University Of Sulaimani
College Of Science
Computer Department

Traffic Signs Recognition in Machin Learning

Prepared By

Mariwan Mahmud – UOS-CS-Dep

2023-2024

What is Traffic Signs Recognition?

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.

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 this Project

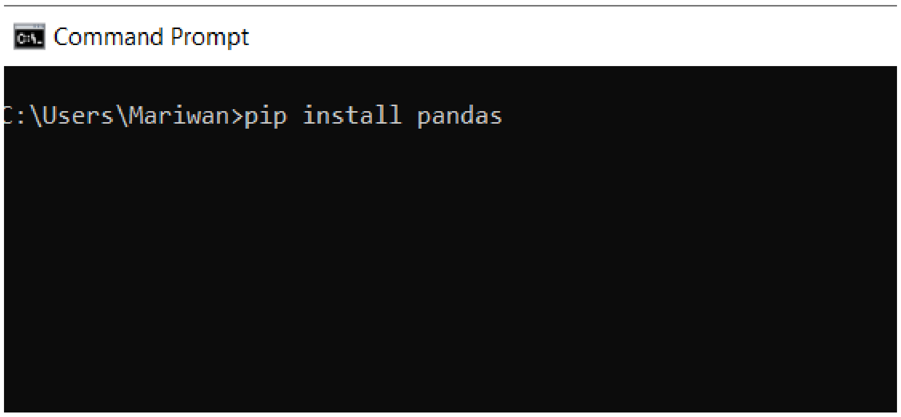
We are utilizing a public dataset from Kaggle for this project, which has over 50,000 photos of various traffic signs. 43 distinct classes are added to its further classification. The dataset exhibits significant variation, with a large number of images in certain classes and few images in others. The dataset is approximately 300 MB in size. The dataset includes two folders: test, which we will use to test our model, and train, which contains images inside each class

Package Requirements

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 pachakge_name ...

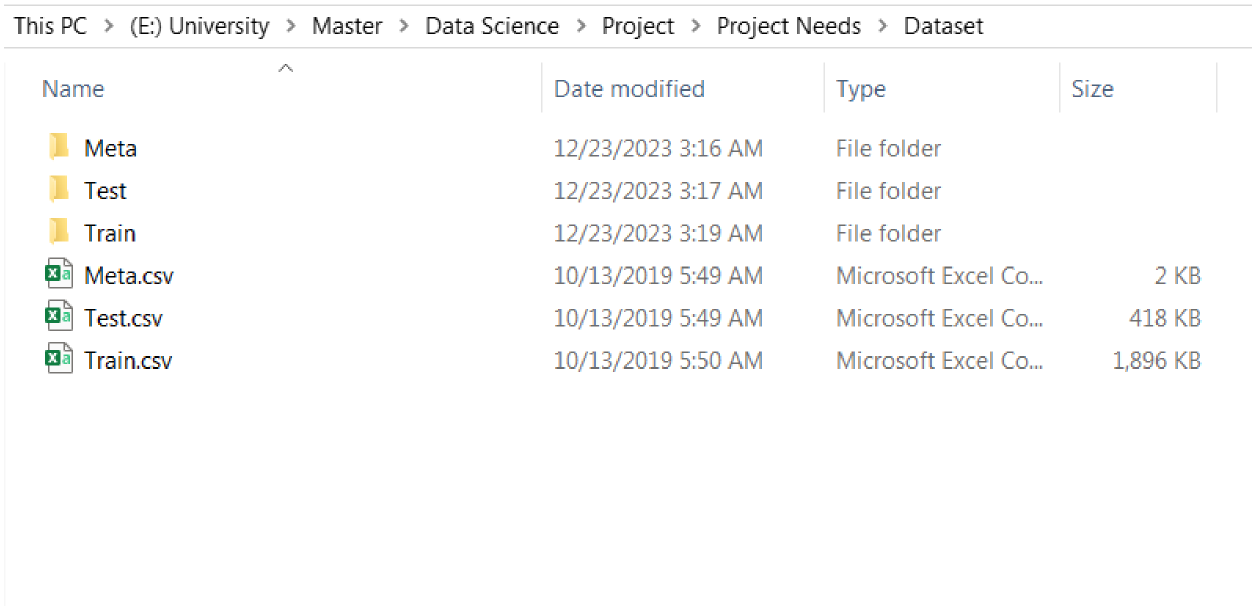


```
C:\Users\Mariwan>pip install pandas
```

Steps to Build this Project

1. **Download And Extract Dataset**

   At the first-time u have to download the dataset and extract the files into a folder such that you will have a train, test and a meta folder. 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

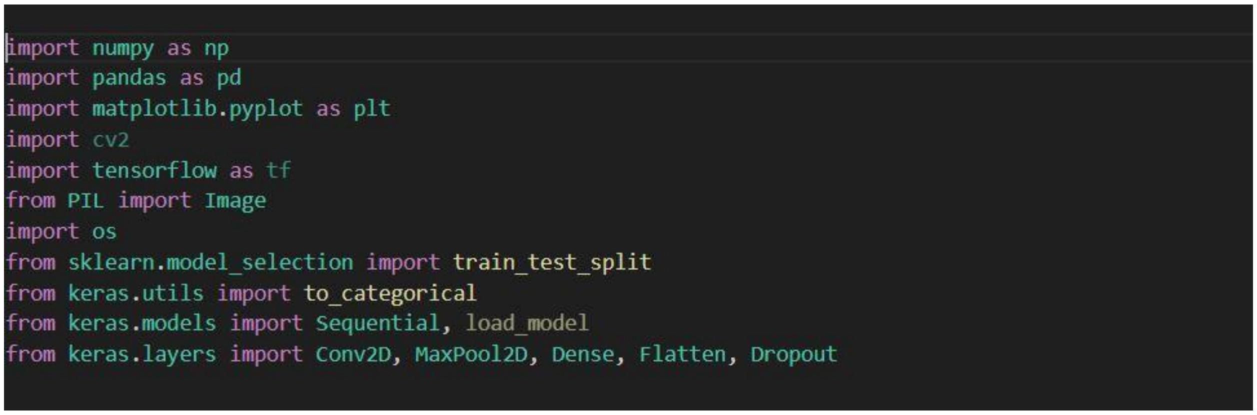


This PC > (E:) University > Master > Data Science > Project > Project Needs > Dataset

