curve right
Classify Image
Upload an image

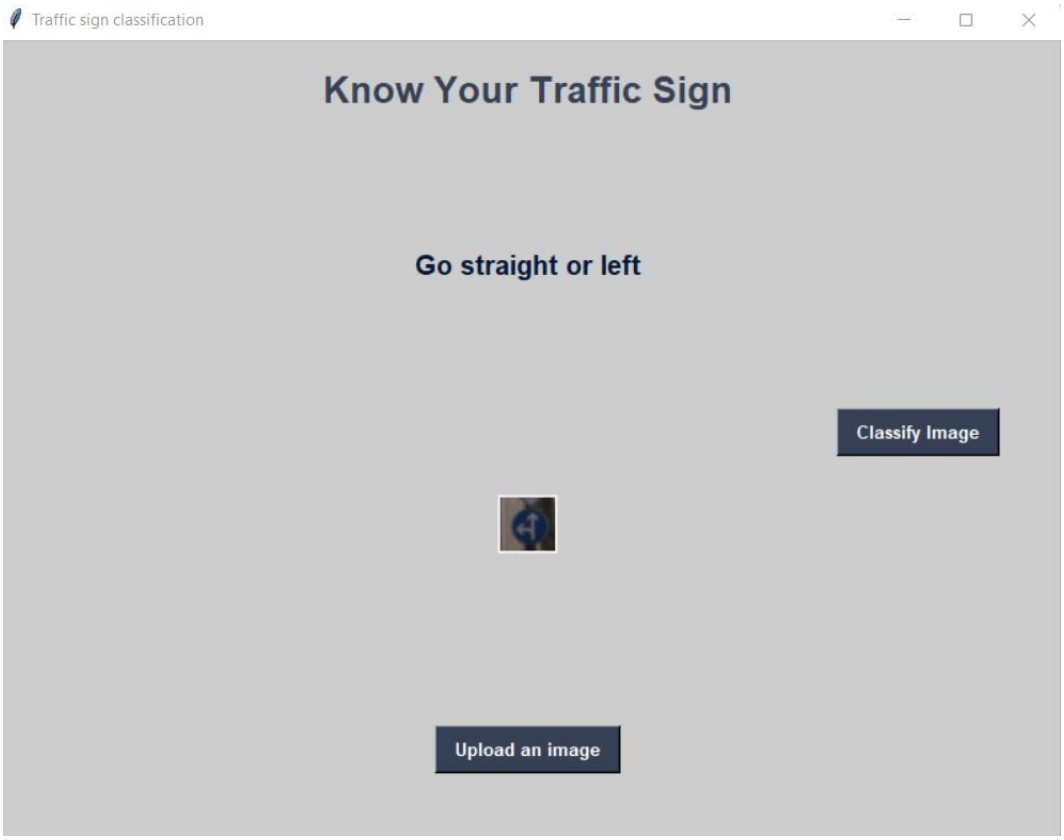

Traffic sign classification
Know Your Traffic Sign
Double curve
Classify Image
Upload an image


Traffic sign classification
Know Your Traffic Sign
End of no passing
Classify Image
Upload an image


Traffic sign classification
Know Your Traffic Sign
End of speed limit (80km/h)
Classify Image
Upload an image

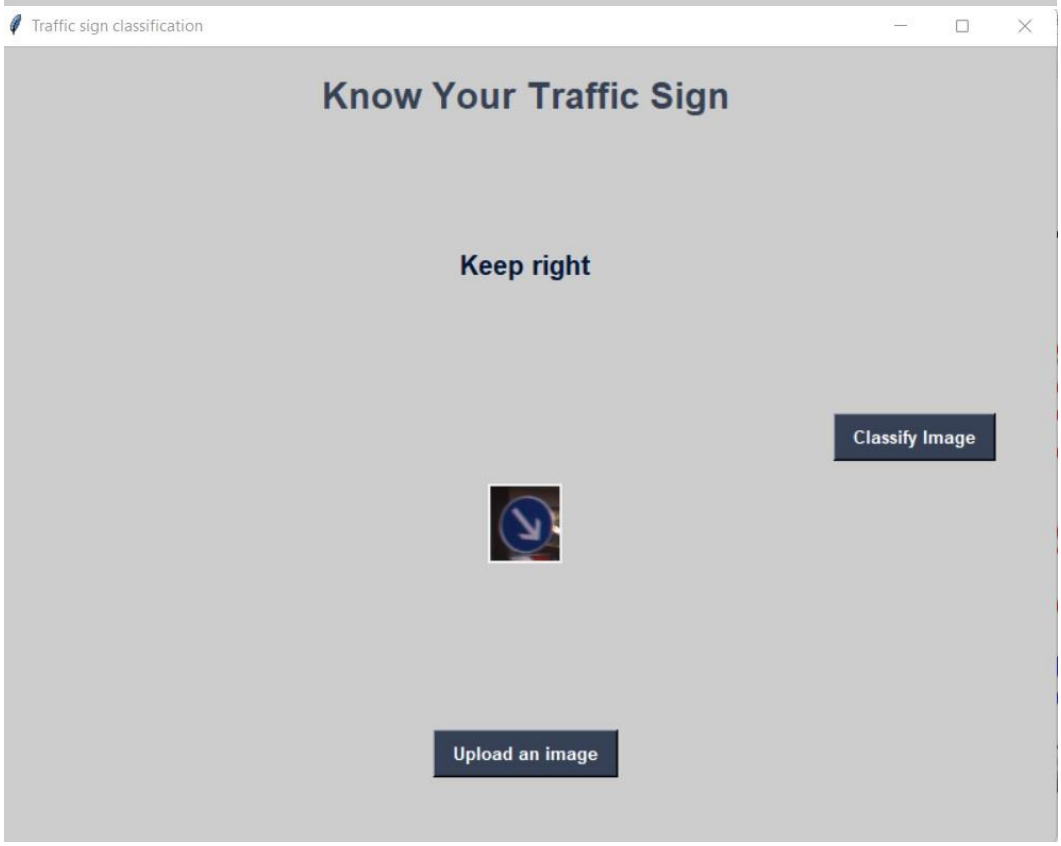

Traffic sign classification
Know Your Traffic Sign
End speed + passing limits
Classify Image
Upload an image

