

Table of Contents

[**Acknowledgment 3**](#_Toc62060664)

[**Introduction 4**](#_Toc62060665)

[**Business Problem Framing 4**](#_Toc62060666)

[**Conceptual Background of the Domain Problem 4**](#_Toc62060667)

[**Motivation for the Problem Undertaken 4**](#_Toc62060668)

[**Analytical Problem Framing 5**](#_Toc62060669)

[**Model/s Development and Evaluation 7**](#_Toc62060670)

[**Run and Evaluate selected models 7**](#_Toc62060671)

[**Key Metrics for success in solving problem under consideration 7**](#_Toc62060672)

[**Visualizations 9**](#_Toc62060673)

[**Conclusion 12**](#_Toc62060674)

[**Learning Outcomes of the Study in respect of Data Science 12**](#_Toc62060675)

## Acknowledgment

Following are the external references which I used:

* [www.w3school.com](http://www.w3school.com)
* [www.stackoverflow.com](http://www.stackoverflow.com)
* [www.google.com](http://www.google.com)
* [www.geeksforgeeks.org](http://www.geeksforgeeks.org)
* <https://www.pyimagesearch.com/2019/11/04/traffic-sign-classification-with-keras-and-deep-learning/>
* <https://medium.com/dataflair/class-data-science-project-for-2020-traffic-signs-recognition-12b09c131742>
* <https://www.google.com/search?q=python-project-traffic-signs-recognition&oq=python-project-traffic-signs-recognition&aqs=chrome..69i57j69i60.2387j0j7&sourceid=chrome&ie=UTF-8>

## Introduction

### Business Problem Framing

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.

### Conceptual Background of the Domain Problem

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 you will use for testing your model.

The ‘train’ folder contains 43 folders each representing a different class. The range of the folder is from 0 to 42. We have to explore the dataset and then build a CNN model. You can use train data for training the model and test the model with the test dataset.

### Motivation for the Problem Undertaken

. Considering the object recognition and interpretation abilities of humans, it is a hard task to try to develop a computer based system which should be able to support people in every day life. There are a lot of conditions which are changing continuously such as luminance and visibility, which are handled by the human recognition system with ease but present serious problems for computer based recognition. Looking at the problem of road and traffic sign recognition shows that the goal is well defined and it seems to be a simple problem. Road signs are located in standard positions and they have standard shapes, standard colours, and their pictograms are known. To see the problem in its full scale, however, a number of parameters that affect the performance of the detection system need to be studied carefully. Road sign images are acquired using a digital camera for the purpose of the current analysis. However, still images captured from a moving camera may suffer from motion blur. Moreover, these images can contain road signs which are partially or totally occluded by other objects such as vehicles or pedestrians. Other problems, such as the presence of objects similar to road signs, such as buildings or billboards, can affect the system and make sign detection difficult. The system should be able to deal with traffic and road signs in a wide range of weather and illumination variant environments such as different seasons, different weather condition e.g. sunny, foggy, rainy and snowy conditions. Different potential difficulties are depicted in one section of this chapter. Using the system in different countries can make the problem even worse. Different countries use different colours and different pictograms. The system should also be adaptive, which means it should allow continuous learning otherwise the training should be repeated for every country. To deal with all these constraints, road sign recognition should be provided with a large number of sign examples to allow the system to respond correctly when a traffic sign is encountered.

### Analytical Problem Framing

#### Data Sources and their formats

I got the data from the flip robo,. The images are the in the format of .png

#### Data Pre-processing Done

In the pre-processing stage, we'll prepare the data to be fed to the Keras model. The first step is clearing the dataset of null values. Then, we'll use one-hot encoding to convert categorical variables to numerical variables. Neural Nets work with numerical data, not categorical. We’ll also split the data into training validatio and testing set. Finally, we'll scale the data/standardize it so that it ranges from -1 to 1. This standardization helps both train the model better and allows it to converge easier.