Name	Date modified	Type	Size
Meta	12/23/2023 3:16 AM	File folder	
Test	12/23/2023 3:17 AM	File folder	
Train	12/23/2023 3:19 AM	File folder	
Meta.csv	10/13/2019 5:49 AM	Microsoft Excel Co...	2 KB
Test.csv	10/13/2019 5:49 AM	Microsoft Excel Co...	418 KB
Train.csv	10/13/2019 5:50 AM	Microsoft Excel Co...	1,896 KB

2. **Download machine learning package and library**

   you can Download machine learning package and library like above command in command prompt. The PIL library is used to open image content into an array.

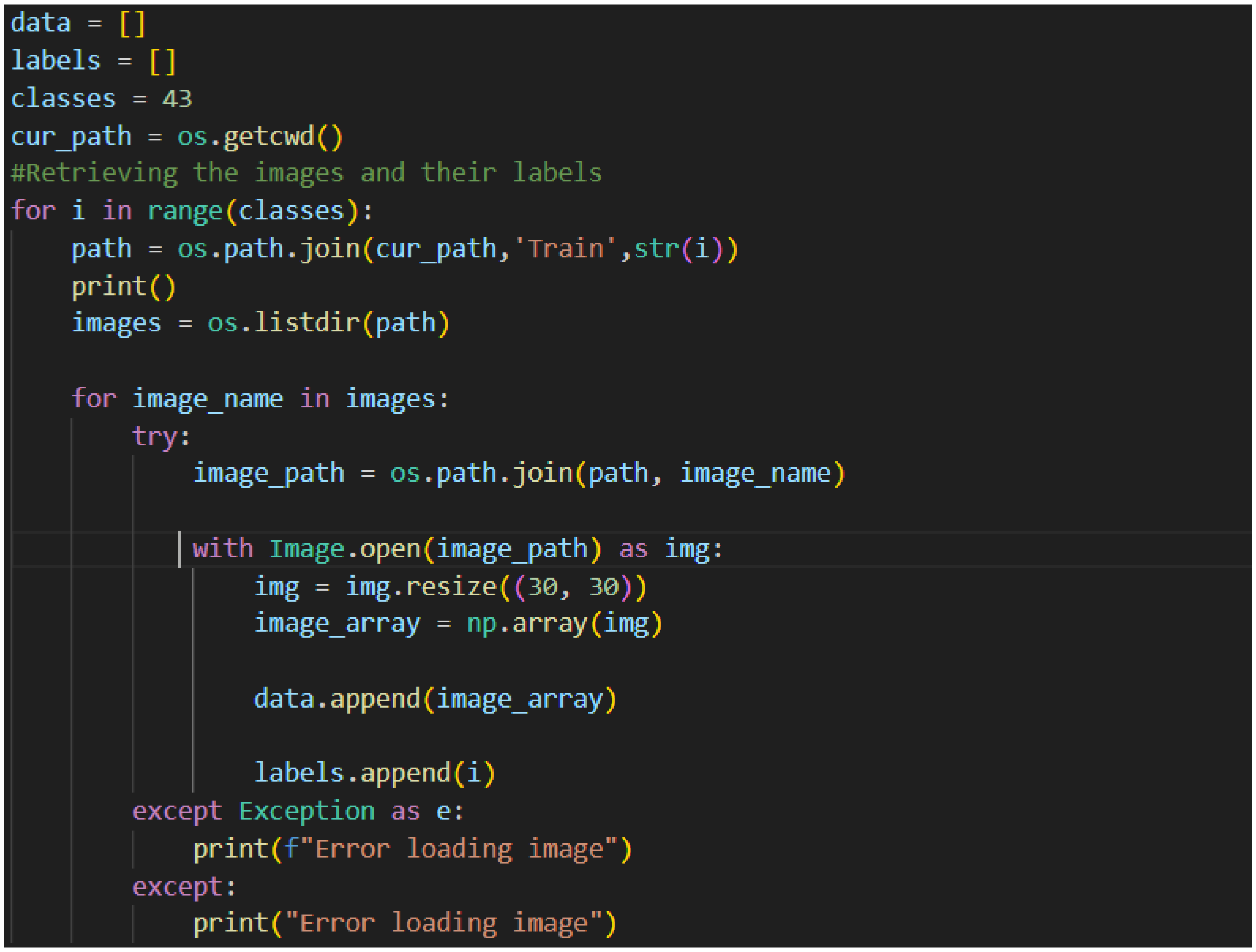


```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import cv2
import tensorflow as tf
from PIL import Image
import os
from sklearn.model_selection import train_test_split
from keras.utils import to_categorical
from keras.models import Sequential, load_model
from keras.layers import Conv2D, MaxPool2D, Dense, Flatten, Dropout
```

3. load and store images

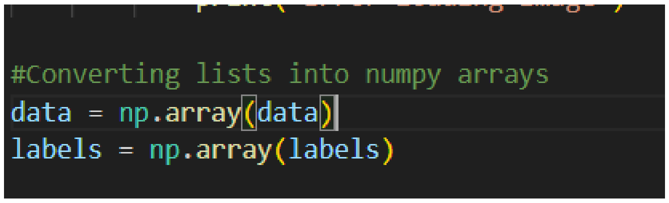
we have stored all the images and their labels into lists (data and labels).