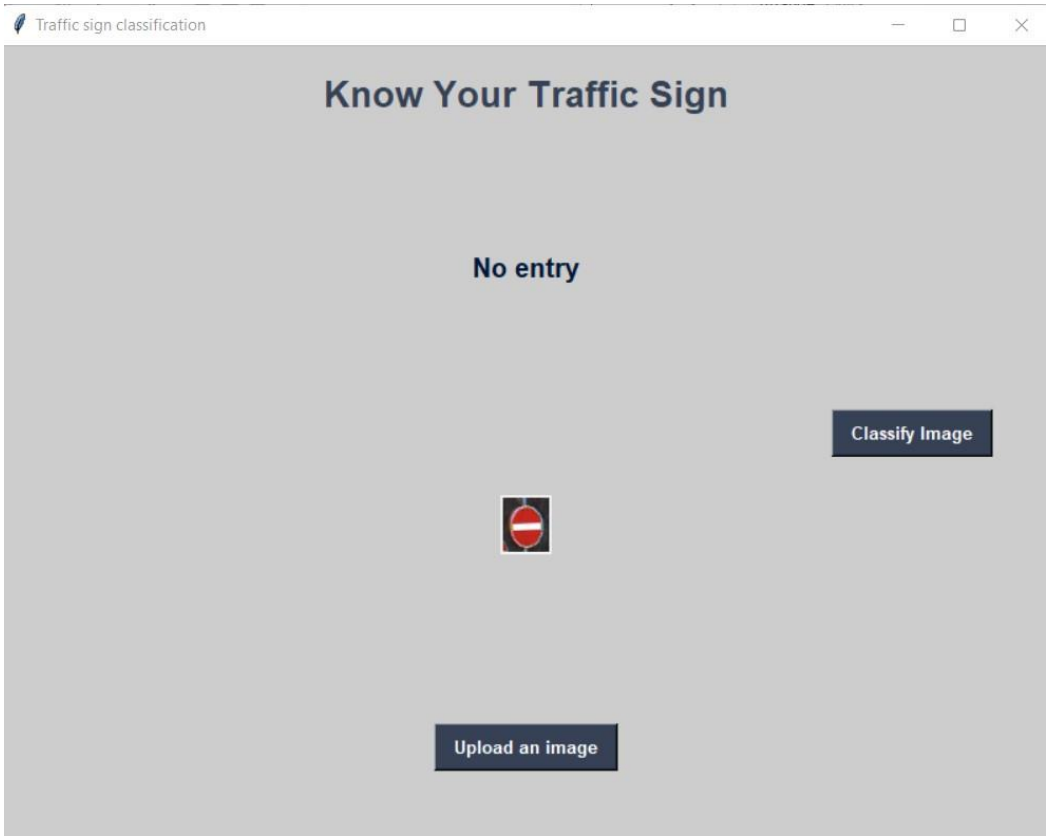

Traffic sign classification
Know Your Traffic Sign
General caution
Classify Image
Upload an image

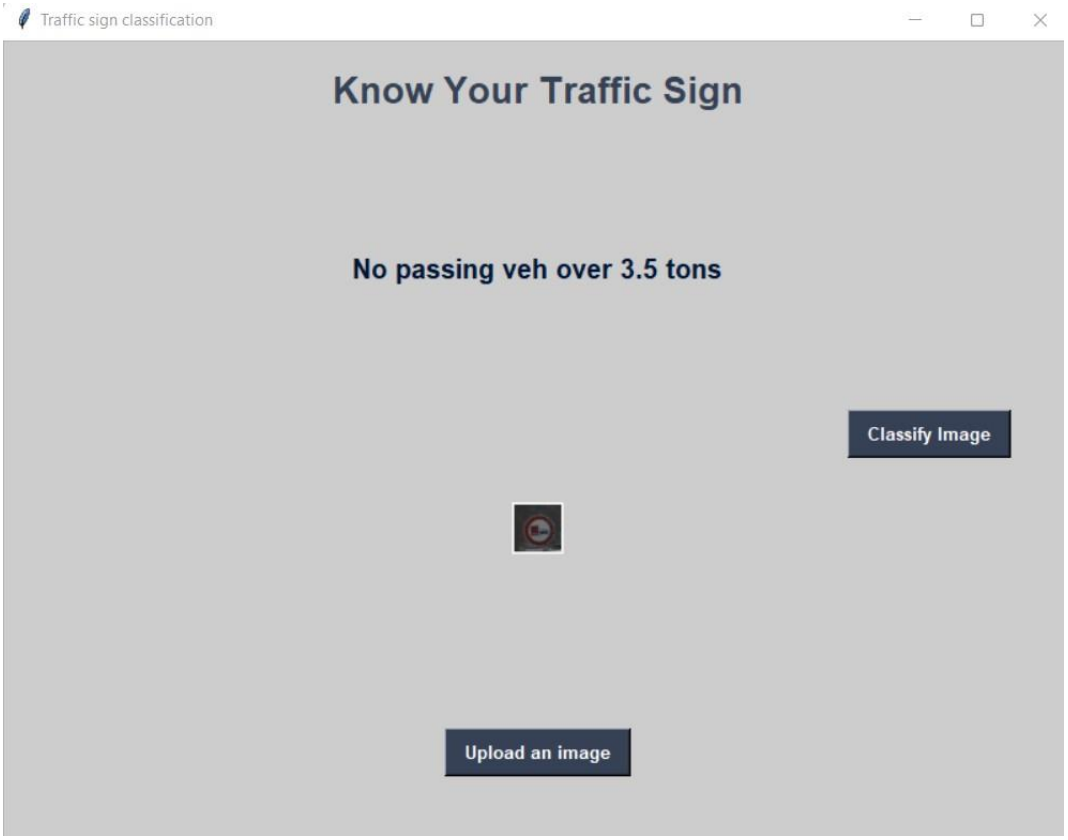

Traffic sign classification
Know Your Traffic Sign
Go straight or left
Classify Image
Upload an image


Traffic sign classification
Know Your Traffic Sign
Go straight or right
Classify Image
Upload an image

