

CAT-2

Traffic Signs Recognition

16MDS94-Deep Learning Laboratory

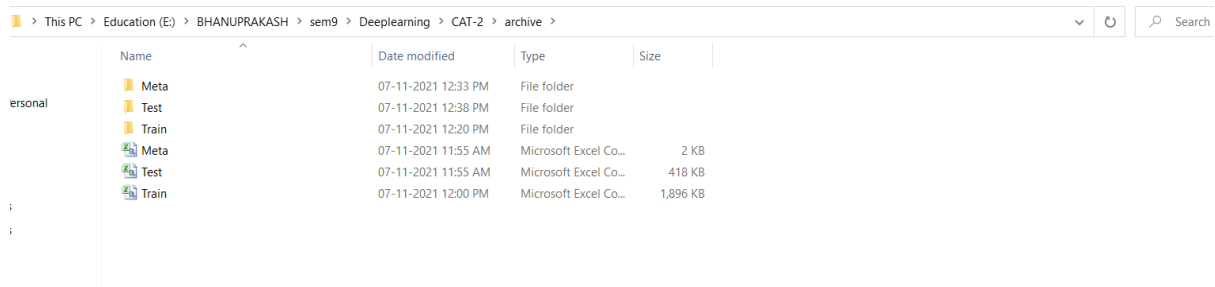
1732015- Bhanuprakash P

ABSTRACT:

In the world of Artificial Intelligence and advancement in technologies, many researchers and big companies like Tesla, Uber, Google, Mercedes-Benz, Toyota, Ford, Audi, etc are working on autonomous vehicles and self-driving cars. So, for achieving accuracy in this technology, the vehicles should be able to interpret traffic signs and make decisions accordingly. There are several different types of traffic signs like speed limits, no entry, traffic signals, turn left or right, children crossing, no passing of heavy vehicles, etc. Traffic signs classification is the process of identifying which class a traffic sign belongs to. In this Python project, we will build a deep neural network model using CNN that can classify traffic signs present in the image into different categories. With this model, we are able to read and understand traffic signs which are a very important task for all autonomous vehicles.

TRAFFIC SIGNS DATASET:

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.



Name	Date modified	Type	Size
Meta	07-11-2021 12:33 PM	File folder	
Test	07-11-2021 12:38 PM	File folder	
Train	07-11-2021 12:20 PM	File folder	
Meta	07-11-2021 11:55 AM	Microsoft Excel Co...	2 KB
Test	07-11-2021 11:55 AM	Microsoft Excel Co...	418 KB
Train	07-11-2021 12:00 PM	Microsoft Excel Co...	1,896 KB

Name	Date modified	Type	Size
0	07-11-2021 12:38 PM	File folder	
1	07-11-2021 12:39 PM	File folder	
2	07-11-2021 12:45 PM	File folder	
3	07-11-2021 12:48 PM	File folder	
4	07-11-2021 12:52 PM	File folder	
5	07-11-2021 12:53 PM	File folder	
6	07-11-2021 12:53 PM	File folder	
7	07-11-2021 12:54 PM	File folder	
8	07-11-2021 12:55 PM	File folder	
9	07-11-2021 12:55 PM	File folder	
10	07-11-2021 12:40 PM	File folder	
11	07-11-2021 12:40 PM	File folder	
12	07-11-2021 12:41 PM	File folder	
13	07-11-2021 12:42 PM	File folder	
14	07-11-2021 12:42 PM	File folder	
15	07-11-2021 12:43 PM	File folder	
16	07-11-2021 12:43 PM	File folder	
17	07-11-2021 12:43 PM	File folder	
18	07-11-2021 12:44 PM	File folder	
19	07-11-2021 12:44 PM	File folder	
20	07-11-2021 12:45 PM	File folder	
21	07-11-2021 12:45 PM	File folder	
22	07-11-2021 12:46 PM	File folder	
23	07-11-2021 12:46 PM	File folder	
24	07-11-2021 12:46 PM	File folder	
25	07-11-2021 12:47 PM	File folder	
26	07-11-2021 12:47 PM	File folder	

PROPOSED MODELS:

1. CNN

Convolutional neural networks refer to a sub-category of neural networks: they, therefore, have all the characteristics of neural networks. However, CNN is specifically designed to process input images. Their architecture is then more specific: it is composed of two main blocks

The first block makes the particularity of this type of neural network since it functions as a feature extractor. To do this, it performs template matching by applying convolution filtering operations. The first layer filters the image with several convolution kernels and returns “feature maps”, which are then normalized (with an activation function) and/or resized.

