**18CS3074P-Deep Learning**

**PROJECT BASED REPORT**

**ON**

**Traffic Sign Prediction Using CNN**

**submitted in partial fulfillment of the requirement for the award of the degree of**

**BACHELOR OF TECHNOLOGY**

**In**

**COMPUTER SCIENCE AND ENGINEERING**

**By**

**P. Sampath Kumar (180030562)**

**A. Sreeja (180030572)**

**P. Shiva (180030646)**

Under the Esteemed Guidance of

******

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**K L (Deemed to be) University**

Green Fields, Vaddeswaram, Guntur District – 522 502

**(2020-2021)**

**INTRODUCTION:**

**ABOUT THE PROJECT:**

* Traffic Sign Classification is an import task for self-driving cars.

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. The main objective of our project is to design and construct a computer-based system which can automatically detect the road signs so as to provide assistance to the user or the machine so that they can take appropriate actions. The proposed approach consists of building a model using convolutional neural networks by extracting traffic signs from an image using color information. We have used convolutional neural networks (CNN) to classify the traffic sign and we used color-based segmentation to extract/crop signs from images.

**AIM OF THE PROJECT:**

To Predict Traffic Sign using Convolutional Neural Network

**SOFTWARE REQUIRED:**

1.Python

2.Jupyter Notebook

**IMPLEMENTATION:**

The implementation has 6 parts in it.,

->Importing Necessary Libraris

->Loading and Formatting Data

->Pre processing the data.

->Dividing into Training, validation and Testing sets.

->Applying One Hot Encoding

->Applying CNN model.

