Traffic Sign Detection

SUBMITTED BY;----

Ajay Dobriyal, 19SCSE1110012

Aniket Krishan, 19SCSE1110017

Abstract

Automatic detection and recognition of traffic signals are very important and may be used with the help of drivers to reduce accidents and ultimately have less motor vehicles. In the world of Artificial Intelligence and technological advances, many researchers and large companies such as Tesla, Uber, Google, Mercedes-Benz,, Toyota, Ford, Audi, etc. they work in private cars and self-driving cars. Therefore, in order to be accurate in this technology, vehicles must be able to interpret road signs and make informed decisions. The proposed system works in real time detecting and recognizing images of traffic signals. Automatic systems will be able to control traffic on both open roads and intersections as well. Accurate visibility and average processing time have improved significantly. These improvements are essential to reduce the risk of accidents and to improve road safety conditions, providing a strong technical guarantee of continuous improvement of smart driving assistance.

Introduction

In this era of computer science, humans are getting additional obsessed with technology. With the improved technology, international firms like Google, Tesla, Uber, Ford, Audi, Toyota, Mercedes-Benz, and lots of additional are acting on automating vehicles. They're attempting to form additional correct autonomous or driverless vehicles. You all would possibly comprehend self-driving cars, wherever the vehicle itself behaves sort of a driver and doesn't want any human steerage to run on the road. This is often not wrong to accept the protection aspects—a likelihood of serious accidents from machines. However no machines are additional correct than humans. Researchers are running several algorithms to confirm 100% road safety and accuracy. One such rule is Traffic Sign Recognition that we have a tendency to state during this journal.

When you press on the road, you see numerous traffic signs like traffic signals, flip left or right, speed limits, no passing of serious vehicles, no entry, kids crossing, etc., that you simply got to follow for a secure drive. Likewise, autonomous vehicles even have to interpret these signs and create choices to realize accuracy. The methodology of recognizing that category a traffic sign belongs to is named Traffic signs classification.

In this Deep Learning project, we are going to build a model for the classification of traffic signs accessible within the image into several classes employing a Convolutional Neural Network(CNN) and Keras library. Neural networks may be accustomed build a traffic sign recognition system capable of recognizing and understanding the various traffic signs and facilitate the vehicles during which they're put in to determine and abide by these rules. In alternative words, the matter statement of this work is: to coach and build a neural network supported model which might acknowledge totally different traffic signs and take the adequate call once foreign during a vehicle. Such a neural network model will facilitate in dominant accidents that area unit caused as a results of over-speeding or symptom of drive

Literature Survey

For making auto-working and good vehicles agile of efforts area unit already heap in type of convolution network trained models with totally different projected methodologies for detection and recognition. A recent approach having wide applications for traffic sign detection and recognizing intelligent transportation systems that informs driver regarding precaution measures and sign connected info, uses color, form and ml algorithm-based ways, and provided comparative info on a similar. Color area, segmentation technique, features, and form detection technique area unit the terms thought-about within the review of the detection module. The paper presents a comparison between these ways and used datasets from totally different countries.

Some necessary datasets and ways price noting area unit quicker Region CNN (FRCNN) and improved LeNet-5 network model that incorporated deep learning models for task accomplishment. Many SSD and SVM based mostly models were conjointly projected. CNN models already trained on ImageNet dataset were used for recognition tasks as well as detection followed by classification of sign. Quicker RCNN was deployed with ResNet model and SSD alongside beginning V2 for performance analysis on GTSRB dataset with performance normal live getting used as frames per second (fps) and mAp. Not a way vital distinction was achieved however performance of YOLO v2 with coconut CNN was over previous two for classification in real time. Another approach was to normalize image at preprocessing stage given as input to VGG-16 of SSD algorithmic program, with model designed with a single softmax layer at the ultimate finish, five convolution layers stacked as initial layer with middle layer being three layers that were totally connected. Coaching batch size was five and 20 with 20,000 iterations to supply 96 accuracy and a learning rate of 0.001. AN approach projected by Dan Ciresan et al. equalized sizes of varied dataset pictures and used distinction normalizing techniques at pre-processing stage to normalize high and low distinction pictures results of that were fed to 9 layer model with three maxpooling and convolution layer between seven hidden layers at input and two totally connected layers by Dan Ciresan et al., that was trained on eight totally different datasets. The grayscale pictures made from model with pictures from totally different datasets with original dataset pictures were used more comprehensive of distinction normalized pictures. Pictures were translated, scaled and revolved at a nominative uniformly distributed vary before each epoch creating recognition rate to be 98.73% with CNN and rate of 99.15% rate for MLP combined with CNN. CNN combined with MLP gave 99.15% and CNN alone gave 98.73% recognition rate. However, the 'no

