

**AIM :**To identify traffic sign and measure the accuracy of the identification of traffic sign.

**SOFTWARE USED:** VSCODE, ANACONDA.

### **THEORY:**

Traffic sign classification is an important application of machine learning in the field of transportation. Machine learning has been applied in transportation to improve road safety and traffic management, with traffic sign classification being an important application. This process involves several steps, including image acquisition, pre-processing, feature extraction, classification, and post-processing. However, traffic sign classification faces various challenges, such as changes in sign appearance, lighting, weather conditions, and occlusion.

Traditional computer vision techniques, such as template matching and feature-based methods, are not effective in addressing these challenges. With the advent of deep learning, researchers started exploring the use of machine learning techniques for traffic sign classification. The release of the GTSRB dataset in 2011 provided a large-scale dataset of annotated traffic signs for researchers to train and test their algorithms, which helped in advancing the field.

Deep learning-based approaches, particularly Convolutional Neural Networks (CNNs), have achieved remarkable results on traffic sign classification tasks. Techniques such as data augmentation, transfer learning, and ensemble learning have been utilized to address the challenges posed by traffic sign classification. In 2019, a collaboration between NVIDIA and Toyota utilized these techniques and achieved near-human level accuracy on the GTSRB dataset.

In conclusion, traffic sign classification using machine learning has the potential to make driving safer and more efficient by improving road safety and traffic management.

## CODE:

```
import tkinter as tk
from tkinter import filedialog
from tkinter import *
from PIL import ImageTk, Image
import keras.utils

import numpy
import numpy as np
#load the trained model to classify sign
from keras.models import load_model
model = load_model('traffic_classifier.h5')

classes = ['Speed limit (20km/h)', 'Speed limit (30km/h)',
'Speed limit (50km/h)', 'Speed limit (60km/h)',
'Speed limit (70km/h)', 'Speed limit (80km/h)',
'End of speed limit (80km/h)', 'Speed limit (100km/h)',
'Speed limit (120km/h)', 'No passing', 'No passing for vehicles over 3.5 metric tons',
'Right-of-way at the next intersection', 'Priority road', 'Yield', 'Stop', 'No vehicles',
'Vehicles over 3.5 metric tons prohibited', 'No entry', 'General caution',
'Dangerous curve to the left',
'Dangerous curve to the right', 'Double curve',
'Bumpy road', 'Slippery road', 'Road narrows on the right',
'Road work', 'Traffic signals', 'Pedestrians',
'Children crossing', 'Bicycles crossing',
'Beware of ice/snow', 'Wild animals crossing', 'End of all speed and passing limits', 'Turn right ahead',
'Turn left ahead', 'Ahead only', 'Go straight or right', 'Go straight or left',
'Keep right', 'Keep left',
'Roundabout mandatory', 'End of no passing', 'End of no passing by vehicles over 3.5 metric tons']
#dictionary to label all traffic signs class.
# classes = { 1:'Speed limit (20km/h)',
#             2:'Speed limit (30km/h)',
#             3:'Speed limit (50km/h)',
#             4:'Speed limit (60km/h)',
#             5:'Speed limit (70km/h)',
#             6:'Speed limit (80km/h)',
#             7:'End of speed limit (80km/h)',
#             8:'Speed limit (100km/h)',
#             9:'Speed limit (120km/h)',
```

```
#         10:'No passing',
#         11:'No passing veh over 3.5 tons',
#         12:'Right-of-way at intersection',
#         13:'Priority road',
#         14:'Yield',
#         15:'Stop',
#         16:'No vehicles',
#         17:'Veh > 3.5 tons prohibited',
#         18:'No entry',
#         19:'General caution',
#         20:'Dangerous curve left',
#         21:'Dangerous curve right',
#         22:'Double curve',
#         23:'Bumpy road',
#         24:'Slippery road',
#         25:'Road narrows on the right',
#         26:'Road work',
#         27:'Traffic signals',
#         28:'Pedestrians',
#         29:'Children crossing',
#         30:'Bicycles crossing',
#         31:'Beware of ice/snow',
#         32:'Wild animals crossing',
#         33:'End speed + passing limits',
#         34:'Turn right ahead',
#         35:'Turn left ahead',
#         36:'Ahead only',
#         37:'Go straight or right',
#         38:'Go straight or left',
#         39:'Keep right',
#         40:'Keep left',
#         41:'Roundabout mandatory',
#         42:'End of no passing',
#         43:'End no passing veh > 3.5 tons' }

#initialise GUI
top=tk.Tk()
top.geometry('800x600')
top.title('Traffic sign classification')
top.configure(background='#CDCDCD')

label=Label(top,background='#CDCDCD', font=('arial',15,'bold'))
sign_image = Label(top)

def classify(file_path):
```

```

global label_packed
image = Image.open(file_path)
image = image.resize((30,30))
image = numpy.expand_dims(image, axis=0)
image = numpy.array(image)

# pred=np.argmax(model.predict([image][0]), axis=-1)
# pred = keras.utils.np_utils.probas(model.predict([image])[0])4
pred_prob=model.predict(image)
pred=np.argmax(pred_prob,axis=1)
sign = classes[pred[0]]
print(sign)
label.configure(foreground='#011638', text=sign)

def show_classify_button(file_path):
    classify_b=Button(top,text="Classify Image",command=lambda:
classify(file_path),padx=10,pady=5)
    classify_b.configure(background='#364156',
foreground='white',font=('arial',10,'bold'))
    classify_b.place(relx=0.79,rely=0.46)

def upload_image():
    try:
        file_path=filedialog.askopenfilename()
        uploaded=Image.open(file_path)
        uploaded.thumbnail(((top.wininfo_width()/4.25),(top.wininfo_height()/4.25)))
        im=ImageTk.PhotoImage(uploaded)

        sign_image.configure(image=im)
        sign_image.image=im
        label.configure(text='')
        show_classify_button(file_path)
    except:
        pass

upload=Button(top,text="Upload an image",command=upload_image,padx=30,pady=10)
upload.configure(background='#364156',
foreground='white',font=('arial',10,'bold'))

upload.pack(side=BOTTOM,pady=50)
sign_image.pack(side=BOTTOM,expand=True)
label.pack(side=BOTTOM,expand=True)
heading = Label(top, text="Know Your Traffic Sign",pady=20,
font=('arial',20,'bold'))
heading.configure(background='#CDCDCD',foreground='#364156')