-> Validating the model

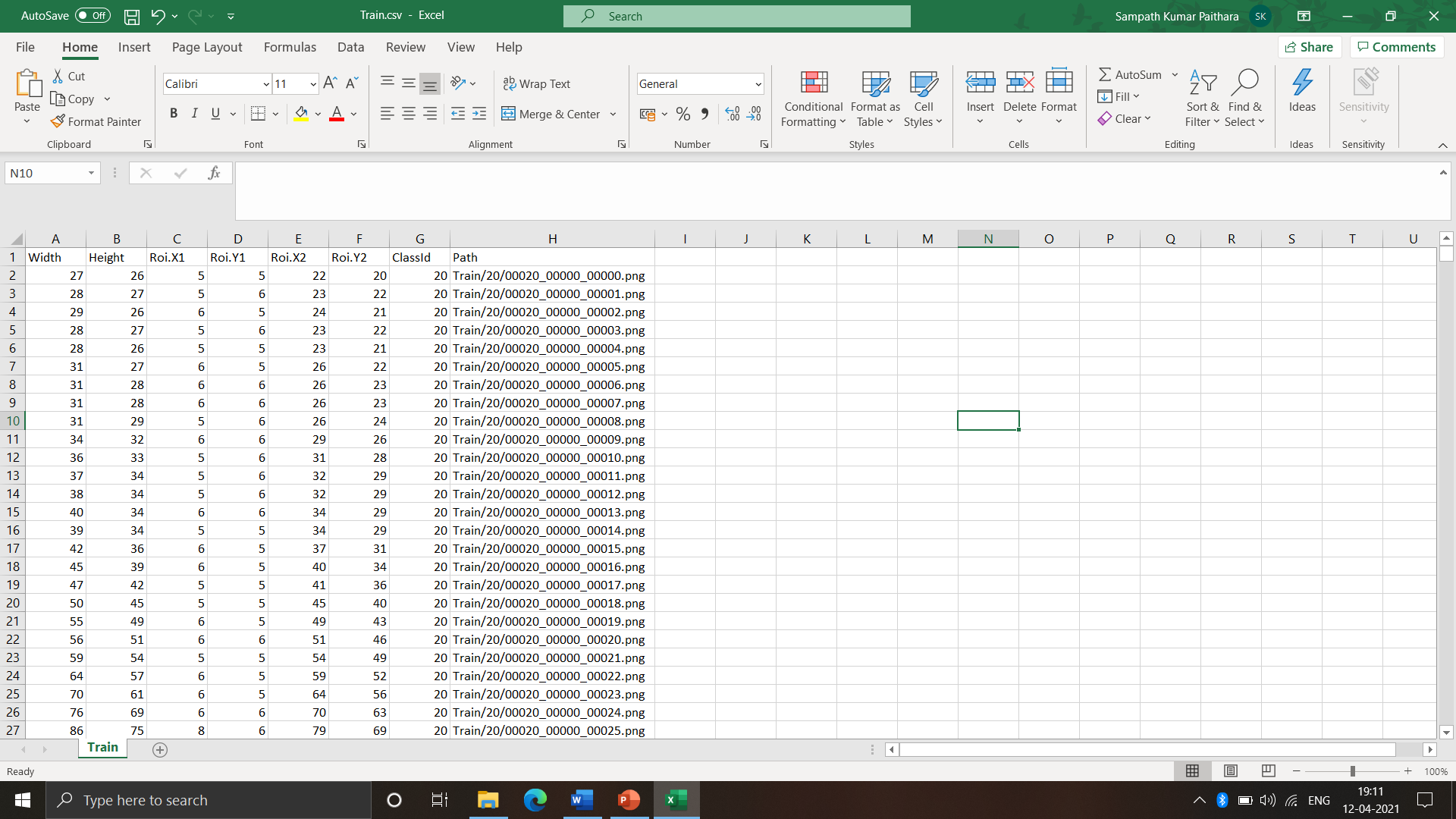
->Testing the model

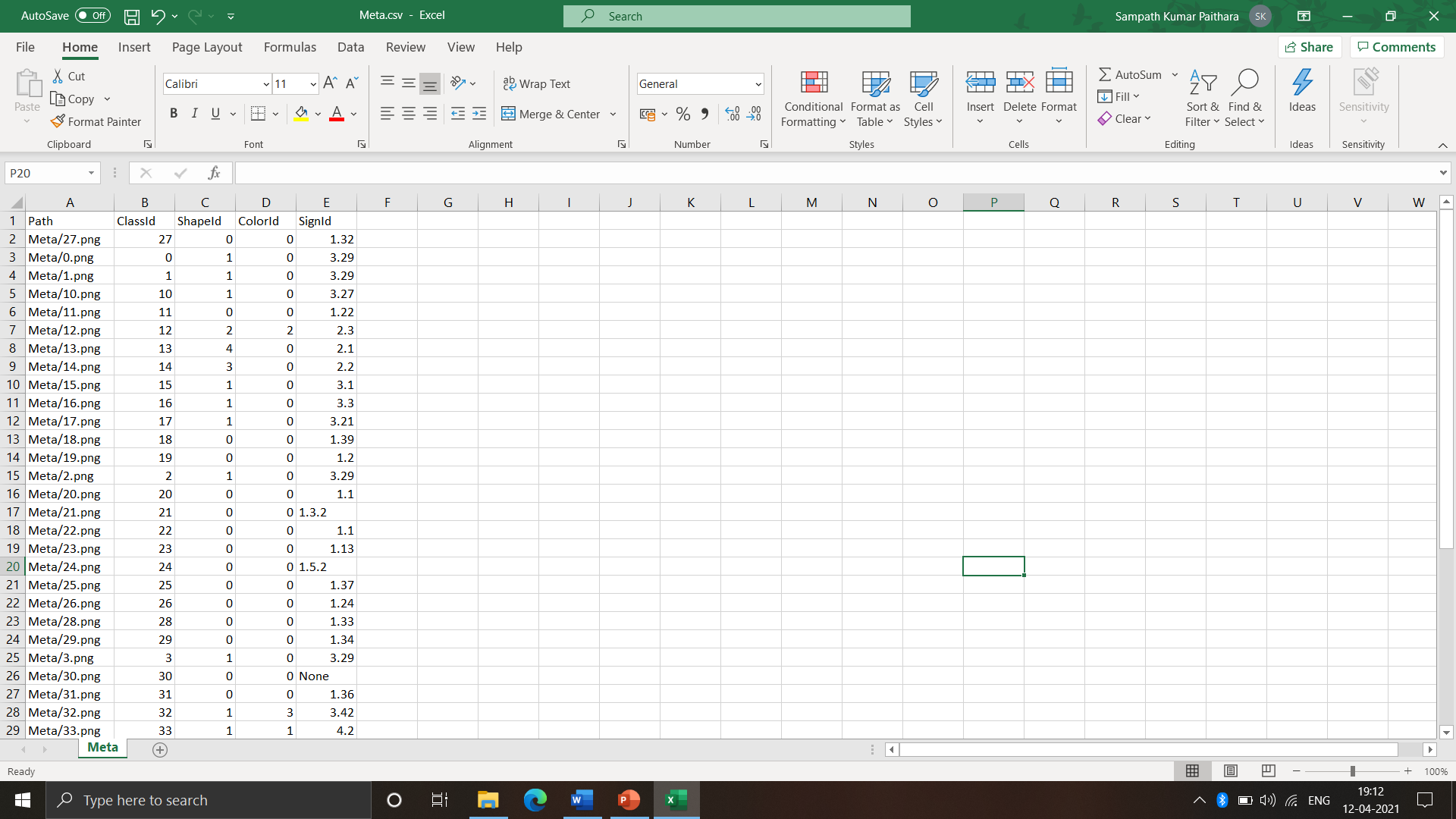
->Predicting Image

->Prediction of test data

**1]Dataset Selection**

Collected it from Kaggle. It has 30,000 Images of 43 different classes





**2]** **Data Preprocessing**

-Removed unnecessary columns

-Filled Null values

-Removed Noisy Data

-Removed improper values

**4]Splitting the Data**

-Separating data into training and testing sets is an important part of evaluating data mining models. ... By using similar data for training and testing, you can minimize the effects of data discrepancies and better understand the characteristics of the model. **i) Training :-**

Training data is the data you use to train an algorithm or machine learning model to predict the outcome you design your model to predict **ii) Validation:-**

Validation data is used to validate the data trained.

**iii) Testing:-**

Test data is used to measure the performance, such as accuracy or efficiency, of the algorithm you are using to train the machine.

**5]Applying One Hot Encoding**

 Data is shuffled so that some of the training model be shuffles and added to testing data. This increases the performance of the model and

**6]Applying CNN:**

Convolutional model is used to train the model. It has 5 layers from which we sipped the Dropout layer which drops some neuron to avoid overfitting.

**7]Validating the data:**

Trained data is validated.

**8] Predicting the tested model**

After testing the data. Model is tested and checked for accuracy.

**CODE:**

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import tensorflow as tf

import keras

import cv2

from PIL import Image

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score

import time

import os

import warnings

warnings.filterwarnings("ignore")df = pd.read\_csv("project.csv")

start = time.time()

input\_path = "C:/Users/paith/Project/"

image\_data = []

image\_labels = []

# Number of total classes

total\_classes = 43

# Dimensions of our images

height = 32

width = 32

channels = 3

# Load the images from the correct path

for i in range(total\_classes):

path = input\_path + "Train/" + str(i)

images = os.listdir(path)

for img in images:

try:

image = cv2.imread(path + '/' + img)

image\_fromarray = Image.fromarray(image, "RGB")

resize\_image = image\_fromarray.resize((height, width))

image\_data.append(np.array(resize\_image))

image\_labels.append(i)

except:

print("Error in Image loading")

# Converting lists into numpy arrays

image\_data = np.array(image\_data)

image\_labels = np.array(image\_labels)

# Time taken to load our images in seconds

end = time.time()

print("Time taken: ", round(end-start, 5), "seconds")

# Shuffling data

shuffle\_indexes = np.arange(image\_data.shape[0])

np.random.shuffle(shuffle\_indexes)

image\_data = image\_data[shuffle\_indexes]

image\_labels = image\_labels[shuffle\_indexes]

# Splitting training and testing dataset

X\_train, X\_valid, y\_train, y\_valid = train\_test\_split(image\_data, image\_labels, test\_size=0.2,

random\_state=2666, shuffle=True)

# Scale the values between 0 and 1

X\_train = X\_train / 255

X\_valid = X\_valid / 255

# The dimensions concur

print("X\_train.shape", X\_train.shape)

print("X\_valid.shape", X\_valid.shape)

print("y\_train.shape", y\_train.shape)

print("y\_valid.shape", y\_valid.shape)

# Converting the labels into one hot encoding

y\_train = keras.utils.to\_categorical(y\_train, total\_classes)

y\_valid = keras.utils.to\_categorical(y\_valid, total\_classes)

# The dimensions concur

print(y\_train.shape)

print(y\_valid.shape)

keras.backend.clear\_session() # Clearing previous session if there was any

np.random.seed(2666)

# Create our model with Keras is straightforward

model = keras.models.Sequential([

keras.layers.Conv2D(filters=18, kernel\_size=(5,5), strides=1, activation="relu",

input\_shape=(height, width, channels)),

keras.layers.MaxPooling2D(pool\_size=(2,2), strides=2),

keras.layers.Conv2D(filters=36, kernel\_size=(5,5), strides=1, activation="relu"),

keras.layers.MaxPooling2D(pool\_size=(2,2), strides=2),

keras.layers.Conv2D(filters=36, kernel\_size=(5, 5), activation="relu"),

keras.layers.MaxPooling2D(pool\_size=(1,1)),

keras.layers.Flatten(),

keras.layers.Dense(units=72, activation="relu"),

keras.layers.Dense(units=43, activation="softmax"),

])

model.summary()

# Compilation of our model

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

epochs = 5

history = model.fit(X\_train, y\_train, batch\_size=32, epochs=epochs, validation\_data=(X\_valid, y\_valid))

validation\_data = (X\_valid, y\_valid)

pd.DataFrame(history.history).plot(figsize=(8, 5))

plt.grid(True)

plt.gca().set\_ylim(0, 1)

plt.show()

# Testing accuracy on the reserved test set

test = pd.read\_csv(input\_path + "/Test.csv")

labels = test["ClassId"].values

test\_imgs = test["Path"].values

# How an image looks like

img\_index = 25

image = Image.open(input\_path + test\_imgs[img\_index])

img = image.resize((height,width))

img = np.array(img) / 255.

img = img.reshape(1, height, width, channels)

print(img.shape)

print(labels[img\_index])

plt.imshow(image)

