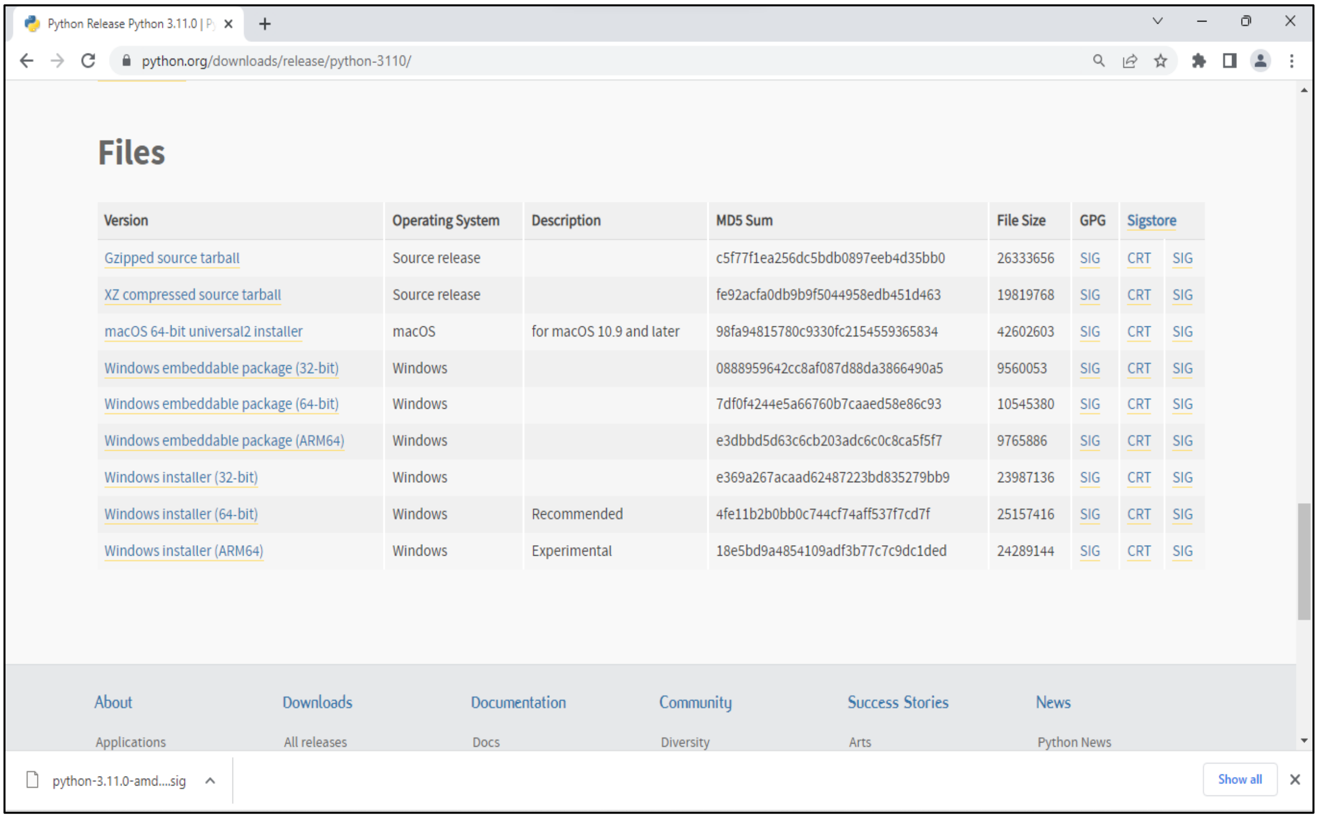
Deep Learning

Part 1: Code and Data Extraction

1. Environment Setup

To install python, download python.exe file from following site https://www.python.org/downloads/release/python-3110/ .

To install Jupyter Notebook after installation, open command promotes and type "pip install notebook."