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)



```python
data = []
labels = []
classes = 43
cur_path = os.getcwd()
#Retrieving the images and their labels
for i in range(classes):
    path = os.path.join(cur_path,'Train',str(i))
    print()
    images = os.listdir(path)

    for image_name in images:
        try:
            image_path = os.path.join(path, image_name)

            with Image.open(image_path) as img:
                img = img.resize((30, 30))
                image_array = np.array(img)

                data.append(image_array)

                labels.append(i)
        except Exception as e:
            print(f"Error loading image")
        except:
            print("Error loading image")
```

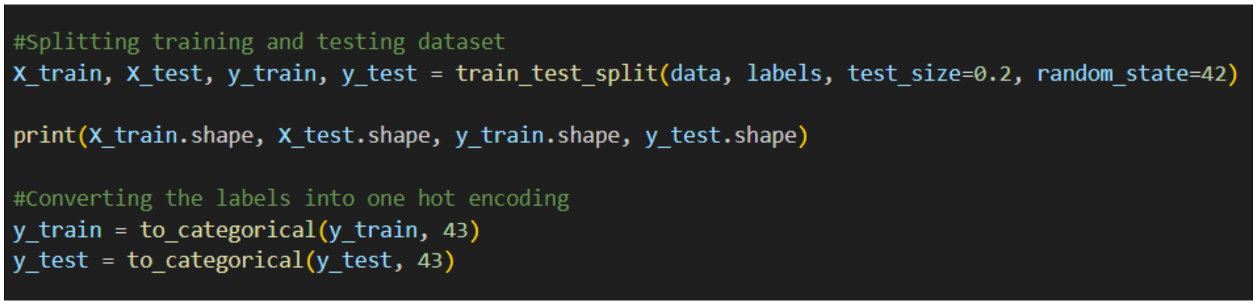
after that u have to convert the image list to array



```python
#Converting lists into numpy arrays
data = np.array(data)
labels = np.array(labels)
```

4. Split Training and Testing Data.

you cand split data 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



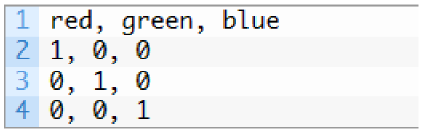
```
#Splitting training and testing dataset
X_train, X_test, y_train, y_test = train_test_split(data, labels, test_size=0.2, random_state=42)

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)
```

5. what is one-hot encoding?

One-hot encoding in machine learning is the conversion of categorical information into a format that may be fed into machine learning algorithms to improve prediction accuracy

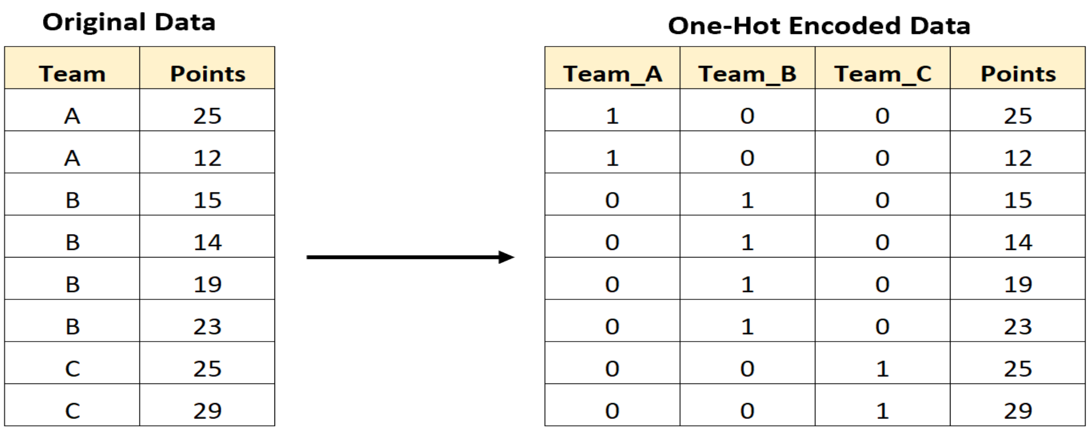
For example, "*red*" is 1, "*green*" is 2, and "*blue*" is 3.
This is called a label encoding or an integer encoding and is easily reversible

In the "*color*" variable example, there are 3 categories and therefore 3 binary variables are needed. A "1" value is placed in the binary variable for the color and "0" values for the other colors



```
red, green, blue
1, 0, 0
0, 1, 0
0, 0, 1
```

Another Example:



Original Data

Team	Points
A	25
A	12
B	15
B	14
B	19
B	23
C	25
C	29

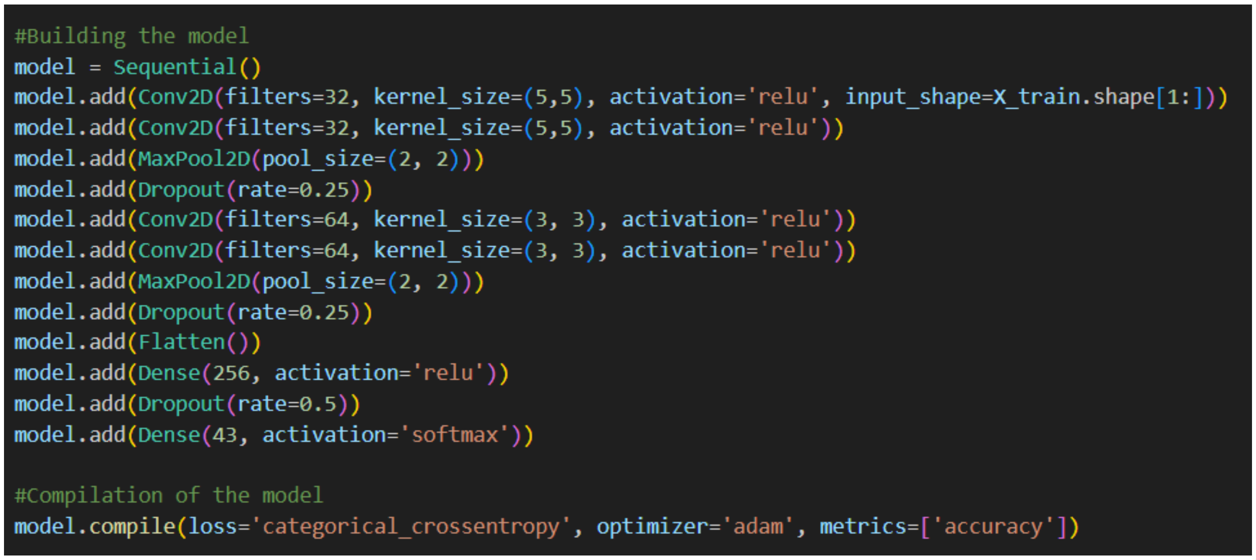
One-Hot Encoded Data

Team_A	Team_B	Team_C	Points
1	0	0	25
1	0	0	12
0	1	0	15
0	1	0	14
0	1	0	19
0	1	0	23
0	0	1	25
0	0	1	29