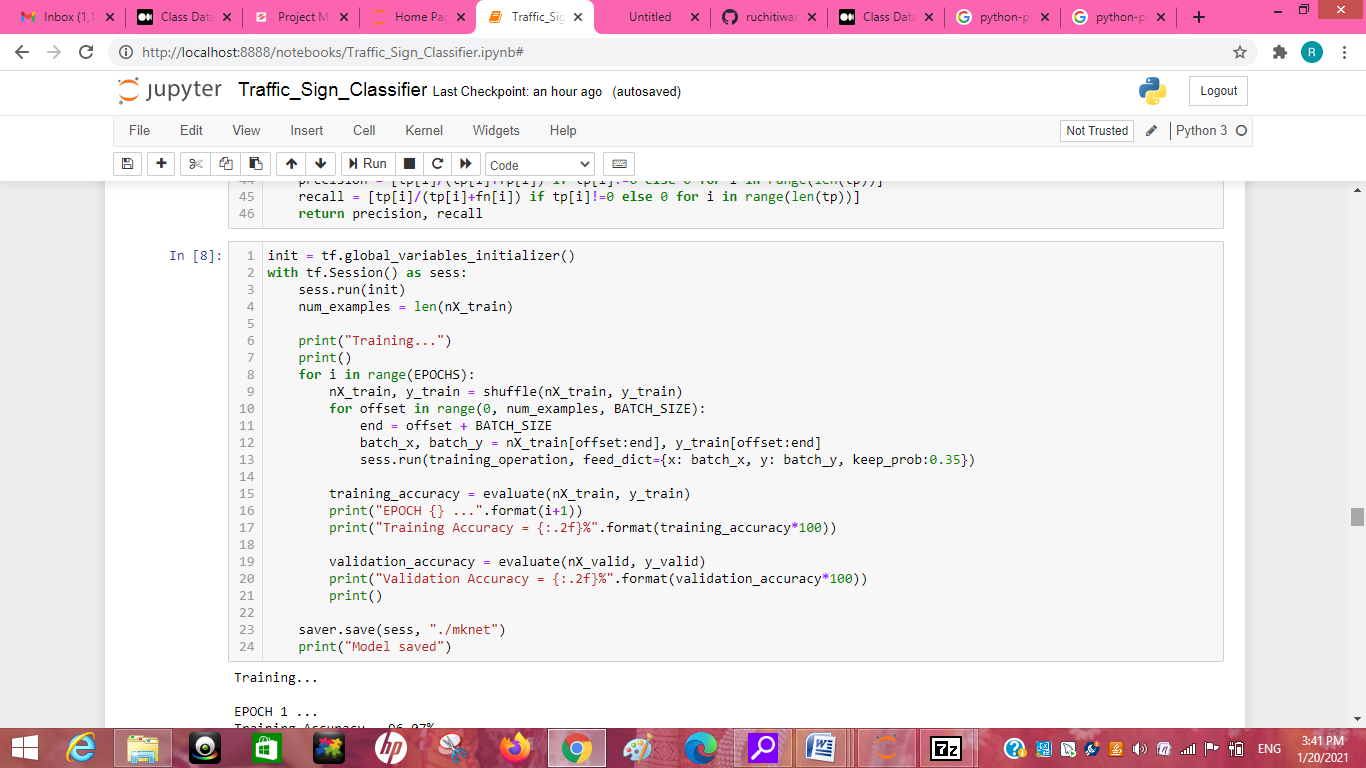
#### Hardware and Software Requirements and Tools Used