Python Release Python 3.11.0 | P

python.org/downloads/release/python-3110/

Files

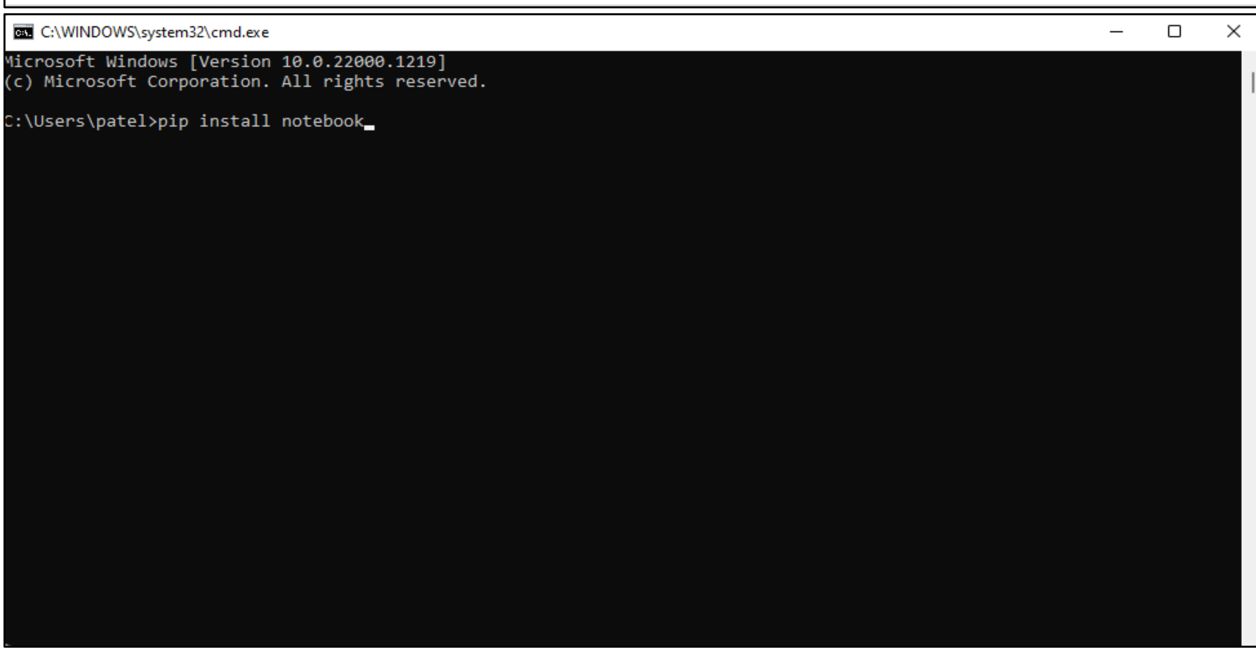
Version	Operating System	Description	MD5 Sum	File Size	GPG	Sigstore	
Gzipped source tarball	Source release		c5f77f1ea256dc5bdb0897eeb4d35bb0	26333656	SIG	CRT	SIG
XZ compressed source tarball	Source release		fe92acfa0db9b9f5044958edb451d463	19819768	SIG	CRT	SIG
macOS 64-bit universal2 installer	macOS	for macOS 10.9 and later	98fa94815780c9330fc2154559365834	42602603	SIG	CRT	SIG
Windows embeddable package (32-bit)	Windows		0888959642cc8af087d88da3866490a5	9560053	SIG	CRT	SIG
Windows embeddable package (64-bit)	Windows		7df0f4244e5a66760b7caaed58e86c93	10545380	SIG	CRT	SIG
Windows embeddable package (ARM64)	Windows		e3dbbd5d63c6cb203adc6c0c8ca5f5f7	9765886	SIG	CRT	SIG
Windows installer (32-bit)	Windows		e369a267acaad62487223bd835279bb9	23987136	SIG	CRT	SIG
Windows installer (64-bit)	Windows	Recommended	4fe11b2b0bb0c744cf74aff537f7cd7f	25157416	SIG	CRT	SIG
Windows installer (ARM64)	Windows	Experimental	18e5bd9a4854109adf3b77c7c9dc1ded	24289144	SIG	CRT	SIG

About | Downloads | Documentation | Community | Success Stories | News

Applications | All releases | Docs | Diversity | Arts | Python News

python-3.11.0-amd....sig

Show all



2. Install TensorFlow, Keras, Sklearn, NumPy and Matplotlib Libraries with following command in notebook

" pip install tensorflow pandas numpy sklearn keras matplotlib"

