PYTHON DEEP LEARNING

PROJECT_BASED_EXAM-1 REPORT

INTRODUCTION:

Main moto of this project is to implement Deep Learning and machine learning algorithms learnt in Python and Deep Learning class so far. Different algorithms used to in this project are CNN, LSTM, PCA, Text classification, Text generation and other Machine learning algorithms.

OBJECTIVE:

To implement different algorithms and models learnt in Python Deep Learning so far. And to apply different Machine Learning algorithms for obtaining efficient results.

METHODS:

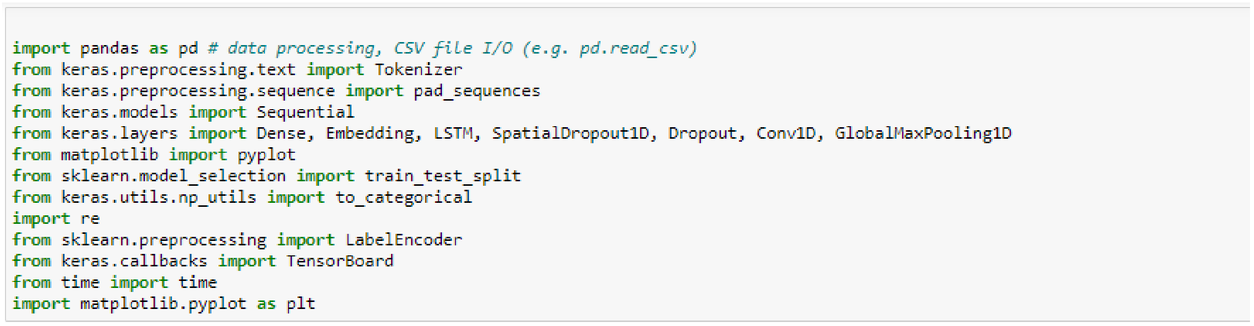
- CNN model: Convolutional neural network is a class of deep neural networks, most generally applied for evaluating visual imagery.
- LSTM model: LSTM model is a type of recurrent neural network that attains state-of-the-art fallouts on challenging prediction issues.
- Image classification.
- Text classification with CNN.
- Text generation with CNN.
- Auto-encoder with CNN.
- PCA

WORK FLOW:

Question 1:

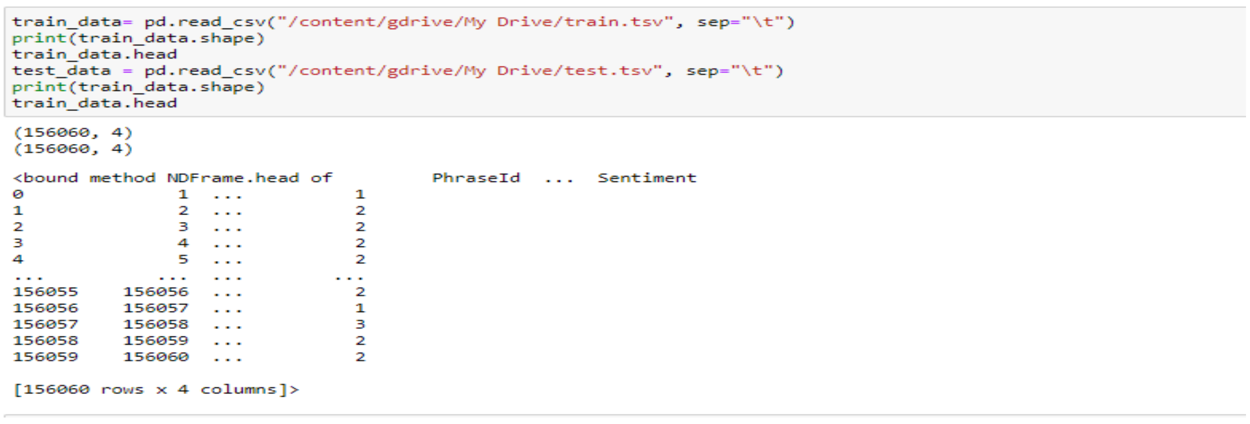
Implement text classification on the review's sentiment dataset using CNN model.

- Import all the necessary libraries.



```
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
from keras.models import Sequential
from keras.layers import Dense, Embedding, LSTM, SpatialDropout1D, Dropout, Conv1D, GlobalMaxPooling1D
from matplotlib import pyplot
from sklearn.model_selection import train_test_split
from keras.utils.np_utils import to_categorical
import re
from sklearn.preprocessing import LabelEncoder
from keras.callbacks import TensorBoard
from time import time
import matplotlib.pyplot as plt
```

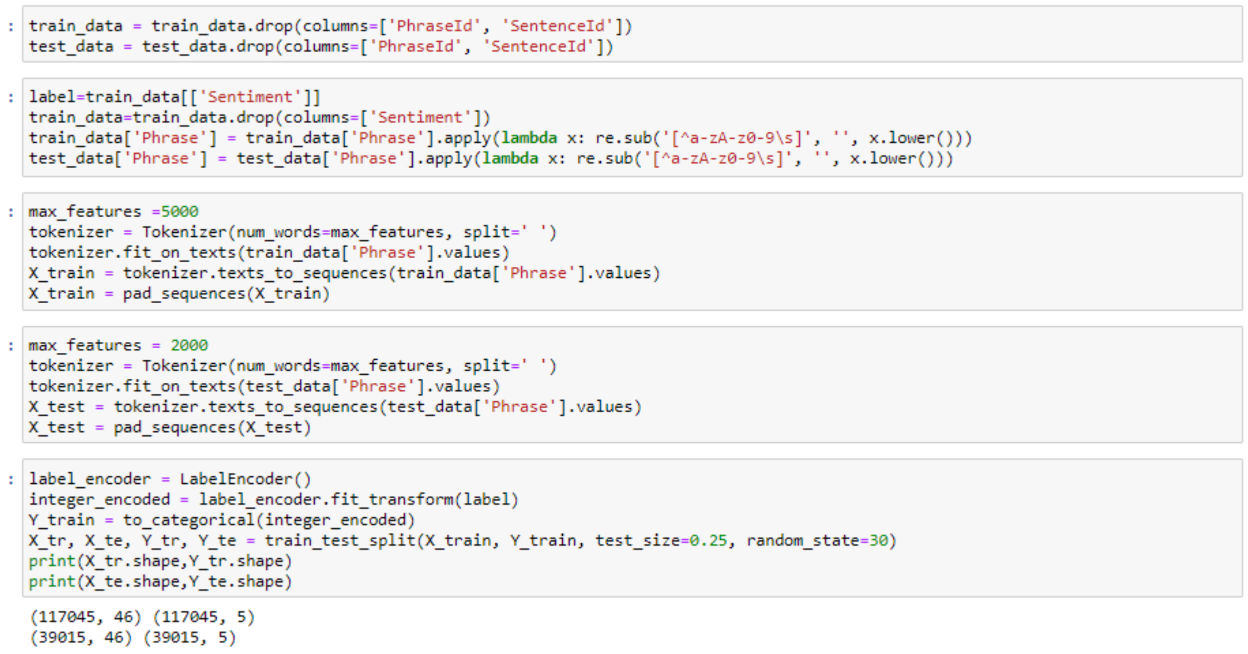
- Read train and test files.
- Print the content in train and test files.



```
train_data= pd.read_csv("/content/gdrive/My Drive/train.tsv", sep="\t")
print(train_data.shape)
train_data.head
test_data = pd.read_csv("/content/gdrive/My Drive/test.tsv", sep="\t")
print(train_data.shape)
train_data.head
```

```
(156060, 4)
(156060, 4)

<bound method NDFrame.head of          PhraseId  ...  Sentiment
0               1  ...          1
1               2  ...          2
2               3  ...          2
3               4  ...          2
4               5  ...          2
...           ...  ...        ...
156055    156056  ...          2
156056    156057  ...          1
156057    156058  ...          3
156058    156059  ...          2
156059    156060  ...          2

[156060 rows x 4 columns]>
```

- Drop multiple columns in pandas dataframe.
- Specify maximum number of features needed to be considered.
- Initialize tokenizer for train data, where it obtains top maximum number of features and filters remaining.
- Repeat the same procedure of test data.
- Specify label encoder.
- Fit the model.
- Split the data to train and test and print the content.



```
: train_data = train_data.drop(columns=['PhraseId', 'SentenceId'])
  test_data = test_data.drop(columns=['PhraseId', 'SentenceId'])
```

```
: label=train_data[['Sentiment']]
  train_data=train_data.drop(columns=['Sentiment'])
  train_data['Phrase'] = train_data['Phrase'].apply(lambda x: re.sub('[^a-zA-z0-9\s]', '', x.lower()))
  test_data['Phrase'] = test_data['Phrase'].apply(lambda x: re.sub('[^a-zA-z0-9\s]', '', x.lower()))
```

```
: max_features =5000
  tokenizer = Tokenizer(num_words=max_features, split=' ')
  tokenizer.fit_on_texts(train_data['Phrase'].values)
  X_train = tokenizer.texts_to_sequences(train_data['Phrase'].values)
  X_train = pad_sequences(X_train)
```

```
: max_features = 2000
  tokenizer = Tokenizer(num_words=max_features, split=' ')
  tokenizer.fit_on_texts(test_data['Phrase'].values)
  X_test = tokenizer.texts_to_sequences(test_data['Phrase'].values)
  X_test = pad_sequences(X_test)
```

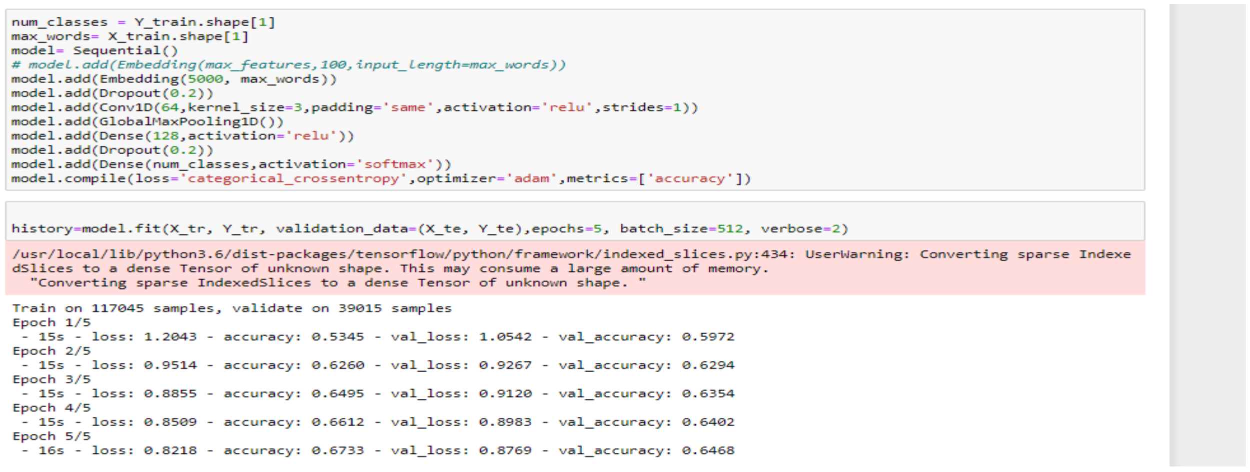
```
: label_encoder = LabelEncoder()
  integer_encoded = label_encoder.fit_transform(label)
  Y_train = to_categorical(integer_encoded)
  X_tr, X_te, Y_tr, Y_te = train_test_split(X_train, Y_train, test_size=0.25, random_state=30)
  print(X_tr.shape,Y_tr.shape)
  print(X_te.shape,Y_te.shape)
```

```
  (117045, 46) (117045, 5)
  (39015, 46) (39015, 5)
```

a. Include Embedding layer in the design of your models and report if that leads to a better performance.