vehicle' sign wasn't classified accurately by each. Approach incorporated was projected for task mentioned afore on basis of CNN with considering adverse environmental things conjointly. Noise because of rain, dirt, shade, and lighting distorted clarity of real time image. Noise in dataset and different challenges detected by virtue of VGG-16 design alike model supported CNN. Totally different ways is used for reducing distortions from pictures, for distinction and noise exploitation CLAHE and take away|to get rid of} error because of rain by using ResNet supported design is accustomed remove rain from the pictures. A deep CNN with U-Net design is employed to localize traffic signs from the preprocessed image. Finally, for the classification task, two convolution blocks, every with two convolution layers, a ReLU activation layer followed by a Max pool layer, with a dropout layer more when every block was used. AN approach to section traffic sign from background scene that involves physicist filter for linear analysis of texture exploitation orientation and frequencies in image at bound region that is incredibly helpful for texture discrimination for special domain. Steps used were reading input, applying physicist filter with edge detection then YCBCR conversion method, feature segmentation feature choice extraction, correlation and performance matrix for performance analysis. It used two datasets particularly GTSDB and GTSRB for each recognition and detection. The information set enclosed scenes from maps and complicated data signs to verify algorithm's ability and potency. Clustering algorithmic program with neural network to extend detection rate with maximizing pooling positions as per projected algorithmic program of Mao and Qian is speculated to provide abundant high time period potency.

Parameters employed

The designed ML model's architecture consists of the following:

-Input: CIFAR-10 dataset's typical image is holder of its dimensions 32 x 32 x 3, where the depth refers to the number of RGB channels in the image.

-CONV: The second layer handles the computation of dot product between the input image region sharing connection and a neuron's weight.

-RELU: The third layer gives the desired dot product which is the resultant of applying a function for activation.

-POOL: The fourth layer is termed as pool layer and is used to perform down-sampling on images, for example, if the image dimensions are 32 x 32 x 12 it will be reduced to 16 x 16 x 12. It can also be said that it reduces spatial dimensions i.e. the height and width of an image.

-Fully connected layer: It is used to calculate class score which leads to

draw the final result on volume (a x b x c), where c represents categories, for example, if c = 10 then this represents categories of CIFAR-10.

The CNN learns the filters' values during the model training process. Some parameters are necessitated to specify here like the number of filters, network architecture being used, size of filter and many other, prior to the training process [2]. Recognition of unseen patterns from image becomes more efficient when number of filters are increased in number. Also, the extraction level and quality of pattern evaluation gets improved to a significant level. Some of the important model parameters are:

Filters- The convolution operation on an image is used to identify dependencies of the image. A feature map is produced as a result when any filter slides over an input image [18]. Different feature maps are generated in turn when convolution operation is performed with another filter. The filters and image are stored for operation as numeric matrices.

Epochs- It is an entity used to signify how many times the entire dataset is passed forward and backward through the neural network. Although the dataset is too big in size to feed to the system even once, but it is required to make multiple submissions to get multiple tests generated optimal results.

Batch size- Different from number of batches, it is a number which represents the number of training examples in a batch. As the whole dataset can't be passed at once to the neural network model, batch numbers are nothing but the division of a dataset so that it is easy to process with accuracy.

Learning rate- There is an iterative optimization algorithm in machine learning called as gradient descent. This iterative algorithm works multiple times to get optimal results. A parameter here in is known as learning rate. This also helps in making an under-fit graph to fit optimally.