# Dictionary to map classes.

classes = {

0:'Speed limit (20km/h)',

1:'Speed limit (30km/h)',

2:'Speed limit (50km/h)',

3:'Speed limit (60km/h)',

4:'Speed limit (70km/h)',

5:'Speed limit (80km/h)',

6:'End of speed limit (80km/h)',

7:'Speed limit (100km/h)',

8:'Speed limit (120km/h)',

9:'No passing',

10:'No passing veh over 3.5 tons',

11:'Right-of-way at intersection',

12:'Priority road',

13:'Yield',

14:'Stop',

15:'No vehicles',

16:'Veh > 3.5 tons prohibited',

17:'No entry',

18:'General caution',

19:'Dangerous curve left',

20:'Dangerous curve right',

21:'Double curve',

22:'Bumpy road',

23:'Slippery road',

24:'Road narrows on the right',

25:'Road work',

26:'Traffic signals',

27:'Pedestrians',

28:'Children crossing',

29:'Bicycles crossing',

30:'Beware of ice/snow',

31:'Wild animals crossing',

32:'End speed + passing limits',

33:'Turn right ahead',

34:'Turn left ahead',

35:'Ahead only',

36:'Go straight or right',

37:'Go straight or left',

38:'Keep right',

39:'Keep left',

40:'Roundabout mandatory',

41:'End of no passing',

42:'End no passing veh > 3.5 tons'

}

# Prediction of this image

pred = model.predict\_classes(img)[0]

print(pred)

sign = classes[pred]

print(sign)

# Load and preprocess test set

start = time.time()

test = pd.read\_csv(input\_path + 'Test.csv')

labels = test["ClassId"].values

imgs = test["Path"].values

data = []

for img in imgs:

try:

image = cv2.imread(input\_path + img)

image\_fromarray = Image.fromarray(image, 'RGB')

resize\_image = image\_fromarray.resize((height, width))

data.append(np.array(resize\_image))

except:

print("Error")

X\_test = np.array(data)

X\_test = X\_test / 255

# Prediction of test set

pred = model.predict\_classes(X\_test)

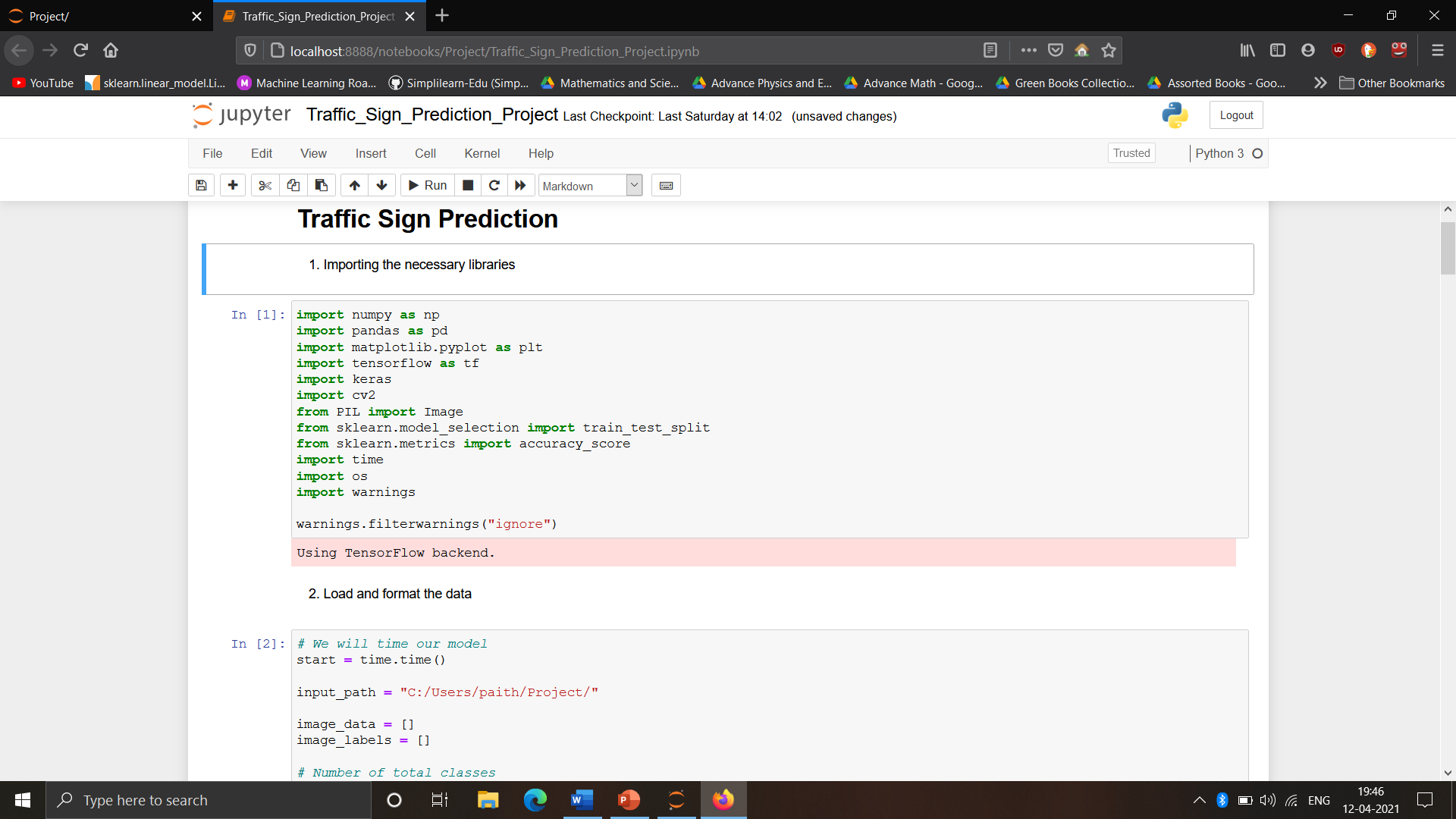
#Accuracy with the test data

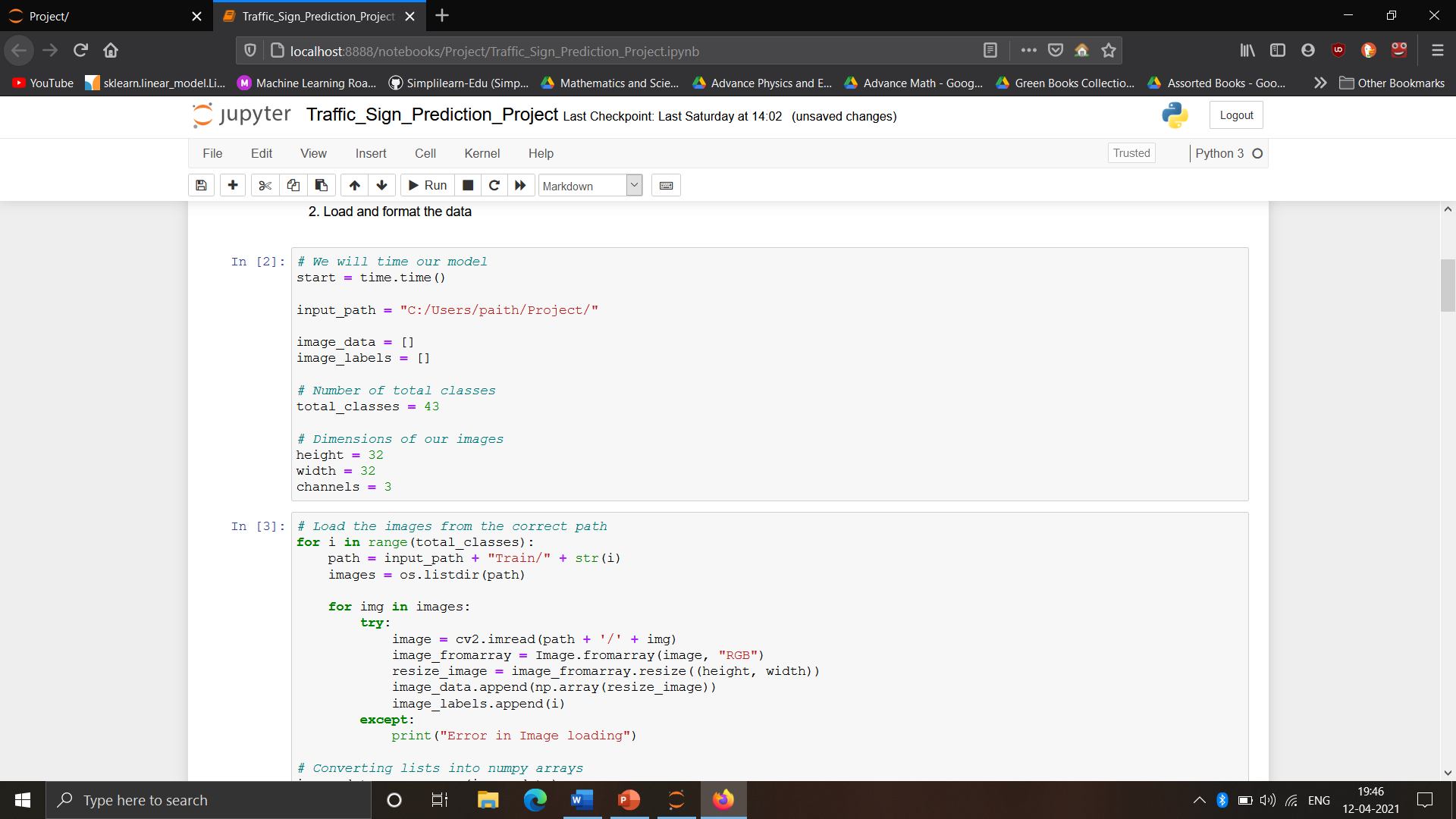
print(accuracy\_score(labels, pred))