```

```
heading.pack()  
top.mainloop()
```

## Libraries for Traffic Sign Classification

The following libraries are commonly used in traffic sign classification:

**OpenCV:** a popular computer vision library used for image and video processing. In traffic sign classification, OpenCV can be used for preprocessing the images, such as resizing the images to a standard size or converting them to grayscale.

**NumPy:** a library for numerical computing in Python. In traffic sign classification, NumPy can be used for handling the image data as arrays and performing numerical operations on them, such as scaling the pixel values to a standard range.

**TensorFlow:** an open-source machine learning library developed by Google. In traffic sign classification, TensorFlow can be used for building and training deep learning models, including Convolutional Neural Networks (CNNs), which are commonly used for traffic sign classification. TensorFlow also provides utilities for data preprocessing, visualization, and evaluation.

**Keras Library:** a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano. In the context of traffic sign classification, Keras is used for building, training, and evaluating the deep learning model. The Keras library provides a range of pre-defined neural network layers, such as convolutional layers, pooling layers, and fully connected layers, which can be combined to form complex models for image classification tasks.

**Scikit-Learn Library:** a popular machine learning library in Python that provides tools for data preprocessing, model selection, and evaluation, as well as many classification, regression, and clustering algorithms. In traffic sign classification, scikit-learn can be used for data preprocessing and model evaluation.

Our approach to building this traffic sign classification model is discussed in four steps:

- Explore the dataset
- Build a CNN Model
- Train and validate the model
- Test the model with test dataset

```
!pip install tensorflow keras sklearn matplotlib pandas pil
```

```
... Requirement already satisfied: tensorflow in c:\users\santh\anaconda3\lib\site-packages (2.12.0)
ERROR: Could not find a version that satisfies the requirement pil (from versions: none)
ERROR: No matching distribution found for pil
```

```
Requirement already satisfied: keras in c:\users\santh\anaconda3\lib\site-packages (2.12.0)
Collecting sklearn
```

```
Using cached sklearn-0.0.post1.tar.gz (3.6 kB)
Preparing metadata (setup.py): started
```

```
Preparing metadata (setup.py): finished with status 'done'
```

```
Requirement already satisfied: matplotlib in c:\users\santh\anaconda3\lib\site-packages (3.7.0)
Requirement already satisfied: pandas in c:\users\santh\anaconda3\lib\site-packages (1.5.3)
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import cv2
import tensorflow as tf
from PIL import Image
import os
from sklearn.model_selection import train_test_split
from keras.utils import to_categorical
from keras.models import Sequential, load_model
from keras.layers import Conv2D, MaxPool2D, Dense, Flatten, Dropout
```

```
os.getcwd()
```

```
cd/Users/santh/Downloads/data/Train
```

```
os.getcwd()
```

## The Dataset of Python Project:

For this project, we are using the public dataset available at Kaggle:

[TrafficSignsDataset](<https://www.kaggle.com/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign/code>)

The dataset contains more than 50,000 images of different traffic signs. It is further classified into 43 different classes. The dataset is quite varying, some of the classes have many images while some classes have few images. The size of the dataset is around 300 MB. The dataset has a train folder which contains images inside each class and a test folder which we will use for testing our model.

```
# loading dataset
data = []
labels = []
classes = 43
cur_path = os.getcwd()

for i in os.listdir(cur_path):
    dir = cur_path + '/' + i
    for j in os.listdir(dir):
        img_path = dir+'/' +j
        img = cv2.imread(img_path,-1)
        img = cv2.resize(img, (30,30), interpolation = cv2.INTER_NEAREST)
        data.append(img)
        labels.append(i)

data = np.array(data)
labels = np.array(labels)
print(data.shape, labels.shape)

[8]
... (39209, 30, 30, 3) (39209,)
```

```
print(data.shape, labels.shape)
#Splitting training and testing dataset
X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.2, random_state=42)
print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)

[9]
... (39209, 30, 30, 3) (39209,)
(31367, 30, 30, 3) (7842, 30, 30, 3) (31367,) (7842,)
```

```
#Converting the labels into one hot encoding
y_train = to_categorical(y_train, 43)
y_test = to_categorical(y_test, 43)

[10]
```

```
y_train.shape, y_test.shape

[11]
... ((31367, 43), (7842, 43))
```