Loss regularization- It refers to the changes or transformations that are made to reduce the generalization error, and not the training error. These modifications are done on the learning algorithm. It is an important factor to prevent over-fitting and maintain precision along with accuracy. Some of the methods used to perform data regularization include using a dropout layer, introducing weight penalty, augmentation on the dataset and incorporating early stopping for effectively tuning hyper-parameters like epochs, number of batches etc.

Dropout rate- Some nodes in neural networks are randomly chosen by the dropout layer for their movement along their incoming and outgoing connections. This layer operates on hidden layer as well as the input layer. It can also be said as an ensembling technique, and the training neural network dropout can be seen as an ensemble collection of 2^n thinned networks. It is useful for different random subsets of neurons which help in learning more robust

features. The different parameters of the CNN model being used are:

0 input layer

1Type- conv2d layer, filters=32, kernelsize=5,5, activation function="relu"

2 Type-conv2d layer, filters=32, kernelsize=5,5, activation function="relu"

3 Type-MaxPool2D, pool_size=2,2

4 Type- DropOut layer , dropout rate=0.25

5 Type-Conv2D layer , filters=64 , kernel_size=3,3 activation function="relu"

6 Type-Conv2D layer , filters=64 , kernel_size=3,3 activation function="relu"

7 Type-MaxPool2D , pool_size=2,2

8 Type-DropOut layer , dropout rate=0.25

9 Type-Flattening Layer

10 Type-Densing layer size=256, activation function="relu"

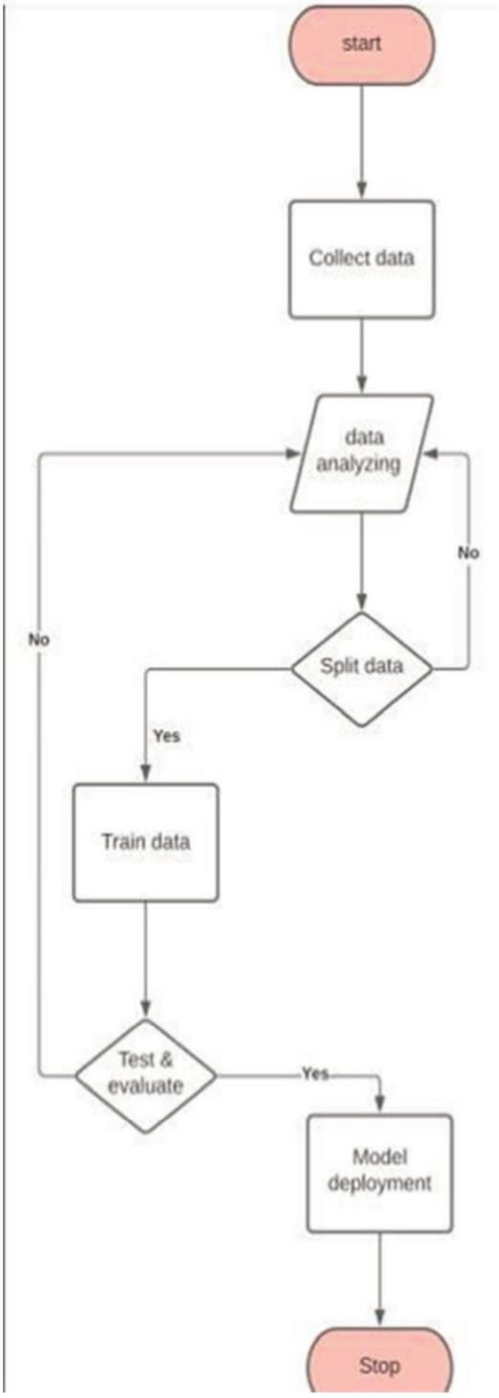
11 Type-DropOut layer , dropout rate=0.5