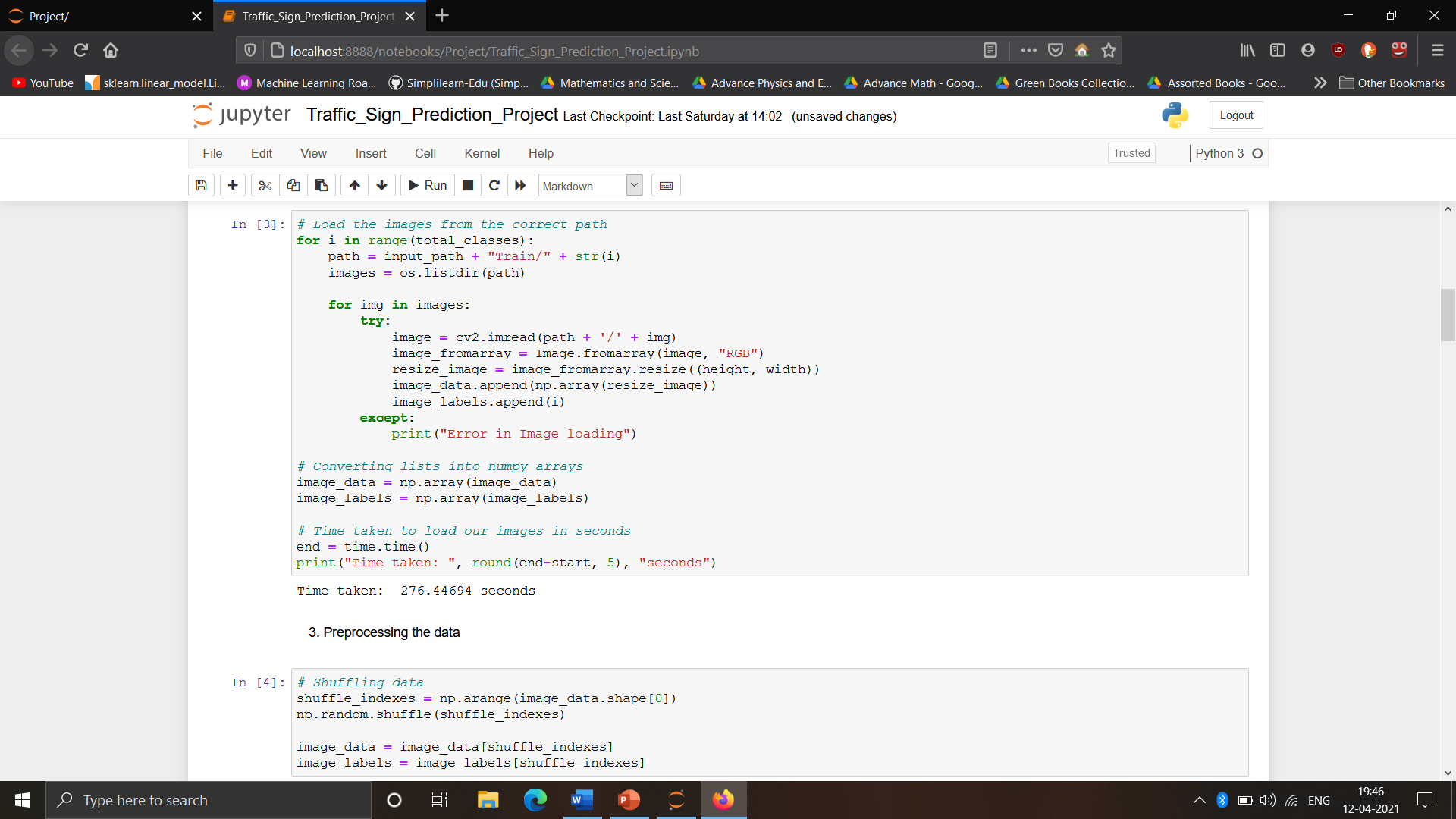
end = time.time()