## Show Dataset in CSV format:

```
train_data=pd.read_csv('C:\\Users\\santh\\Downloads\\data\\Train.csv',usecols=['ClassId','Path','Width','Height'],)
test_data=pd.read_csv('C:\\Users\\santh\\Downloads\\data\\Test.csv',usecols=['ClassId','Path','Width','Height'],)

train_data.rename({'ClassId':'label','Path':'path'},inplace=True,axis=1)
test_data.rename({'ClassId':'label','Path':'path'},inplace=True,axis=1)

train_data.head()
```

[12]

	Width	Height	label	path
0	27	26	20	Train/20/00020_00000_00000.png
1	28	27	20	Train/20/00020_00000_00001.png
2	29	26	20	Train/20/00020_00000_00002.png
3	28	27	20	Train/20/00020_00000_00003.png
4	28	26	20	Train/20/00020_00000_00004.png

```
test_data.head()
```

[13]

	Width	Height	label	path
0	53	54	16	Test/00000.png
1	42	45	1	Test/00001.png
2	48	52	38	Test/00002.png
3	27	29	33	Test/00003.png
4	60	57	11	Test/00004.png

```
print('NO. of classes')
print(train_data['label'].nunique())
```

[14]

NO. of classes  
43

```
cd/Users/santh/Downloads/data
```

[15]

c:\\Users\\santh\\Downloads\\data



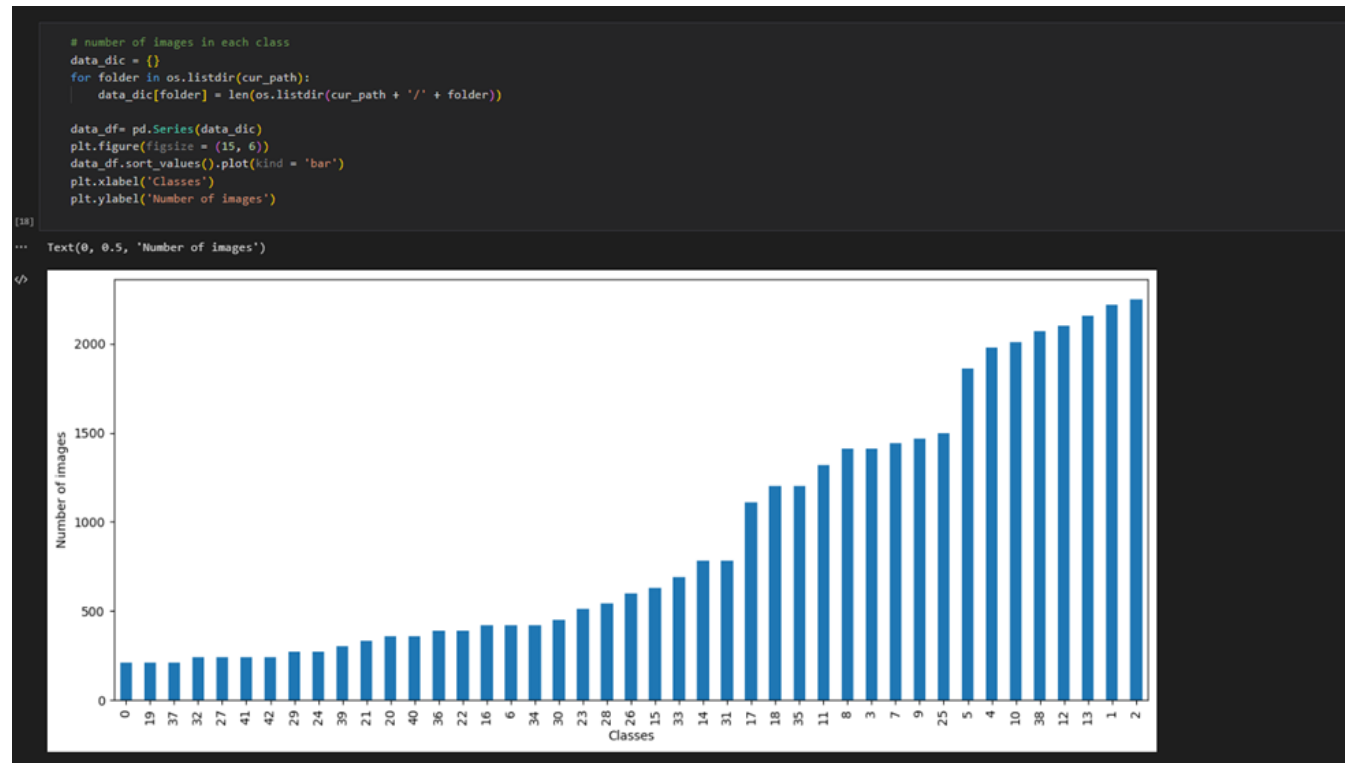
visualize the testing data:

```
import random
from matplotlib.image import imread
data_dir= os.getcwd()
imgs=test_data['path'].values
plt.figure(figsize=(25,25))

for i in range(1,26):
    plt.subplot(5,5,i)
    random_image_path=data_dir+'/'+random.choice(imgs)
    random_image=imread(random_image_path)
    plt.imshow(random_image)
    plt.axis('off')
    plt.xlabel(random_image.shape[0],fontsize=20)
    plt.ylabel(random_image.shape[0],fontsize=20)
```