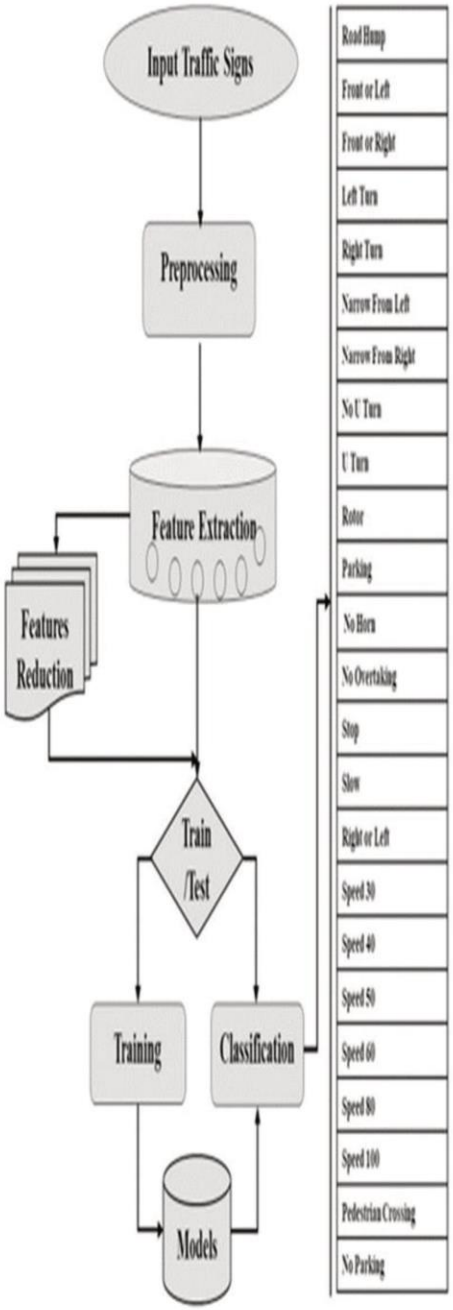
12 Type-Densing layer size=43, activation function="SoftMax"

Dataset

The image dataset is consists of more than 50,000 pictures of various traffic signs(speed limit, crossing, traffic signals, etc.) Around 43 different classes are present in the dataset for image classification. The dataset classes vary in size like some class has very few images while others have a vast number of images. The dataset doesn't take much time and space to download as the file size is around 314.36 MB. It contains two separate folders, train and test, where the train folder is consists of classes, and every category contains various images.


Label 0 (15)
Label 1 (110)
Label 2 (13)
Label 3 (15)
Label 4 (15)
Label 5 (11)
Label 6 (18)
Label 7 (157)
Label 8 (27)
Label 9 (18)
Label 10 (21)
Label 11 (7)
Label 12 (18)
Label 13 (90)
Label 14 (43)
Label 15 (9)
Label 16 (9)
Label 17 (79)
Label 18 (81)
Label 19 (231)
Label 20 (42)
Label 21 (43)
Label 22 (375)
Label 23 (15)
Label 24 (48)
Label 25 (42)
Label 26 (6)
Label 27 (18)
Label 28 (125)
Label 29 (33)
Label 30 (37)
Label 31 (63)
Label 32 (316)
Label 33 (12)
Label 34 (46)
Label 35 (60)
Label 36 (18)
Label 37 (98)
Label 38 (285)
Label 39 (196)
Label 40 (242)
Label 41 (148)
Label 42 (35)
Label 43 (30)
Label 44 (48)
Label 45 (74)
Label 46 (44)
Label 47 (147)
Label 48 (11)
Label 49 (12)
Label 50 (15)
Label 51 (27)
Label 52 (27)
Label 53 (199)
Label 54 (118)
Label 55 (12)
Label 56 (95)
Label 57 (78)
Label 58 (15)
Label 59 (42)
Label 60 (9)
Label 61 (282)


start
Collect data
data analyzing
No
Split data
No
Yes
Train data
Test & evaluate
Yes
Model deployment
Stop


Input Traffic Signs
Preprocessing
Feature Extraction
Features Reduction
Train /Test
Training
Classification
Models
Road Hump
Front or Left
Front or Right
Left Turn
Right Turn
Narrow From Left
Narrow From Right
No U Turn
U Turn
Rotor
Parking
No Horn
No Overtaking
Stop
Slow
Right or Left
Speed 30
Speed 40
Speed 50
Speed 60
Speed 80
Speed 100
Pedestrian Crossing
No Parking