Step 1 : Install Libraries

In [5]:
```
pip install tensorflow pandas numpy sklearn keras
```

```
n.3.10_qbz5n2kfra8p0\localcache\local-packages\python310\site-packages (from tensorflow) (3.19.6)
Requirement already satisfied: libclang>=13.0.0 in c:\users\patel\appdata\local\packages\pythonsoftwarefoundation.python.3.1
0_qbz5n2kfra8p0\localcache\local-packages\python310\site-packages (from tensorflow) (14.0.6)
Requirement already satisfied: astunparse>=1.6.0 in c:\users\patel\appdata\local\packages\pythonsoftwarefoundation.python.3.
10_qbz5n2kfra8p0\localcache\local-packages\python310\site-packages (from tensorflow) (1.6.3)
Requirement already satisfied: tensorboard<2.11,>=2.10 in c:\users\patel\appdata\local\packages\pythonsoftwarefoundation.pyt
hon.3.10_qbz5n2kfra8p0\localcache\local-packages\python310\site-packages (from tensorflow) (2.10.1)
Requirement already satisfied: keras-preprocessing>=1.1.1 in c:\users\patel\appdata\local\packages\pythonsoftwarefoundation.
python.3.10_qbz5n2kfra8p0\localcache\local-packages\python310\site-packages (from tensorflow) (1.1.2)
Requirement already satisfied: flatbuffers>=2.0 in c:\users\patel\appdata\local\packages\pythonsoftwarefoundation.python.3.1
0_qbz5n2kfra8p0\localcache\local-packages\python310\site-packages (from tensorflow) (22.10.26)
Requirement already satisfied: tensorflow-estimator<2.11,>=2.10.0 in c:\users\patel\appdata\local\packages\pythonsoftwarefou
ndation.python.3.10_qbz5n2kfra8p0\localcache\local-packages\python310\site-packages (from tensorflow) (2.10.0)
Requirement already satisfied: google-pasta>=0.1.1 in c:\users\patel\appdata\local\packages\pythonsoftwarefoundation.python.
3.10_qbz5n2kfra8p0\localcache\local-packages\python310\site-packages (from tensorflow) (0.2.0)
Requirement already satisfied: grpcio<2.0,>=1.24.3 in c:\users\patel\appdata\local\packages\pythonsoftwarefoundation.python.
3.10_qbz5n2kfra8p0\localcache\local-packages\python310\site-packages (from tensorflow) (1.50.0)
Requirement already satisfied: python-dateutil>=2.8.1 in c:\users\patel\appdata\local\packages\pythonsoftwarefoundation.pyth
on.3.10_qbz5n2kfra8p0\localcache\local-packages\python310\site-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\patel\appdata\local\packages\pythonsoftwarefoundation.python.3.10_qb
```

Import all the libraries as shown in below figure using "import" keyword

Step 2: Import Libarires

In [14]:
```
import numpy as np
import matplotlib.pyplot as plt

from PIL import Image
import os
import pandas as pd

from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import train_test_split
from tensorflow.keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Conv2D, MaxPool2D, Dense, Flatten, Dropout

from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
```

3. **Download the data from [GTSRB - German Traffic Sign Recognition Benchmark dataset from Kaggle]() and extract the zip folder to PC**

4. **Read the data from Local Drive to program**

   Save the data directory (images folders) to a variable; in following programme, that variable is called "cur path." Then, using the FOR loop, iterate through all the images and folders.

Step 3 : Add Training Data Set into Program and Normalize it

```
In [15]: data = []
labels = []
classes = 43
cur_path = "C:\\Users\\patel\\OneDrive\\download\\archive\\Train\\"
data_folder = 'data'

# Loading training dataset
for i in range(classes):
    path = os.path.join(cur_path,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:
            print("Error loading image")

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

# Normalizing data via Min-Max normalizer
scaler = MinMaxScaler()
ascolumns = data.reshape(-1, 3)
t = scaler.fit_transform(ascolumns)
data = t.reshape(data.shape)
print(data.shape, labels.shape)

(39209, 30, 30, 3) (39209,)
```

5. **Convolutional Neural Network Model Building**

   Models consist of **Convolution layer** with kernel size of (3,3) and Activation function 'relu'. Next, **pooling layer** with pool size of (3,3), then **Drop Out** layer with learning rate 0.25. Then, model have flatten, dropout and two dense layers at the end.