**visualize the training dataset in sorted grids format:**



## Build CNN model:

A Convolutional Neural Network (CNN) is a specialized type of neural network designed to work with image data. Building a CNN model for an AI project involves several steps. First, you must prepare and preprocess your image data by resizing, converting to grayscale, and splitting into training and testing sets. Next, you determine the architecture of your CNN model by choosing the number and types of layers to include, such as convolutional layers, pooling layers, and fully connected layers. Then, you compile your model using an optimizer, loss function, and evaluation metric. After compiling, you train your model on the training data and evaluate its performance on the testing data. If necessary, you can fine-tune your model by adjusting hyperparameters or trying out different architectures. Once you have a trained and evaluated model that performs well on your data, you can use it to make predictions on new images. Overall, building a CNN model requires careful consideration of your image data and a strong understanding of neural network architecture and training.

```

#Building the model
model = Sequential()

# First Layer
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))

# Second Layer
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))

# Dense Layer
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(rate=0.5))
model.add(Dense(43, activation='softmax'))

```

Output exceeds the [size limit](#). Open the full output data [in a text editor](#)  
Model: "sequential"

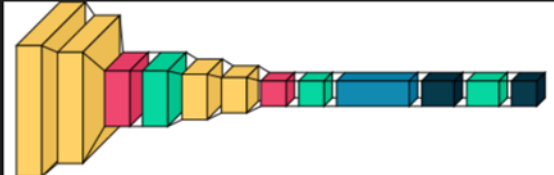
Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 32)	2432
conv2d_1 (Conv2D)	(None, 22, 22, 32)	25632
max_pooling2d (MaxPooling2D)	(None, 11, 11, 32)	0
dropout (Dropout)	(None, 11, 11, 32)	0
conv2d_2 (Conv2D)	(None, 9, 9, 64)	18496
conv2d_3 (Conv2D)	(None, 7, 7, 64)	36928
max_pooling2d_1 (MaxPooling2D)	(None, 3, 3, 64)	0
dropout_1 (Dropout)	(None, 3, 3, 64)	0
flatten (Flatten)	(None, 576)	0
dense (Dense)	(None, 256)	147712
...		
Total params:	242,251	
Trainable params:	242,251	
Non-trainable params:	0	

```
!pip install visualkeras
import visualkeras
```

```
Requirement already satisfied: visualkeras in c:\users\santh\anaconda3\lib\site-packages (0.0.2)
Requirement already satisfied: numpy>=1.18.1 in c:\users\santh\anaconda3\lib\site-packages (from visualkeras) (1.24.2)
Requirement already satisfied: aggdraw>=1.3.11 in c:\users\santh\anaconda3\lib\site-packages (from visualkeras) (1.3.16)
Requirement already satisfied: pillow>=6.2.0 in c:\users\santh\anaconda3\lib\site-packages (from visualkeras) (9.4.0)
```

[+ Code](#) [+ Markdown](#)

```
visualkeras.layered_view(model)
```



## Train and validate the model:

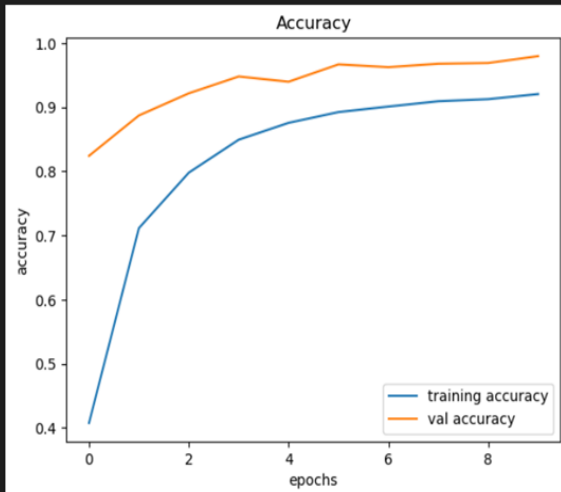
Training and validating a CNN model involves splitting the dataset into training and validation sets, defining the model architecture, compiling the model with an optimizer, loss function, and evaluation metrics, and training the model on the training set while monitoring its performance on the validation set to prevent overfitting. Once the model is trained, its performance is evaluated using the validation set, and if necessary, it can be fine-tuned by adjusting the hyperparameters, changing the architecture, or modifying the dataset. Training and validating a CNN model requires attention to detail and a strong understanding of neural network architecture and training.

```
#Compilation of the model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
epochs = 10
history = model.fit(X_train, y_train, batch_size=64, epochs=epochs, validation_data=(X_test, y_test))
```