- Import embedding from keras layers.
- Add embedding hidden layer.

- Fit the data and compile the model.



```
num_classes = Y_train.shape[1]
max_words= X_train.shape[1]
model= Sequential()
# model.add(Embedding(max_features,100,input_length=max_words))
model.add(Embedding(5000, max_words))
model.add(Dropout(0.2))
model.add(Conv1D(64,kernel_size=3,padding='same',activation='relu',strides=1))
model.add(GlobalMaxPooling1D())
model.add(Dense(128,activation='relu'))
model.add(Dropout(0.2))
model.add(Dense(num_classes,activation='softmax'))
model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
```

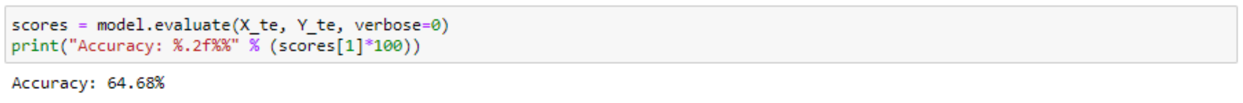
```
history=model.fit(X_tr, Y_tr, validation_data=(X_te, Y_te),epochs=5, batch_size=512, verbose=2)
```

```
/usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/indexed_slices.py:434: UserWarning: Converting sparse Indexe
dSlices to a dense Tensor of unknown shape. This may consume a large amount of memory.
  "Converting sparse IndexedSlices to a dense Tensor of unknown shape. "
```

```
Train on 117045 samples, validate on 39015 samples
Epoch 1/5
 - 15s - loss: 1.2043 - accuracy: 0.5345 - val_loss: 1.0542 - val_accuracy: 0.5972
Epoch 2/5
 - 15s - loss: 0.9514 - accuracy: 0.6260 - val_loss: 0.9267 - val_accuracy: 0.6294
Epoch 3/5
 - 15s - loss: 0.8855 - accuracy: 0.6495 - val_loss: 0.9120 - val_accuracy: 0.6354
Epoch 4/5
 - 15s - loss: 0.8509 - accuracy: 0.6612 - val_loss: 0.8983 - val_accuracy: 0.6402
Epoch 5/5
 - 16s - loss: 0.8218 - accuracy: 0.6733 - val_loss: 0.8769 - val_accuracy: 0.6468
```

b. Plot loss of the model and report if you see any overfitting problem.

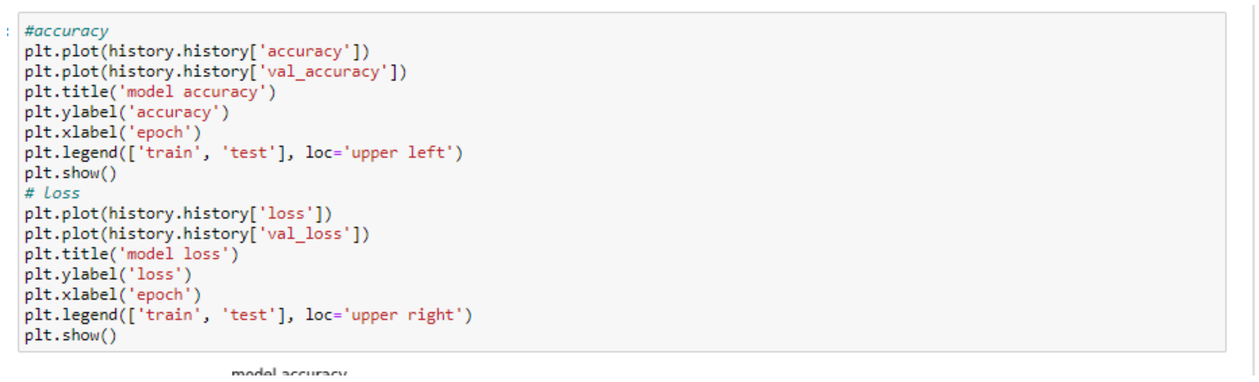
- Calculate accuracy and print accuracy value.



```
scores = model.evaluate(X_te, Y_te, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

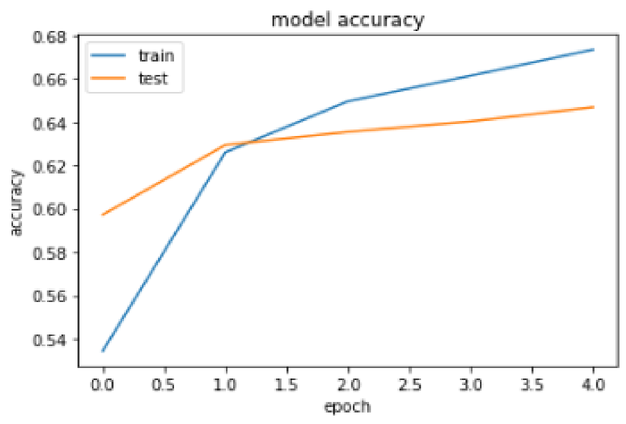
```
Accuracy: 64.68%
```

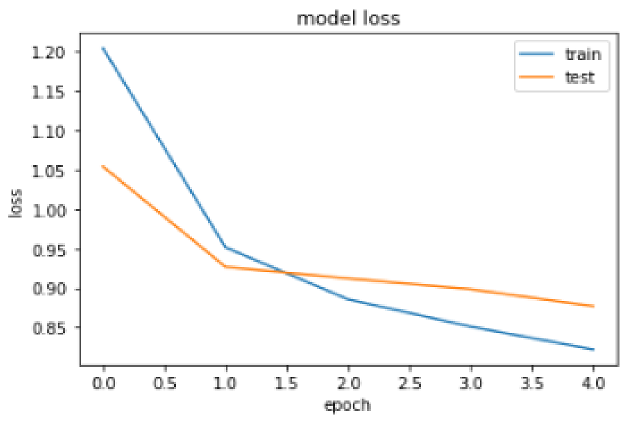
- Plot loss and accuracy.



```
#accuracy
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
# loss
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper right')
plt.show()
```

PLOTS:





Inference: From above procedure, we can draw that loss has increased and hence it is an overfitting model.

c. What techniques you can apply to fix overfitting model.

- Adding hidden embedding layers and learning rate reduced this overfitting problem.

```
from keras.optimizers import adam
#in the above model evaluation, loss has been increased hence it is overfitting model, to overcome we will include learning rate
s=adam(lr=0.001)
model1= Sequential()
model1.add(Embedding(5000,max_words))
model1.add(Dropout(0.2))
model1.add(Conv1D(64,kernel_size=3,padding='same',activation='relu',strides=1))
model1.add(GlobalMaxPooling1D())
model1.add(Dense(128,activation='relu'))
model1.add(Dropout(0.2))
model1.add(Dense(num_classes,activation='softmax'))
model1.compile(loss='binary_crossentropy',optimizer=s,metrics=['accuracy'])
```

```
history1=model1.fit(X_tr, Y_tr, validation_data=(X_te, Y_te),epochs=5, batch_size=51, verbose=1)
```

```
/usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/indexed_slices.py:434: UserWarning: Converting sparse Indexe
dSlices to a dense Tensor of unknown shape. This may consume a large amount of memory.
  "Converting sparse IndexedSlices to a dense Tensor of unknown shape. "
```

```
Train on 117045 samples, validate on 39015 samples
Epoch 1/5
117045/117045 [==============================] - 32s 273us/step - loss: 0.3496 - accuracy: 0.8466 - val_loss: 0.3227 - val_accu
racy: 0.8548
Epoch 2/5
117045/117045 [==============================] - 32s 270us/step - loss: 0.3067 - accuracy: 0.8633 - val_loss: 0.3084 - val_accu
racy: 0.8611
Epoch 3/5
117045/117045 [==============================] - 32s 272us/step - loss: 0.2915 - accuracy: 0.8704 - val_loss: 0.3050 - val_accu
racy: 0.8623
Epoch 4/5
117045/117045 [==============================] - 32s 270us/step - loss: 0.2810 - accuracy: 0.8758 - val_loss: 0.3050 - val_accu
racy: 0.8630
Epoch 5/5
117045/117045 [==============================] - 33s 280us/step - loss: 0.2738 - accuracy: 0.8787 - val_loss: 0.3043 - val_accu
racy: 0.8637
```

Question 2: Implement text classification on the 20news_group dataset using LSTM model

a. Include Embedding layer in the design of your models and report if that leads to a better performance.

- Import the necessary libraries.

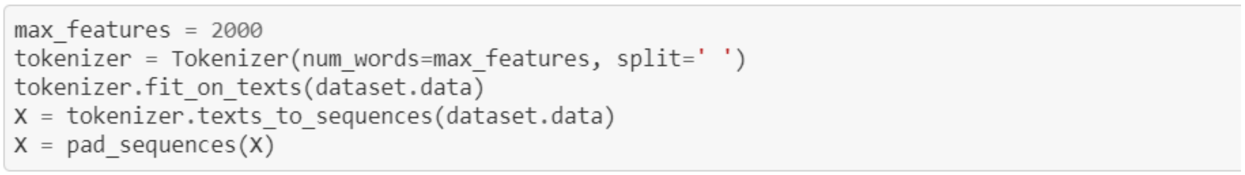
```
In [1]: from os import listdir
        from os.path import isfile, join
        import string
        from sklearn.datasets import load_files
        from sklearn.model_selection import train_test_split
        from sklearn import preprocessing
        from sklearn.model_selection import train_test_split
        from keras.preprocessing.sequence import pad_sequences
        from keras.preprocessing.text import Tokenizer
        from keras.utils.np_utils import to_categorical
        from sklearn.preprocessing import LabelEncoder
        from keras.models import Sequential
        from keras.layers import Dense, Embedding, LSTM, SpatialDropout1D,Input
```

- Then downloaded the 20 newgroups dataset and then load that into dataframe.