Model 1

```
In [20]: model = Sequential()
model.add(Conv2D(filters=32, kernel_size=(3,3), activation='relu', input_shape=X_train.shape[1:]))
model.add(MaxPool2D(pool_size=(3, 3)))
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'))
model.summary()
print(len(model.layers))

Model: "sequential_8"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_8 (Conv2D)           (None, 28, 28, 32)        896

 max_pooling2d_8 (MaxPooling  (None, 9, 9, 32)         0
 2D)

 dropout_8 (Dropout)         (None, 9, 9, 32)          0

 flatten_1 (Flatten)         (None, 2592)              0

 dense_8 (Dense)             (None, 256)               663808

 dropout_9 (Dropout)         (None, 256)               0

 dense_9 (Dense)             (None, 43)                11051

=================================================================
Total params: 675,755
Trainable params: 675,755
Non-trainable params: 0
_________________________________________________________________
7
```

6. Compilation of Model

Model created in the preceding phase needs to be completed before prediction. Run the command shown in the image below to assemble it, and then use the resulting model on the training and validation data sets.

Step 5: Compilation of the model

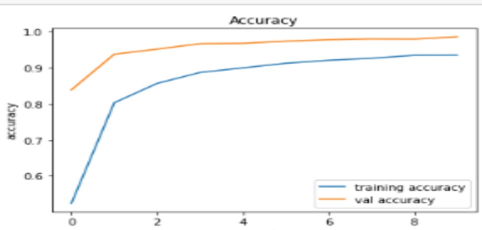
```
In [21]: model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
         epochs = 10
         history = model.fit(X_train, y_train, batch_size=32, epochs=epochs, validation_data=(X_val, y_val))
         model.save('traffic_classifier.h5')
```

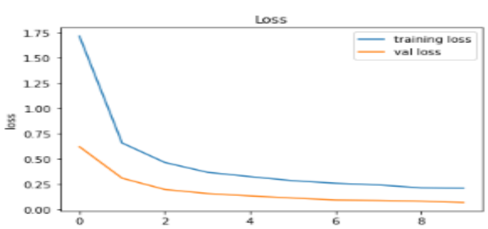
```
Epoch 1/10
981/981 [==============================] - 23s 22ms/step - loss: 1.6172 - accuracy: 0.5529 - val_loss: 0.5431 - val_accuracy:
0.8795
Epoch 2/10
981/981 [==============================] - 21s 21ms/step - loss: 0.6008 - accuracy: 0.8173 - val_loss: 0.2968 - val_accuracy:
0.9348
Epoch 3/10
981/981 [==============================] - 19s 20ms/step - loss: 0.4285 - accuracy: 0.8691 - val_loss: 0.1882 - val_accuracy:
0.9589
Epoch 4/10
981/981 [==============================] - 22s 22ms/step - loss: 0.3485 - accuracy: 0.8946 - val_loss: 0.1536 - val_accuracy:
0.9676
Epoch 5/10
981/981 [==============================] - 18s 19ms/step - loss: 0.3021 - accuracy: 0.9072 - val_loss: 0.1285 - val_accuracy:
0.9753
Epoch 6/10
981/981 [==============================] - 20s 20ms/step - loss: 0.2735 - accuracy: 0.9173 - val_loss: 0.1199 - val_accuracy:
0.9723
Epoch 7/10
981/981 [==============================] - 19s 20ms/step - loss: 0.2385 - accuracy: 0.9266 - val_loss: 0.0915 - val_accuracy:
0.9796
Epoch 8/10
981/981 [==============================] - 19s 20ms/step - loss: 0.2265 - accuracy: 0.9294 - val_loss: 0.0898 - val_accuracy:
0.9793
Epoch 9/10
981/981 [==============================] - 19s 19ms/step - loss: 0.2037 - accuracy: 0.9374 - val_loss: 0.0859 - val_accuracy:
0.9763
Epoch 10/10
981/981 [==============================] - 21s 21ms/step - loss: 0.1973 - accuracy: 0.9390 - val_loss: 0.0655 - val_accuracy:
0.9847
```