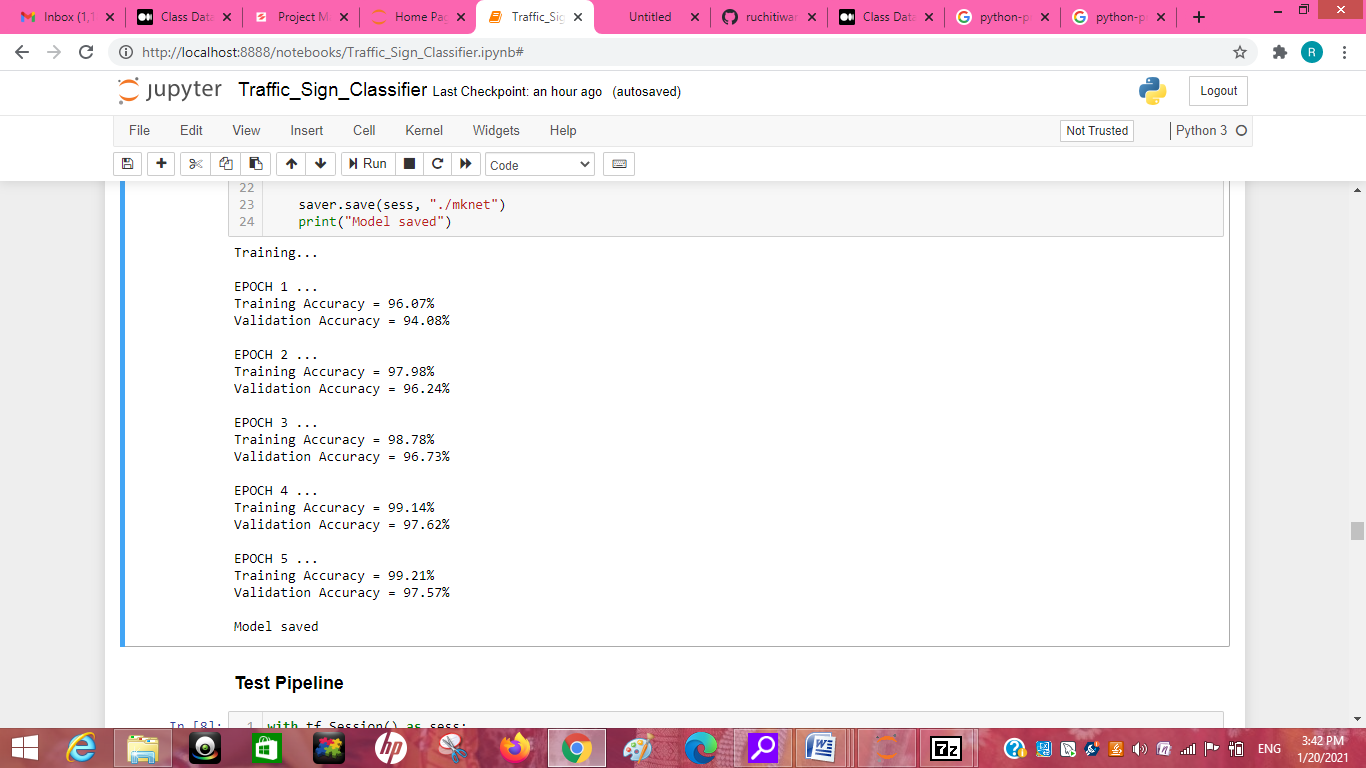
* Hardware – Laptop
* Software - anaconda jupyter notebook
* Libraries- numpy, pandas, seaborn, matplotlib.pyplot,warnings,
* Import requests:- Requests will allow you to send HTTP/1.1 requests using Python. With it, you can add content like headers, form data, multipart files, and parameters via simple Python libraries. It also allows you to access the response data of Python in the same way.
* Import Tensorflow:- Tensorflow is an open-source library developed by Google primarily for deep learning applications. It also supports traditional machine learning. TensorFlow was originally developed for large numerical computations without keeping deep learning in mind.
* Import random: - random imports the random module, which contains a variety of things to do with random number generation. Among these is the random () function, which generates random numbers between 0 and 1. Doing the import this way this requires you to use the syntax random.

* Import cv2:- OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library. OpenCV was built to provide a common infrastructure for computer vision applications and to accelerate the use of machine perception in the commercial products.
* Import os :- The OS module in python provides functions for interacting with the operating system. OS, comes under Python's standard utility modules. This module provides a portable way of using operating system dependent functionality. Path\* modules include many functions to interact with the file system.
* Import tqdm: - loading a page or doing a transaction, it always eases your mind whenever you see that small progress bar giving you an estimation of how long the process would take to complete or render. If you have a simple progress bar in your script or code, it looks very pleasing to the eye and gives proper feedback to the user whenever he executes the code. You can use the Python external library tqdm, to create simple & hassle-free progress bars which you can add in your code and make it look lively
* Import pickle:-Pickle is used for serializing and de-serializing Python object structures, also called marshalling or flattening. Serialization refers to the process of converting an object in memory to a byte stream that can be stored on disk or sent over a network
* Import time:- The time() function returns the number of seconds passed since epoch.