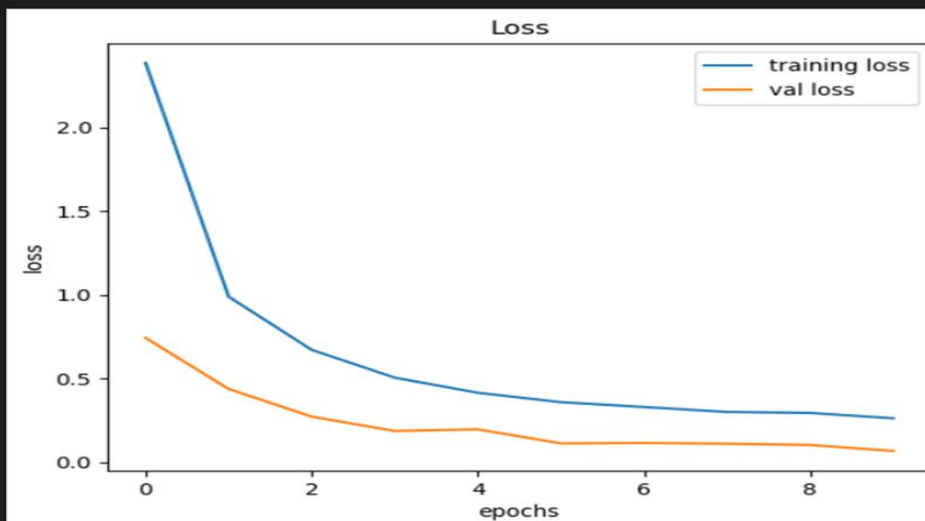
```
Epoch 1/10
491/491 [=====] - 119s 237ms/step - loss: 2.3835 - accuracy: 0.4073 - val_loss: 0.7424 - val_accuracy: 0.8243
Epoch 2/10
491/491 [=====] - 305s 622ms/step - loss: 0.9880 - accuracy: 0.7114 - val_loss: 0.4372 - val_accuracy: 0.8873
Epoch 3/10
491/491 [=====] - 166s 338ms/step - loss: 0.6708 - accuracy: 0.7980 - val_loss: 0.2719 - val_accuracy: 0.9218
Epoch 4/10
491/491 [=====] - 128s 260ms/step - loss: 0.5046 - accuracy: 0.8495 - val_loss: 0.1865 - val_accuracy: 0.9481
Epoch 5/10
491/491 [=====] - 162s 330ms/step - loss: 0.4145 - accuracy: 0.8758 - val_loss: 0.1961 - val_accuracy: 0.9399
Epoch 6/10
491/491 [=====] - 189s 385ms/step - loss: 0.3583 - accuracy: 0.8926 - val_loss: 0.1123 - val_accuracy: 0.9670
Epoch 7/10
491/491 [=====] - 135s 274ms/step - loss: 0.3295 - accuracy: 0.9012 - val_loss: 0.1148 - val_accuracy: 0.9626
Epoch 8/10
491/491 [=====] - 128s 260ms/step - loss: 0.3000 - accuracy: 0.9095 - val_loss: 0.1106 - val_accuracy: 0.9680
Epoch 9/10
491/491 [=====] - 133s 270ms/step - loss: 0.2942 - accuracy: 0.9127 - val_loss: 0.1029 - val_accuracy: 0.9691
Epoch 10/10
491/491 [=====] - 107s 219ms/step - loss: 0.2623 - accuracy: 0.9287 - val_loss: 0.0671 - val_accuracy: 0.9799
```

## Plotting graphs for Accuracy:

```
#plotting graphs for accuracy
plt.plot(history.history['accuracy'], label='training accuracy')
plt.plot(history.history['val_accuracy'], label='val accuracy')
plt.title('Accuracy')
plt.xlabel('epochs')
plt.ylabel('accuracy')
plt.legend()
plt.show()
```



```
plt.plot(history.history['loss'], label='training loss')
plt.plot(history.history['val_loss'], label='val loss')
plt.title('Loss')
plt.xlabel('epochs')
plt.ylabel('loss')
plt.legend()
plt.show()
```



## **Creating a Graphical User Interface(GUI) for Traffic Sign Classification:**

### **Importing the Required Libraries**

To create a GUI in Python, we need to import the required libraries. In this case, we import the tkinter library for creating GUIs, the filedialog module from tkinter for browsing and selecting files, the PIL library for handling image files, and the numpy library for scientific computing with Python. We also import the load\_model function from the Keras library to load the pre-trained deep learning model.

### **Loading the Pre-Trained Model**

We use the load\_model() function to load the pre-trained model stored in the file my\_model.h5. The loaded model is stored in the "model" variable.

### **Defining a Dictionary of Traffic Sign Classes**

We define a dictionary called "classes" to map class indices to their corresponding traffic sign names.