The accuracy and loss between the epochs and data sets are depicted in the following graph. Additionally, based on graph properties, it was projected that the model does neither overfit nor underfit for the settings of the supplied parameters.

```
In [13]: 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()
```





7. Computation Accuracy, Precision, Recall and F Score on test data set

Import the test data files from the directory into the programme using the same logic as when importing the train dataset. This will allow you to test the model on real test data. After that, normalise it before providing data to the model, and based on the confusion matrix, compute the accuracy, precision, recall, and F1 score after prediction.

Step 6: Prediction

In [22]:
```
test_path = "C:\\Users\\patel\\OneDrive\\download\\archive\\"
path = os.path.join(test_path)
data = "C:\\Users\\patel\\OneDrive\\download\\archive\\"
y_test = pd.read_csv(data +'Test.csv')
labels = y_test["ClassId"].values
imgs = y_test["Path"].values


data=[]

for img in imgs:
    image = Image.open(path + img)
    image = image.resize((30,30))
    data.append(np.array(image))
X_test = np.array(data)

# Normalizing test set
ascolumns = X_test.reshape(-1, 3)
t = scaler.transform(ascolumns)
X_test = t.reshape(X_test.shape)
```

In [25]:
```
# Predicting on test set
pred = np.argmax(model.predict(X_test),axis=1)
print(labels,pred)
cm = confusion_matrix(labels, pred)
print('Confusion Matrix:')
print(cm)

# accuracy: (tp + tn) / (p + n)
accuracy = accuracy_score(labels, pred)
print('Accuracy: %f' % accuracy)
# precision tp / (tp + fp)
precision = precision_score(labels, pred, average='macro')
print('Precision: %f' % precision)
# recall: tp / (tp + fn)
recall = recall_score(labels, pred, average='macro')
print('Recall: %f' % recall)
# f1: 2 tp / (2 tp + fp + fn)
f1 = f1_score(labels, pred, average='macro')
print('F1 score: %f' % f1)
```

```
395/395 [==============================] - 2s 4ms/step
[16  1 38 ...  6  7 10] [16  1 38 ...  3  7 10]
```

Following are model results:

```
 [  0   0   0 ...   0   1  85]]
Accuracy: 0.915281
Precision: 0.903170
Recall: 0.890970
F1 score: 0.887330
```

Part 2: Parameter Tunning

1. Epochs

According to the https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/ , is a hyperparameter that defines the number times that the learning algorithm will work through the entire training dataset. It has been discovered that when the program's epoch count is changed, the parameters indicated above also experience a slight difference.

```
[ 0 0 0 ... 0
Accuracy: 0.923040
Precision: 0.917071
Recall: 0.890027
F1 score: 0.896512
```

Epochs = 10

```
Accuracy: 0.919715
Precision: 0.916329
Recall: 0.879134
F1 score: 0.889379
```

Epochs = 15

```
[ 0 0 0 ... 0
Accuracy: 0.923278
Precision: 0.906873
Recall: 0.896159
F1 score: 0.897020
```

Epochs = 20

```
Accuracy: 0.926287
Precision: 0.913793
Recall: 0.901873
F1 score: 0.904342
```

Epochs = 25

2. Batch Size

According to the https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/, it is a hyperparameter that defines the number of samples to work through before updating the internal model parameters. Set batch sizes of 32, 64, and 128 cannot be used to determine whether they have a positive or negative impact on the model.

```
[ 0 0 0 ... 0
Accuracy: 0.923040
Precision: 0.917071
Recall: 0.890027
F1 score: 0.896512
```

Batch Size = 32

```
Accuracy: 0.918606
Precision: 0.905883
Recall: 0.882762
F1 score: 0.885726
```

Batch Size = 64

```
Accuracy: 0.926920
Precision: 0.910046
Recall: 0.897507
F1 score: 0.896900
```

Batch Size = 128

3. Number of Layers

The model becomes increasingly accurate and exact after further Convolution, Dropout, and polling layers are added.

```
[ 0 0 0 ... 0
Accuracy: 0.923040
Precision: 0.917071
Recall: 0.890027
F1 score: 0.896512
```