6. Build a CNN model

will build a CNN model (Convolutional Neural Network). CNN is best for image classification purposes. And We compile the model with Adam optimizer

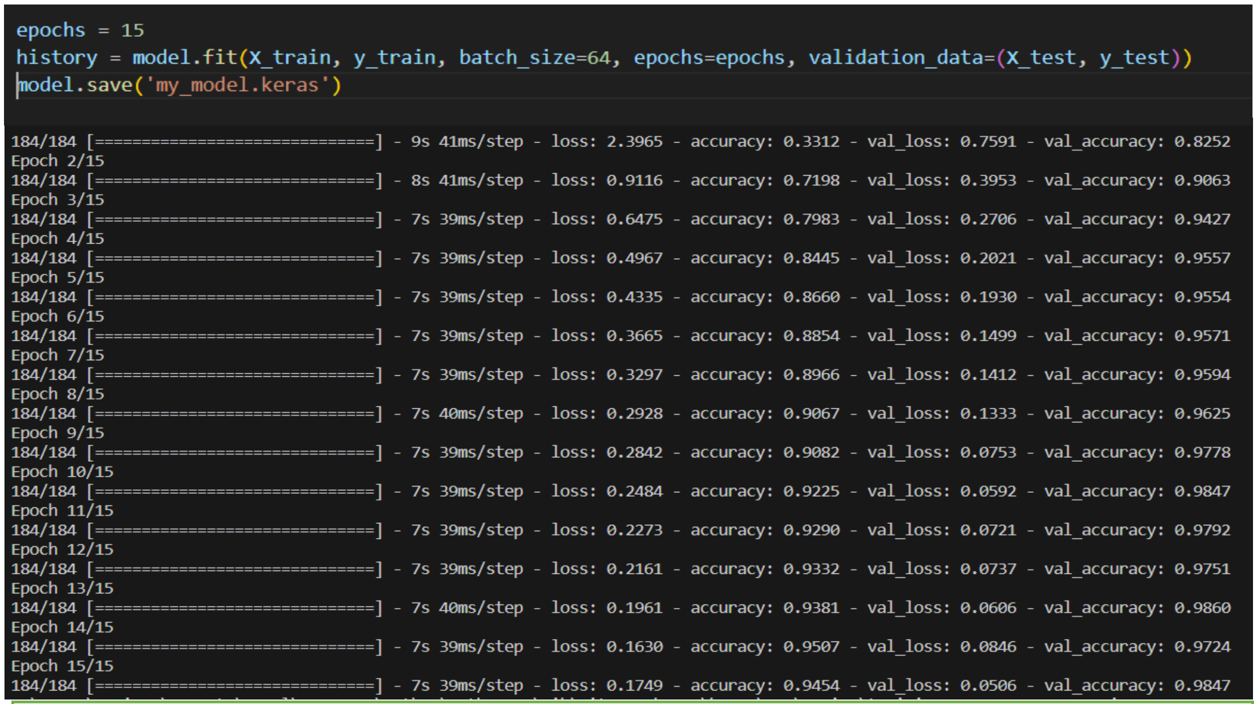
The architecture of our model is:



```
#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'])
```

7. Train and validate the model

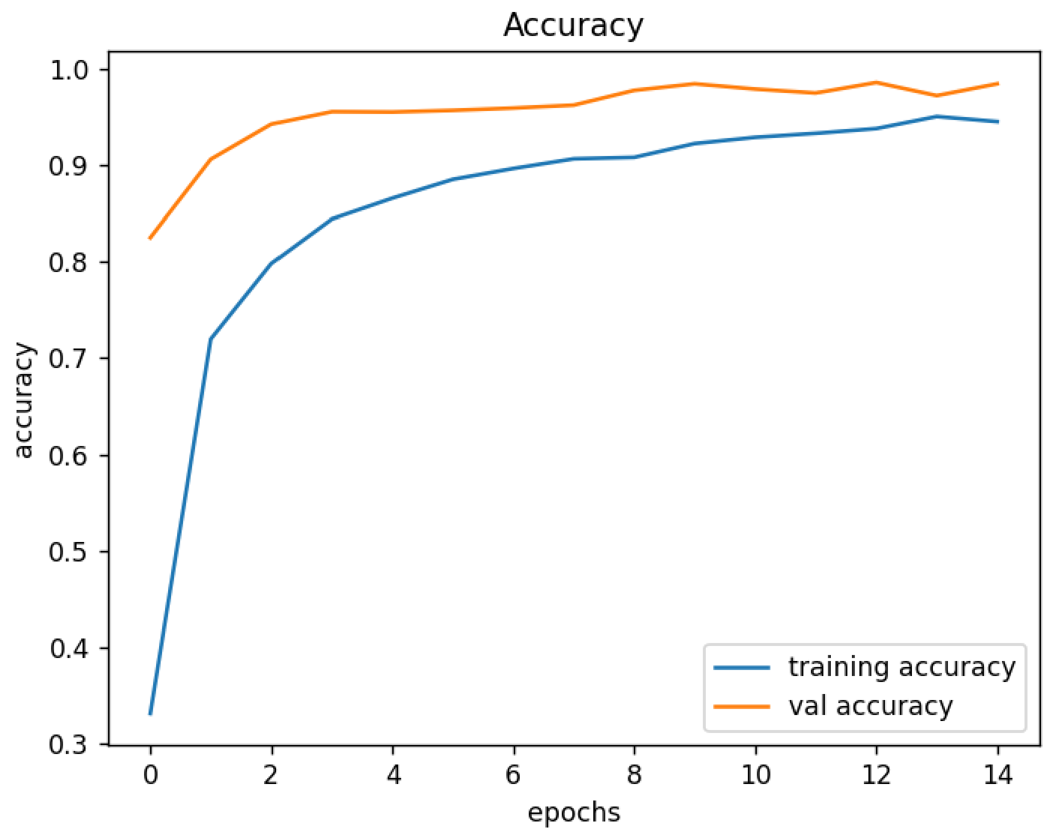
After building the model architecture, we then train the model using model. Fit (). I tried with batch size 32 and 64. Our model performed better with 64 batch size. And after 15 epochs the accuracy was stable