## Model/s Development and Evaluation

### Run and Evaluate selected models





### Key Metrics for success in solving problem under consideration

Tensor Board is a handy application that allows you to view aspects of our model,

The way that we use Tensor Board with Keras is via a Keras callback. There are actually quite a few Keras callbacks, and you can make your own. Definitely check the others out: [Keras Callbacks](https://www.tensorflow.org/api_docs/python/tf/keras/callbacks/TensorBoard). For example, Model Checkpoint is another useful one. For now, however, we're going to be focused on the Tensor Board callback.

The Convolution Neural Network gained popularity through its use with image data, and is currently the state of the art for detecting what an image is, or what is contained in the image.

The basic CNN structure is as follows: Convolution -> Pooling -> Convolution -> Pooling -> Fully Connected Layer -> Output

Convolution is the act of taking the original data, and creating feature maps from it.Pooling is down-sampling, most often in the form of "max-pooling," where we select a region, and then take the maximum value in that region, and that becomes the new value for the entire region. Fully Connected Layers are typical neural networks, where all nodes are "fully connected." The convolution layers are not fully connected like a traditional neural network.

**directory**: Directory where the data is located. If labels is "inferred", it should contain subdirectories, each containing images for a class. Otherwise, the directory structure is ignored.

**labels**: Either "inferred" (labels are generated from the directory structure), or a list/tuple of integer labels of the same size as the number of image files found in the directory. Labels should be sorted according to the alphanumeric order of the image file paths (obtained via os.walk(directory) in Python).

**label\_mode**: - 'int': means that the labels are encoded as integers (e.g. for sparse\_categorical\_crossentropy loss). - 'categorical' means that the labels are encoded as a categorical vector (e.g. for categorical\_crossentropy loss). - 'binary' means that the labels (there can be only 2) are encoded as float32 scalars with values 0 or 1 (e.g. for binary\_crossentropy). - None (no labels).

**class\_names**: Only valid if "labels" is "inferred". This is the explict list of class names (must match names of subdirectories). Used to control the order of the classes (otherwise alphanumerical order is used).

**color\_mode**: One of "grayscale", "rgb", "rgba". Default: "rgb". Whether the images will be converted to have 1, 3, or 4 channels.

**batch\_size**: Size of the batches of data. Default: 32 or 36.

**image\_size**: Size to resize images to after they are read from disk. Defaults to (256, 256). Since the pipeline processes batches of images that must all have the same size, this must be provided.