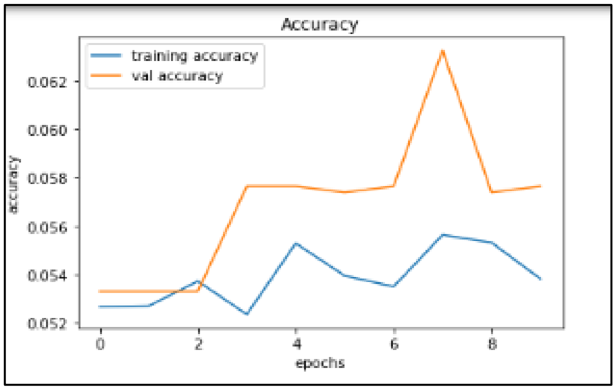
Number of Layers = 7

```
[ 0 0 0 ... 0
Accuracy: 0.961441
Precision: 0.944992
Recall: 0.936860
F1 score: 0.938008
```

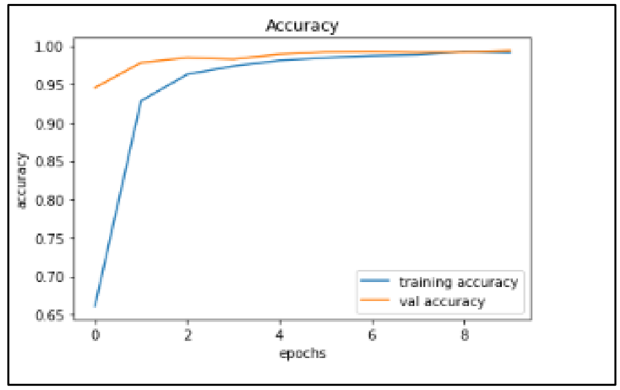
Number of Layers = 10

4. Learning Rate

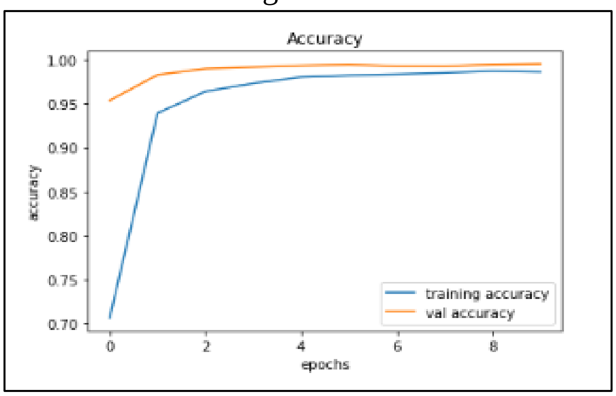
According to the Machine Learning, The learning rate controls how quickly the model is adapted to the problem. For this model, the default value and 0.001 show the highest performance, whereas values higher than 0.001 show unpredictable behaviour.



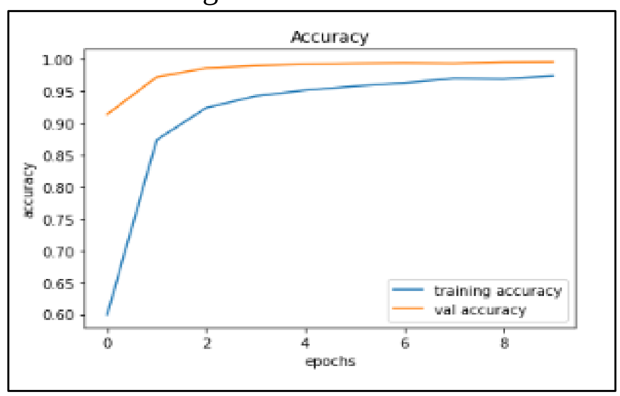
Learning rate = 0.005



Learning rate = 0.0005



Learning rate = 0.001



Learning rate = Default

Conclusion:

From the prior tuning and observation, it can be inferred that the CNN model positively depends on layers and performs better consistently for models with more layers. And for a model to produce reliable results, the learning rate should be 0.001 or less than it. In addition, neither epochs nor batch size significantly affect the output parameters.