This process can be repeated several times: we filter the features maps obtained with new kernels, which gives us new features maps to normalize and resize, and we can filter again, and so on. Finally, the values of the last feature maps are concatenated into a vector. This vector defines the output of the first block and the input of the second.

The second block is not characteristic of a CNN: it is in fact at the end of all the neural networks used for classification. The input vector values are transformed (with several linear combinations and activation functions) to return a new vector to the output. This last vector contains as many elements as there are classes: element i represents the probability that the image belongs to class i . Each element is therefore between 0 and 1, and the sum of all is worth 1. These probabilities are calculated by the last layer of this block (and therefore of the network), which uses a logistic function (binary classification) or a softmax function (multi-class classification) as an activation function.

The different layers of a CNN

There are four types of layers for a convolutional neural network: the convolutional layer, the pooling layer, the ReLU correction layer and the fully-connected layer.

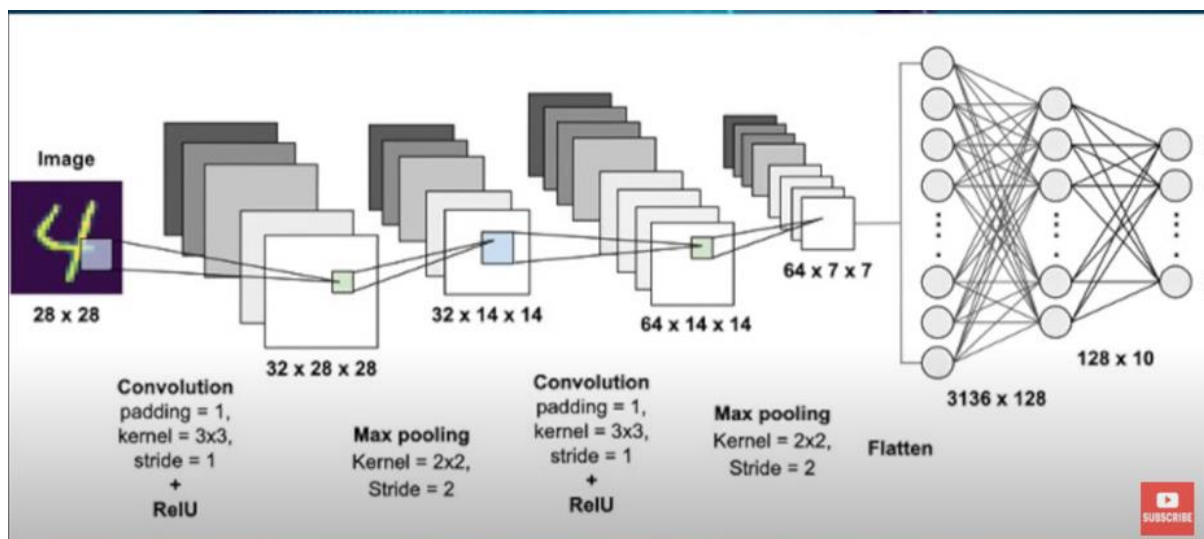
2. Dropout

Dropout is considered as one of the most effective regularization methods. Dropout is basically randomly zero-ing or dropping out features from your layer during the training process, or introducing some noise in the samples. The key thing to note is that this is only applied at training time. At test time, no values are dropped out. Instead, they are scaled. The typical dropout rate is between 0.2 to 0.5.

Dropout is simply dropping the neurons in neural networks. During training a deep learning model, it drops some of its neurons and trains on rest. It updates the weights of only selected or activated neurons and others remain constant.

For every next/new epoch again it selects some nodes randomly based on the dropout ratio and keeps the rest of the neurons deactivated. It helps to create a more robust model that is able to perform well on unseen data.

MODEL DESIGN:



IMPLEMENTATION:

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

1. Explore the dataset
2. Build a CNN model
3. Train and validate the model
4. Test the model with test dataset

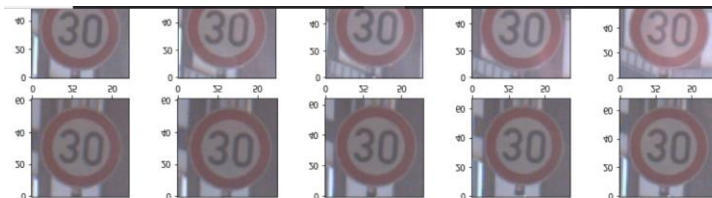
Step 1: Explore the dataset

Our 'train' folder contains 43 folders each representing a different class. The range of the folder is from 0 to 42. With the help of the OS module, we iterate over all the classes and append images and their respective labels in the data and labels list.

