



LOVELY
PROFESSIONAL
UNIVERSITY

TRAFFIC SIGN RECOGNITION

Markanda Meher

11904889

INT 248

Jupyter Notebook:



Model.ipynb

TRAFFIC SIGN

Traffic signs are the salient speakers, preventing drivers and pedestrians from fatal risks on the road. Understanding traffic signs is crucial for optimum safety on the road. These traffic signs communicate the basic rules and regulations of road safety in the form of extremely simple graphics that can be easily understood within seconds. In fact, any person applying for a driving license, for example, needs to be completely aware of all the traffic signs to pass the theoretical exam of a driving test.

Different Types of Traffic Signs:

1) Mandatory Traffic Signs



2) Cautionary Traffic Signs



3) Informatory Traffic Signs



DATASET

Name:

GTSRB - German Traffic Sign Recognition Benchmark

URL:

<https://www.kaggle.com/datasets/meowmeowmeow/meowmeow/gtsrb-german-traffic-sign>


The German Traffic Sign Benchmark is a multi-class, single-image classification challenge held at the International Joint Conference on Neural Networks (IJCNN) 2011.

Properties:

- Single-image, multi-class classification problem
- More than 40 classes
- More than 50,000 images in total
- Large, lifelike database

Data Explorer

Version 1 (314.36 MB)

▸  Meta

▸  Test

 Train

 Meta.csv

 Test.csv

 Train.csv

 Meta.csv

 Test.csv

 Train.csv

- Train Folder contains 43 different folders(labels) of Images for Training and Testing
- Test Folder contains Images for Testing our Model
- Train.csv has Class ID and Path to image for Training Sample
- Test.csv also has Class ID and Path to image for Test Sample

Import required libraries

```
In [1]: 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
os.chdir('D:\Traffic Sign Recognition\Model')
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
```

NumPy: For Numeric Arrays

Pandas: For Data Cleaning

Matplotlib: For Data Visualization

Cv2: For Image Processing

TensorFlow: For Numerical Computation in Deep Learning

Pre-processing the images

```
In [4]: 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 Exception as e:
        print(e)
```

Here we are appending Images in the form of a NumPy array from the Directory after resizing and appending labels for that image.

Dataset Splitting

```
In [9]: X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.2, random_state=0)

In [10]: print(X_train.shape, X_test.shape, y_train.shape, y_test.shape)
(31367, 30, 30, 3) (7842, 30, 30, 3) (31367,) (7842,)
```

Here we are splitting the dataset into Training and Testing where we are taking 20% of the total dataset into Validation Testing using sklearn train_test_split function.

Convert labels to one hot encoding

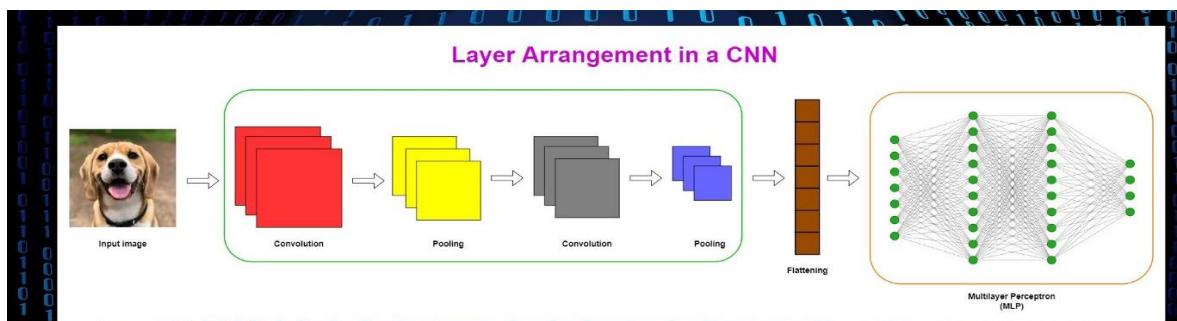
```
In [11]: y_train = to_categorical(y_train, 43)
          y_test = to_categorical(y_test, 43)
```

Keras provides numpy utility library, which provides functions to perform actions on numpy arrays. Using the method to_categorical(), a numpy array (or) a vector which has integers that represent different categories, can be converted into a numpy array (or) a matrix which has binary values and has columns equal to the number of categories in the data.

Model

```
In [12]: 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))
# We have 43 classes that's why we have defined 43 in the dense
model.add(Dense(43, activation='softmax'))
```

A CNN can be instantiated as a Sequential model because each layer has exactly one input and output and is stacked together to form the entire network.



Keras Conv2D is a 2D Convolution Layer, this layer creates a convolution kernel that is wind with layers input which helps produce a tensor of outputs.

Relu activation function Applies the rectified linear unit activation function. With default values, this returns the standard ReLU activation: $\max(x, 0)$, the element-wise maximum of 0 and the input tensor.

Max Pooling is a pooling operation that calculates the maximum value for patches of a feature map, and uses it to create a downsampled (pooled) feature map.

Dropout is a technique where randomly selected neurons are ignored during training. They are “dropped out” randomly. This means that their contribution to the activation of downstream neurons is temporally removed on the forward pass, and any weight updates are not applied to the neuron on the backward pass.