```
dataset = load_files("./20_newsgroups",
                             description= None, load_content = True,
                             encoding='latin1', decode_error='strict', shuffle=True,
                             random_state=42)
len(dataset.data)
```

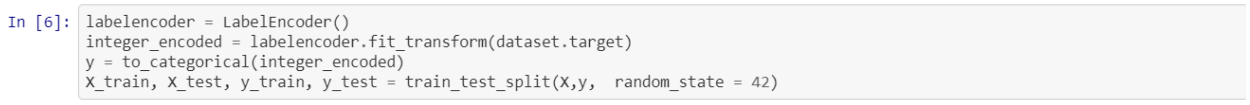
```
538
```

- Apply the tokenizer to the dataset.



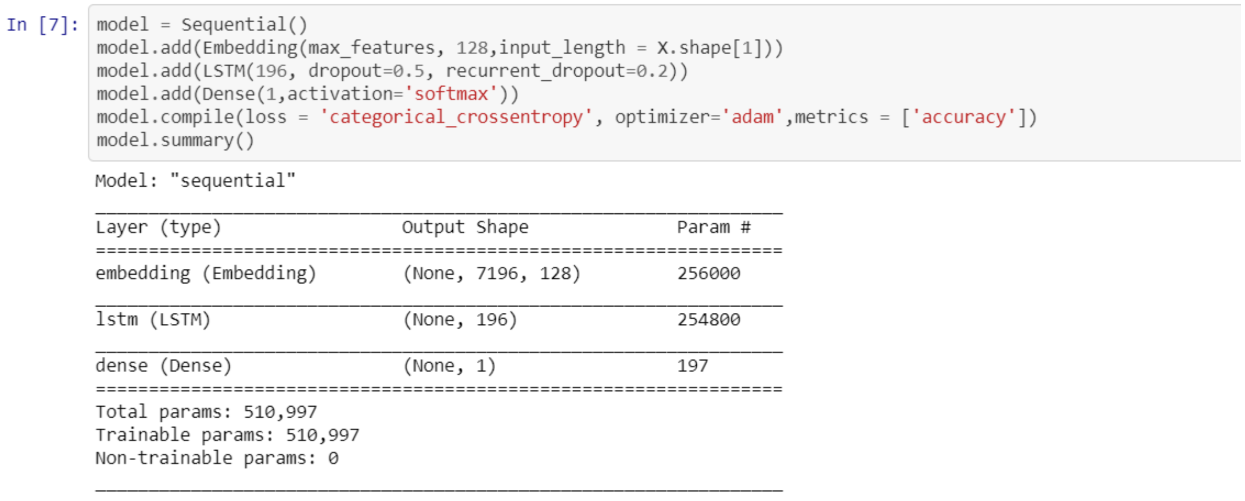
```
max_features = 2000
tokenizer = Tokenizer(num_words=max_features, split=' ')
tokenizer.fit_on_texts(dataset.data)
X = tokenizer.texts_to_sequences(dataset.data)
X = pad_sequences(X)
```

- Apply label encoder to convert categorical data into numeric features.



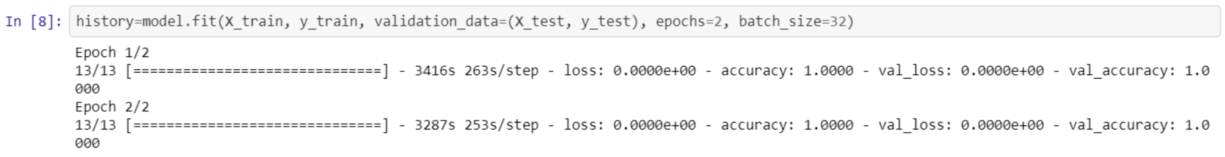
```
In [6]: labelencoder = LabelEncoder()
        integer_encoded = labelencoder.fit_transform(dataset.target)
        y = to_categorical(integer_encoded)
        X_train, X_test, y_train, y_test = train_test_split(X,y,  random_state = 42)
```

- Initialize the LSTM model and add the embedding layer.



```
In [7]: model = Sequential()
        model.add(Embedding(max_features, 128,input_length = X.shape[1]))
        model.add(LSTM(196, dropout=0.5, recurrent_dropout=0.2))
        model.add(Dense(1,activation='softmax'))
        model.compile(loss = 'categorical_crossentropy', optimizer='adam',metrics = ['accuracy'])
        model.summary()

        Model: "sequential"
        _________________________________________________________________
        Layer (type)                 Output Shape              Param #
        =================================================================
        embedding (Embedding)        (None, 7196, 128)         256000
        _________________________________________________________________
        lstm (LSTM)                  (None, 196)               254800
        _________________________________________________________________
        dense (Dense)                (None, 1)                 197
        =================================================================
        Total params: 510,997
        Trainable params: 510,997
        Non-trainable params: 0
        _________________________________________________________________
```

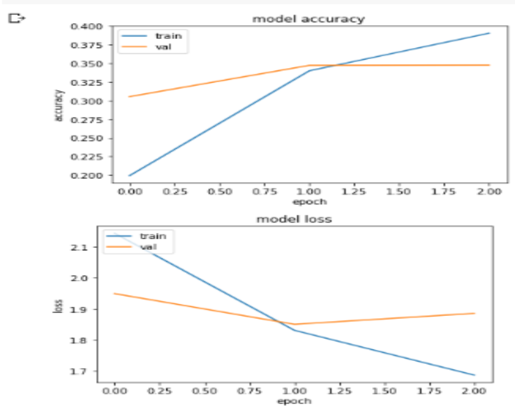
- Plot the history object and predict the loss and accuracy.



```
In [8]: history=model.fit(X_train, y_train, validation_data=(X_test, y_test), epochs=2, batch_size=32)

        Epoch 1/2
        13/13 [==============================] - 3416s 263s/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0
        000
        Epoch 2/2
        13/13 [==============================] - 3287s 253s/step - loss: 0.0000e+00 - accuracy: 1.0000 - val_loss: 0.0000e+00 - val_accuracy: 1.0
        000
```

b. Plot loss of the model and report if you see any overfitting problem.

- Plot the loss and accuracy.



Question 3: Implement image classification with CNN model, using a dataset.

a. **Report your classification result with and without doing scaling**

- Install scikit-image and opencv-python.
- Import all the necessary libraries.

```
import math
import matplotlib.pyplot as plt
import scipy
import cv2
import numpy as np
import glob
import os
import pandas as pd
import tensorflow as tf
import itertools
import random
from random import shuffle
from tqdm import tqdm
from PIL import Image
from scipy import ndimage
from pathlib import Path
from sklearn.metrics import classification_report, confusion_matrix
from sklearn import metrics
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D, Activation, Dropout, Flatten, Dense
from keras.callbacks import ModelCheckpoint, EarlyStopping
```

- Now read train and the test files.
- From labels.txt, add the label names to the column.

```
train_dir = Path('./Spieces data/training/training/')
test_dir = Path('./Spieces data/validation/validation/')
```

```
cols = ['Label','Latin Name', 'Common Name','Train Images', 'Validation Images']
labels = pd.read_csv("./Spieces data/monkey_labels.txt", names=cols, skiprows=1)
labels
```

	Label	Latin Name	Common Name	Train Images	Validation Images
0	n0	alouatta_palliata\t	mantled_howler	131	26
1	n1	erythrocebus_patas\t	patas_monkey	139	28
2	n2	cacajao_calvus\t	bald_uakari	137	27
3	n3	macaca_fuscata\t	japanese_macaque	152	30
4	n4	cebuella_pygmea\t	pygmy_marmoset	131	26
5	n5	cebus_capucinus\t	white_headed_capuchin	141	28
6	n6	mico_argentatus\t	silvery_marmoset	132	26
7	n7	saimiri_sciureus\t	common_squirrel_monkey	142	28
8	n8	aotus_nigriceps\t	black_headed_night_monkey	133	27
9	n9	trachypithecus_johnii	nilgiri_langur	132	26

- Print the content in 'common name' column to check.

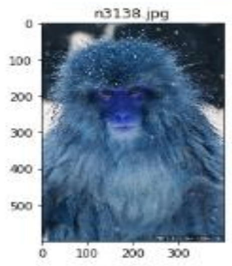
```
labels = labels['Common Name']
labels
```

```
0      mantled_howler
1      patas_monkey
2      bald_uakari
3      japanese_macaque
4      pygmy_marmoset
5      white_headed_capuchin
6      silvery_marmoset
7      common_squirrel_monkey
8      black_headed_night_monkey
9      nilgiri_langur
Name: Common Name, dtype: object
```

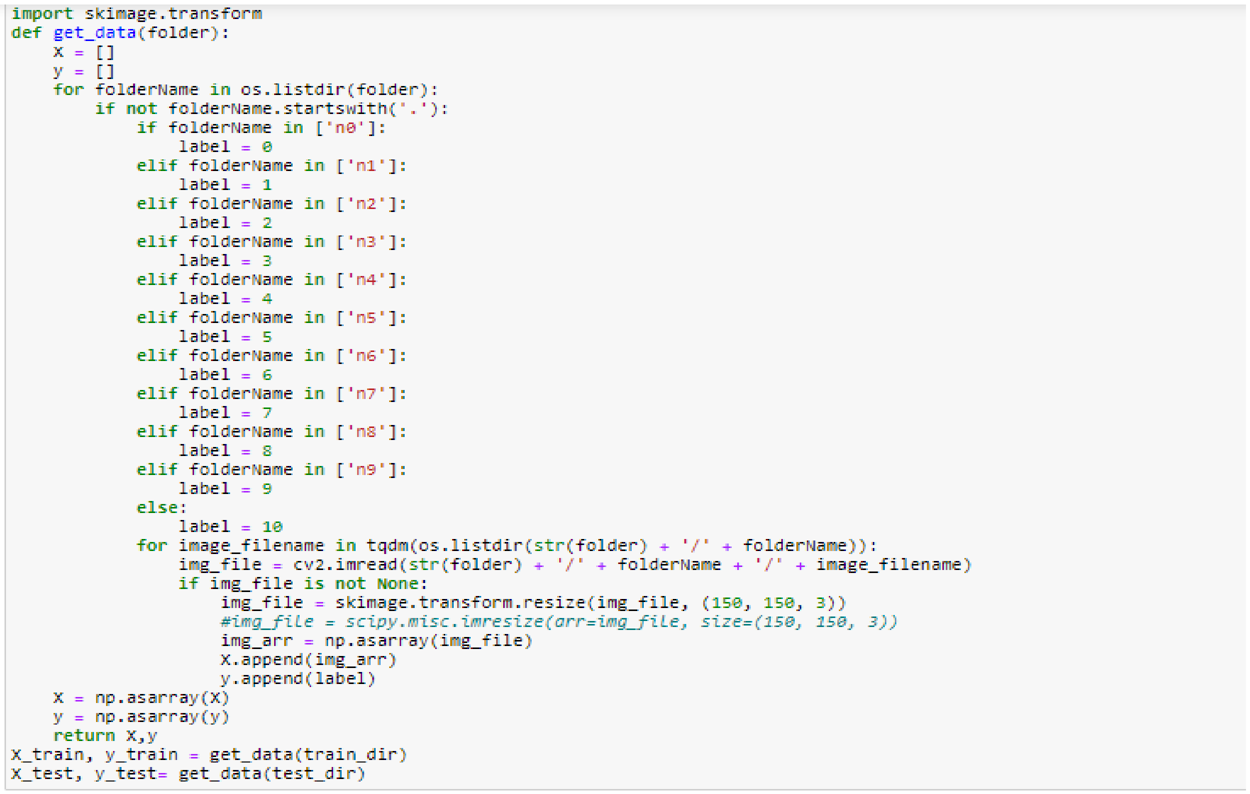
- Now Print the images with the different labels.