Install all these packages to begin with the project:

```
pip install tensorflow

pip install tensorflow keras

pip install tensorflow sklearn

pip install tensorflow matplotlib

pip install tensorflow pandas

pip install tensorflow pil
```

Building the Model

We need to follow the below 4 steps to build our traffic sign classification model:

- Dataset exploration
- CNN model building
- Model training and validation
- Model testing

Dataset exploration

Around 43 subfolders(ranging from 0 to 42) are available in our 'train' folder, and each subfolder represents a different class. We have an OS module that helps in the iteration of all the images with their respective classes and labels. To open the contents of ideas into an array, we are using the PIL library.

```
import numpy as np

import pandas as pd

 import matplotlib.pyplot as plt

 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

from keras.layers import Conv2D, MaxPool2D,
Dense, Flatten, Dropout

data = []

labels = []

classes = 43

cur_path = os.getcwd()

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)

    data.append(image)

    labels.append(i)

  except:

      print("Error loading image")

data = np.array(data)

labels = np.array(labels)
```

In the end, we've to store each image with its corresponding labels into lists.

A NumPy array is required to feed the info to the model, therefore an convert this list into an array. Now, the form of our an is (39209, 30, 30, 3), where 39209 represents the amount of pictures, 30*30 represents the image sizes into pixels, and also the last three represents the RGB value(availability of colored data).print(data.shape, labels.shape)

#Splitting training and testing dataset

X_t1, X_t2, y_t1, y_t2 = train_test_split(data, labels, test_size=0.2, random_state=42)

print(X_t1.shape, X_t2.shape, y_t1.shape, y_t2.shape)

#Converting the labels into one hot encoding

y_t1 = to_categorical(y_t1, 43)

y_t2 = to_categorical(y_t2, 43)

CNN model building

We build a CNN model to classify the images into their respective categories.

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)
- Dense Fully connected layer (256 nodes, activation="relu")
- Dropout layer (rate=0.5)
- Dense layer (43 nodes, activation=" softmax")

- #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'))
- #Compilation of the model
- model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

Model training and validation

To train our model, we will use the model.fit() method that works well after the successful building of model architecture. With the help of 64 batch sizes, we got 95%accuracy on training sets and acquired stability after 15 epochs.

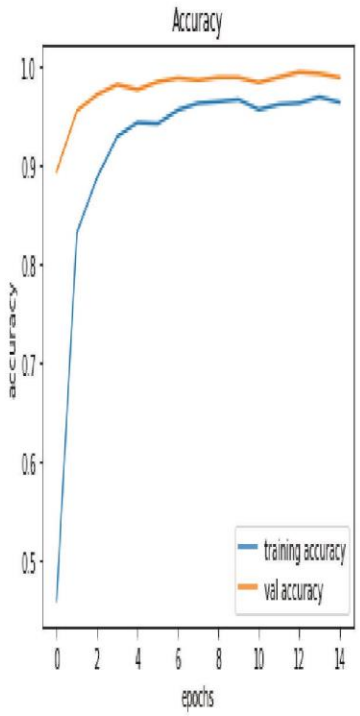
epochs = 15

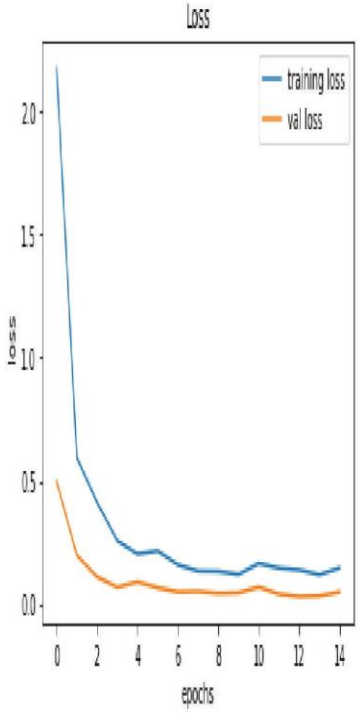
history = model.fit(X_train, y_train, batch_size=64, epochs=epochs, validation_data=(X_test, y_test))