**shuffle**: Whether to shuffle the data. Default: True. If set to False, sorts the data in alphanumeric order.

**seed**: Optional random seed for shuffling and transformations.

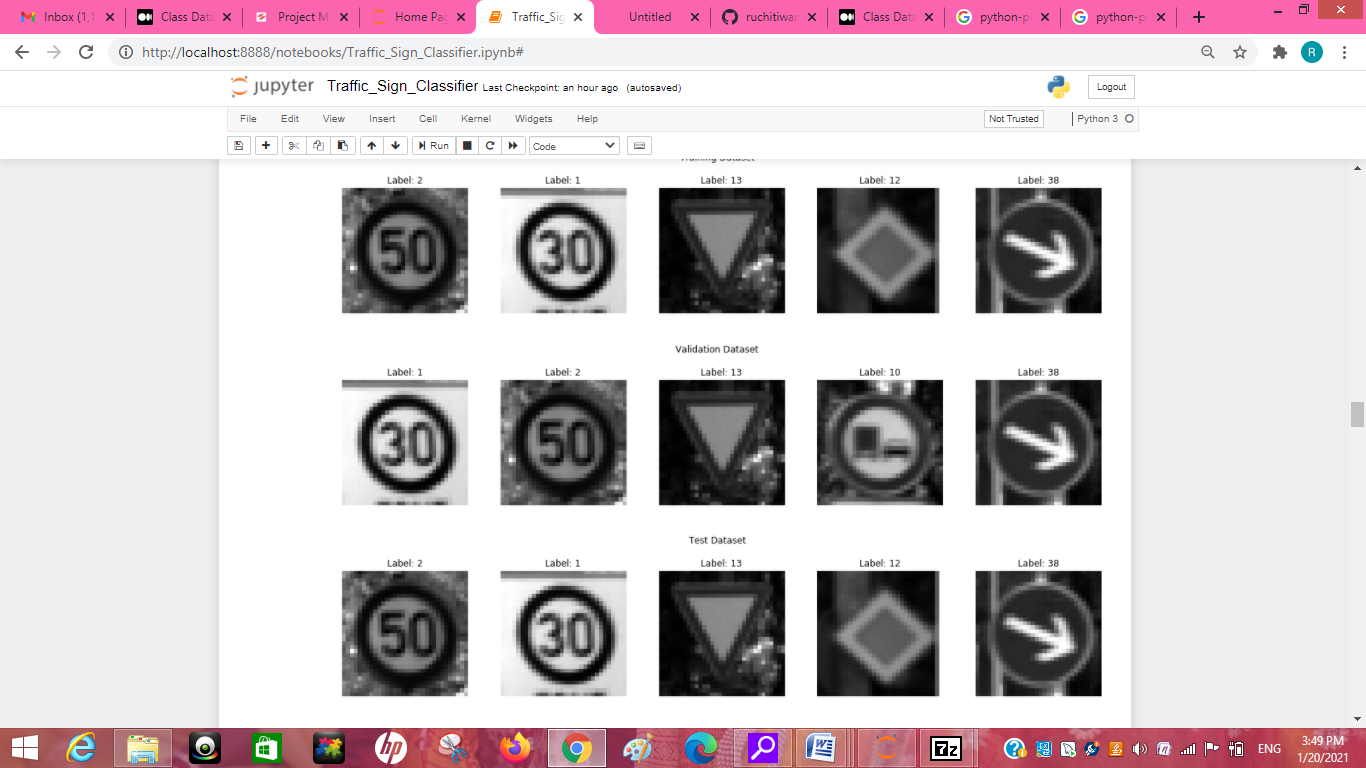
**validation\_split**: Optional float between 0 and 1, fraction of data to reserve for validation.

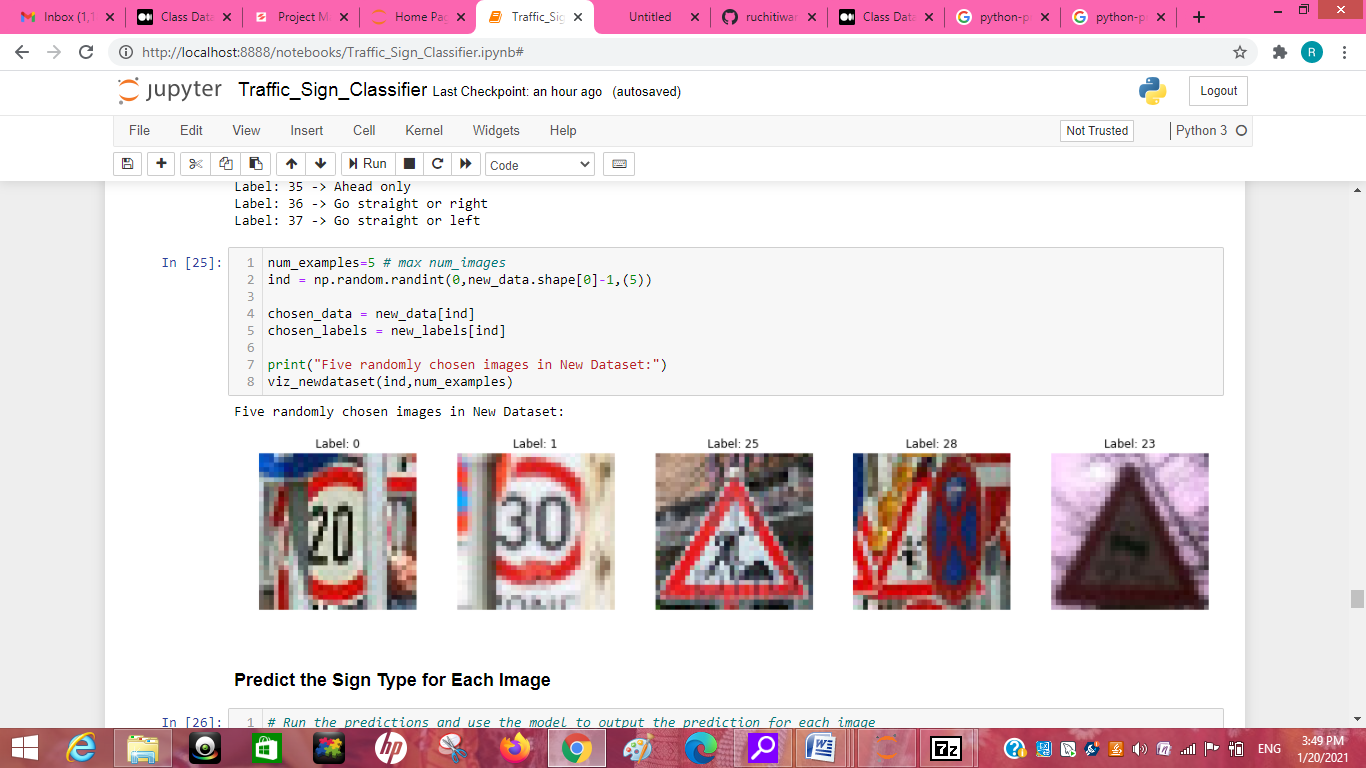
**subset**: One of "training" or "validation". Only used if validation\_split is set.