```
def image_show(num_image,label):
    for i in range(num_image):
        imgdir = Path('./Spieces data/training/training/' + label)
        #print(imgdir)
        imgfile = random.choice(os.listdir(imgdir))
        #print(imgfile)
        img = cv2.imread('./Spieces data/training/training/'+ label +'/'+ imgfile)
       # print(img.shape)
        #print(label)
        plt.figure(i)
        plt.imshow(img)
        plt.title(imgfile)
    plt.show()
```

```
print(labels[3])
image_show(3,'n3')
```

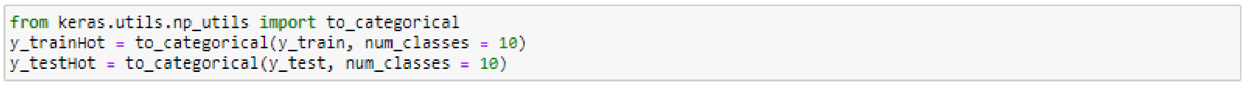


- Import tqdm and then append all the foldernames to read the image data and store in array.

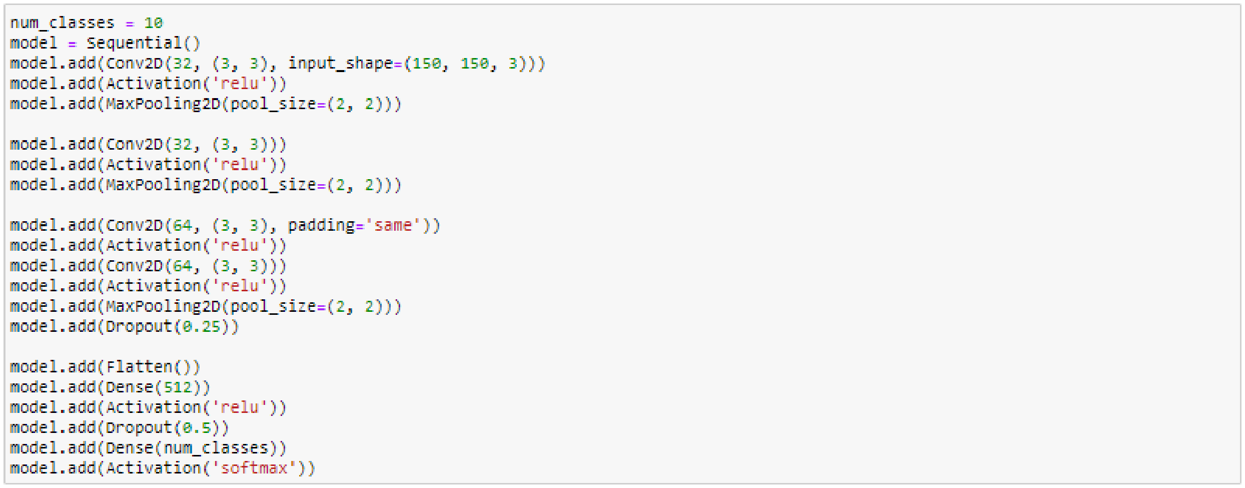


```
import skimage.transform
def get_data(folder):
    X = []
    y = []
    for folderName in os.listdir(folder):
        if not folderName.startswith('.'):
            if folderName in ['n0']:
                label = 0
            elif folderName in ['n1']:
                label = 1
            elif folderName in ['n2']:
                label = 2
            elif folderName in ['n3']:
                label = 3
            elif folderName in ['n4']:
                label = 4
            elif folderName in ['n5']:
                label = 5
            elif folderName in ['n6']:
                label = 6
            elif folderName in ['n7']:
                label = 7
            elif folderName in ['n8']:
                label = 8
            elif folderName in ['n9']:
                label = 9
            else:
                label = 10
            for image_filename in tqdm(os.listdir(str(folder) + '/' + folderName)):
                img_file = cv2.imread(str(folder) + '/' + folderName + '/' + image_filename)
                if img_file is not None:
                    img_file = skimage.transform.resize(img_file, (150, 150, 3))
                    #img_file = scipy.misc.imresize(arr=img_file, size=(150, 150, 3))
                    img_arr = np.asarray(img_file)
                    X.append(img_arr)
                    y.append(label)
    X = np.asarray(X)
    y = np.asarray(y)
    return X,y
X_train, y_train = get_data(train_dir)
X_test, y_test= get_data(test_dir)
```

- Using hot encoder, convert categorical columns to numerical columns.
- Initialize the sequential model.

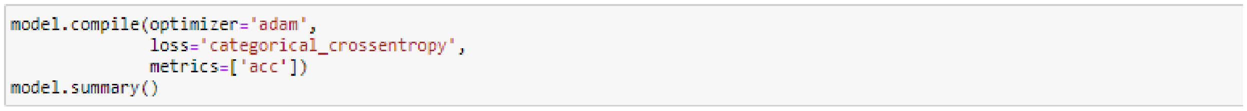


```
from keras.utils.np_utils import to_categorical
y_trainHot = to_categorical(y_train, num_classes = 10)
y_testHot = to_categorical(y_test, num_classes = 10)
```

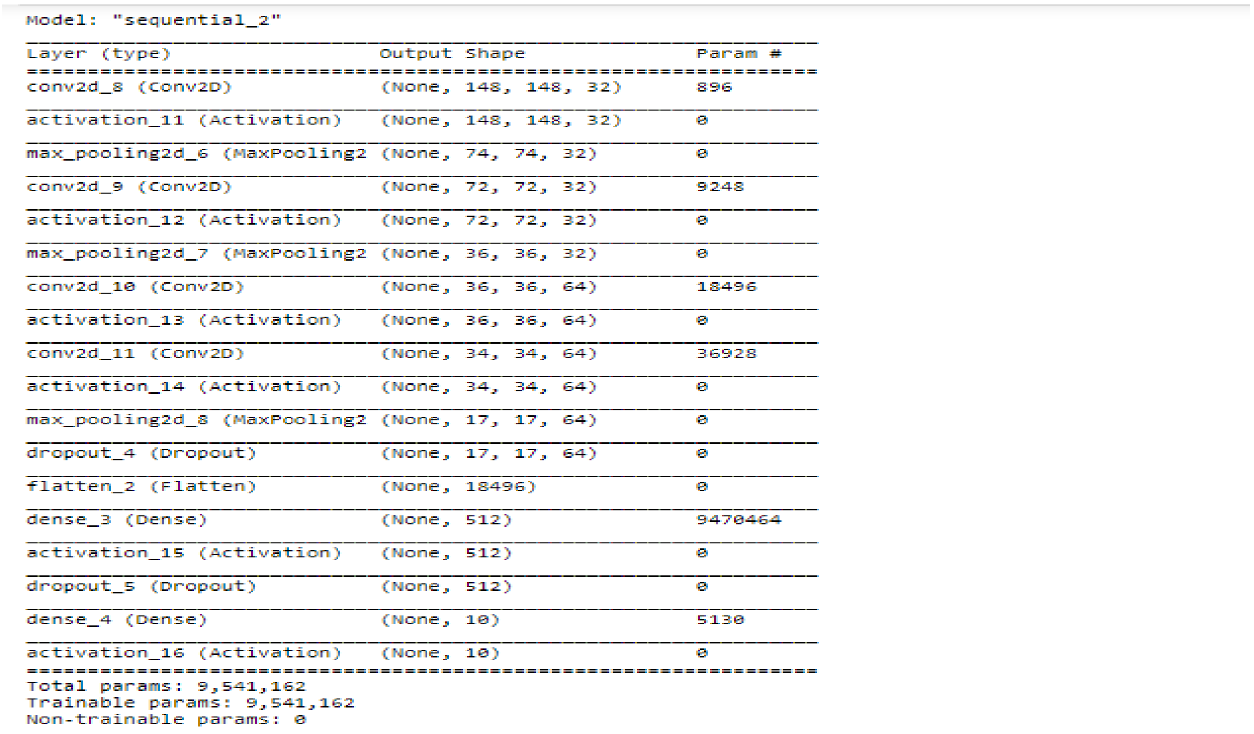


```
num_classes = 10
model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape=(150, 150, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))

model.add(Conv2D(64, (3, 3), padding='same'))
model.add(Activation('relu'))
model.add(Conv2D(64, (3, 3)))
model.add(Activation('relu'))
model.add(MaxPooling2D(pool_size=(2, 2)))
model.add(Dropout(0.25))

model.add(Flatten())
model.add(Dense(512))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(num_classes))
model.add(Activation('softmax'))
```

- Compile the model and get the summary.



```
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['acc'])
model.summary()
```



```
Model: "sequential_2"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_8 (Conv2D)            (None, 148, 148, 32)      896
_________________________________________________________________
activation_11 (Activation)   (None, 148, 148, 32)      0
_________________________________________________________________
max_pooling2d_6 (MaxPooling2 (None, 74, 74, 32)        0
_________________________________________________________________
conv2d_9 (Conv2D)            (None, 72, 72, 32)        9248
_________________________________________________________________
activation_12 (Activation)   (None, 72, 72, 32)        0
_________________________________________________________________
max_pooling2d_7 (MaxPooling2 (None, 36, 36, 32)        0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 36, 36, 64)        18496
_________________________________________________________________
activation_13 (Activation)   (None, 36, 36, 64)        0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 34, 34, 64)        36928
_________________________________________________________________
activation_14 (Activation)   (None, 34, 34, 64)        0
_________________________________________________________________
max_pooling2d_8 (MaxPooling2 (None, 17, 17, 64)        0
_________________________________________________________________
dropout_4 (Dropout)          (None, 17, 17, 64)        0
_________________________________________________________________
flatten_2 (Flatten)          (None, 18496)             0
_________________________________________________________________
dense_3 (Dense)              (None, 512)               9470464
_________________________________________________________________
activation_15 (Activation)   (None, 512)               0
_________________________________________________________________
dropout_5 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_4 (Dense)              (None, 10)                5130
_________________________________________________________________
activation_16 (Activation)   (None, 10)                0
=================================================================
Total params: 9,541,162
Trainable params: 9,541,162
Non-trainable params: 0
```

- Plot the history object and predict the accuracy and loss.

```
history = model.fit(X_train,y_trainHot, epochs=5, validation_data=(X_test,y_testHot), verbose=1,batch_size = 1)
# y_testHot
```

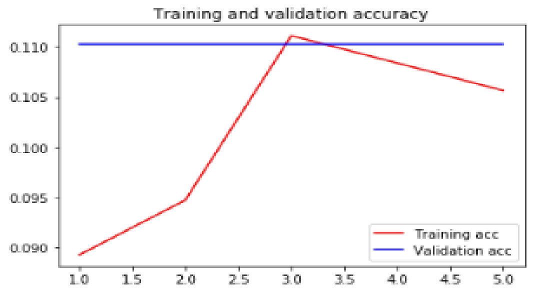
```
Epoch 1/5
1098/1098 [==============================] - 120s 109ms/step - loss: 2.3119 - acc: 0.0893 - val_loss: 2.3020 - val_acc: 0.1103
Epoch 2/5
1098/1098 [==============================] - 121s 110ms/step - loss: 2.3038 - acc: 0.0947 - val_loss: 2.3017 - val_acc: 0.1103
Epoch 3/5
1098/1098 [==============================] - 124s 113ms/step - loss: 2.3036 - acc: 0.1111 - val_loss: 2.3016 - val_acc: 0.1103
Epoch 4/5
1098/1098 [==============================] - 124s 113ms/step - loss: 2.3032 - acc: 0.1084 - val_loss: 2.3016 - val_acc: 0.1103
Epoch 5/5
1098/1098 [==============================] - 126s 115ms/step - loss: 2.3039 - acc: 0.1056 - val_loss: 2.3016 - val_acc: 0.1103
```

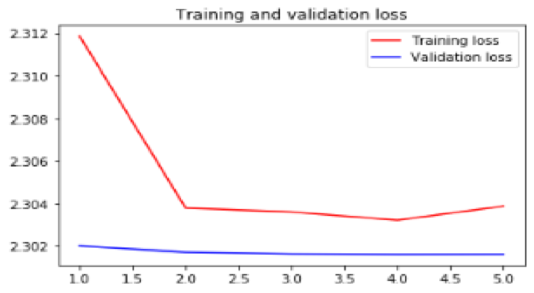
```
score = model.evaluate(X_test,y_testHot, verbose=0)
print('accuracy:', score[1], '\n')
```

```
accuracy: 0.11029411852359772
```

- Now plot the loss and accuracy.