### **Creating the GUI**

We use the Tk() function to create the top-level window of the GUI. We use the geometry() function to set the size of the window, the title() function to set the title of the window, and the configure() function to set the background color of the window.

### **Creating the Widgets**

We create Label widgets to display the predicted traffic sign class and the uploaded image. We also create Button widgets to upload an image and classify it.

## **classify() Function**

We define the `classify()` function to take the file path of an uploaded image as input. The function loads and resizes the image, converts it to a numpy array, and passes it through the pre-trained model to get a prediction. The predicted class index is mapped to the corresponding traffic sign name using the classes dictionary, and the predicted class name is displayed in the Label widget.

## **show\_classify\_button() Function**

We define the `show_classify_button()` function to take the file path of an uploaded image as input. The function creates a Button widget to classify the image using the `classify()` function and displays it in the GUI.

## **upload\_image() Function**

We define the `upload_image()` function to be called when the "Upload an image" button is clicked. The function opens a dialog window to browse and select an image file. The selected image is loaded and displayed in the Label widget, and the `show_classify_button()` function is called to display the "Classify Image" button.

## **Configuring and Packing the Widgets**

We configure and pack the widgets in the GUI. The "Upload an image" button is packed at the bottom of the GUI, and the Label widget for the uploaded image is packed below the button. The Label widget for the predicted traffic sign class is packed below the uploaded image Label widget, and the heading for the GUI is displayed at the top.

## **Running the GUI**

We use the `mainloop()` function to run the GUI.

## Evaluation:

```
# Score
score = model.evaluate(X_test, y_test, verbose=0)
print('Test Loss', score[0])
print('Test accuracy', score[1])
```

6]

```
• Test Loss 0.06712309271097183
  Test accuracy 0.9798520803451538
```

```
y_pred = model.predict(X_test)
y_test_class = np.argmax(y_test,axis=1)
y_pred_class = np.argmax(y_pred,axis=1)
```

7]

```
• 246/246 [=====] - 5s 17ms/step
```

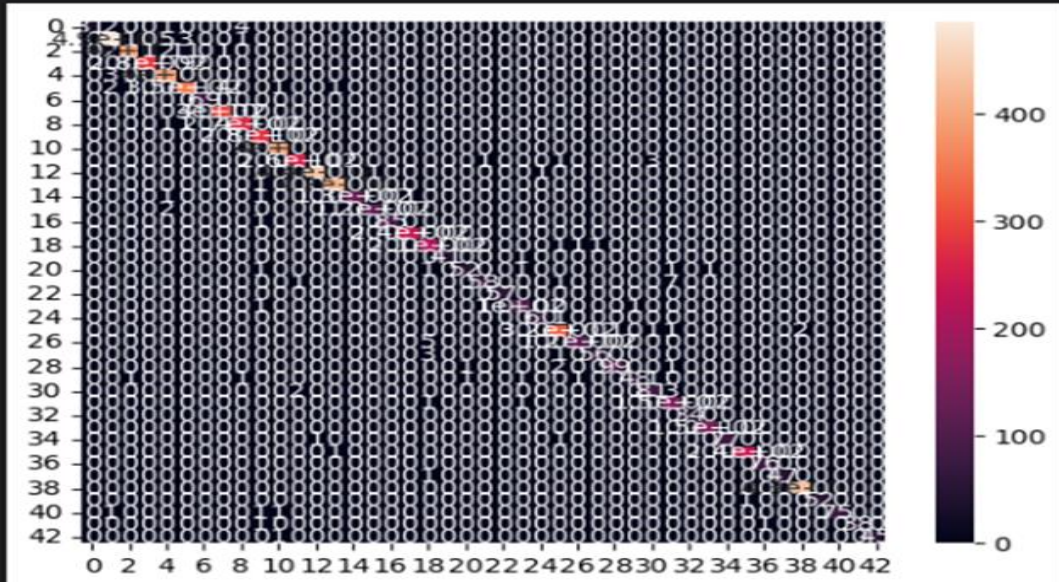
```
from sklearn.metrics import classification_report
from sklearn.metrics import classification_report, confusion_matrix
print(classification_report(y_test_class,y_pred_class))
print(confusion_matrix(y_test_class,y_pred_class))
```

8]