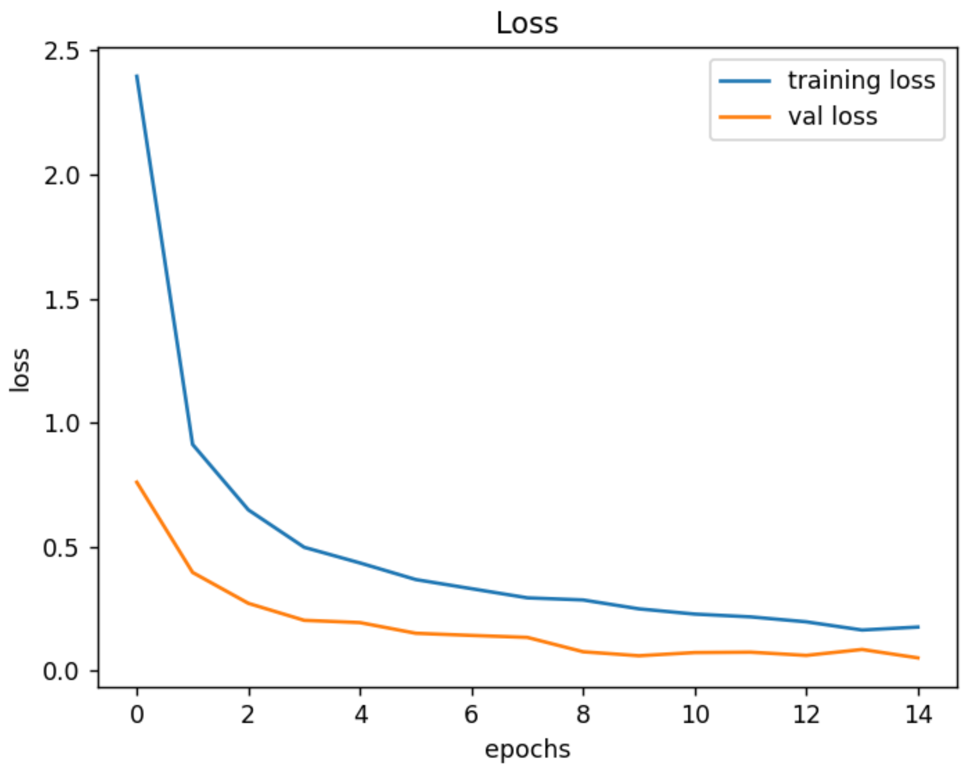


```
epochs = 15
history = model.fit(X_train, y_train, batch_size=64, epochs=epochs, validation_data=(X_test, y_test))
model.save('my_model.keras')
```