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()
```

Output





Model testing

A folder named " test" is accessible in our dataset; within that, we got the most operating comma-separated file called " test.csv". It contains 2 things, the image methods, and their individual category labels. We will use the pandas' python library to extract the image path with corresponding labels. Next, we want to size our pictures to 30×30 pixels to predict the model and build a numpy array full of image knowledge. To grasp however the model predicts the particular labels, we want to import accuracy_score from the sklearn.metrics. At last, we tend to are line of work the Keras model.save() methodology to stay our trained model.

#testing accuracy on test dataset

```
from sklearn.metrics import accuracy_score
y_test = pd.read_csv('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 =
np.argmax(model.predict(X_test),axis=1)
#Accuracy with the test data
from sklearn.metrics import accuracy_score
print(accuracy_score(labels, pred))
```

Output

```
0.9623911322248614
```

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

GUI for Traffic Signs Classifier

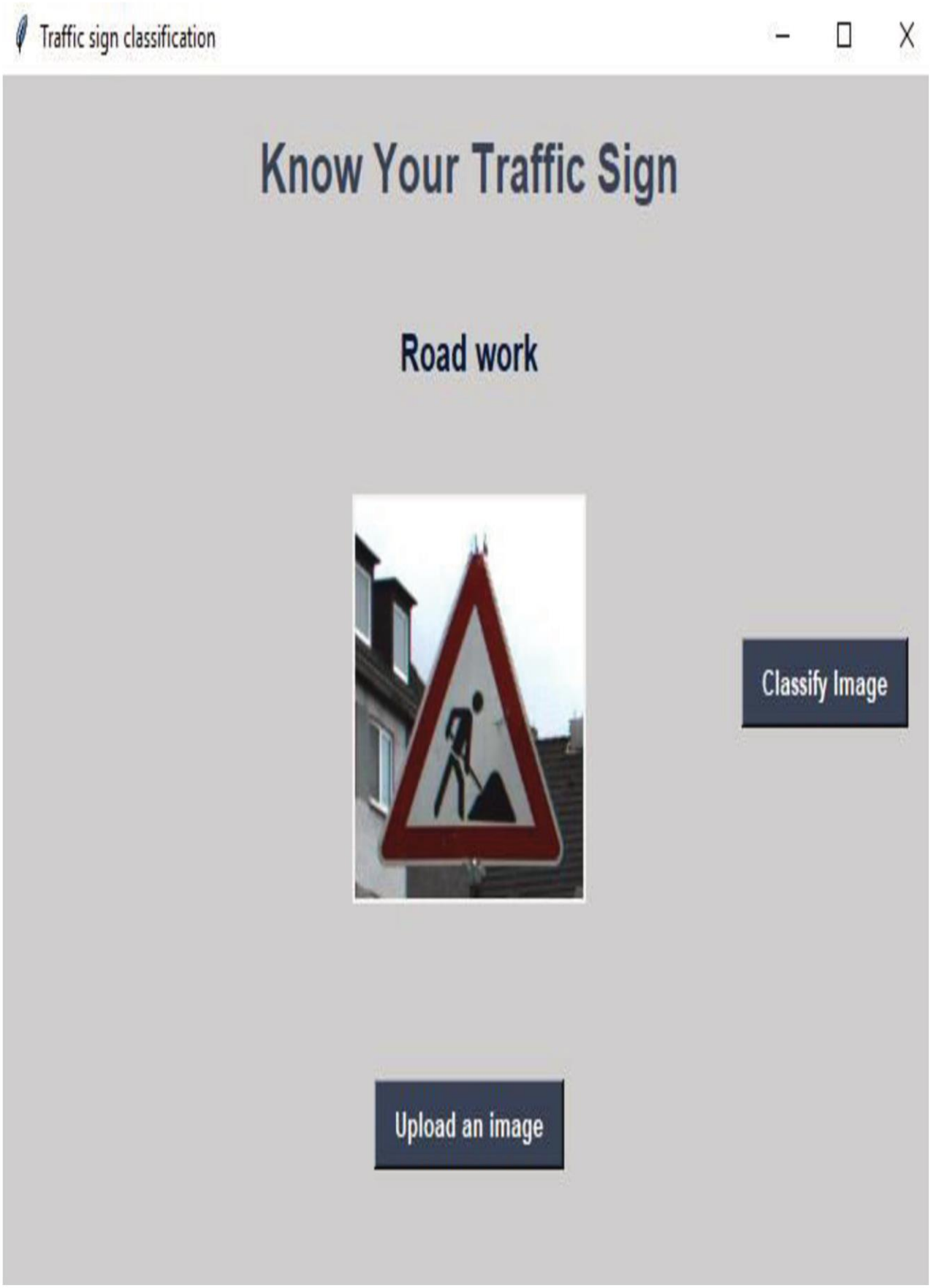
We will use a standard python library called Tkinter to build a graphical user interface(GUI) for our traffic signs recognizer. We need to create a separate python file named" gui.py" for this purpose. Firstly, we need to load our trained model 'traffic_classifier.h5' with the Keras library's help of the deep learning technique. After that, we build the GUI to upload images and a classifier button to determine which class our image belongs. We create classify() function for this purpose; whence we click on the GUI button, this function is called implicitly. To predict the traffic sign, we need to provide the same resolutions of shape we used at the model training time. So, in the classify() method, we convert the image into the dimension of shape (1 * 30 * 30 * 3). The model.predict_classes(image) function is used for image prediction, it returns the class number(0-42) for every image. Then, we can extract the information from the dictionary using this class number.