Dense implements the operation: $\text{output} = \text{activation}(\text{dot}(\text{input}, \text{kernel}) + \text{bias})$ where activation is the element-wise activation function passed as the activation argument, kernel is a weights matrix created by the layer, and bias is a bias vector created by the layer (only applicable if use_bias is True).

The softmax function is used as the activation function in the output layer of neural network models that predict a multinomial probability distribution. That is, softmax is used as the activation function for multi-class classification problems where class membership is required on more than two class labels.

Model Compilation

```
In [13]: #Compilation of the model
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
```

Adaptive Moment Estimation is an algorithm for optimization technique for gradient descent. The method is really efficient when working with large problem involving a lot of data or parameters. It requires less memory and is efficient.

Model Training and Checking Accuracy in Validation Data

```
epochs = 20
history = model.fit(X_train, y_train, batch_size=32, epochs=epochs, validation_data=(X_test, y_test))
```

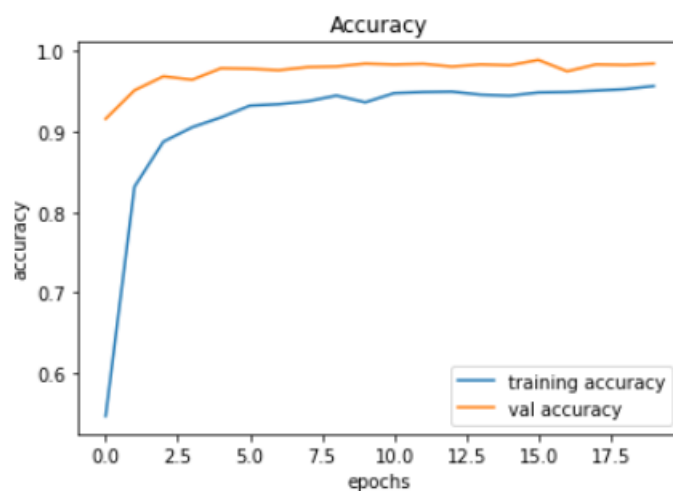


```

981/981 [=====] - 86s 88ms/step - loss: 0.2098 - accuracy: 0.9459 - val_loss: 0.0645 - val_accuracy: 0.9836
Epoch 15/20
981/981 [=====] - 87s 89ms/step - loss: 0.2161 - accuracy: 0.9447 - val_loss: 0.0735 - val_accuracy: 0.9825
Epoch 16/20
981/981 [=====] - 86s 88ms/step - loss: 0.2075 - accuracy: 0.9487 - val_loss: 0.0430 - val_accuracy: 0.9892
Epoch 17/20
981/981 [=====] - 89s 90ms/step - loss: 0.2032 - accuracy: 0.9493 - val_loss: 0.0870 - val_accuracy: 0.9750
Epoch 18/20
981/981 [=====] - 89s 90ms/step - loss: 0.1995 - accuracy: 0.9511 - val_loss: 0.0622 - val_accuracy: 0.9836
Epoch 19/20
981/981 [=====] - 91s 93ms/step - loss: 0.1893 - accuracy: 0.9528 - val_loss: 0.0594 - val_accuracy: 0.9829
Epoch 20/20
981/981 [=====] - 88s 90ms/step - loss: 0.1681 - accuracy: 0.9568 - val_loss: 0.0572 - val_accuracy: 0.9846

```

After training our model, Accuracy on Validation data is 98.46%



Testing Model on Test Data

```

In [17]: def testing(testcsv):
          y_test = pd.read_csv(testcsv)
          label = 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)
          return X_test,label

```

```

In [19]: X_test, label = testing('Test.csv')

```

```

In [23]: predict_x=model.predict(X_test)
          Y_pred=np.argmax(predict_x,axis=1)
          Y_pred

```

```

395/395 [=====] - 14s 35ms/step

```

```

Out[23]: array([16,  1, 38, ...,  6,  7, 10], dtype=int64)

```

Here we are making a function testing where we are taking a csv file and split the csv file into location of images(Path) and labels(ClassId) because we have our Test.csv file in that format, after that we are predicting our output on Test Data.

The `numpy.argmax()` function returns indices of the max element of the array in a particular axis.

Test Accuracy

```
In [24]: from sklearn.metrics import accuracy_score
print(accuracy_score(label, Y_pred))

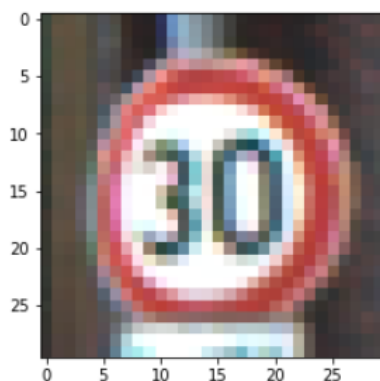
0.9558194774346793
```

Here we can see our Accuracy on the Test Dataset is 95%

Predicting on an Image

```
In [31]: plot, prediction = test_on_img(r'D:\Traffic Sign Recognition\Model\Test\00001.png')
s = [str(i) for i in prediction]
a = int("".join(s))
print("Predicted traffic sign is: ", classes[a])
plt.imshow(plot)
plt.show()

1/1 [=====] - 0s 438ms/step
Predicted traffic sign is: Speed limit (30km/h)
```



Here we can see in the Image 30km/hr is the sign and Our model also predict it Correctly.