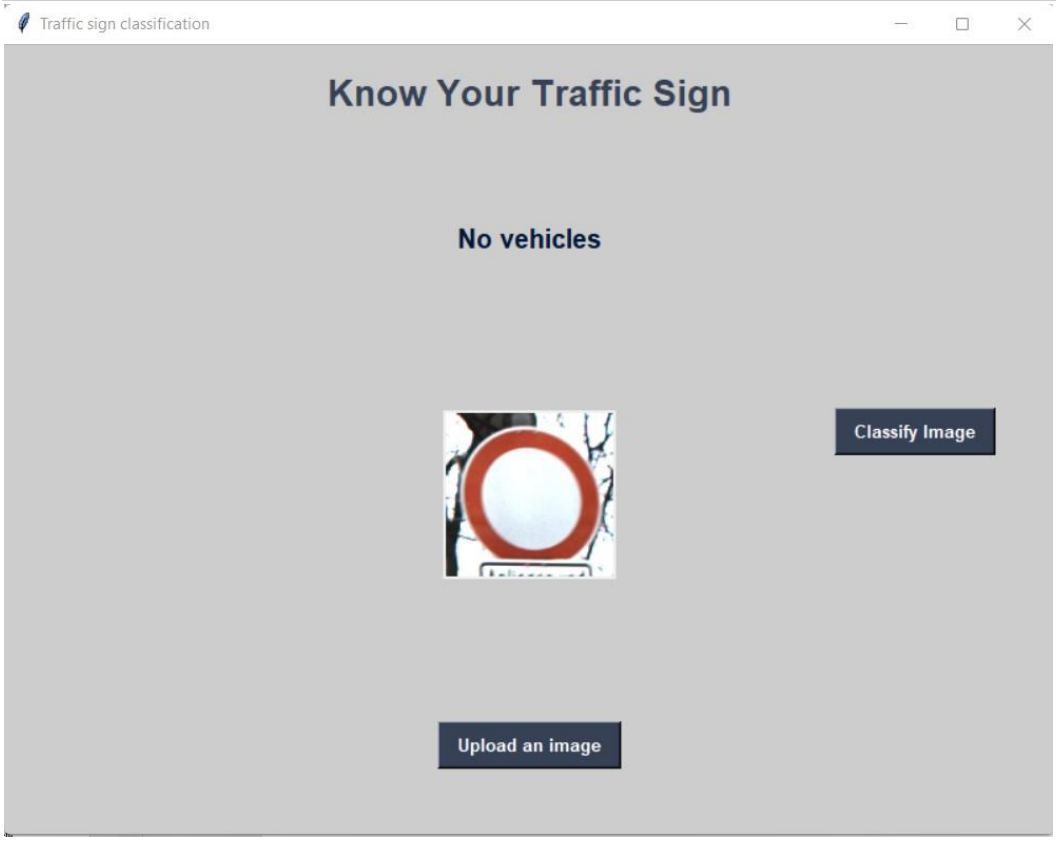

Traffic sign classification
Know Your Traffic Sign
Keep left
Classify Image
Upload an image

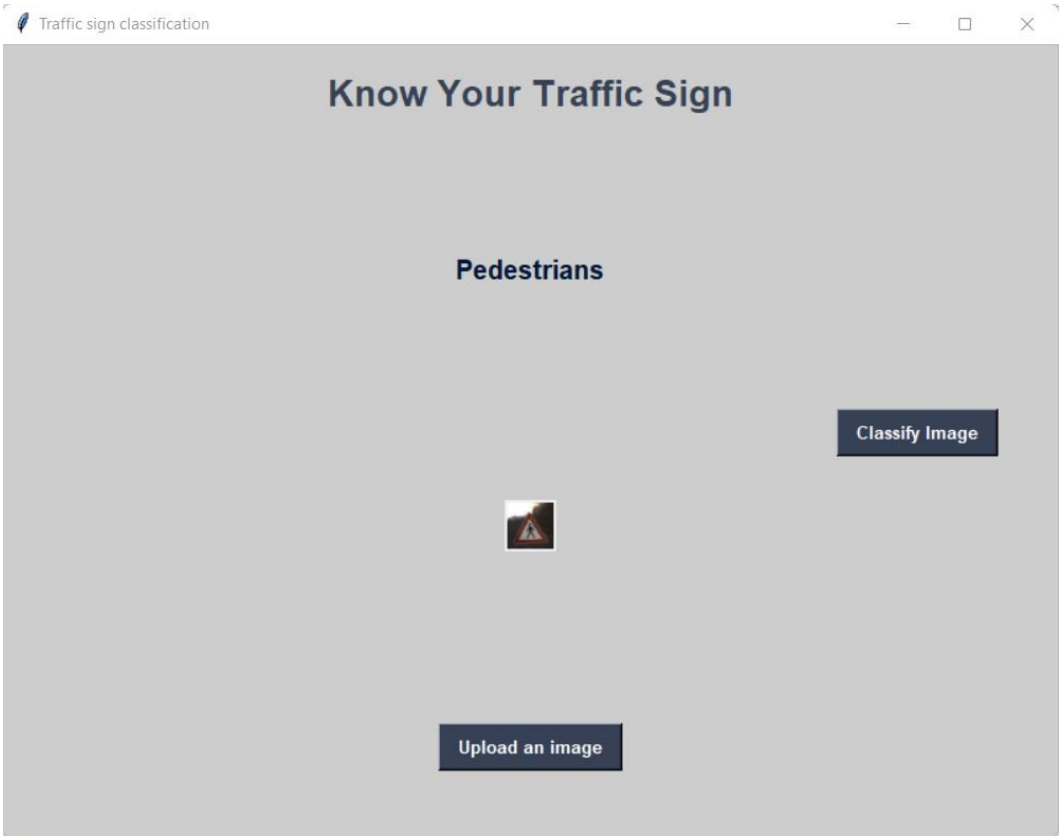

Traffic sign classification
Know Your Traffic Sign
Keep right
Classify Image
Upload an image

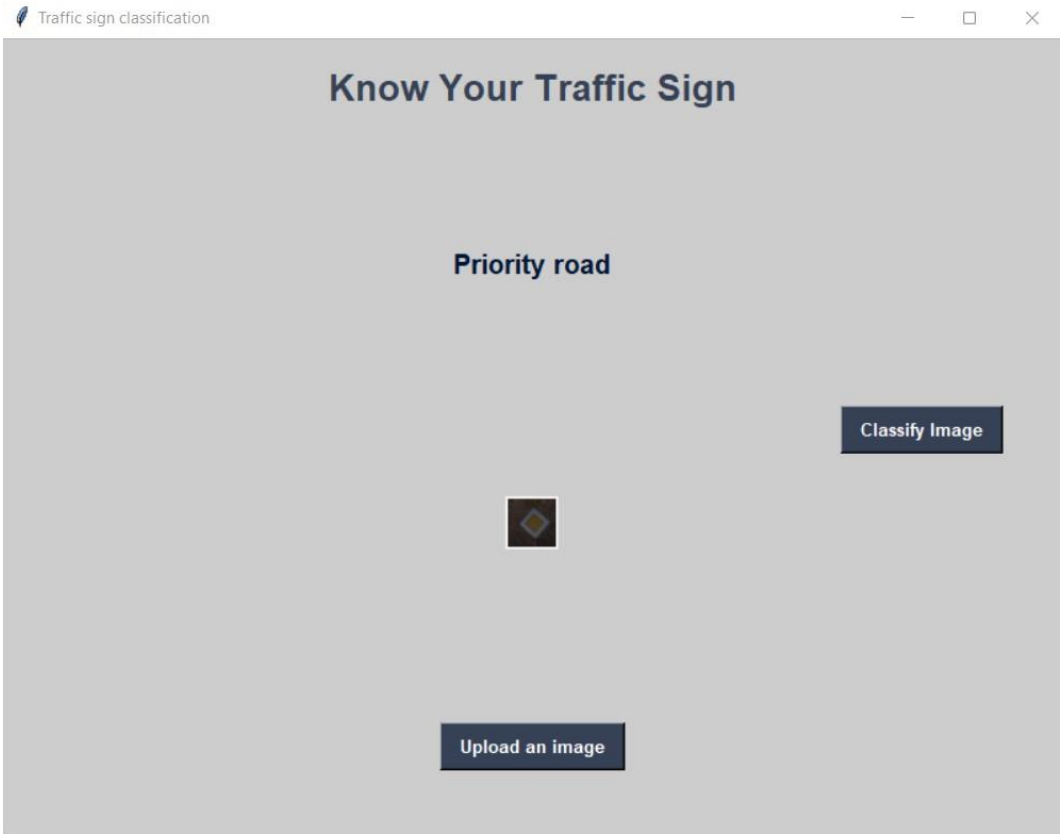

Traffic sign classification
Know Your Traffic Sign
No entry
Classify Image
Upload an image