```
acc = history.history['acc']
val_acc = history.history['val_acc']
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(1, len(acc) + 1)

plt.title('Training and validation accuracy')
plt.plot(epochs, acc, 'red', label='Training acc')
plt.plot(epochs, val_acc, 'blue', label='Validation acc')
plt.legend()

plt.figure()
plt.title('Training and validation loss')
plt.plot(epochs, loss, 'red', label='Training loss')
plt.plot(epochs, val_loss, 'blue', label='Validation loss')

plt.legend()

plt.show()
```





- Fit it to the standard scaler and then initialize the model.

```
#with scaling
from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
scaler.fit(y_trainHot)
y_trainHot1 = scaler.transform(y_trainHot)
y_testHot1 = scaler.transform(y_testHot)
```

```
num_classes = 10
model1 = Sequential()
model1.add(Conv2D(32, (3, 3), input_shape=(150, 150, 3)))
model1.add(Activation('relu'))
model1.add(MaxPooling2D(pool_size=(2, 2)))

model1.add(Conv2D(32, (3, 3)))
model.add(Activation('relu'))
model1.add(MaxPooling2D(pool_size=(2, 2)))

model1.add(Conv2D(64, (3, 3), padding='same'))
model.add(Activation('relu'))
model1.add(Conv2D(64, (3, 3)))
model1.add(Activation('relu'))
model1.add(MaxPooling2D(pool_size=(2, 2)))
model1.add(Dropout(0.25))

model1.add(Flatten())
model1.add(Dense(512))
model1.add(Activation('relu'))
model1.add(Dropout(0.5))
model1.add(Dense(num_classes))
model1.add(Activation('softmax'))
```

```
model1.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['acc'])
model1.summary()
```

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_20 (Conv2D)           (None, 148, 148, 32)      896
_________________________________________________________________
activation_29 (Activation)   (None, 148, 148, 32)      0
_________________________________________________________________
max_pooling2d_15 (MaxPooling (None, 74, 74, 32)        0
_________________________________________________________________
conv2d_21 (Conv2D)           (None, 72, 72, 32)        9248
_________________________________________________________________
max_pooling2d_16 (MaxPooling (None, 36, 36, 32)        0
_________________________________________________________________
conv2d_22 (Conv2D)           (None, 36, 36, 64)        18496
_________________________________________________________________
conv2d_23 (Conv2D)           (None, 34, 34, 64)        36928
_________________________________________________________________
activation_32 (Activation)   (None, 34, 34, 64)        0
_________________________________________________________________
max_pooling2d_17 (MaxPooling (None, 17, 17, 64)        0
_________________________________________________________________
dropout_10 (Dropout)         (None, 17, 17, 64)        0
_________________________________________________________________
flatten_5 (Flatten)          (None, 18496)             0
_________________________________________________________________
dense_9 (Dense)              (None, 512)               9470464
_________________________________________________________________
activation_33 (Activation)   (None, 512)               0
_________________________________________________________________
dropout_11 (Dropout)         (None, 512)               0
_________________________________________________________________
dense_10 (Dense)             (None, 10)                5130
_________________________________________________________________
activation_34 (Activation)   (None, 10)                0
=================================================================
Total params: 9,541,162
Trainable params: 9,541,162
Non-trainable params: 0
```

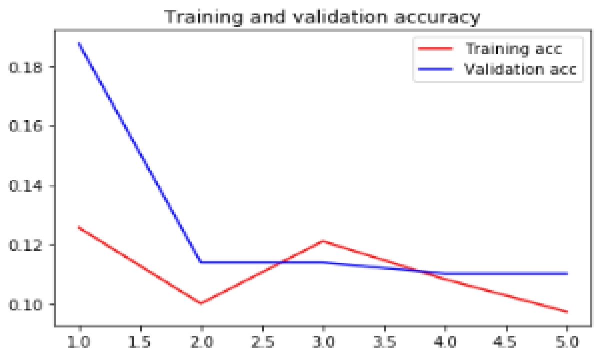
- Plot the history object and predict the loss and accuracy.

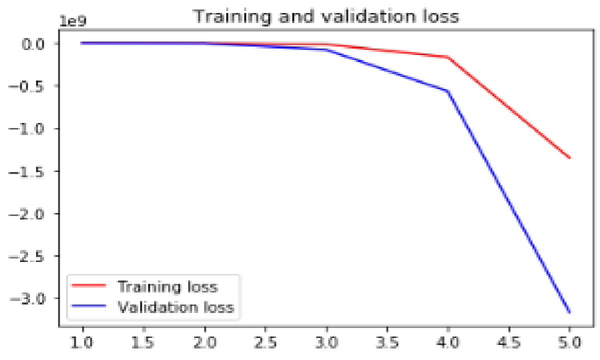
```
history1 = model1.fit(X_train,y_trainHot1, epochs=5, validation_data=(X_test,y_testHot1), verbose=1)
```

```
Epoch 1/5
35/35 [==============================] - 44s 1s/step - loss: -1292.2365 - acc: 0.1257 - val_loss: -10082.3311 - val_acc: 0.1875
Epoch 2/5
35/35 [==============================] - 47s 1s/step - loss: -343748.3125 - acc: 0.1002 - val_loss: -2857977.0000 - val_acc: 0.
1140
Epoch 3/5
35/35 [==============================] - 48s 1s/step - loss: -13003757.0000 - acc: 0.1211 - val_loss: -75697256.0000 - val_acc:
0.1140
Epoch 4/5
35/35 [==============================] - 49s 1s/step - loss: -164611680.0000 - acc: 0.1084 - val_loss: -566307328.0000 - val_ac
c: 0.1103
Epoch 5/5
35/35 [==============================] - 49s 1s/step - loss: -1352128896.0000 - acc: 0.0974 - val_loss: -3170531072.0000 - val_
acc: 0.1103
```

- Plot the loss and accuracy.

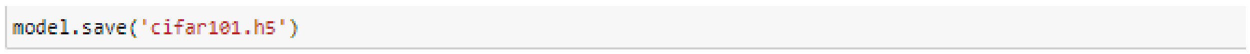
```
acc = history1.history['acc']
val_acc = history1.history['val_acc']
loss = history1.history['loss']
val_loss = history1.history['val_loss']
epochs = range(1, len(acc) + 1)

plt.title('Training and validation accuracy')
plt.plot(epochs, acc, 'red', label='Training acc')
plt.plot(epochs, val_acc, 'blue', label='Validation acc')
plt.legend()

plt.figure()
plt.title('Training and validation loss')
plt.plot(epochs, loss, 'red', label='Training loss')
plt.plot(epochs, val_loss, 'blue', label='Validation loss')

plt.legend()

plt.show()
```



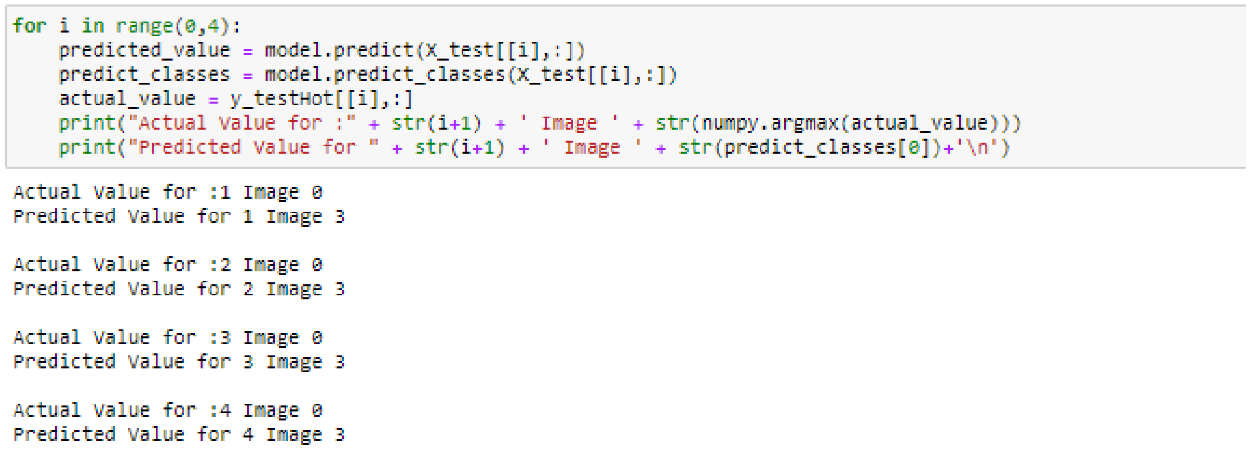


b. Save the model and then predict on one of the test data. Report the prediction and check if it has been predicted correctly or not.

- Now save the model and print the actual and predicted values for images.)



```
model.save('cifar101.h5')
```

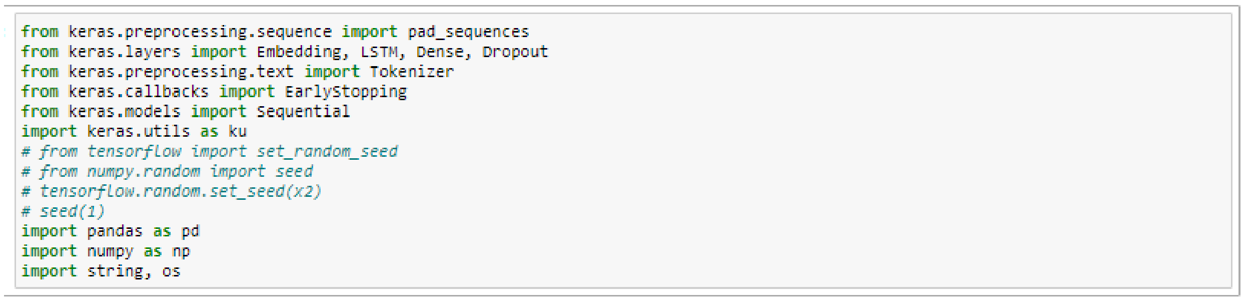


```
for i in range(0,4):
    predicted_value = model.predict(X_test[[i],:])
    predict_classes = model.predict_classes(X_test[[i],:])
    actual_value = y_testHot[[i],:]
    print("Actual Value for :" + str(i+1) + ' Image ' + str(numpy.argmax(actual_value)))
    print("Predicted Value for " + str(i+1) + ' Image ' + str(predict_classes[0])+'\n')
```

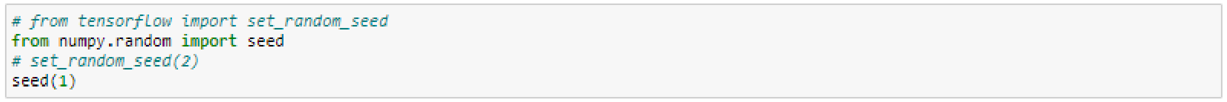
```
Actual Value for :1 Image 0
Predicted Value for 1 Image 3

Actual Value for :2 Image 0
Predicted Value for 2 Image 3

Actual Value for :3 Image 0
Predicted Value for 3 Image 3

Actual Value for :4 Image 0
Predicted Value for 4 Image 3
```

Question 4: The purpose of this question is to learn about text generation. Use New York Times Comments and Headlines to train a text generation language model which can be used to generate News Headlines.