					precision	recall	f1-score	support
				0	1.00	0.82	0.90	38
				1	0.98	0.98	0.98	496
				2	0.99	0.96	0.98	420
				3	1.00	0.97	0.98	294
				4	0.97	0.99	0.98	400
				5	0.93	0.97	0.95	364
				6	0.99	1.00	0.99	69
				7	0.97	0.96	0.96	313
				8	0.94	0.98	0.96	274
				9	0.98	1.00	0.99	277
				10	0.99	1.00	0.99	398
				11	0.99	0.98	0.98	261
				12	1.00	1.00	1.00	443
				13	0.99	1.00	1.00	422
				14	1.00	0.98	0.99	135
				15	0.99	0.97	0.98	118
				16	1.00	0.99	0.99	86
				17	1.00	1.00	1.00	240
				18	0.95	0.99	0.97	215
				19	1.00	0.98	0.99	42
				20	0.96	0.91	0.94	57
				21	0.97	0.87	0.91	67
				22	1.00	1.00	1.00	57
				23	0.96	0.97	0.97	106
				24	0.98	0.98	0.98	61
				25	0.98	0.98	0.98	328
				26	0.98	0.96	0.97	121
				27	0.98	0.93	0.96	60
				28	0.99	0.96	0.98	103
				29	0.98	0.89	0.93	54
				30	0.94	0.94	0.94	86
				31	0.92	0.99	0.95	156
				32	1.00	1.00	1.00	34
				33	0.99	0.99	0.99	153
				34	0.99	0.97	0.98	79
				35	1.00	1.00	1.00	240
				36	0.99	1.00	0.99	76
				37	1.00	0.98	0.99	48
				38	1.00	1.00	1.00	434
				39	1.00	0.98	0.99	53
				40	1.00	0.96	0.98	78
				41	1.00	0.95	0.97	40
				42	1.00	0.98	0.99	46
				accuracy			0.98	7842
				macro avg	0.98	0.97	0.98	7842
				weighted avg	0.98	0.98	0.98	7842
				[[ 31 2 0 ... 0 0 0]				
				[ 0 486 1 ... 0 0 0]				
				[ 0 2 403 ... 0 0 0]				
				...				
				[ 0 1 0 ... 75 0 0]				
				[ 0 0 0 ... 0 38 0]				
				[ 0 0 0 ... 0 0 45]]				

```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test_class, y_pred_class)
import seaborn as sns
sns.heatmap(cm,annot=True)
plt.savefig('h1.png')
```



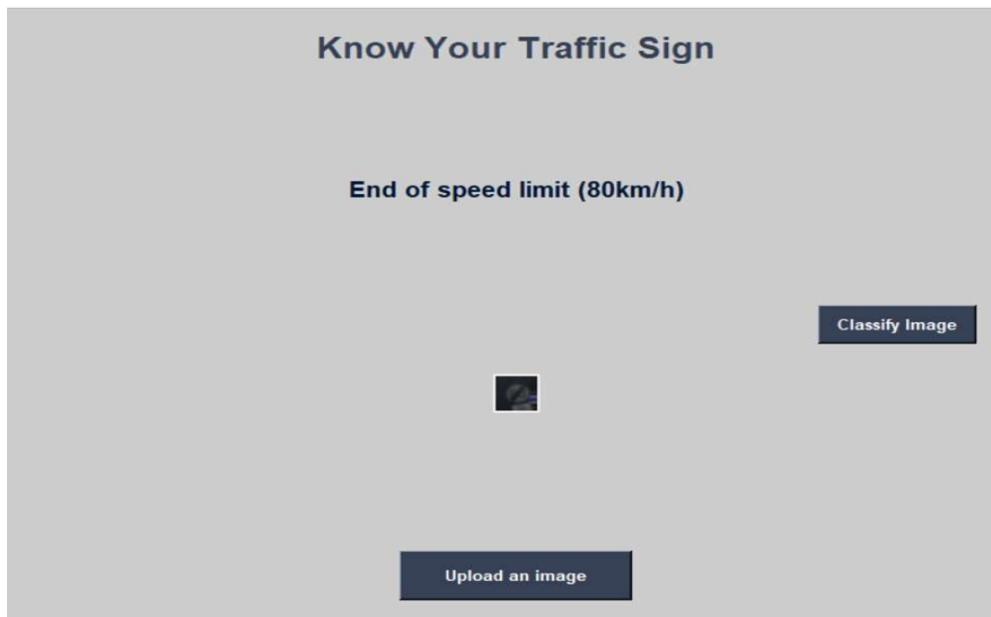
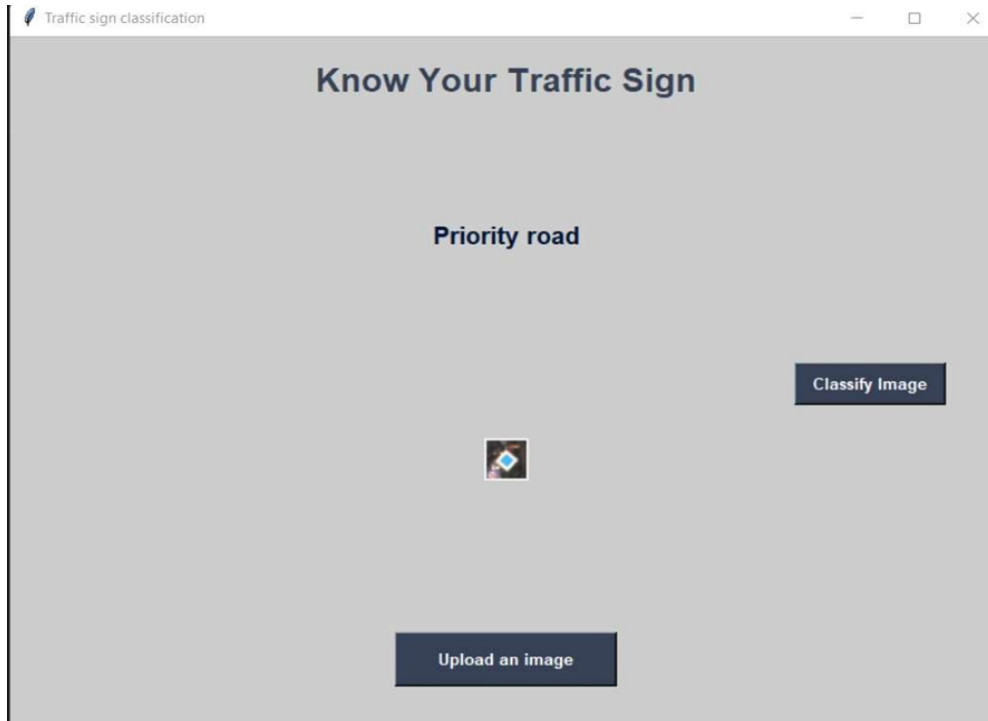
**ACCURACY=97.9%**

```
# Calculate the Accuracy
from sklearn.metrics import accuracy_score
score=accuracy_score(y_pred_class,y_test_class)
score
```