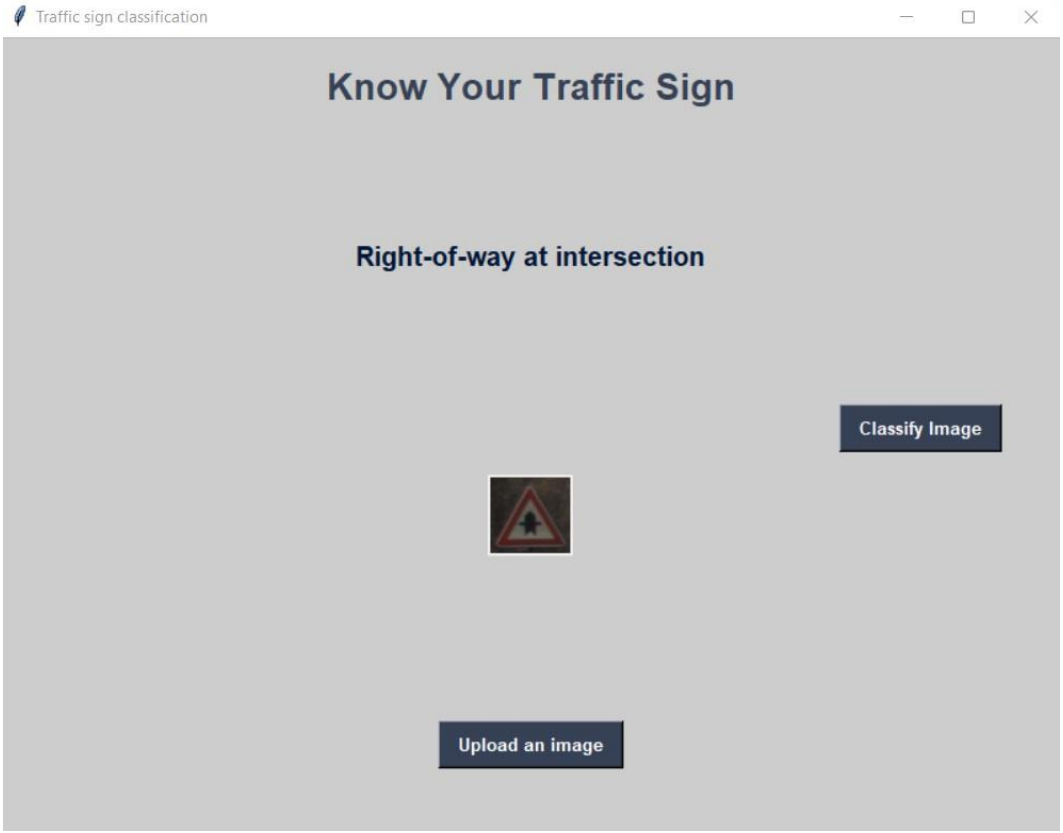

Traffic sign classification
Know Your Traffic Sign
No passing veh over 3.5 tons
Classify Image
Upload an image

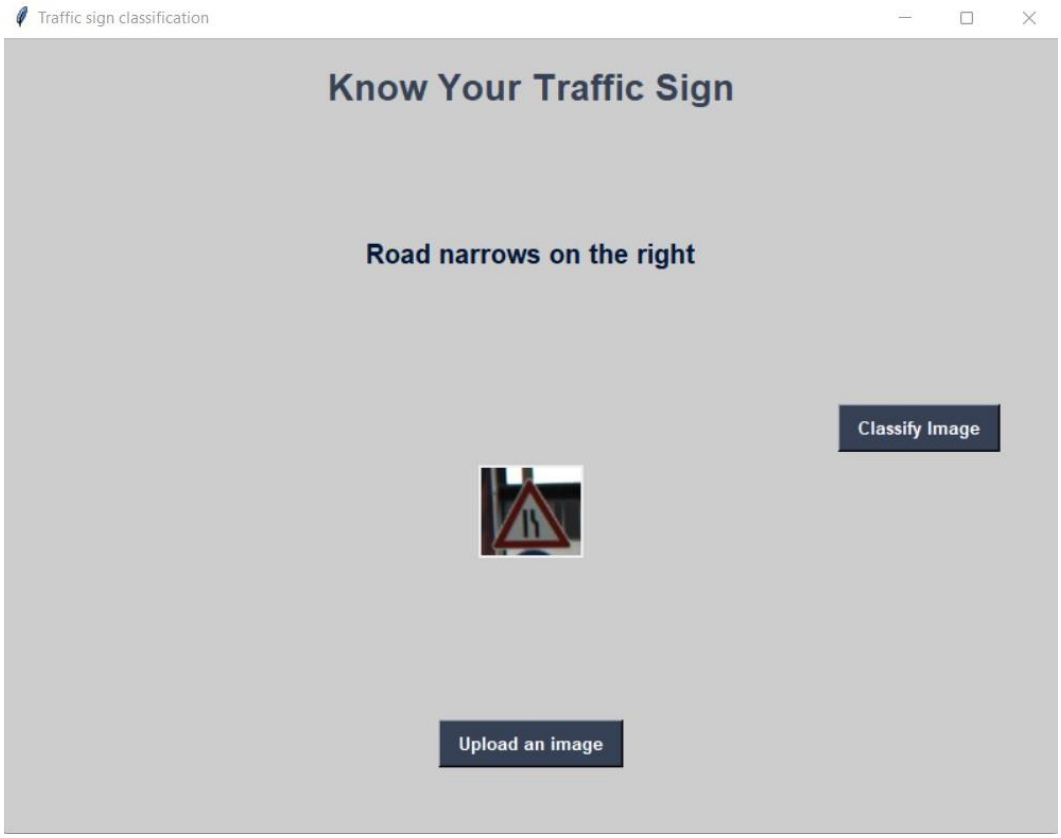

Traffic sign classification
Know Your Traffic Sign
No passing
Classify Image
Upload an image