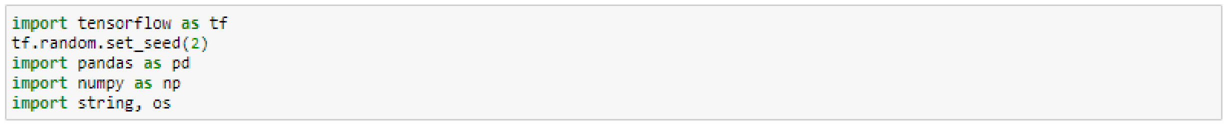
- Import all the necessary libraries.



```
from keras.preprocessing.sequence import pad_sequences
from keras.layers import Embedding, LSTM, Dense, Dropout
from keras.preprocessing.text import Tokenizer
from keras.callbacks import EarlyStopping
from keras.models import Sequential
import keras.utils as ku
# from tensorflow import set_random_seed
# from numpy.random import seed
# tensorflow.random.set_seed(x2)
# seed(1)
import pandas as pd
import numpy as np
import string, os
```



```
# from tensorflow import set_random_seed
from numpy.random import seed
# set_random_seed(2)
seed(1)
```



```
import tensorflow as tf
tf.random.set_seed(2)
import pandas as pd
import numpy as np
import string, os
```

- Read the files.

```
dir = './newyork_headline/'
headlines = []
for filename in os.listdir(dir):
    if 'Articles' in filename:
        print(filename)
        articledata = pd.read_csv(dir + filename)
        headlines.extend(list(articledata.headline.values))


headlines = [h for h in headlines if h != "Unknown"]
len(headlines)
```

```
ArticlesFeb2017.csv
ArticlesFeb2018.csv
ArticlesJan2017.csv
ArticlesJan2018.csv

3493
```

- Obtain train and test data.
- Normalize the text data.
- Print ten headlines.

```
def clean_text(txt):
    txt = "".join(v for v in txt if v not in string.punctuation).lower()
    txt = txt.encode("utf8").decode("ascii",'ignore')
    return txt

data = [clean_text(x) for x in headlines]
data[:10]
```

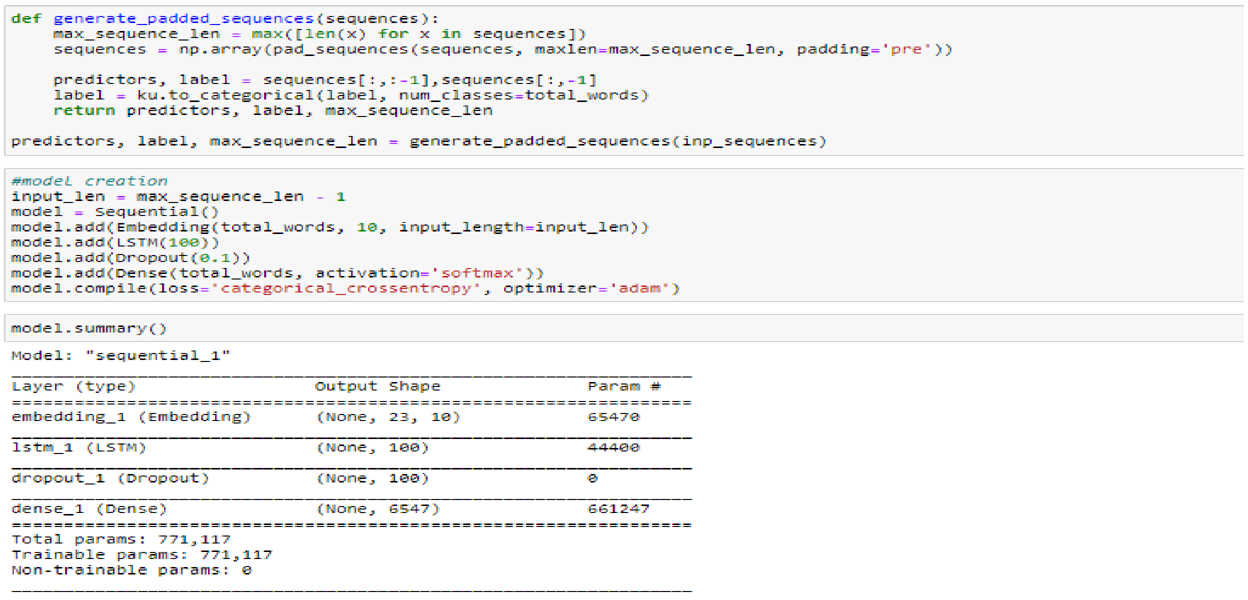
```
['nfl vs politics has been battle all season long',
 'voice vice veracity',
 'a standups downward slide',
 'new york today a groundhog has her day',
 'a swimmers communion with the ocean',
 'trail activity',
 'super bowl',
 'trumps mexican shakedown',
 'pences presidential pet',
 'fruit of a poison tree']
```

- Initialize tokenizer.
- Input sequences will be created using the list of tokens.
- Obtain sequence of tokens and print ten of them.

```
tokenizer = Tokenizer()

def get_sequence_of_tokens(data):
    ## tokenization
    tokenizer.fit_on_texts(data)
    total_words = len(tokenizer.word_index) + 1
    print(total_words)

    input_sequences = []
    for line in data:
        token_list = tokenizer.texts_to_sequences([line])[0]
        for i in range(1, len(token_list)):
            n_gram_sequence = token_list[:i+1]
            input_sequences.append(n_gram_sequence)
    return input_sequences, total_words

inp_sequences, total_words = get_sequence_of_tokens(data)
inp_sequences[:10]
```

```
6547

[[636, 139],
 [636, 139, 140],
 [636, 139, 140, 84],
 [636, 139, 140, 84, 815],
 [636, 139, 140, 84, 815, 292],
 [636, 139, 140, 84, 815, 292, 62],
 [636, 139, 140, 84, 815, 292, 62, 46],
 [636, 139, 140, 84, 815, 292, 62, 46, 129],
 [428, 2539],
 [428, 2539, 2540]]
```

- Will be creating predictors and labels.
- Padding the sequences and obtaining the predictors and targets.
- Create a model with input sequence length as one less than maximum sequence length.
- Initialize LSTM model.
- As we have multiple classes initialize softmax activation layer.
- Then obtain model summary.

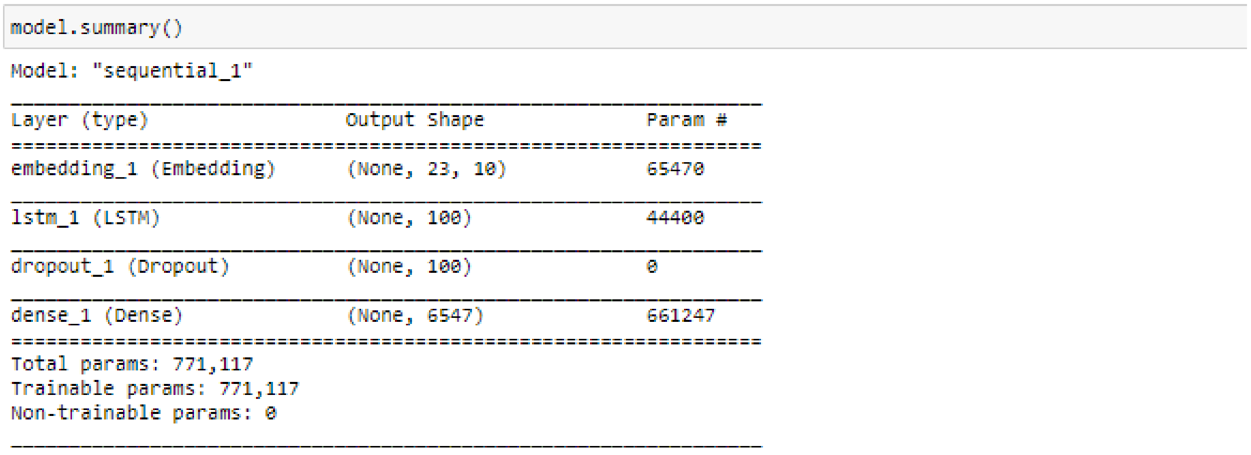


```
def generate_padded_sequences(sequences):
    max_sequence_len = max([len(x) for x in sequences])
    sequences = np.array(pad_sequences(sequences, maxlen=max_sequence_len, padding='pre'))

    predictors, label = sequences[:,:-1],sequences[:,-1]
    label = ku.to_categorical(label, num_classes=total_words)
    return predictors, label, max_sequence_len

predictors, label, max_sequence_len = generate_padded_sequences(inp_sequences)
```

```
#model creation
input_len = max_sequence_len - 1
model = Sequential()
model.add(Embedding(total_words, 10, input_length=input_len))
model.add(LSTM(100))
model.add(Dropout(0.1))
model.add(Dense(total_words, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam')
```

```
model.summary()
```

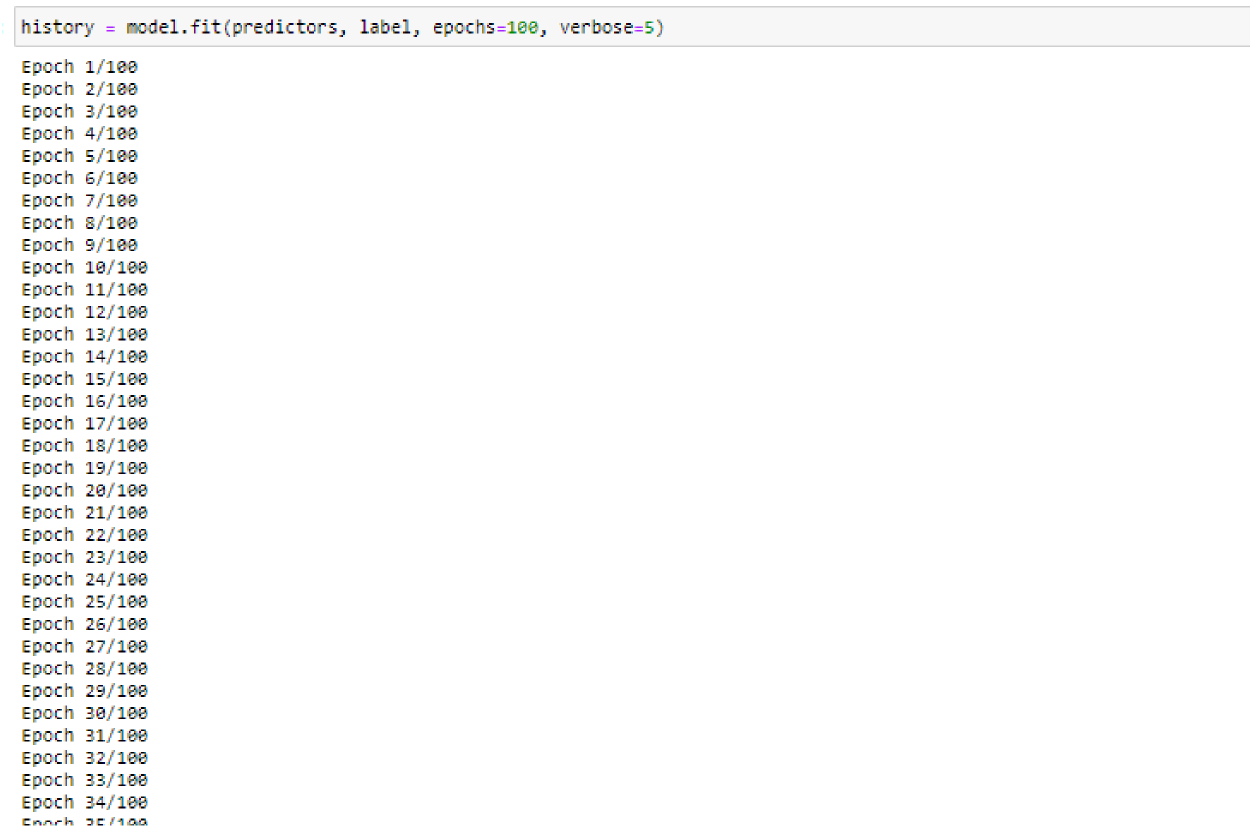
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 23, 10)            65470
_________________________________________________________________
lstm_1 (LSTM)                (None, 100)               44400
_________________________________________________________________
dropout_1 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_1 (Dense)              (None, 6547)              661247
=================================================================
Total params: 771,117
Trainable params: 771,117
Non-trainable params: 0
_________________________________________________________________
```