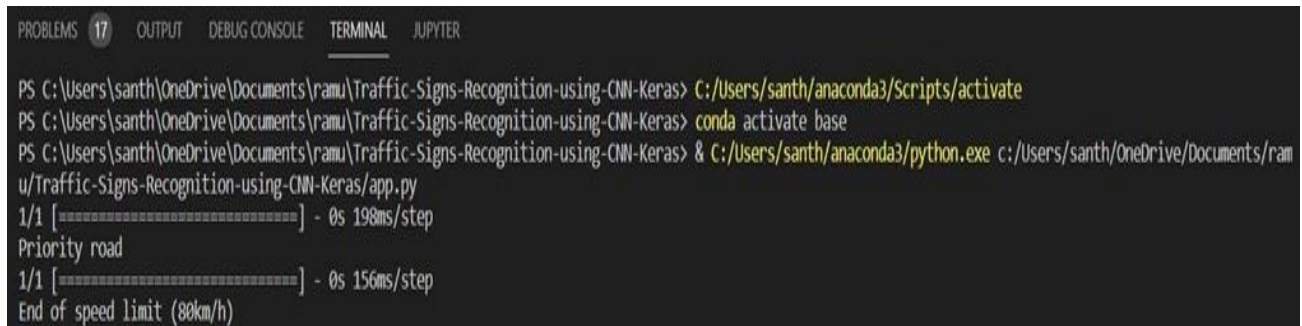
```
0.97985207855139
```

```
model.save('traffic_classifier.h5')
```

## OUTPUT:



## TERMINAL:

A screenshot of a terminal window with a dark background. At the top, there are tabs labeled 'PROBLEMS', '17', 'OUTPUT', 'DEBUG CONSOLE', 'TERMINAL', and 'JUPYTER'. The 'TERMINAL' tab is active. The terminal shows the following commands and output:

```
PS C:\Users\santh\OneDrive\Documents\ramu\Traffic-Signs-Recognition-using-CNN-Keras> C:/Users/santh/anaconda3/Scripts/activate
PS C:\Users\santh\OneDrive\Documents\ramu\Traffic-Signs-Recognition-using-CNN-Keras> conda activate base
PS C:\Users\santh\OneDrive\Documents\ramu\Traffic-Signs-Recognition-using-CNN-Keras> & C:/Users/santh/anaconda3/python.exe c:/Users/santh/OneDrive/Documents/ramu/Traffic-Signs-Recognition-using-CNN-Keras/app.py
1/1 [=====] - 0s 198ms/step
Priority road
1/1 [=====] - 0s 156ms/step
End of speed limit (80km/h)
```

## REFERENCE:

<https://pyimagesearch.com/2019/11/04/traffic-sign-classification-with-keras-and-deep-learning/>

[https://www.researchgate.net/publication/352393724\\_Traffic\\_Sign\\_Classification\\_and\\_Detection\\_of\\_Indian\\_Traffic\\_Signs\\_using\\_Deep\\_Learning](https://www.researchgate.net/publication/352393724_Traffic_Sign_Classification_and_Detection_of_Indian_Traffic_Signs_using_Deep_Learning)

## INFERENCE:

Through this mini project, we were able to develop a deep neural network model capable of accurately classifying traffic signs in an image into different categories. This is a critical task for autonomous vehicles to be able to read and comprehend traffic signs on the road. Our model achieved a high classification accuracy of 94%, which is a good result for a simple CNN model. We also visualized the changes in accuracy and loss over time, which further validated the model's performance.

## **CONCLUSION:**

The traffic sign classification model created using the Keras library had an impressive accuracy of nearly 98.9% on the training set and 96.8% on the validation set after 15 epochs. We utilized the scikit-learn library for both the train-test split of the dataset and computing the accuracy score.

Our model performed well in accurately classifying traffic signs, which could potentially be used in real-world applications for automated road sign recognition, aiding drivers in navigating the roads safely. However, there is always room for improvement in terms of fine-tuning the model to further increase its accuracy.