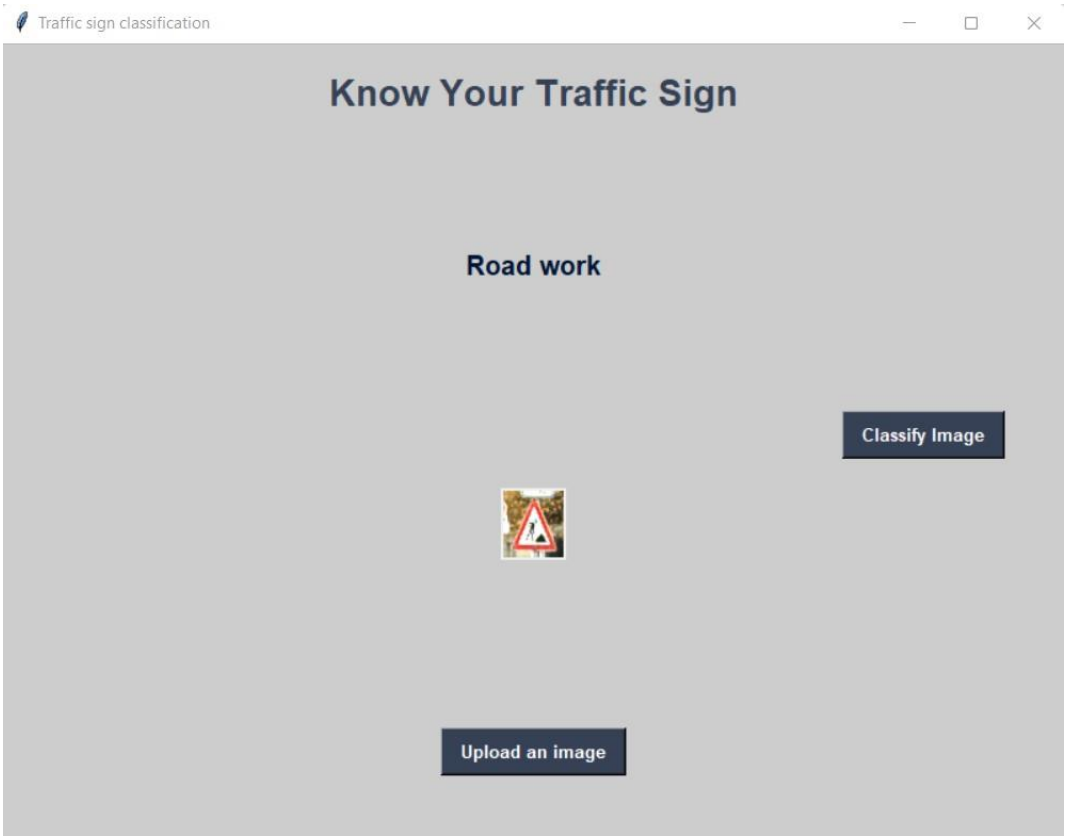

Traffic sign classification
Know Your Traffic Sign
No vehicles
Classify Image
Upload an image

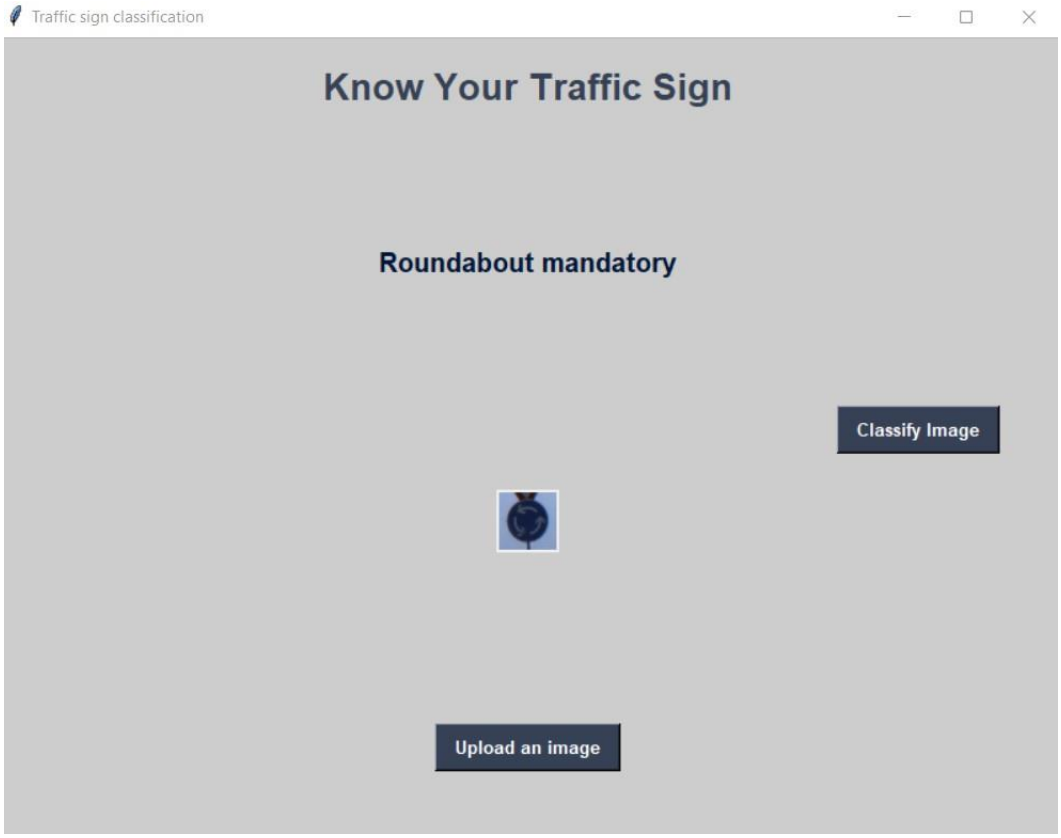

Traffic sign classification
Know Your Traffic Sign
Pedestrians
Classify Image
Upload an image

