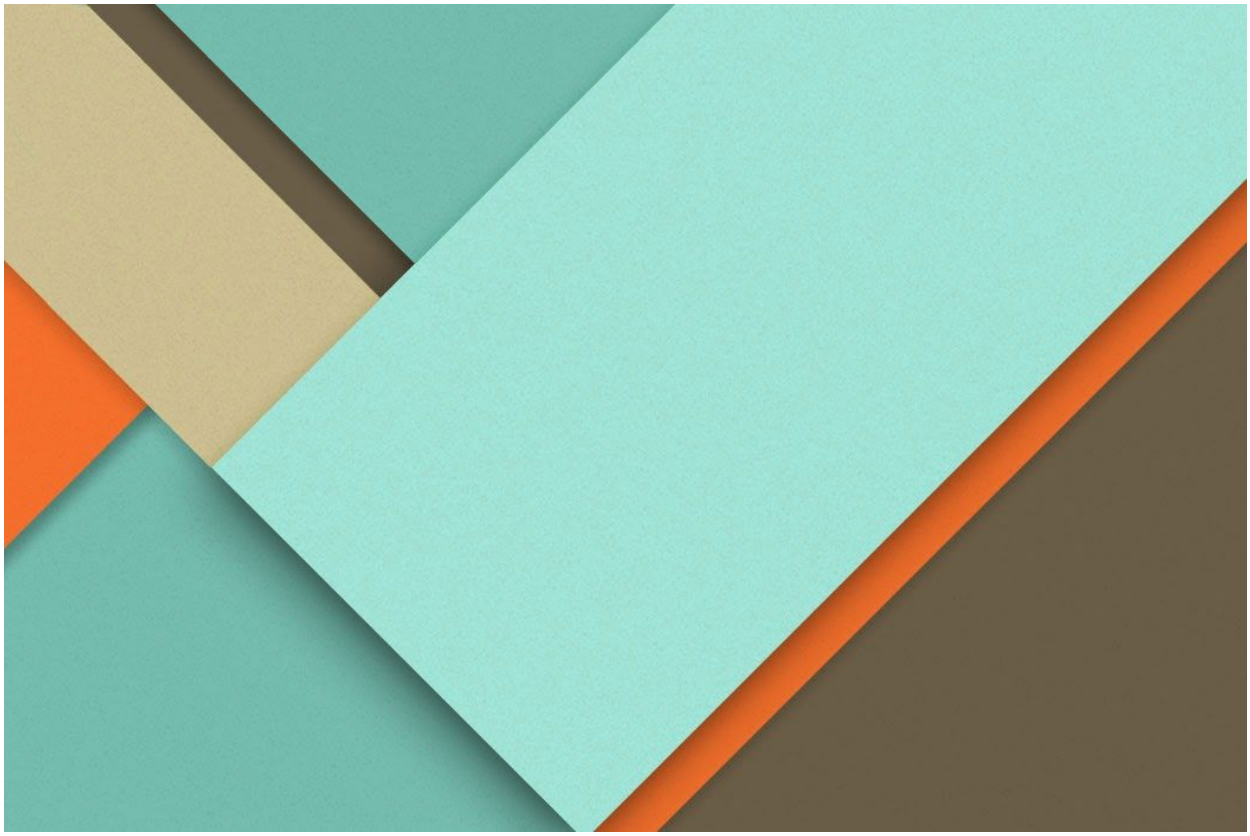


Traffic Sign Recognition Using CNN & Keras

14.08.2020

Divyank Singh

Overview

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.

Goals

1. In this Python project , we 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.

Final Report

For this python project of detecting traffic signals using CNN we achieved an overall accuracy of 95%.

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).

To classify the images into their respective categories, we will build a CNN model (Convolutional Neural Network). 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")

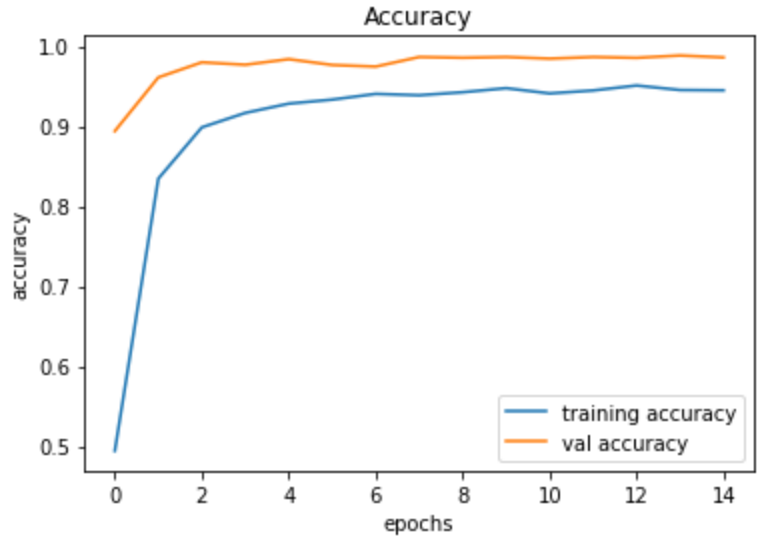
Dropout layer (rate=0.5)

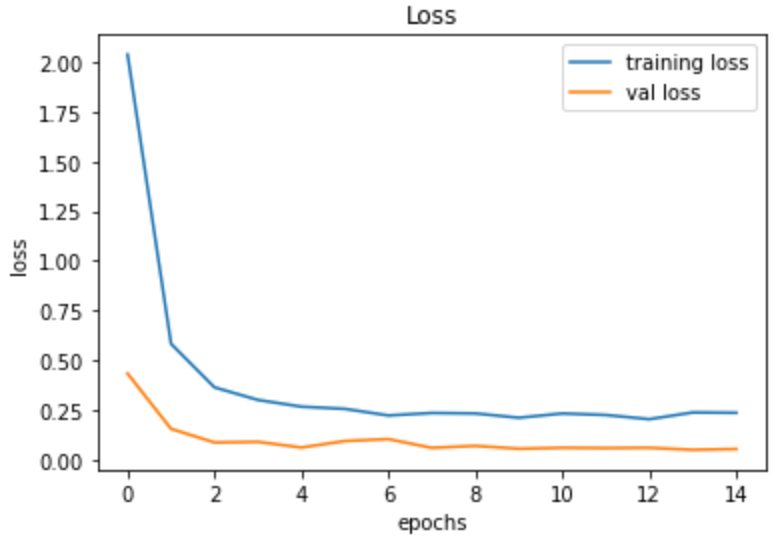
Dense layer (43 nodes, activation="softmax")

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

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.

This is the graph of Accuracy & Loss




Loss
training loss
val loss
loss
epochs
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00
0
2
4
6
8
10
12
14