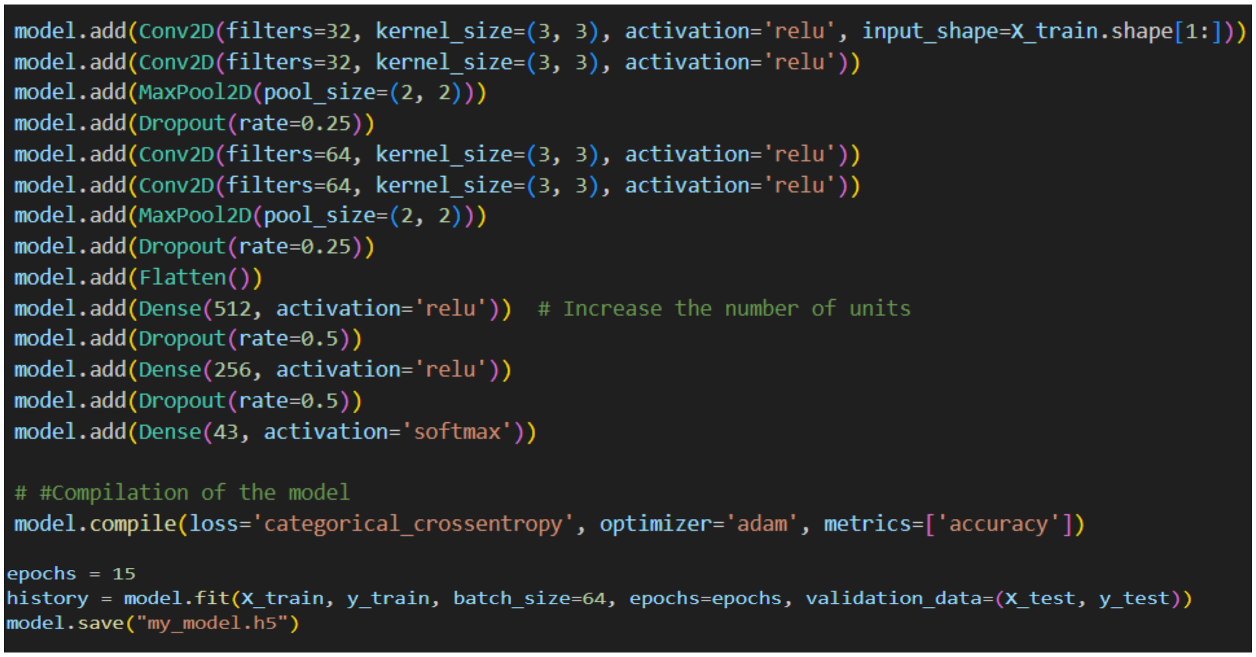
```
184/184 [==============================] - 9s 41ms/step - loss: 2.3965 - accuracy: 0.3312 - val_loss: 0.7591 - val_accuracy: 0.8252
Epoch 2/15
184/184 [==============================] - 8s 41ms/step - loss: 0.9116 - accuracy: 0.7198 - val_loss: 0.3953 - val_accuracy: 0.9063
Epoch 3/15
184/184 [==============================] - 7s 39ms/step - loss: 0.6475 - accuracy: 0.7983 - val_loss: 0.2706 - val_accuracy: 0.9427
Epoch 4/15
184/184 [==============================] - 7s 39ms/step - loss: 0.4967 - accuracy: 0.8445 - val_loss: 0.2021 - val_accuracy: 0.9557
Epoch 5/15
184/184 [==============================] - 7s 39ms/step - loss: 0.4335 - accuracy: 0.8660 - val_loss: 0.1930 - val_accuracy: 0.9554
Epoch 6/15
184/184 [==============================] - 7s 39ms/step - loss: 0.3665 - accuracy: 0.8854 - val_loss: 0.1499 - val_accuracy: 0.9571
Epoch 7/15
184/184 [==============================] - 7s 39ms/step - loss: 0.3297 - accuracy: 0.8966 - val_loss: 0.1412 - val_accuracy: 0.9594
Epoch 8/15
184/184 [==============================] - 7s 40ms/step - loss: 0.2928 - accuracy: 0.9067 - val_loss: 0.1333 - val_accuracy: 0.9625
Epoch 9/15
184/184 [==============================] - 7s 39ms/step - loss: 0.2842 - accuracy: 0.9082 - val_loss: 0.0753 - val_accuracy: 0.9778
Epoch 10/15
184/184 [==============================] - 7s 39ms/step - loss: 0.2484 - accuracy: 0.9225 - val_loss: 0.0592 - val_accuracy: 0.9847
Epoch 11/15
184/184 [==============================] - 7s 39ms/step - loss: 0.2273 - accuracy: 0.9290 - val_loss: 0.0721 - val_accuracy: 0.9792
Epoch 12/15
184/184 [==============================] - 7s 39ms/step - loss: 0.2161 - accuracy: 0.9332 - val_loss: 0.0737 - val_accuracy: 0.9751
Epoch 13/15
184/184 [==============================] - 7s 40ms/step - loss: 0.1961 - accuracy: 0.9381 - val_loss: 0.0606 - val_accuracy: 0.9860
Epoch 14/15
184/184 [==============================] - 7s 39ms/step - loss: 0.1630 - accuracy: 0.9507 - val_loss: 0.0846 - val_accuracy: 0.9724
Epoch 15/15
184/184 [==============================] - 7s 39ms/step - loss: 0.1749 - accuracy: 0.9454 - val_loss: 0.0506 - val_accuracy: 0.9847
```

Accuracy: 0.9454


Accuracy
accuracy
epochs
training accuracy
val accuracy
1.0
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0
2
4
6
8
10
12
14

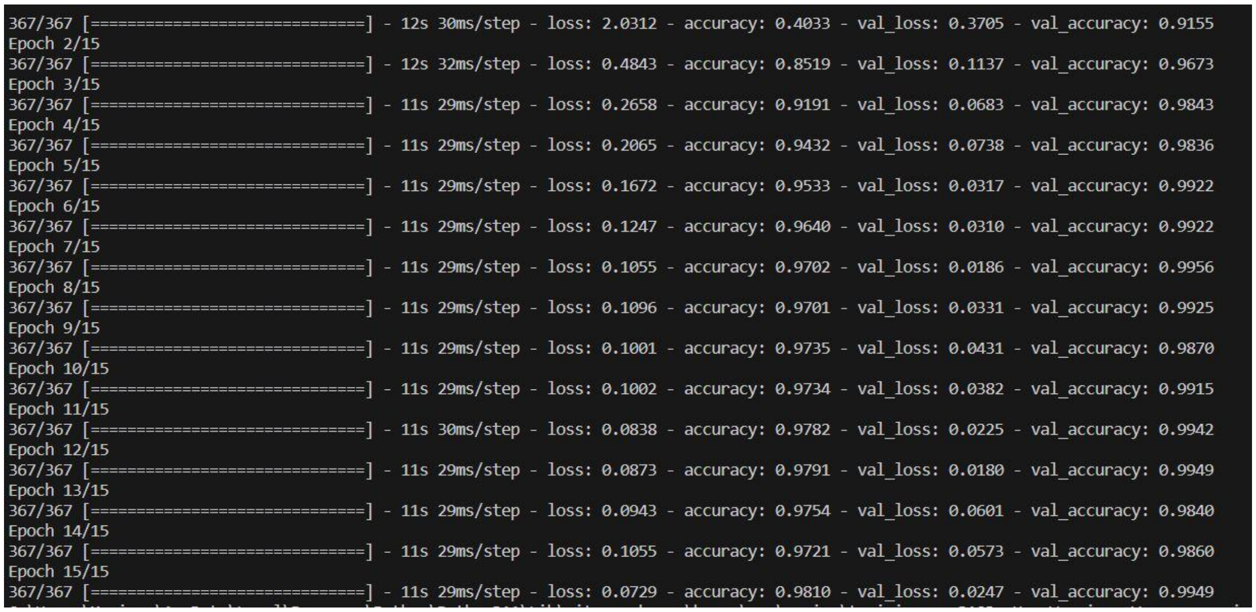

Loss
loss
epochs
training loss
val loss
2.5
2.0
1.5
1.0
0.5
0.0
0
2
4
6
8
10
12
14

and you can improve the accuracy by changing kernel_size parameter to (3,3)