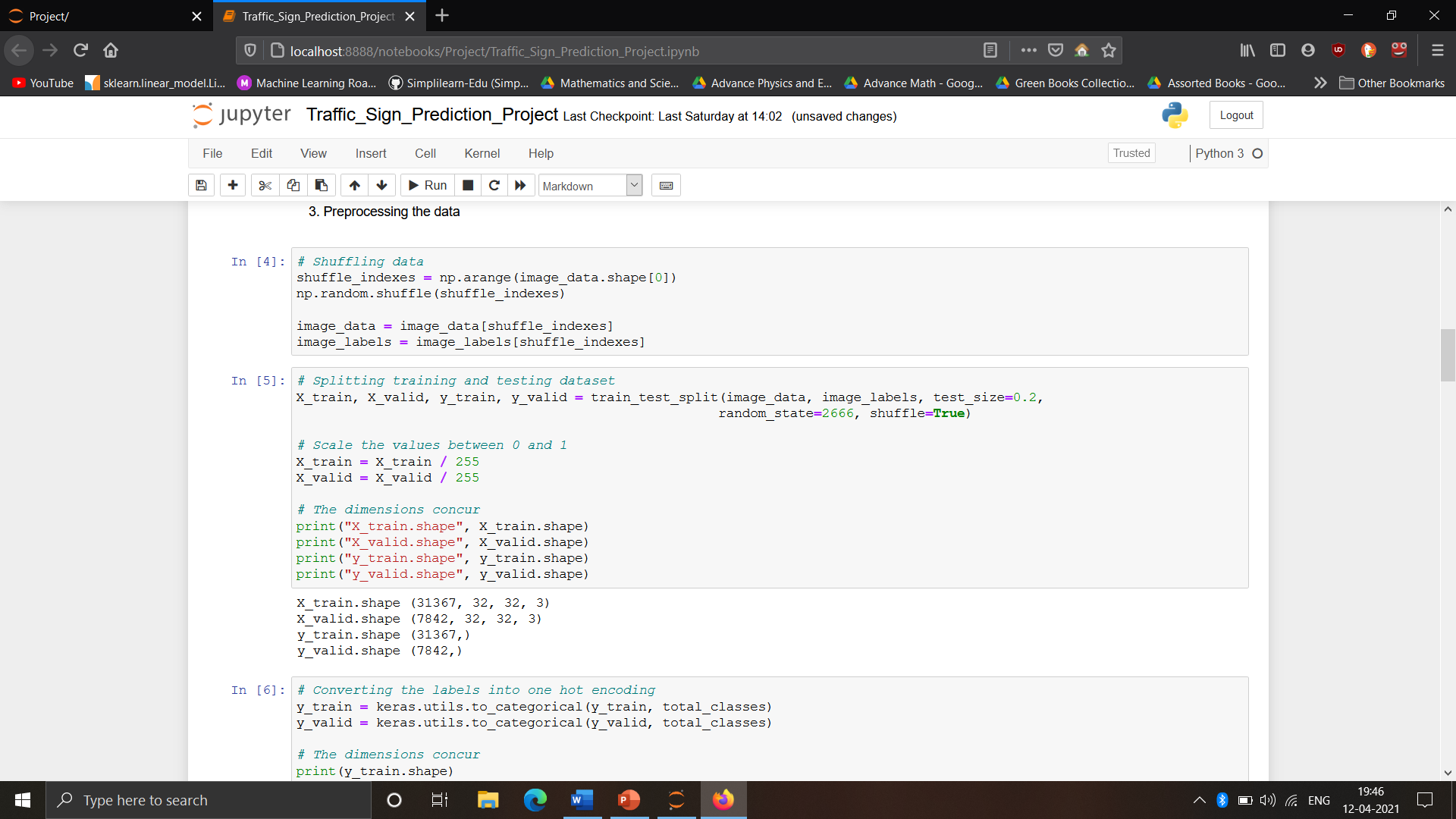
print("Time taken: ", round(end-start,5), "seconds")