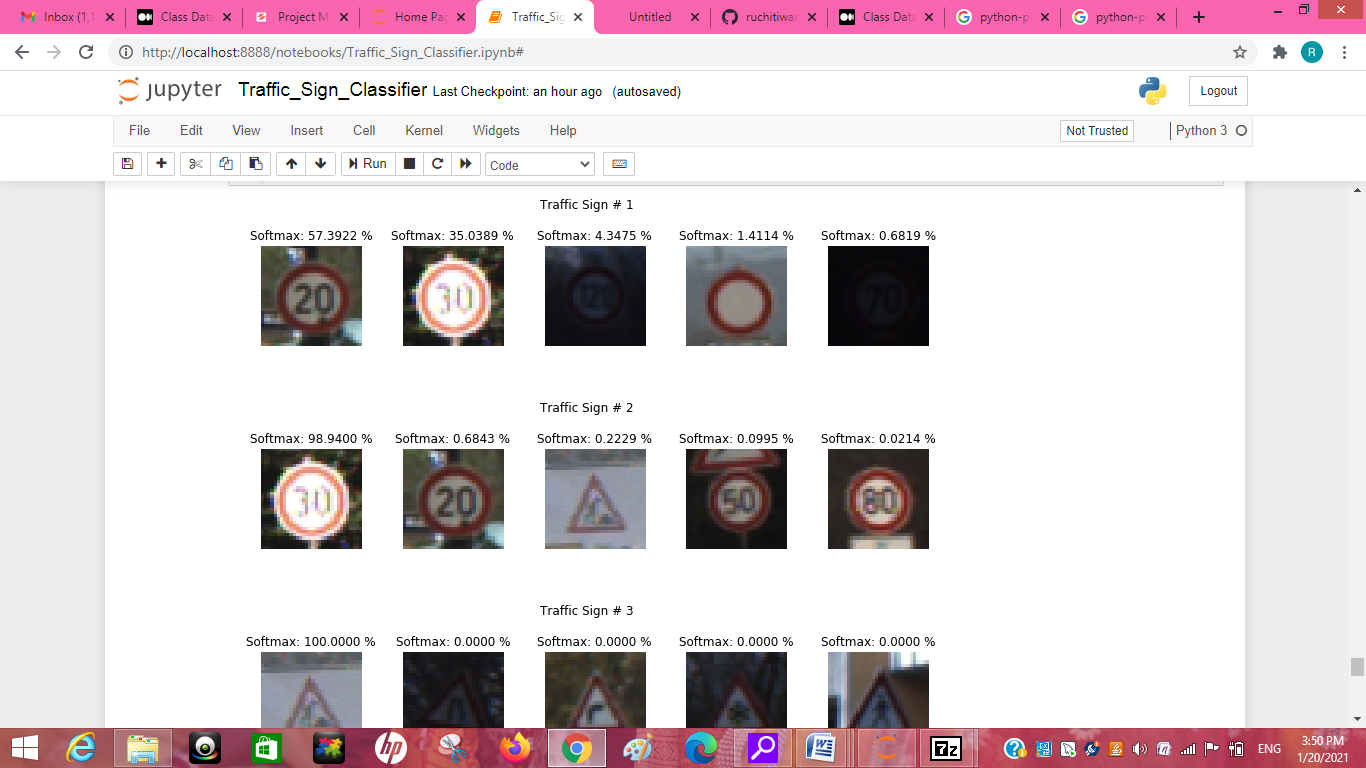
**interpolation**: String, the interpolation method used when resizing images. Defaults to bilinear. Supports bilinear, nearest, bicubic, area, lanczos3, lanczos5, gaussian, mitchellcubic.