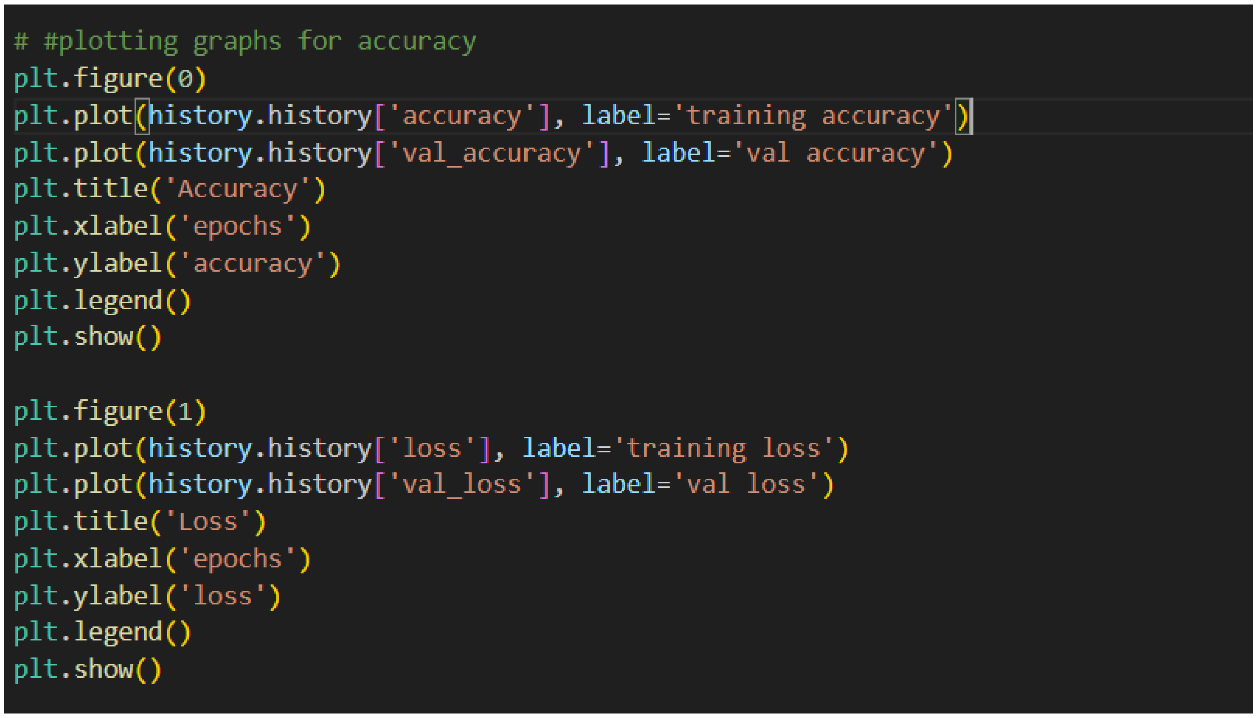
```
model.add(Conv2D(filters=32, kernel_size=(3, 3), activation='relu', input_shape=X_train.shape[1:]))
model.add(Conv2D(filters=32, kernel_size=(3, 3), activation='relu'))
model.add(MaxPool2D(pool_size=(2, 2)))
model.add(Dropout(rate=0.25))
model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu'))
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(512, activation='relu'))  # Increase the number of units
model.add(Dropout(rate=0.5))
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'])

epochs = 15
history = model.fit(X_train, y_train, batch_size=64, epochs=epochs, validation_data=(X_test, y_test))
model.save("my_model.h5")
```

result:

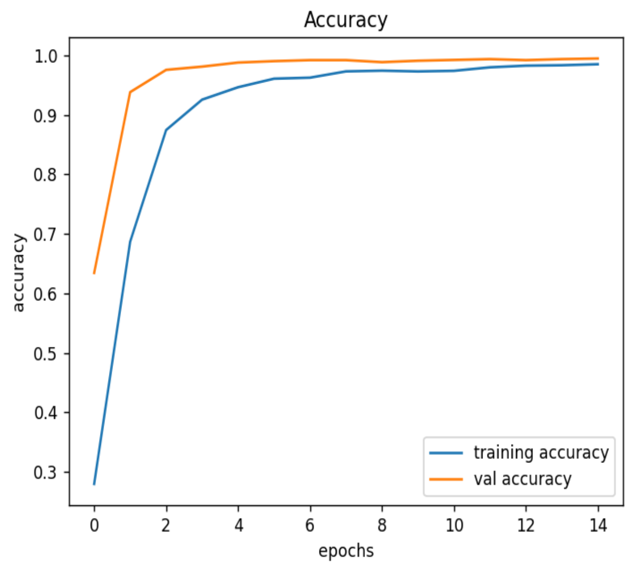


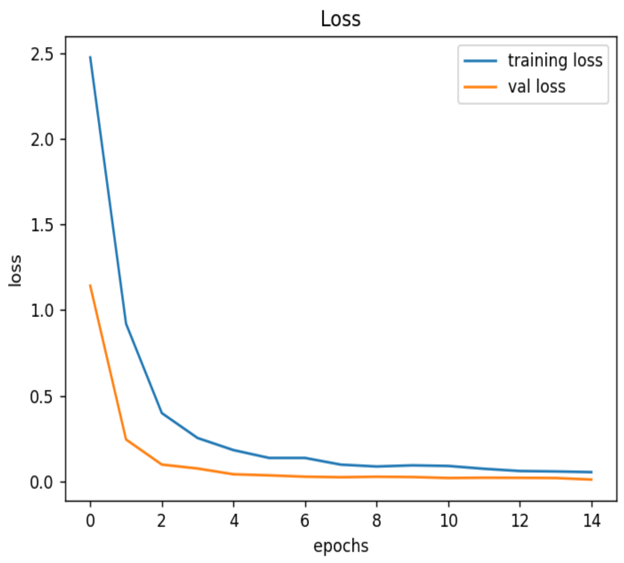
```
367/367 [==============================] - 12s 30ms/step - loss: 2.0312 - accuracy: 0.4033 - val_loss: 0.3705 - val_accuracy: 0.9155
Epoch 2/15
367/367 [==============================] - 12s 32ms/step - loss: 0.4843 - accuracy: 0.8519 - val_loss: 0.1137 - val_accuracy: 0.9673
Epoch 3/15
367/367 [==============================] - 11s 29ms/step - loss: 0.2658 - accuracy: 0.9191 - val_loss: 0.0683 - val_accuracy: 0.9843
Epoch 4/15
367/367 [==============================] - 11s 29ms/step - loss: 0.2065 - accuracy: 0.9432 - val_loss: 0.0738 - val_accuracy: 0.9836
Epoch 5/15
367/367 [==============================] - 11s 29ms/step - loss: 0.1672 - accuracy: 0.9533 - val_loss: 0.0317 - val_accuracy: 0.9922
Epoch 6/15
367/367 [==============================] - 11s 29ms/step - loss: 0.1247 - accuracy: 0.9640 - val_loss: 0.0310 - val_accuracy: 0.9922
Epoch 7/15
367/367 [==============================] - 11s 29ms/step - loss: 0.1055 - accuracy: 0.9702 - val_loss: 0.0186 - val_accuracy: 0.9956
Epoch 8/15
367/367 [==============================] - 11s 29ms/step - loss: 0.1096 - accuracy: 0.9701 - val_loss: 0.0331 - val_accuracy: 0.9925
Epoch 9/15
367/367 [==============================] - 11s 29ms/step - loss: 0.1001 - accuracy: 0.9735 - val_loss: 0.0431 - val_accuracy: 0.9870
Epoch 10/15
367/367 [==============================] - 11s 29ms/step - loss: 0.1002 - accuracy: 0.9734 - val_loss: 0.0382 - val_accuracy: 0.9915
Epoch 11/15
367/367 [==============================] - 11s 30ms/step - loss: 0.0838 - accuracy: 0.9782 - val_loss: 0.0225 - val_accuracy: 0.9942
Epoch 12/15
367/367 [==============================] - 11s 29ms/step - loss: 0.0873 - accuracy: 0.9791 - val_loss: 0.0180 - val_accuracy: 0.9949
Epoch 13/15
367/367 [==============================] - 11s 29ms/step - loss: 0.0943 - accuracy: 0.9754 - val_loss: 0.0601 - val_accuracy: 0.9840
Epoch 14/15
367/367 [==============================] - 11s 29ms/step - loss: 0.1055 - accuracy: 0.9721 - val_loss: 0.0573 - val_accuracy: 0.9860
Epoch 15/15
367/367 [==============================] - 11s 29ms/step - loss: 0.0729 - accuracy: 0.9810 - val_loss: 0.0247 - val_accuracy: 0.9949
```

8. **we plot the graph for accuracy and the loss.**



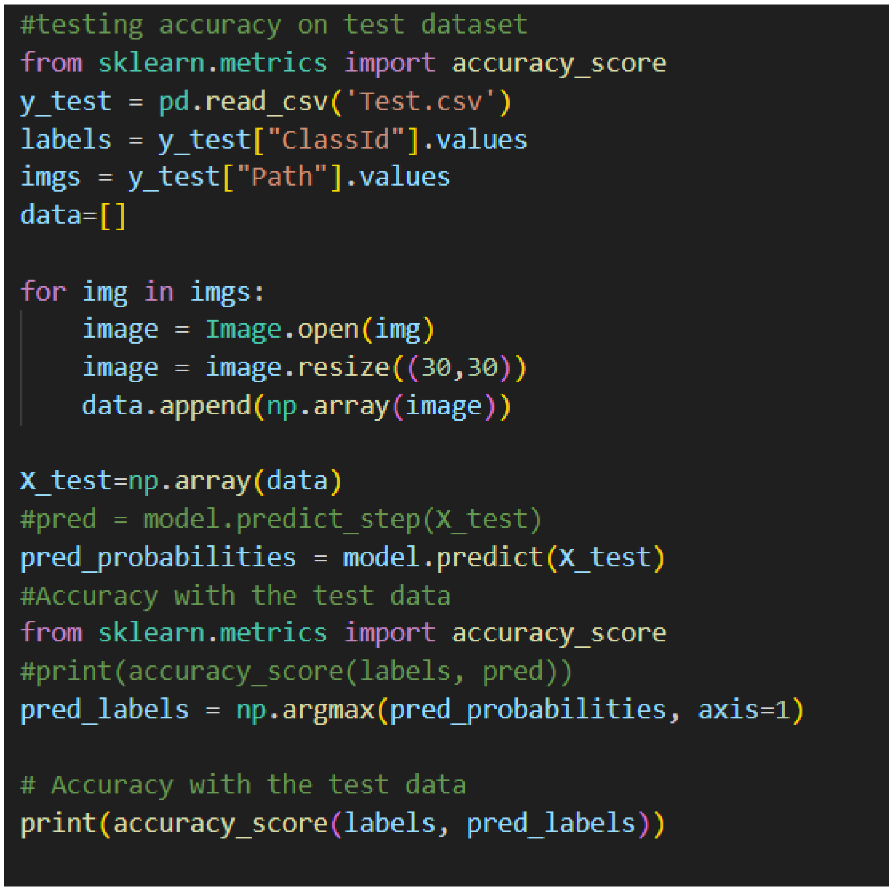
```
# #plotting graphs for accuracy
plt.figure(0)
plt.plot(history.history['accuracy'], label='training accuracy')
plt.plot(history.history['val_accuracy'], label='val accuracy')
plt.title('Accuracy')
plt.xlabel('epochs')
plt.ylabel('accuracy')
plt.legend()
plt.show()

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()
```



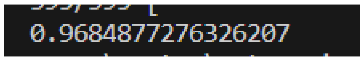


9. Step 4: Test our model with test dataset

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,



```python
#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_step(X_test)
pred_probabilities = model.predict(X_test)
#Accuracy with the test data
from sklearn.metrics import accuracy_score
#print(accuracy_score(labels, pred))
pred_labels = np.argmax(pred_probabilities, axis=1)

# Accuracy with the test data
print(accuracy_score(labels, pred_labels))
```

Accuracy: 

```
0.9684877276326207
```