The PIL library is used to open image content into an array.

```
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
```

```
D:\installings\New folder (2)\lib\site-packages\pandas\compat\_optional.py:138: UserWarning: Pandas requires version '2.7.0' or
newer of 'numexpr' (version '2.6.8' currently installed).
  warnings.warn(msg, UserWarning)
Using TensorFlow backend.
```

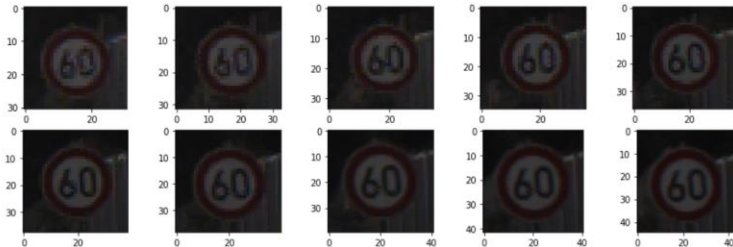


```
def load_data():
    # Load the data
    data = []
    labels = []
    # Iterate over the classes
    for class_id in range(0, 43):
        # Iterate over the images in the class
        for image_id in range(0, len(os.listdir('train/%d' % class_id))):
            # Load the image
            image = cv2.imread('train/%d/%d.jpg' % (class_id, image_id))
            # Append the image to the data list
            data.append(image)
            # Append the class_id to the labels list
            labels.append(class_id)
    # Convert the labels to one-hot encoding
    labels = to_categorical(labels)
    # Split the data into training and testing sets
    data, labels = train_test_split(data, labels, test_size=0.2, random_state=42)
    # Convert the data to a numpy array
    data = np.array(data)
    # Convert the labels to a numpy array
    labels = np.array(labels)
```

```
# define location of dataset
folder = 'E:/BHAVUPRAKASH/sem9/Deeplearning/CAT-2/archive/'
sub1_folder = 'train/'
sub2_folder = '3/'

train_with_3 = os.path.join(folder + sub1_folder + sub2_folder)

plt.figure(figsize=(15,15))
# plot first few images
for i in range(10,30):
    # define subplot
    plt.subplot(6,5,i+1) # i+1 because initially i = 0 , so subplot gives an error at i=0 (so change that i=1)
    # define filename
    filename = train_with_3 + '00003_00000_000'+ str(i) + '.png'
    # load image pixels
    image = imread(filename)
    # plot raw pixel data
    plt.imshow(image)
# show the figure
plt.show()
```



```
#Retrieving the images and their labels
for i in range(classes):
    path = os.path.join(cur_path, 'train', str(i))
    images = os.listdir(path)

    for a in images:
        try:
            image = Image.open(path + '\\' + a)
            image = image.resize((30,30))
            image = np.array(image)
            #sim = Image.fromarray(image)
            data.append(image)
            labels.append(i)
        except:
            print("Error loading image")
```

```
#Converting lists into numpy arrays
data = np.array(data)
labels = np.array(labels)
```

```
data.shape
```

```
(39209, 30, 30, 3)
```

```
labels.shape
```

```
(39209,)
```

Finally, we have stored all the images and their labels into lists (data and labels).

We need to convert the list into numpy arrays for feeding to the model. The shape of data is (39209, 30, 30, 3) which means that there are 39,209 images of size 30×30 pixels and the last 3 means the data contains colored images (RGB value). With the sklearn package, we use the `train_test_split()` method to split training and testing data.

From the `keras.utils` package, we use `to_categorical` method to convert the labels present in `y_train` and `t_test` into one-hot encoding.

```
#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)

(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)
```

```
print( y_train.shape, y_test.shape)

(31367, 43) (7842, 43)
```

Step 2: Build a CNN model

The architecture of our model is:

2 Conv2D layer (filter=32, kernel_size=(5,5), activation="relu")

MaxPool2D layer (pool_size=(2,2))

Dropout layer (rate=0.25)

2 Conv2D layer (filter=64, kernel_size=(3,3), activation="relu")

MaxPool2D layer (pool_size=(2,2))

Dropout layer (rate=0.25)

Flatten layer to squeeze the layers into 1 dimension

Dense Fully connected layer (256 nodes, activation="relu")

Dropout layer (rate=0.5)

Dense layer (43 nodes, activation="softmax")