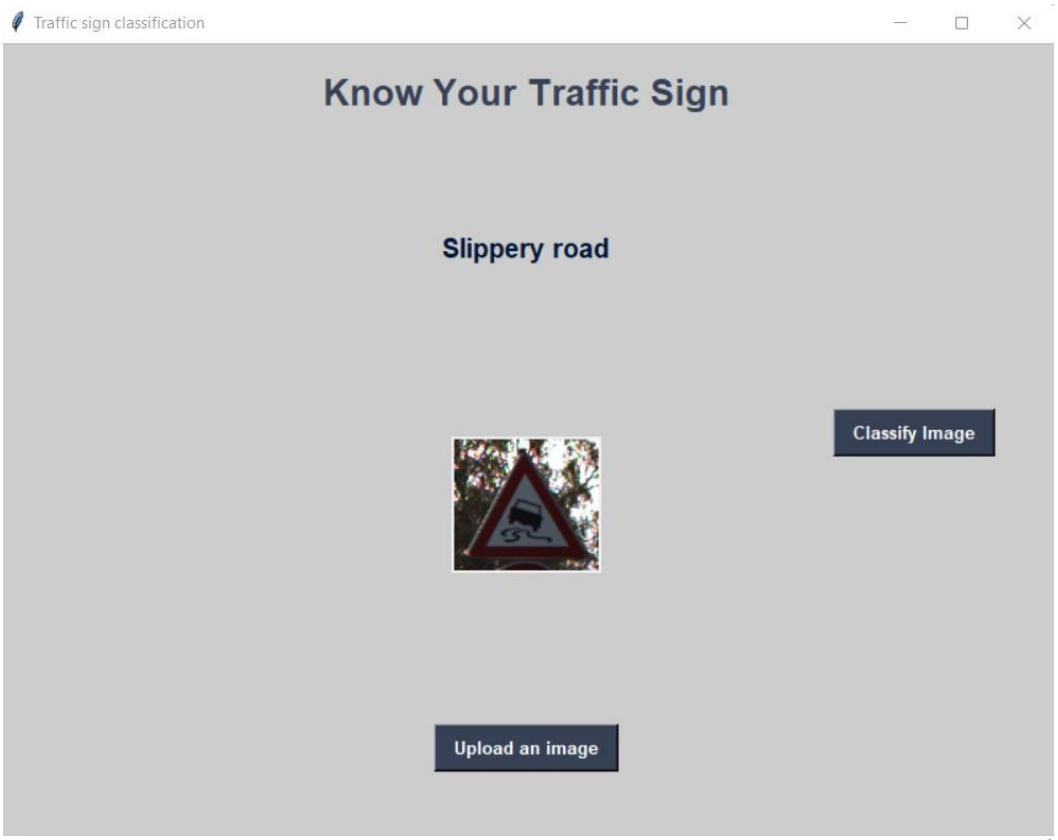

Traffic sign classification
Know Your Traffic Sign
Priority road
Classify Image
Upload an image


Traffic sign classification
Know Your Traffic Sign
Right-of-way at intersection
Classify Image
Upload an image

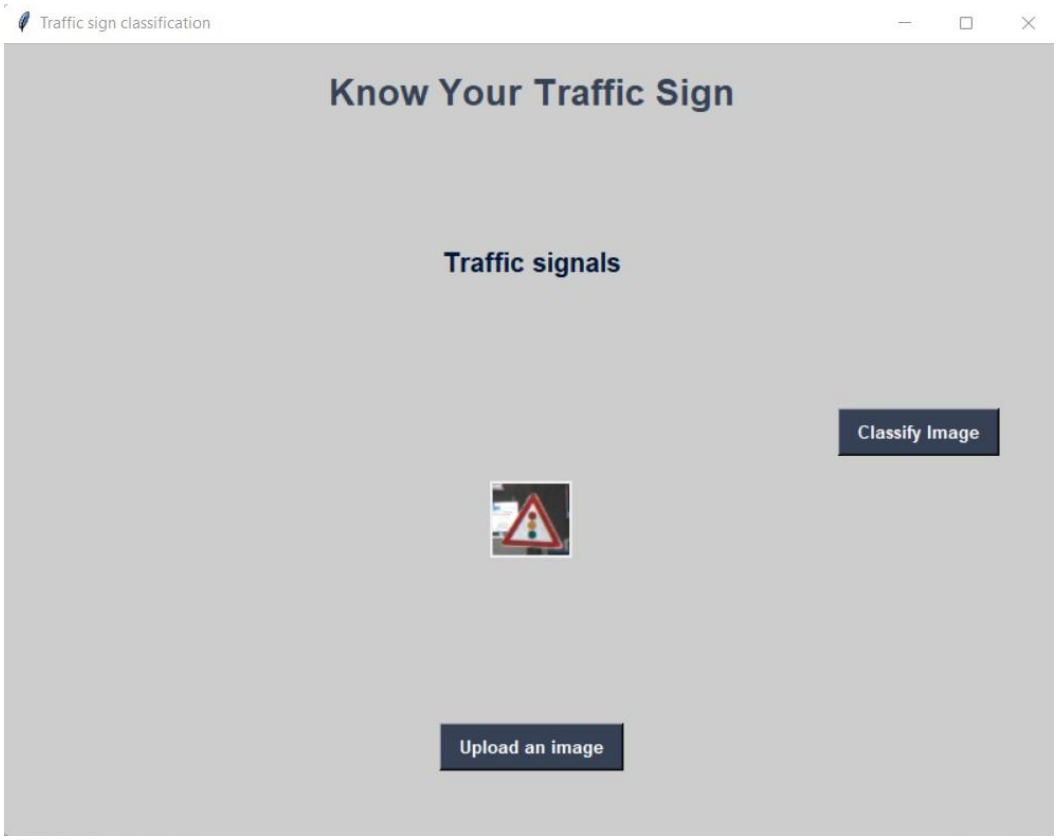

Traffic sign classification
Know Your Traffic Sign
Road narrows on the right
Classify Image
Upload an image