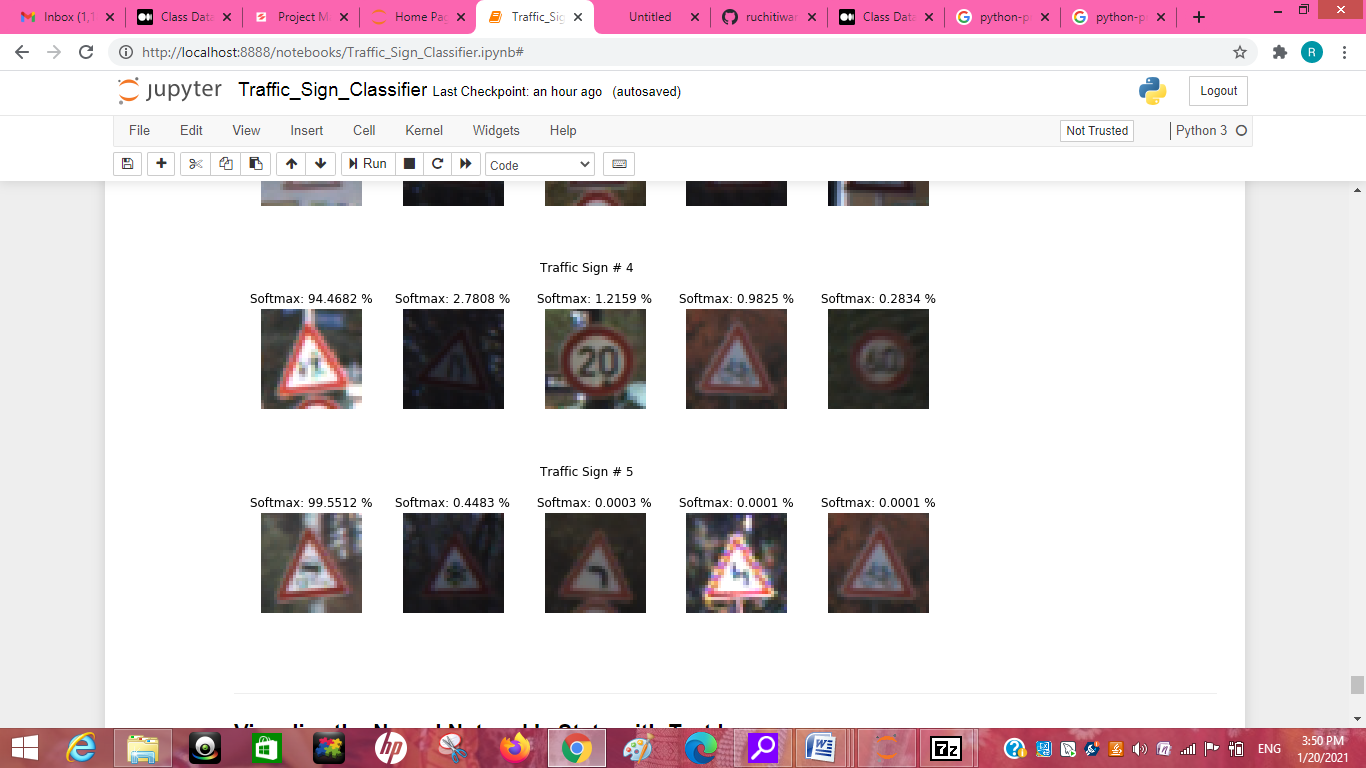
**follow\_links**: Whether to visits subdirectories pointed to by symlinks. Defaults to False.