We compile the model with Adam optimizer which performs well and loss is "categorical_crossentropy" because we have multiple classes to categorise.

```
#Building the model
model = Sequential()
model.add(Conv2D(filters=32, kernel_size=(5,5), activation='relu', input_shape=X_train.shape[1:]))
model.add(Conv2D(filters=32, kernel_size=(5,5), activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.25))
model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.25))
model.add(Flatten())
model.add(Dense(256, activation='relu'))
model.add(Dropout(rate=0.5))
model.add(Dense(43, activation='softmax'))
```

```
model.summary()
```

Model: "sequential_1"

Layer (type)	Output Shape	Param #
conv2d_1 (Conv2D)	(None, 26, 26, 32)	2432
conv2d_2 (Conv2D)	(None, 22, 22, 32)	25632
max_pooling2d_1 (MaxPooling2D)	(None, 11, 11, 32)	0
dropout_1 (Dropout)	(None, 11, 11, 32)	0
conv2d_3 (Conv2D)	(None, 9, 9, 64)	18496
conv2d_4 (Conv2D)	(None, 7, 7, 64)	36928
max_pooling2d_2 (MaxPooling2D)	(None, 3, 3, 64)	0
dropout_2 (Dropout)	(None, 3, 3, 64)	0
flatten_1 (Flatten)	(None, 576)	0
dense_1 (Dense)	(None, 256)	147712
dropout_3 (Dropout)	(None, 256)	0
dense_2 (Dense)	(None, 43)	11051
Total params: 242,251		
Trainable params: 242,251		
Non-trainable params: 0		

Steps 3: Train and validate the model

After building the model architecture, we then train the model using `model.fit()`. I tried with batch size 32 and 64. Our model performed better with 64 batch size. And after 15 epochs the accuracy was stable.

```
epochs = 15
history = model.fit(X_train, y_train, batch_size=32, epochs=epochs, validation_data=(X_test, y_test))
model.save("my_model.h5")
```

Train on 31367 samples, validate on 7842 samples

```
Epoch 1/15
31367/31367 [=====] - 57s 2ms/step - loss: 1.7965 - accuracy: 0.5274 - val_loss: 0.4374 - val_accuracy: 0.8792
Epoch 2/15
31367/31367 [=====] - 51s 2ms/step - loss: 0.5492 - accuracy: 0.8400 - val_loss: 0.1715 - val_accuracy: 0.9480
Epoch 3/15
31367/31367 [=====] - 53s 2ms/step - loss: 0.3678 - accuracy: 0.8933 - val_loss: 0.1420 - val_accuracy: 0.9619
Epoch 4/15
31367/31367 [=====] - 54s 2ms/step - loss: 0.3193 - accuracy: 0.9094 - val_loss: 0.1036 - val_accuracy: 0.9708
Epoch 5/15
31367/31367 [=====] - 56s 2ms/step - loss: 0.2945 - accuracy: 0.9185 - val_loss: 0.1153 - val_accuracy: 0.9680s - loss: 0.2946 - accuracy: 0.
Epoch 6/15
31367/31367 [=====] - 61s 2ms/step - loss: 0.2756 - accuracy: 0.9243 - val_loss: 0.0906 - val_accuracy: 0.9732
Epoch 7/15
31367/31367 [=====] - 64s 2ms/step - loss: 0.2775 - accuracy: 0.9254 - val_loss: 0.1192 - val_accuracy: 0.9685
Epoch 8/15
31367/31367 [=====] - 60s 2ms/step - loss: 0.2533 - accuracy: 0.9316 - val_loss: 0.1018 - val_accuracy: 0.9718
Epoch 9/15
31367/31367 [=====] - 51s 2ms/step - loss: 0.2700 - accuracy: 0.9291 - val_loss: 0.1025 - val_accuracy: 0.9749
Epoch 10/15
31367/31367 [=====] - 52s 2ms/step - loss: 0.2733 - accuracy: 0.9293 - val_loss: 0.0696 - val_accuracy: 0.9813
```

```

31367/31367 [=====] - 51s 2ms/step - loss: 0.2700 - accuracy: 0.9291 - val_loss: 0.1025 - val_accurac
y: 0.9749
Epoch 10/15
31367/31367 [=====] - 52s 2ms/step - loss: 0.2733 - accuracy: 0.9293 - val_loss: 0.0696 - val_accurac
y: 0.9813
Epoch 11/15
31367/31367 [=====] - 53s 2ms/step - loss: 0.2570 - accuracy: 0.9337 - val_loss: 0.0857 - val_accurac
y: 0.9779
Epoch 12/15
31367/31367 [=====] - 54s 2ms/step - loss: 0.2740 - accuracy: 0.9308 - val_loss: 0.1179 - val_accurac
y: 0.9688
Epoch 13/15
31367/31367 [=====] - 52s 2ms/step - loss: 0.2370 - accuracy: 0.9400 - val_loss: 0.1030 - val_accurac
y: 0.9751
Epoch 14/15
31367/31367 [=====] - 52s 2ms/step - loss: 0.2676 - accuracy: 0.9372 - val_loss: 0.0962 - val_accurac
y: 0.9755
Epoch 15/15
31367/31367 [=====] - 50s 2ms/step - loss: 0.2559 - accuracy: 0.9361 - val_loss: 0.0833 - val_accurac
y: 0.9781