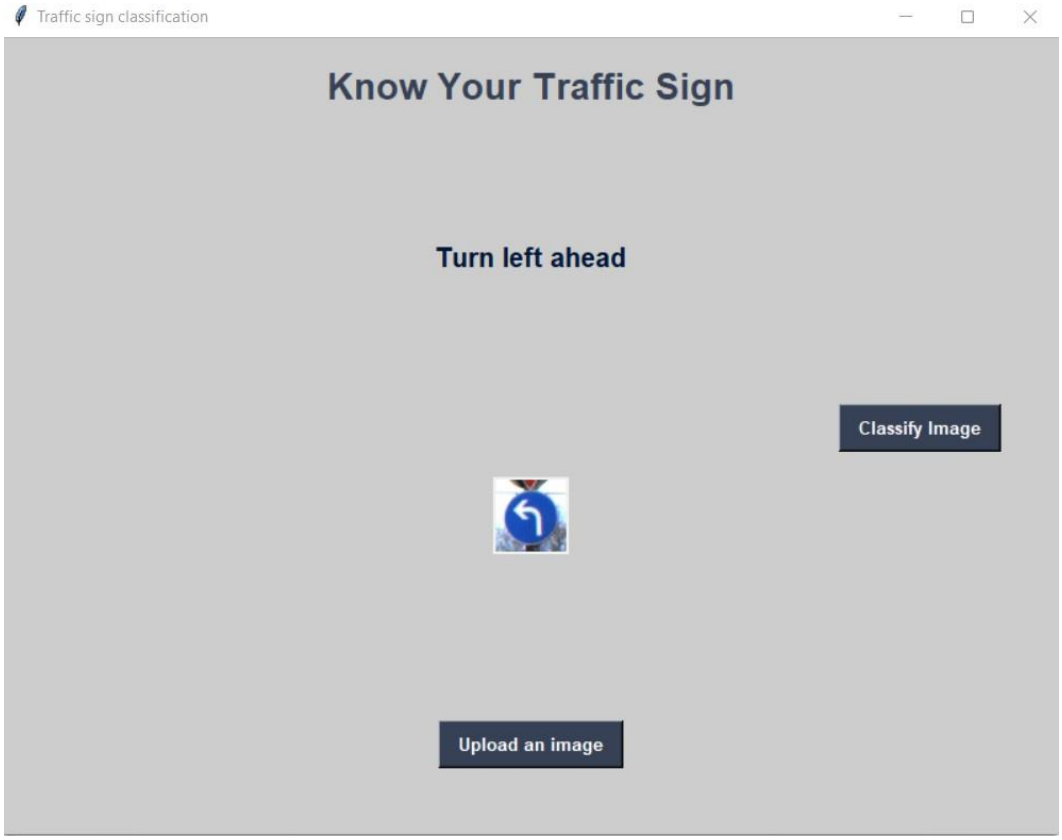

Traffic sign classification
Know Your Traffic Sign
Road work
Classify Image
Upload an image

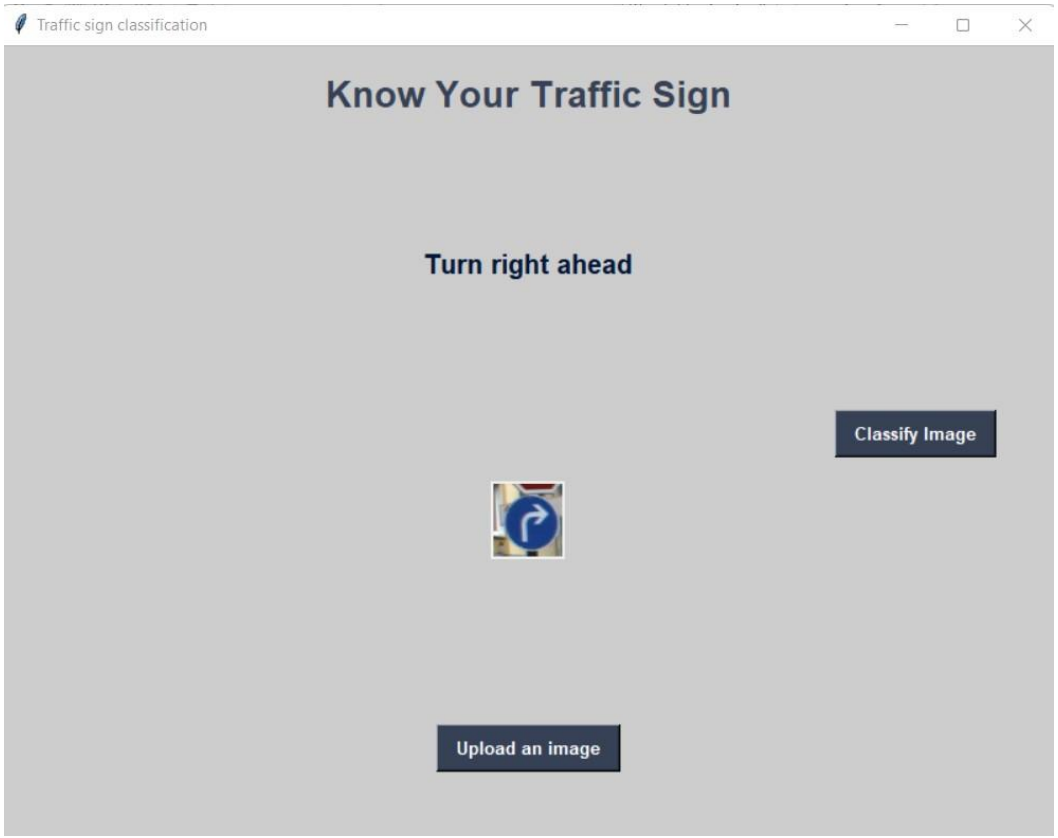

Traffic sign classification
Know Your Traffic Sign
Roundabout mandatory
Classify Image
Upload an image


Traffic sign classification
Know Your Traffic Sign
Slippery road
Classify Image
Upload an image

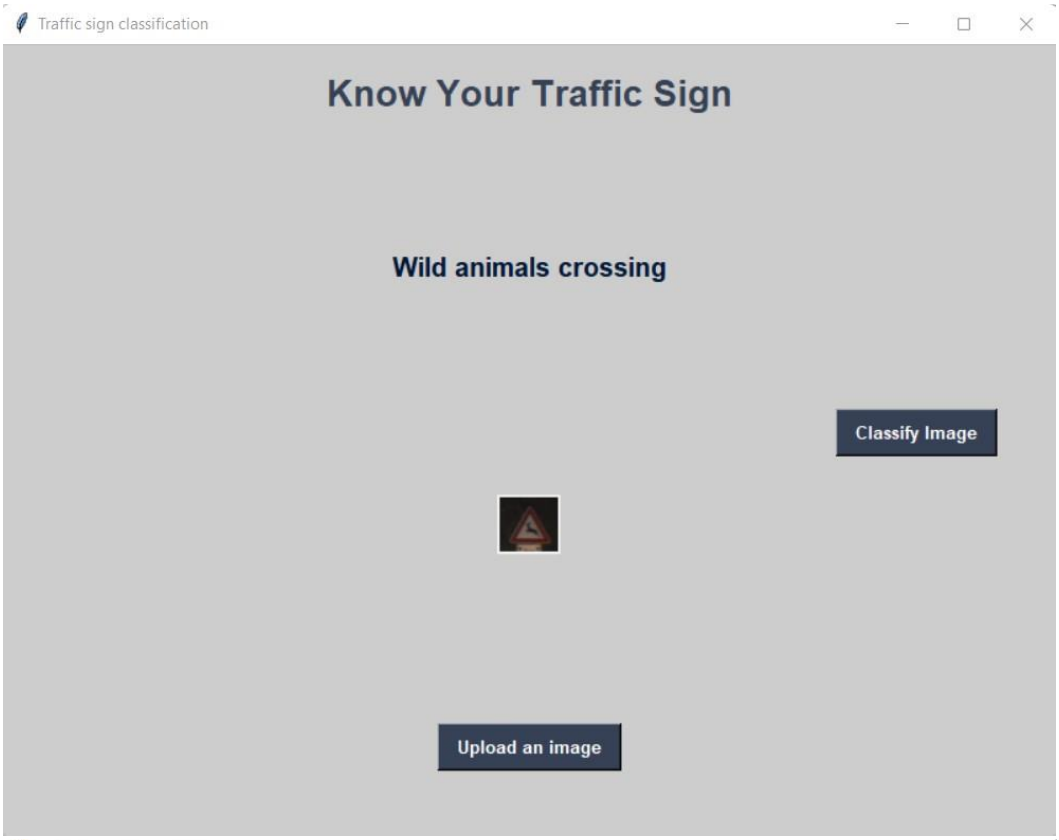

Traffic sign classification
Know Your Traffic Sign
Stop
Classify Image
STOP
Upload an image