```
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('traffic_classifier.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',

            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])

    sign = classes[pred+1]

    print(sign)

    label.configure(foreground='#011638
', text=sign)

def show_classify_button(file_path):

    classify_b=Button(top,text="Classif
y Image",command=lambda:
classify(file_path),padx=10,pady=5)

    classify_b.configure(background='#3
64156',
foreground='white',font=('arial',10,'bo
ld'))

    classify_b.place(relx=0.79,rely=0.4
6)

def upload_image():

    try:

        file_path=filedialog.askopenfil
ename()

        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,pad
y=5)

upload.configure(background='#364156',
foreground='white',font=('arial',10,'bo
ld'))

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


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

Conclusion

We have successfully implemented Convolutional Neural Network to Traffic Signal Recognition function greatly more than 90% accuracy on average. We've covered how deep it is reading can be used to distinguish road signs at the top accuracy, use of a variety of pre-processing as well different model viewing and testing techniques properties. We built CNN easy to understand model to see road signs accurately. Our model reached close to 90% accuracy in the test set there it is good to consider the limit of the power of integration once with simple structures. There is still much work to be done to be done, a link that includes modern Advanced Learning programs using the latest and most complex buildings like Google Net or ResNet. But apparently this comes in handy Additional calculation costs, on the other hand.

REFERENCE

Cire¸san, D., Meier, U., Masci, J., & Schmidhuber, J. (2012). Multi-column deep neural network for trac sign classification. Neural Networks, 32 , 333– 338.

Collobert, R., Kavukcuoglu, K., & Farabet, C. (2011). Torch7: A matlab-like environment for machine learning. In BigLearn, NIPS Workshop EPFL- CONF-192376.

https://data-flair.training/blogs/python-project-traffic-signs-recognition/

https://www.javatpoint.com/matplotlib

https://www.javatpoint.com/python-pandas

https://www.geeksforgeeks.org/machine-learning/

https://www.analyticsvidhya.com/blog/2021/12/traffic-signs-recognition-using-cnn-and-keras-in-python/

Y. Xie, L. F Liu, C. H. Li, and Y. Y. Qu. "Unifying visual saliency with HOG feature learning for traffic sign detection." In IEEE Intelligent Vehicles Symposium, , 2009, pp. 24-29. [5]

https://www.javatpoint.com/python-pillow#:~:text=Python%20pillow%20library%20is%20used,or%20create%20new%20images%2C%20etc.

R. Qian, B. Zhang, Y. Yue, Z. Wang, D. Coenen, "Robust Chinese traffic sign detection and recognition with deep convolutional neural network," IEEE 11th International Conference on Natural Computation, pp. 791-796, January 2016.