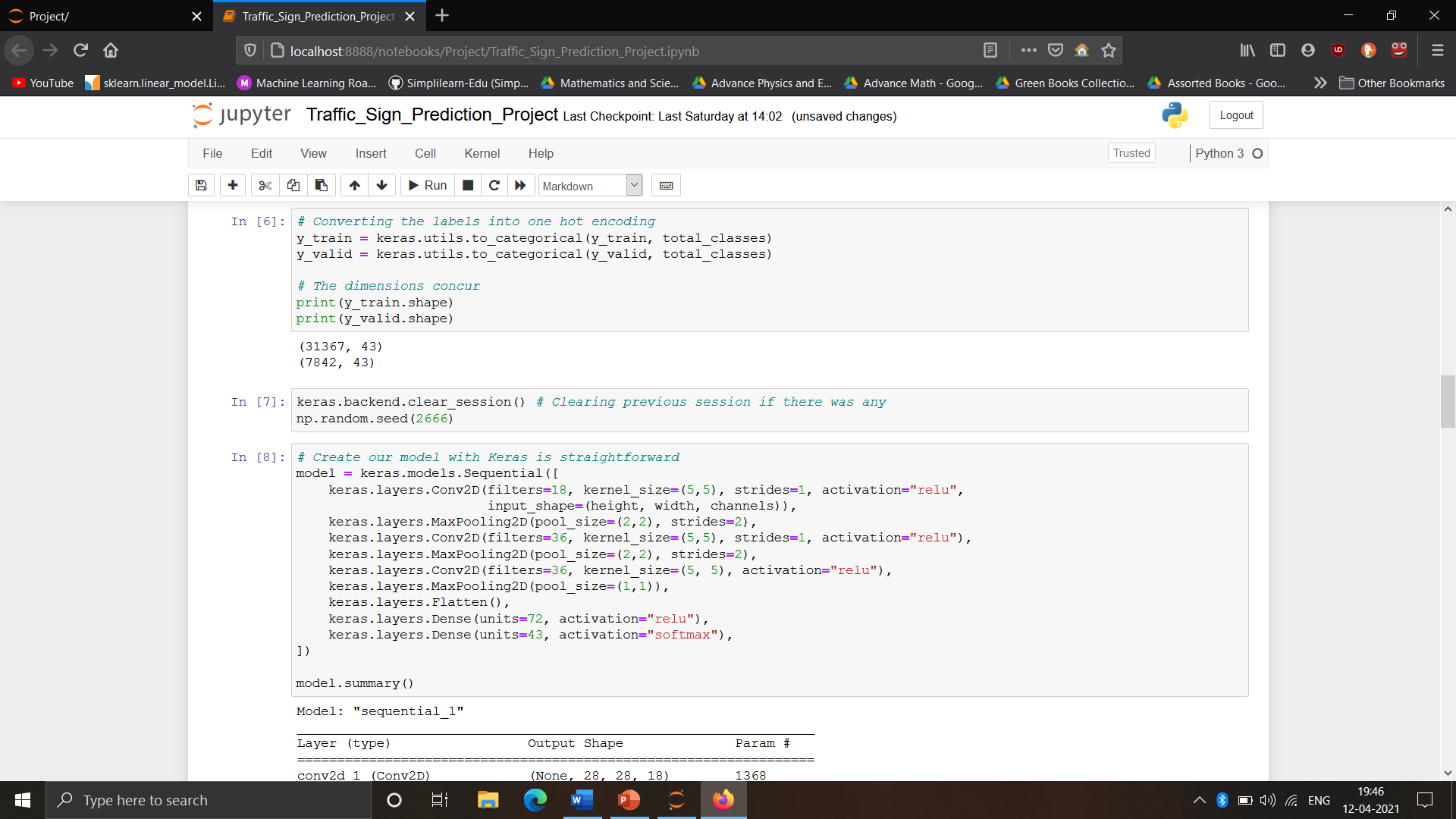
**Screenshots:**

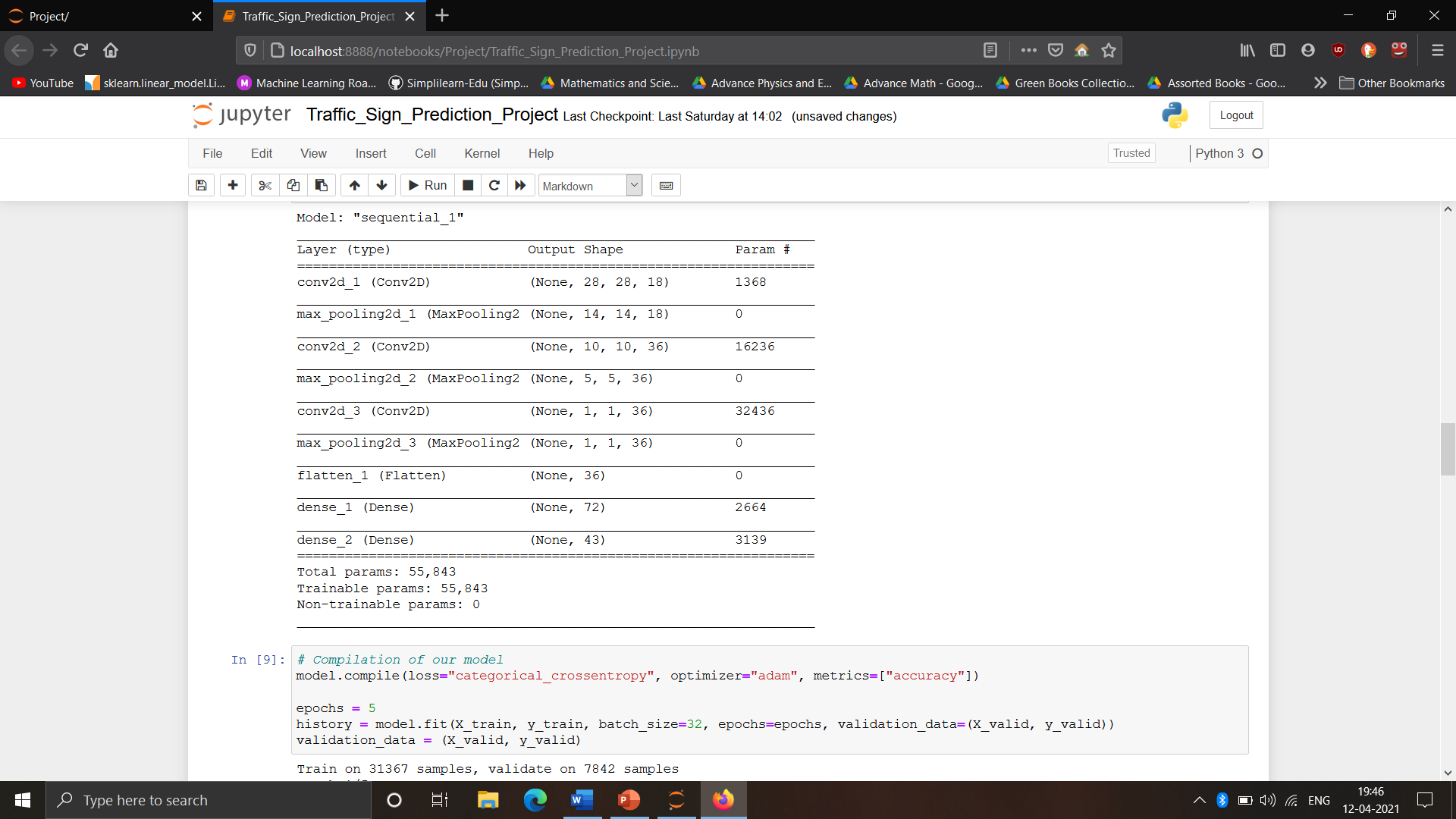


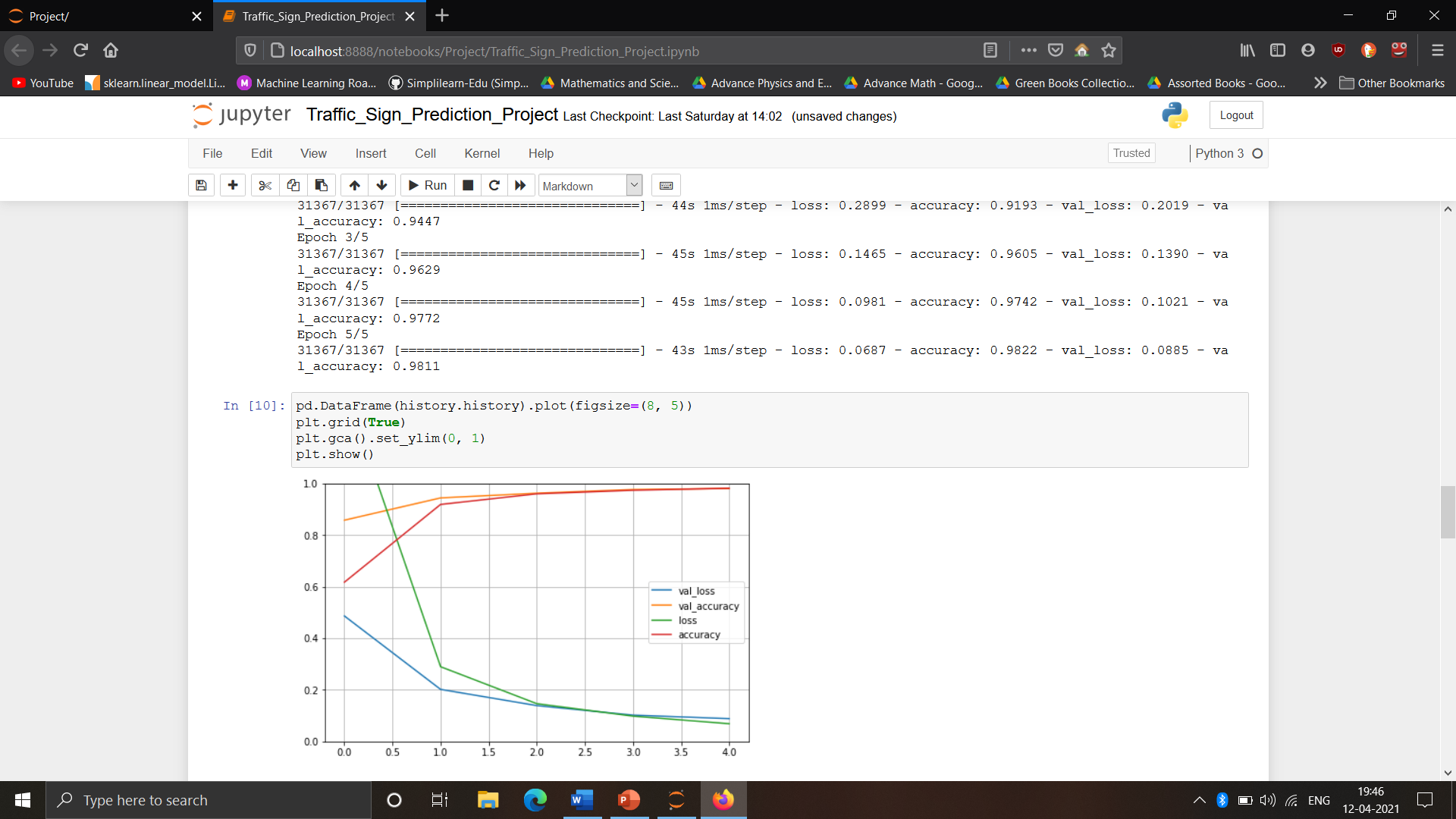


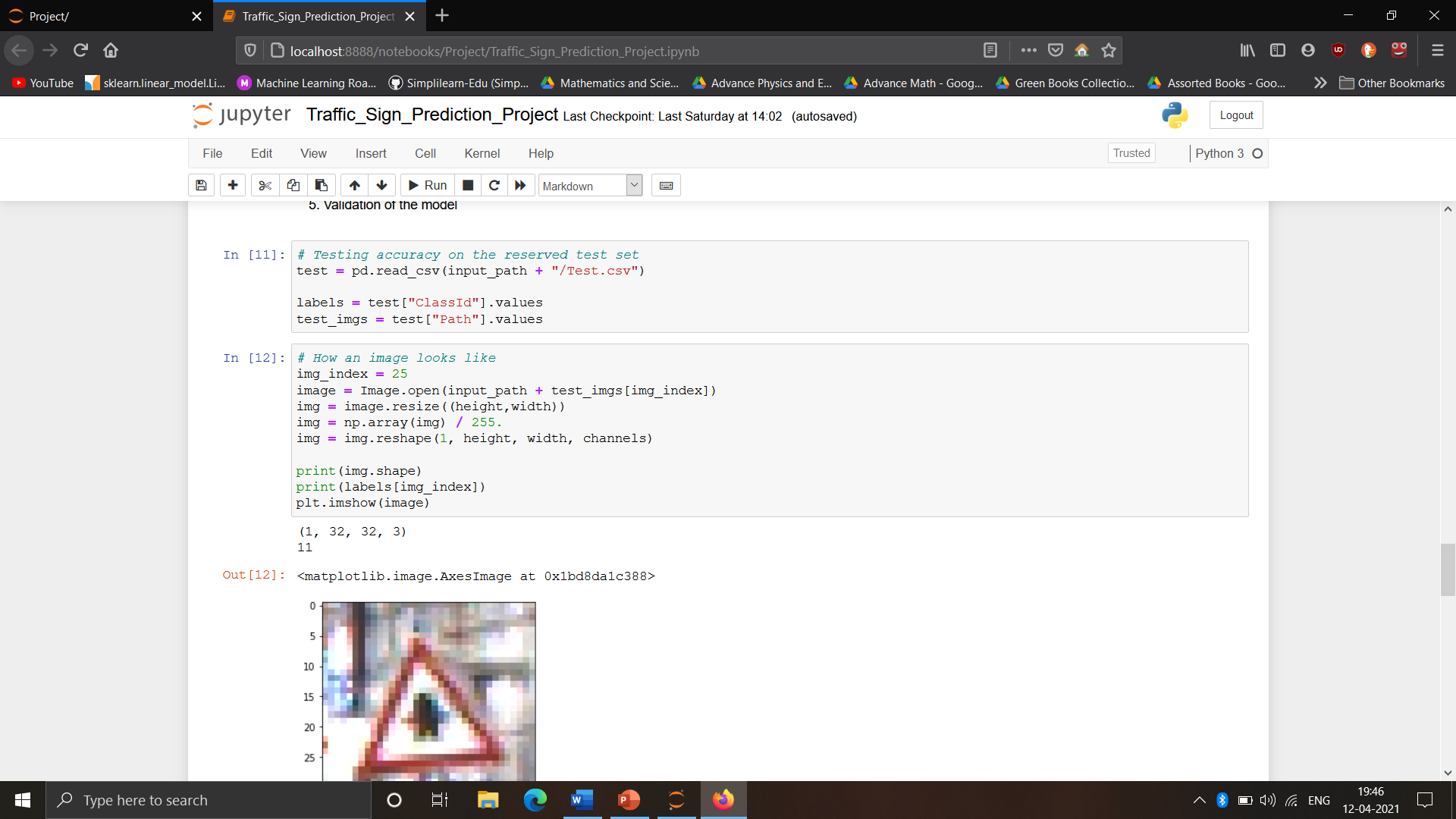


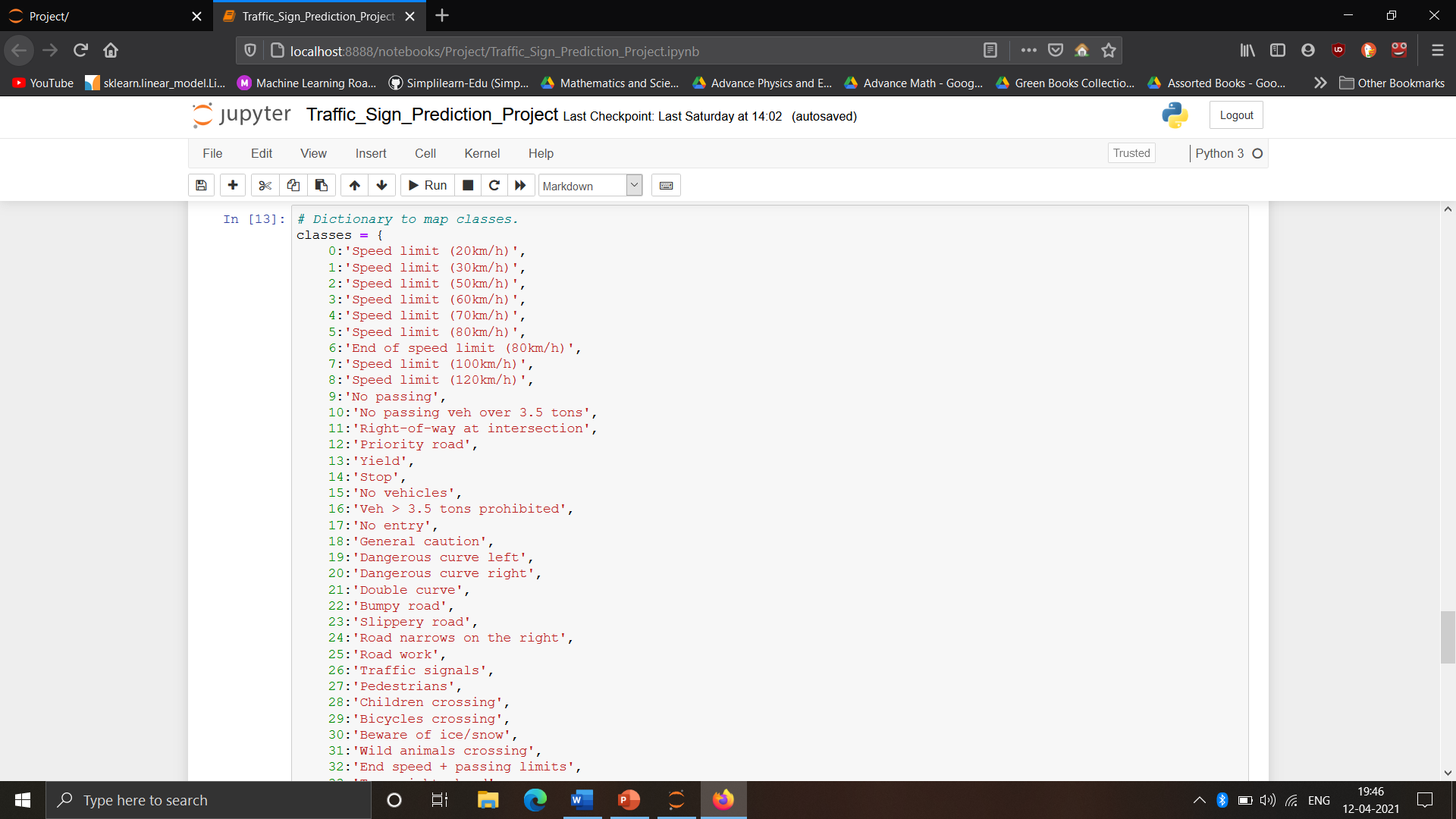


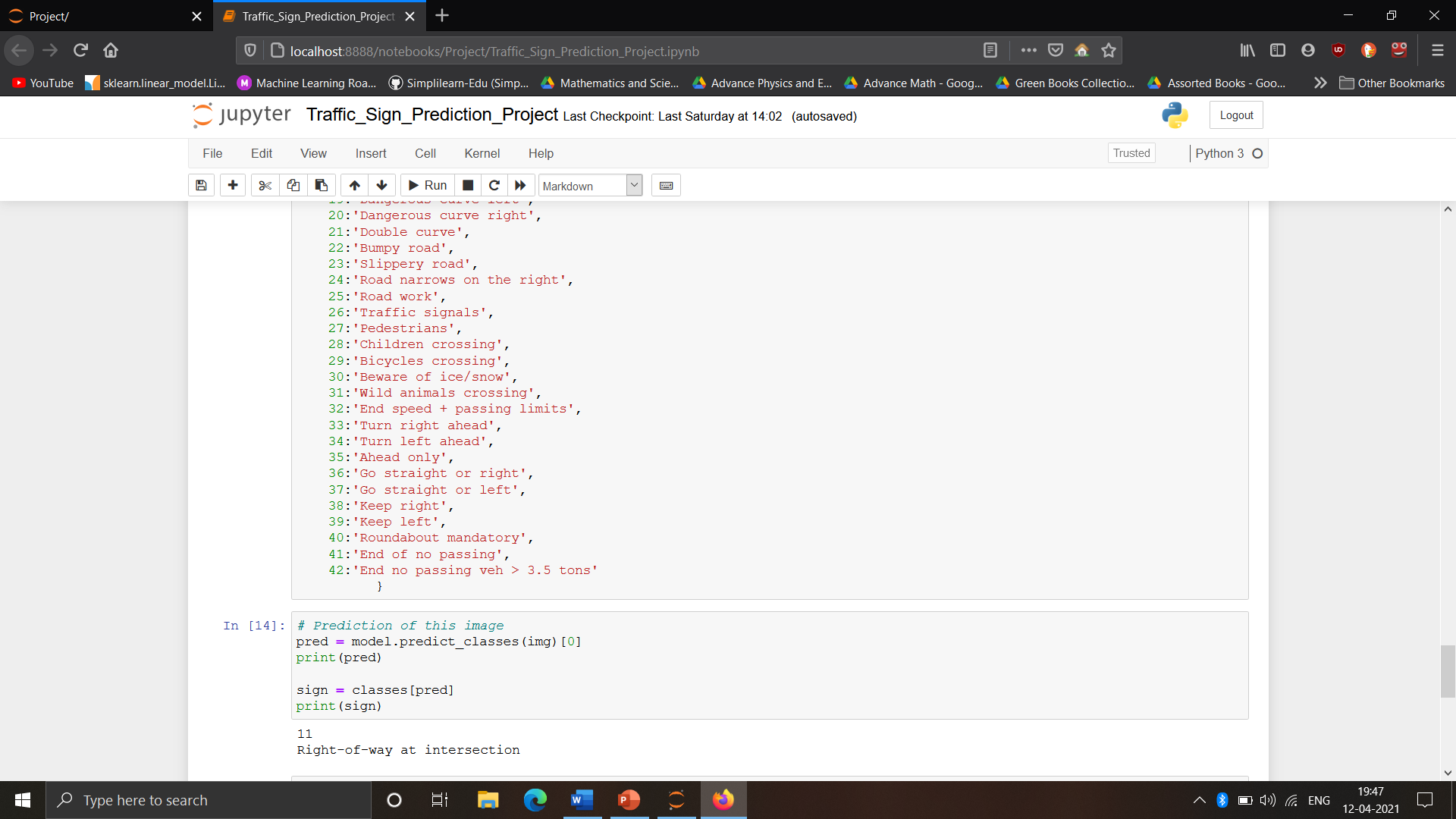


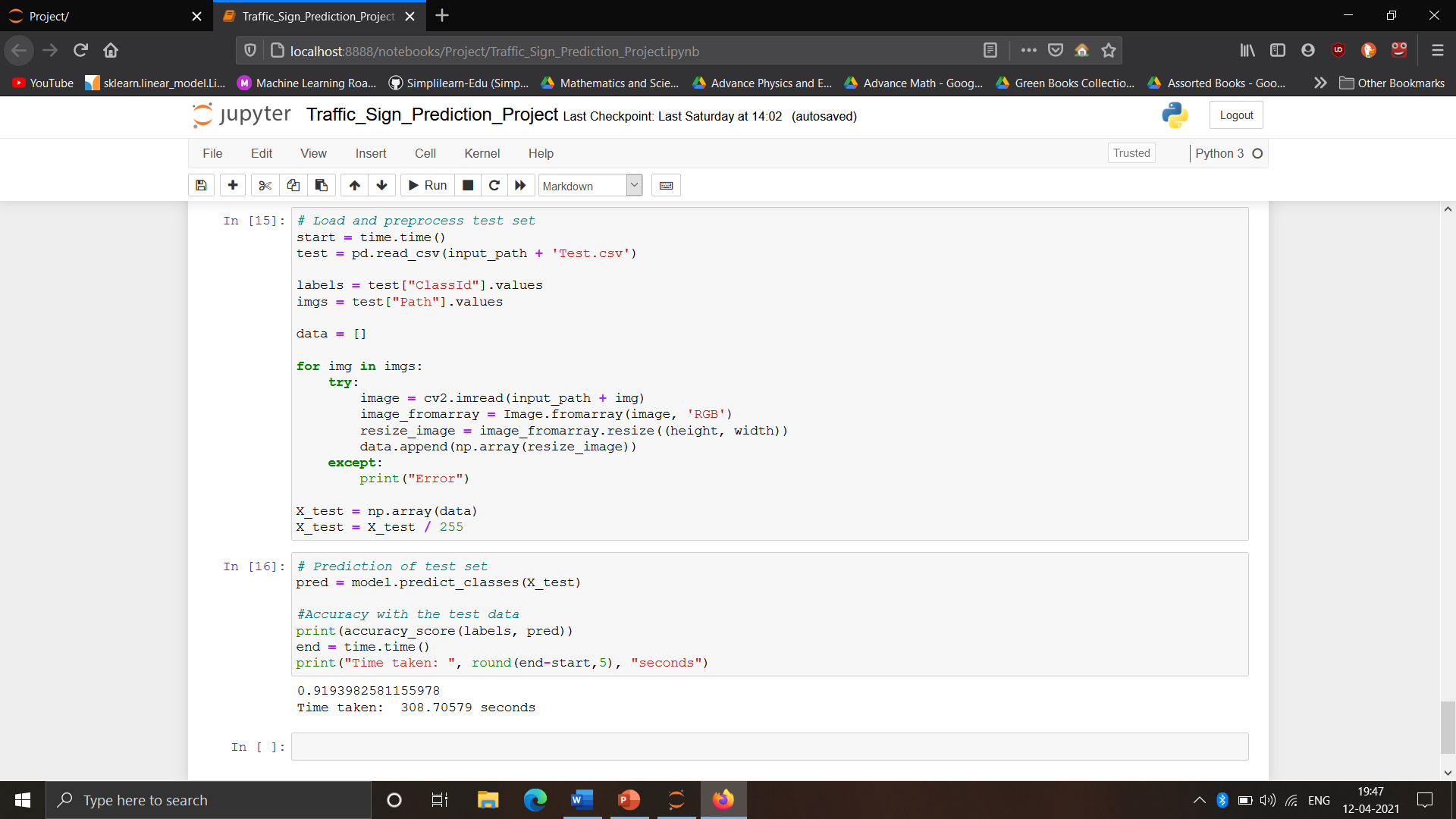












**CONCLUSION:**

From the following results we can see that the CNN is doing a good job in classifying different types of traffic signs when the extracted signs are cropped perfectly from the image. Our approach fails to give good results when the extracted signs from test images are cropped incorrectly. Another drawback of our approach is that when the color of the traffic signs vary which may be due to bad weather conditions and poor camera quality, the image masks obtained are not perfect and hence the signs are not detected properly. Future improvements can be made for extracting signs from test images by using advanced segmentation methods

**REFERENCES:**

**-**https://www.kaggle.com/

-http://docs.opencv.org/2.4/doc/tutorials/imgproc/shapedescriptors/find\_contours/find\_contours.html

-https://en.wikipedia.org/wiki/Convolutional\_neural\_network