MINI-PROJECT

***Traffic Signs Recognition using CNN & Keras***

REPORT

\

AIM:

In this project we are implementing the CNN deep network and Keras which can be trained with the best data base known as, German traffic sign recognition benchmark (GTSRB) and the link to the data base is provided to achieve a great efficiency and trained model can be helpful in the automation of vehicles.

INTRODUCTION:

You must have heard about the self-driving cars in which the passenger can fully depend on the car for traveling. But to achieve level 5 autonomous, it is necessary for vehicles to understand and follow all traffic rules.

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 example, we will build a deep neural network model 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.

### The Dataset of Python Project:

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

The link for the data set is (<https://www.kaggle.com/meowmeowmeowmeowmeow/gtsrb-german-traffic-sign>)

The data set is named

# GTSRB - German Traffic Sign Recognition Benchmark

# 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.

### Prerequisites:

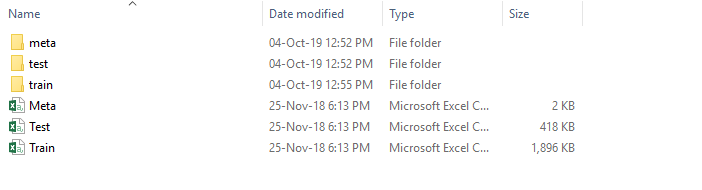
This project requires prior knowledge of Keras, Matplotlib, Scikit-learn, Pandas, PIL and image classification.

To install the necessary packages used for this Python data science project, enter the below command in your terminal:

\*\*\*\*pip install tensorflow keras sklearn matplotlib pandas pil

## STEPS INVOLVED IN THIS PROJECT:

To get started with the project. And in data folder such that you will have a train, test and a meta folder.



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

**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

data = []

labels = []

classes = 43

cur\_path = os.chdir("/content/Train.csv")

cur\_path = os.getcwd()

#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)

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.

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)

#Converting the labels into one hot encoding

y\_train = to\_categorical(y\_train, 43)

y\_test = to\_categorical(y\_test, 43)

From the above code we will get the shape of the data

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

(31367, 30, 30, 3) (7842, 30, 30, 3) (31367,)(7842,)

**Step 2: Build a CNN model:**

To classify the images into their respective categories, we will build a CNN model ([Convolutional Neural Network](https://data-flair.training/blogs/convolutional-neural-networks-tutorial/)). CNN is best for image classification purposes.

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'))

#Compilation of the model

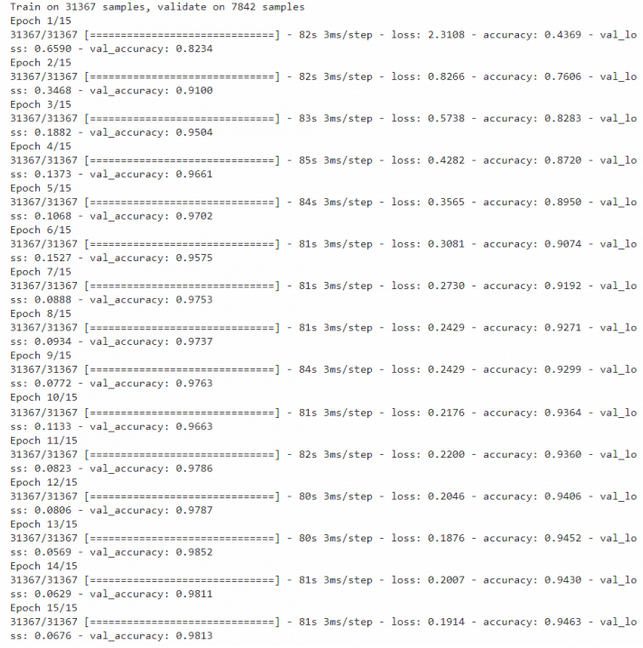
model.compile(loss='categorical\_crossentropy', optimizer='adam', metrics=['accuracy'])

**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))



Our model got a 95% 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')

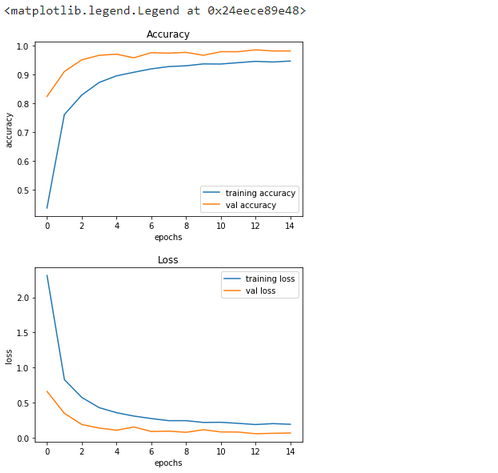
plt.title('Loss')

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.

#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 = model.predict\_classes(X\_test)

#Accuracy with the test data

from sklearn.metrics import accuracy\_score

print(accuracy\_score(labels, pred))

RESULT: The value is accuracy of the above model

**0.9602533650039589**

The above model can be saved as a model using keras.model.save as we have imported it earlier in the initial steps

model.save(‘traffic\_classifier.h5’)

Conclusion:

From the above model saved we can further use it automate the model with the GUI preferred to it as scans the image from the camera and it browse through it and tells the system to understand according to the program that has done to follow up when that situation occurs.