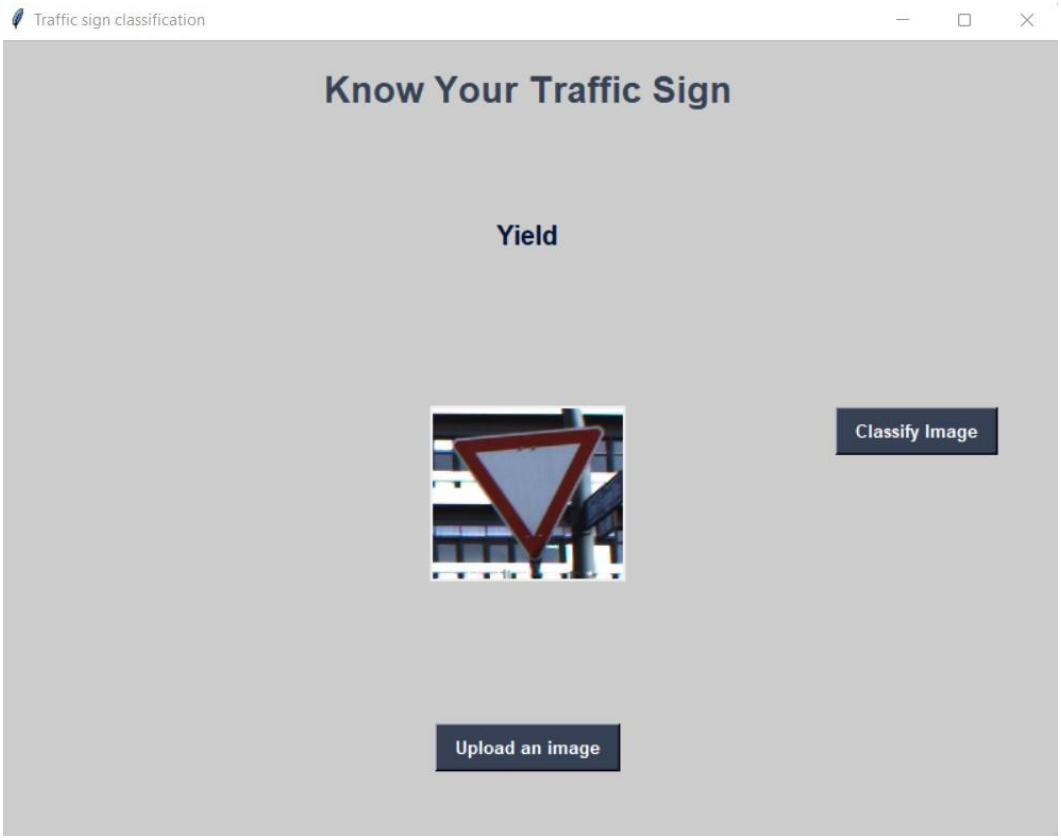

Traffic sign classification
Know Your Traffic Sign
Traffic signals
Classify Image
Upload an image


Traffic sign classification
Know Your Traffic Sign
Turn left ahead
Classify Image
Upload an image


Traffic sign classification
Know Your Traffic Sign
Turn right ahead
Classify Image
Upload an image


Traffic sign classification
Know Your Traffic Sign
Veh > 3.5 tons prohibited
Classify Image
Upload an image


Traffic sign classification
Know Your Traffic Sign
Wild animals crossing
Classify Image
Upload an image


Traffic sign classification
Know Your Traffic Sign
Yield
Classify Image
Upload an image

6. CONCLUSION AND FUTURE SCOPE

Conclusion

A method is described that uses a mix of colour manipulation and Convolutional Neural Networks (CNN). The CNN with fixed and learnable layers got good results while working on a picture preprocessed by colour transformation. The following are the advantages of the CNN we used: For starters, a fixed layer can limit the amount of locations that the classifier must deal with, thereby speeding up the detection process. Second, because the ROIs created by the fixed filter are so near to the boundaries of traffic signs, the problem of alignment is avoided, which would otherwise reduce the performance of the supervised convolution network. Third, it has been demonstrated that a CNN with a proper architecture learnt in a supervised manner is capable of extracting features for traffic sign categorization. The outcomes of our trial backed up our conclusion. The best results achieved for training the dataset with the modified CNN 8-layers model were 97.43% accuracyThe best results achieved for training the dataset with the modified CNN 8-layers model were 97.43% accuracy.

Future scope

- The traffic signs emphasise the use of different travel demand management strategies to reduce traffic loads on the current road network.
- Traffic signs should be used to clear encroachment, reduce congestion, and enhance traffic flow.
- Road signs notify road user of regulation and provide warning and guidance needed for safe, uniform and efficient operation.

7.REFERENCES

[1] Keras Documentation: https://keras.io/guides/

[2] Research Paper: https://ieeexplore.ieee.org/document/7859723

[3] Dataset: https://www.kaggle.com/valentynsichkar/traffic-signs-classification-with-cnn

[4]https://wwwresearchgate.net/publication/352393724_Traffic_Sign_Classification_and_Detection_of_Indian_Traffic_Signs_using_Deep_Learning

[5]https://github.com/qubvel/train_imagenet/blob/master/src/nn_models/se_models2/se_mobilenets .py

[6] https://complex-valued-neural-networks.readthedocs.io/en/latest/

[7]https://complex-valued-neural-networks.readthedocs.io/en/latest/layers/complex_conv.html#complex-conv-2d

[8] https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv2D

[9] https://faroit.com/keras-docs/2.0.7/layers/convolutional/

[10] https://faroit.com/keras-docs/2.1.5/layers/convolutional/

[11] https://github.com/cihanbilge/AstrocyteSegmentation/blob/master/conv2d_LC_layer.py

[12] https://github.com/gao-lab/vConv/blob/main/corecode/vConv_core.py

[13] https://complex-valued-neural-networks.readthedocs.io/en/latest/layers/complex_conv.html

[14] https://complex-valued-neural-networks.readthedocs.io/en/latest/

[15] https://docs.w3cub.com/tensorflow~2.3/keras/layers/conv2d

[16] https://faroit.com/keras-docs/

[17]https://www.ijraset.com/research-paper/traffic-sign-classification-using-cnn#:~:text=Traffic%20Sign%20Classification%20is%20very,its%20high%20accuracy%20and%20precision.

[18] https://huggingface.co/