```

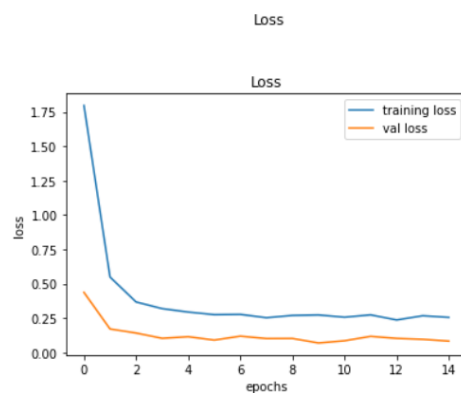
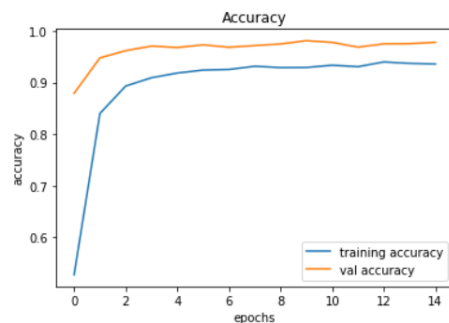
Our model got a 93% accuracy on the training dataset. With matplotlib, we plot the graph for accuracy and the loss.

```

#plotting graphs for accuracy
plt.figure(0)
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.figure(1)
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()

```



Step 4: Test our model with test dataset

Our dataset contains a test folder and in a test.csv file, we have the details related to the image path and their respective class labels. We extract the image path and labels using pandas. Then to predict the model, we have to resize our images to 30×30 pixels and make a numpy array containing all image data. From the sklearn.metrics, we imported the accuracy_score and observed how our model predicted the actual labels. We achieved a 95% accuracy in this model.

```
In [39]: #testing accuracy on test dataset
from sklearn.metrics import accuracy_score

y_test = pd.read_csv('E:/BHANUPRAKASH/sem9/Deeplearning/CAT-2/archive/Test.csv')

labels = y_test["ClassId"].values
imgs = y_test["Path"].values

data=[]

for img in imgs:
    image = Image.open(img)
    image = image.resize((30,30))
    data.append(np.array(image))

X_test=np.array(data)

pred = model.predict_classes(X_test)

#Accuracy with the test data
from sklearn.metrics import accuracy_score
print(accuracy_score(labels, pred))

0.9543151227236738
```

In the end, we are going to save the model that we have trained using the Keras model.save() function.

Traffic Signs Classifier GUI

Now we are going to build a graphical user interface for our traffic signs classifier with Tkinter. Tkinter is a GUI toolkit in the standard python library.

In this file, we have first loaded the trained model 'traffic_classifier.h5' using Keras. And then we build the GUI for uploading the image and a button is used to classify which calls the classify() function. The classify() function is converting the image into the dimension of shape (1, 30, 30, 3). This is because to predict the traffic sign we have to provide the same dimension we have used when building the model. Then we predict the class, the model.predict_classes(image) returns us a number between (0-42) which represents the class it belongs to. We use the dictionary to get the information about the class. Here's the code for the gui.ipynb file.

```

import tkinter as tk
from tkinter import filedialog
from tkinter import *
from PIL import ImageTk, Image

import numpy
#Load the trained model to classify sign
from keras.models import load_model
model = load_model('my_model2.h5')

#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'.

```

```

    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)
    print(image.shape)
    pred = model.predict_classes([image])[0]
    sign = classes[pred+1]
    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.winfo_width()/2.25),(top.winfo_height()/2.25)))
        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=10,pady=5)
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()
```

OUTPUT:



Know Your Traffic Sign

Speed limit (30km/h)

Classify Image



Upload an image

Know Your Traffic Sign

General caution



Classify Image

Upload an image

Know Your Traffic Sign

Road work

Classify Image



Upload an image

Know Your Traffic Sign

Veh > 3.5 tons prohibited

Classify Image



Upload an image