```
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 23, 10)            65470
_________________________________________________________________
lstm_1 (LSTM)                (None, 100)               44400
_________________________________________________________________
dropout_1 (Dropout)          (None, 100)               0
_________________________________________________________________
dense_1 (Dense)              (None, 6547)              661247
=================================================================
Total params: 771,117
Trainable params: 771,117
Non-trainable params: 0
_________________________________________________________________
```

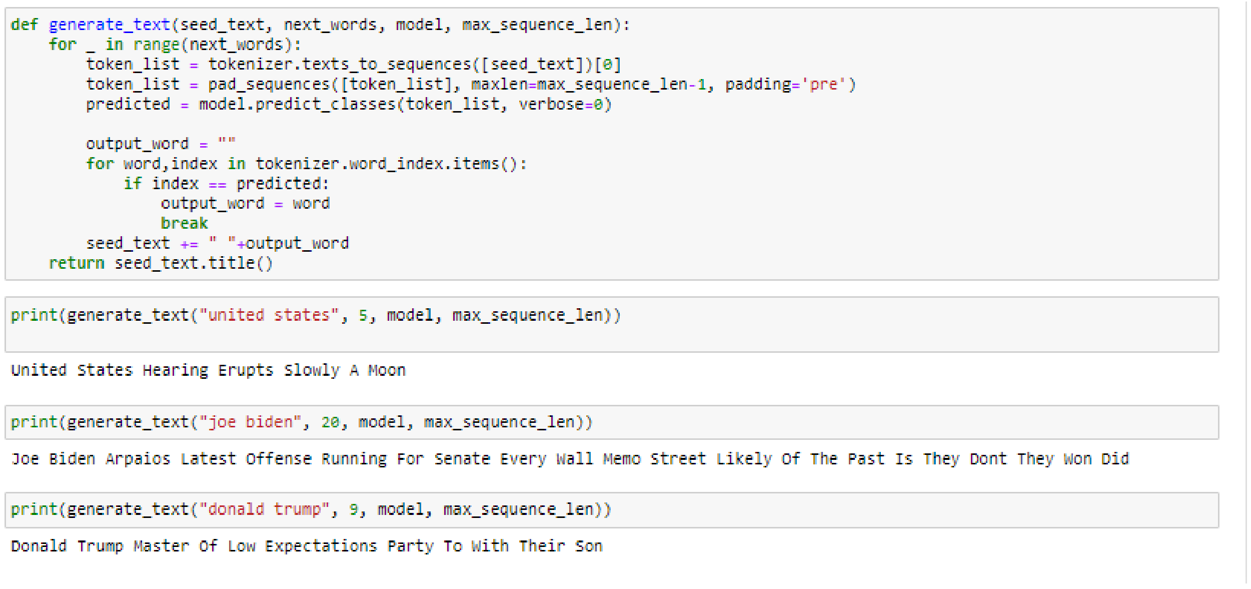
- Now fit the model on training data.
- Return the history object which hold the values of loss and accuracy metrics values.



```
history = model.fit(predictors, label, epochs=100, verbose=5)
```

```
Epoch 1/100
Epoch 2/100
Epoch 3/100
Epoch 4/100
Epoch 5/100
Epoch 6/100
Epoch 7/100
Epoch 8/100
Epoch 9/100
Epoch 10/100
Epoch 11/100
Epoch 12/100
Epoch 13/100
Epoch 14/100
Epoch 15/100
Epoch 16/100
Epoch 17/100
Epoch 18/100
Epoch 19/100
Epoch 20/100
Epoch 21/100
Epoch 22/100
Epoch 23/100
Epoch 24/100
Epoch 25/100
Epoch 26/100
Epoch 27/100
Epoch 28/100
Epoch 29/100
Epoch 30/100
Epoch 31/100
Epoch 32/100
Epoch 33/100
Epoch 34/100
```

- Create a model and train the model which can generate sequence of next words.
- Consider this model and predict the token list.



```
def generate_text(seed_text, next_words, model, max_sequence_len):
    for _ in range(next_words):
        token_list = tokenizer.texts_to_sequences([seed_text])[0]
        token_list = pad_sequences([token_list], maxlen=max_sequence_len-1, padding='pre')
        predicted = model.predict_classes(token_list, verbose=0)

        output_word = ""
        for word,index in tokenizer.word_index.items():
            if index == predicted:
                output_word = word
                break
        seed_text += " "+output_word
    return seed_text.title()
```

```
print(generate_text("united states", 5, model, max_sequence_len))
```

```
United States Hearing Erupts Slowly A Moon
```

```
print(generate_text("joe biden", 20, model, max_sequence_len))
```

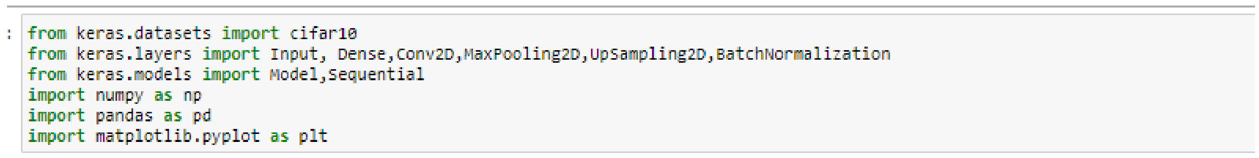
```
Joe Biden Arpaios Latest Offense Running For Senate Every Wall Memo Street Likely Of The Past Is They Dont They Won Did
```

```
print(generate_text("donald trump", 9, model, max_sequence_len))
```

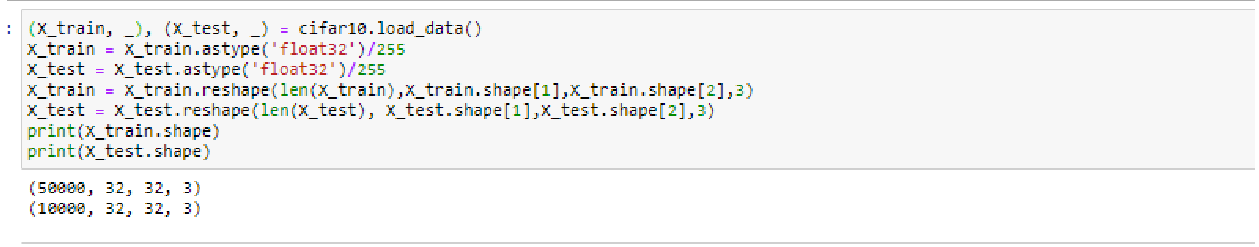
```
Donald Trump Master Of Low Expectations Party To With Their Son
```

Question 5: Apply Autoencoder on the Cifar_10datasetand then pass the result of Autoencoder to CNN or LSTM or three layers model to classify data.

- First import all the necessary libraries.
- Normalize train and test data and then reshape.

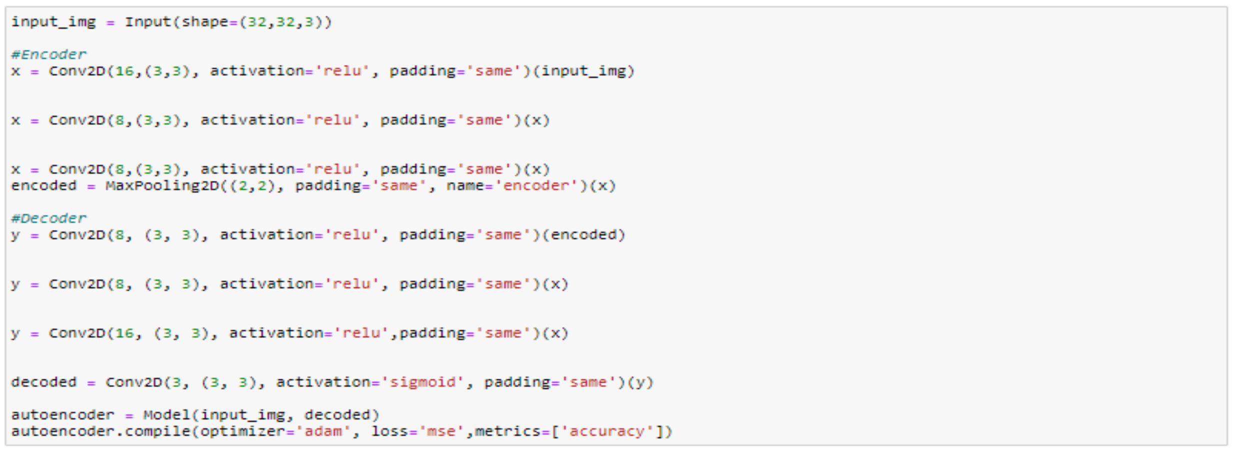