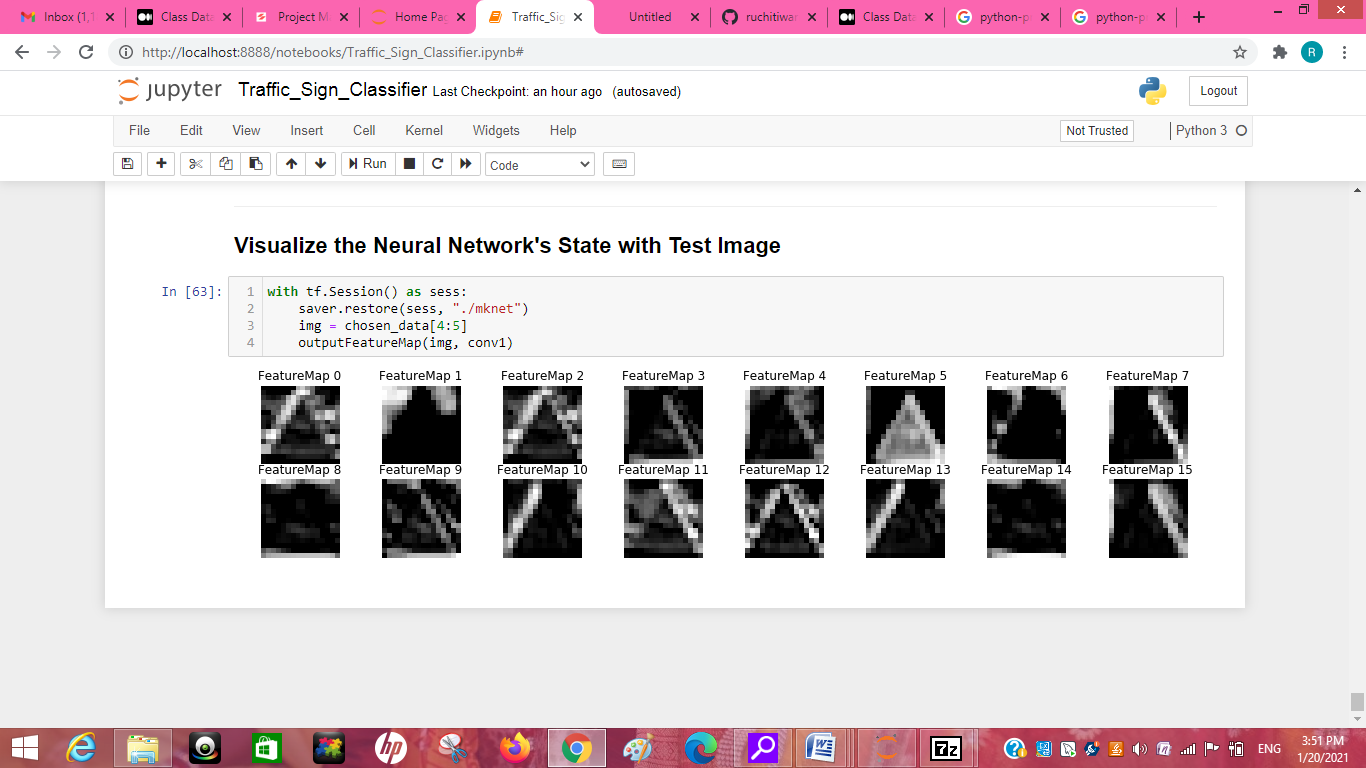
### Visualizations











## Conclusion

### Learning Outcomes of the Study in respect of Data Science

In this Python project with source code, we have successfully classified the traffic signs classifier with 95% accuracy and also visualized how our accuracy and loss changes with time, which is pretty good from a simple CNN model.

Our dataset contains a test folder , 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. 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.