```
: from keras.datasets import cifar10
  from keras.layers import Input, Dense,Conv2D,MaxPooling2D,UpSampling2D,BatchNormalization
  from keras.models import Model,Sequential
  import numpy as np
  import pandas as pd
  import matplotlib.pyplot as plt
```



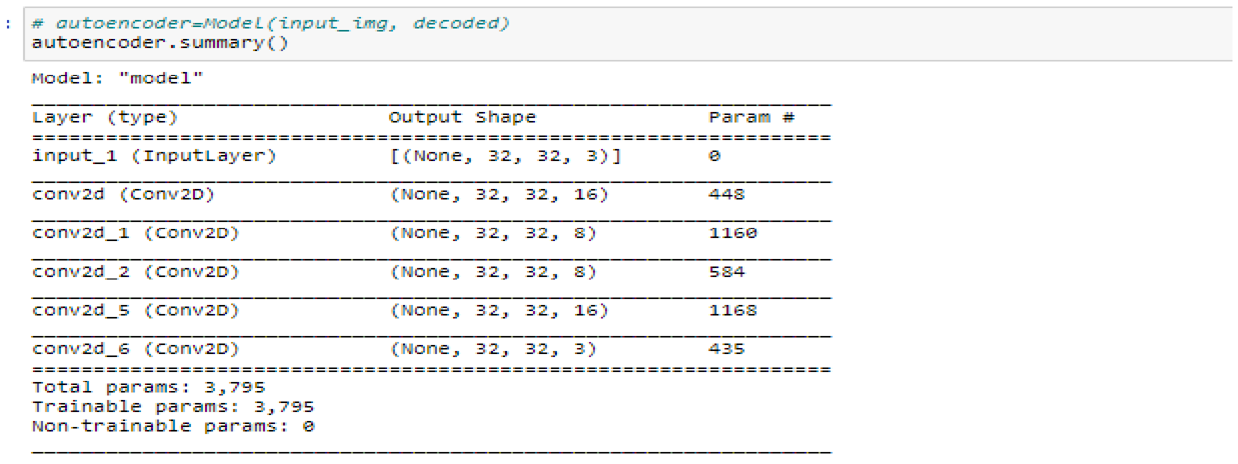
```
: (X_train, _), (X_test, _) = cifar10.load_data()
  X_train = X_train.astype('float32')/255
  X_test = X_test.astype('float32')/255
  X_train = X_train.reshape(len(X_train),X_train.shape[1],X_train.shape[2],3)
  X_test = X_test.reshape(len(X_test), X_test.shape[1],X_test.shape[2],3)
  print(X_train.shape)
  print(X_test.shape)

  (50000, 32, 32, 3)
  (10000, 32, 32, 3)
```

- Add the encoding layer with padding same.
- Also add the decoding layer with activation sigmoid.



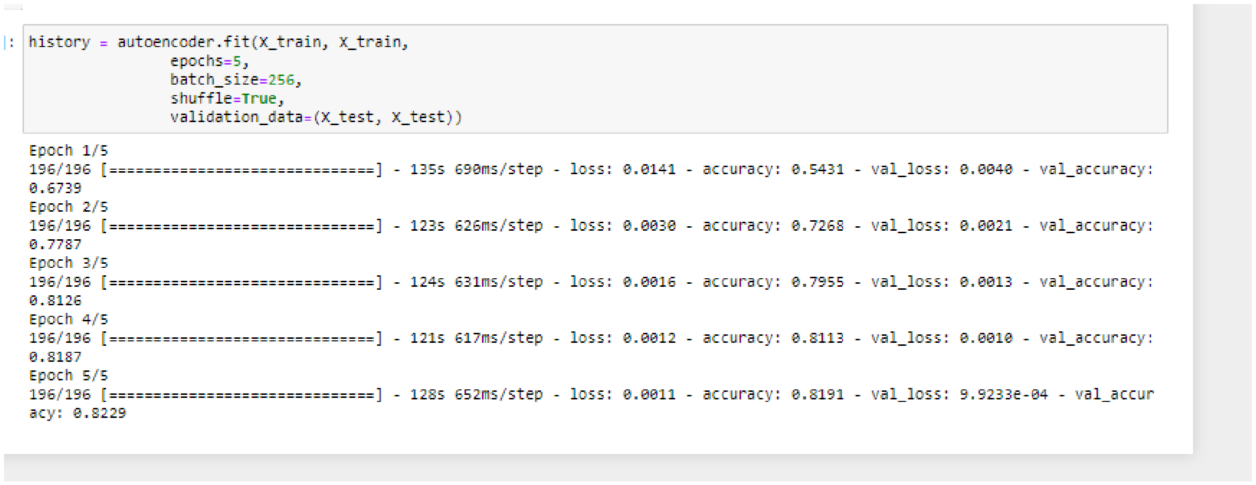
```
input_img = Input(shape=(32,32,3))

#Encoder
x = Conv2D(16,(3,3), activation='relu', padding='same')(input_img)


x = Conv2D(8,(3,3), activation='relu', padding='same')(x)


x = Conv2D(8,(3,3), activation='relu', padding='same')(x)
encoded = MaxPooling2D((2,2), padding='same', name='encoder')(x)

#Decoder
y = Conv2D(8, (3, 3), activation='relu', padding='same')(encoded)


y = Conv2D(8, (3, 3), activation='relu', padding='same')(x)


y = Conv2D(16, (3, 3), activation='relu',padding='same')(x)


decoded = Conv2D(3, (3, 3), activation='sigmoid', padding='same')(y)

autoencoder = Model(input_img, decoded)
autoencoder.compile(optimizer='adam', loss='mse',metrics=['accuracy'])
```

- Now compile the autoencoder and print the summary.



```
: # autoencoder=Model(input_img, decoded)
  autoencoder.summary()

  Model: "model"
  _________________________________________________________________
  Layer (type)                 Output Shape              Param #
  =================================================================
  input_1 (InputLayer)         [(None, 32, 32, 3)]       0
  _________________________________________________________________
  conv2d (Conv2D)              (None, 32, 32, 16)        448
  _________________________________________________________________
  conv2d_1 (Conv2D)            (None, 32, 32, 8)         1160
  _________________________________________________________________
  conv2d_2 (Conv2D)            (None, 32, 32, 8)         584
  _________________________________________________________________
  conv2d_5 (Conv2D)            (None, 32, 32, 16)        1168
  _________________________________________________________________
  conv2d_6 (Conv2D)            (None, 32, 32, 3)         435
  =================================================================
  Total params: 3,795
  Trainable params: 3,795
  Non-trainable params: 0
  _________________________________________________________________
```

- Plot the history object and then predict the loss and accuracy.

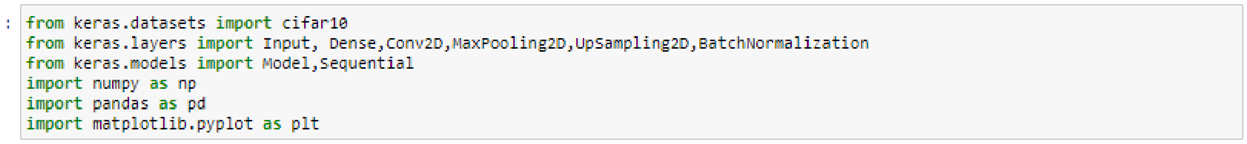


```
]: history = autoencoder.fit(X_train, X_train,
                epochs=5,
                batch_size=256,
                shuffle=True,
                validation_data=(X_test, X_test))
```

```
Epoch 1/5
196/196 [==============================] - 135s 690ms/step - loss: 0.0141 - accuracy: 0.5431 - val_loss: 0.0040 - val_accuracy:
0.6739
Epoch 2/5
196/196 [==============================] - 123s 626ms/step - loss: 0.0030 - accuracy: 0.7268 - val_loss: 0.0021 - val_accuracy:
0.7787
Epoch 3/5
196/196 [==============================] - 124s 631ms/step - loss: 0.0016 - accuracy: 0.7955 - val_loss: 0.0013 - val_accuracy:
0.8126
Epoch 4/5
196/196 [==============================] - 121s 617ms/step - loss: 0.0012 - accuracy: 0.8113 - val_loss: 0.0010 - val_accuracy:
0.8187
Epoch 5/5
196/196 [==============================] - 128s 652ms/step - loss: 0.0011 - accuracy: 0.8191 - val_loss: 9.9233e-04 - val_accur
acy: 0.8229
```

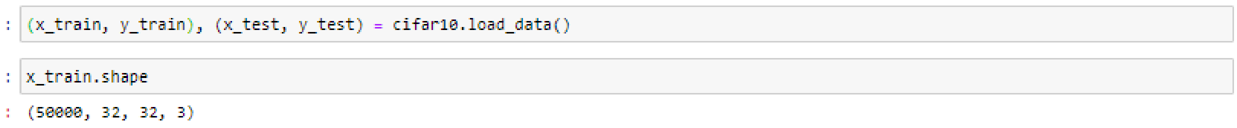
a) Repeat the same thing with PCA (apply PCA on the dataset and then pass the result to CNN or LSTM or three layers model)

- Import the necessary libraries.



```
: from keras.datasets import cifar10
  from keras.layers import Input, Dense,Conv2D,MaxPooling2D,UpSampling2D,BatchNormalization
  from keras.models import Model,Sequential
  import numpy as np
  import pandas as pd
  import matplotlib.pyplot as plt
```

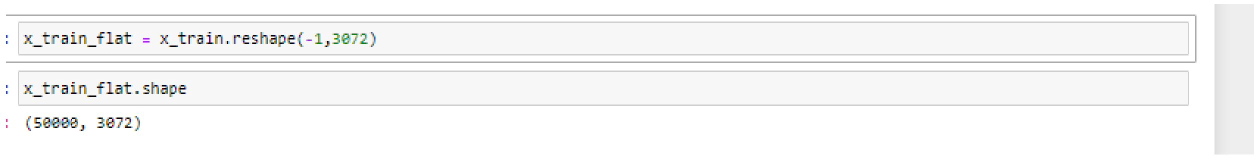
- Read the train and the test data.
- Reshape the train and the test data into two dimensional to fit into PCA.



```
: (x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

```
: x_train.shape
```

```
: (50000, 32, 32, 3)
```

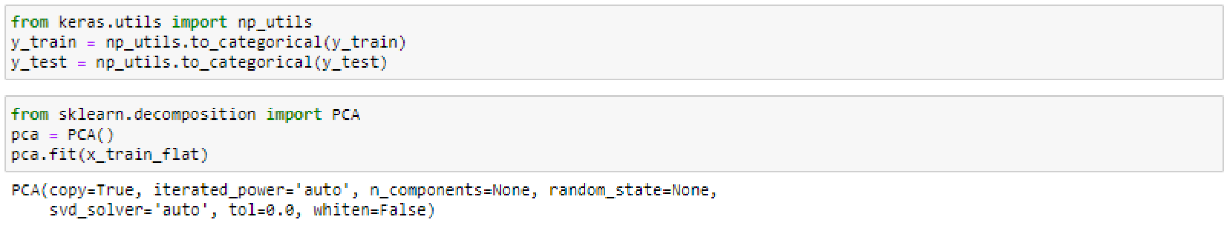


```
: x_train_flat = x_train.reshape(-1,3072)
```

```
: x_train_flat.shape
```

```
: (50000, 3072)
```

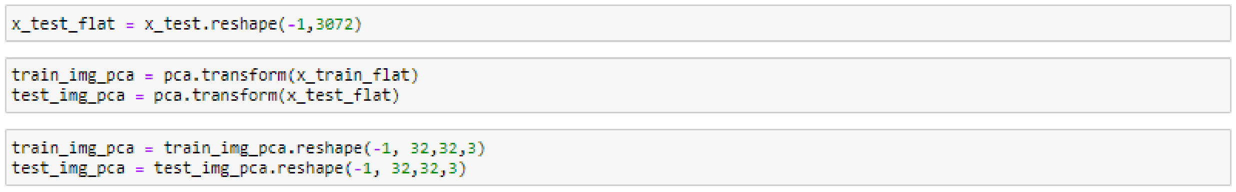
- Now import PCA and then fit x_train_flat and x_test_flat into PCA



```
from keras.utils import np_utils
y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)
```

```
from sklearn.decomposition import PCA
pca = PCA()
pca.fit(x_train_flat)
```

```
PCA(copy=True, iterated_power='auto', n_components=None, random_state=None,
    svd_solver='auto', tol=0.0, whiten=False)
```

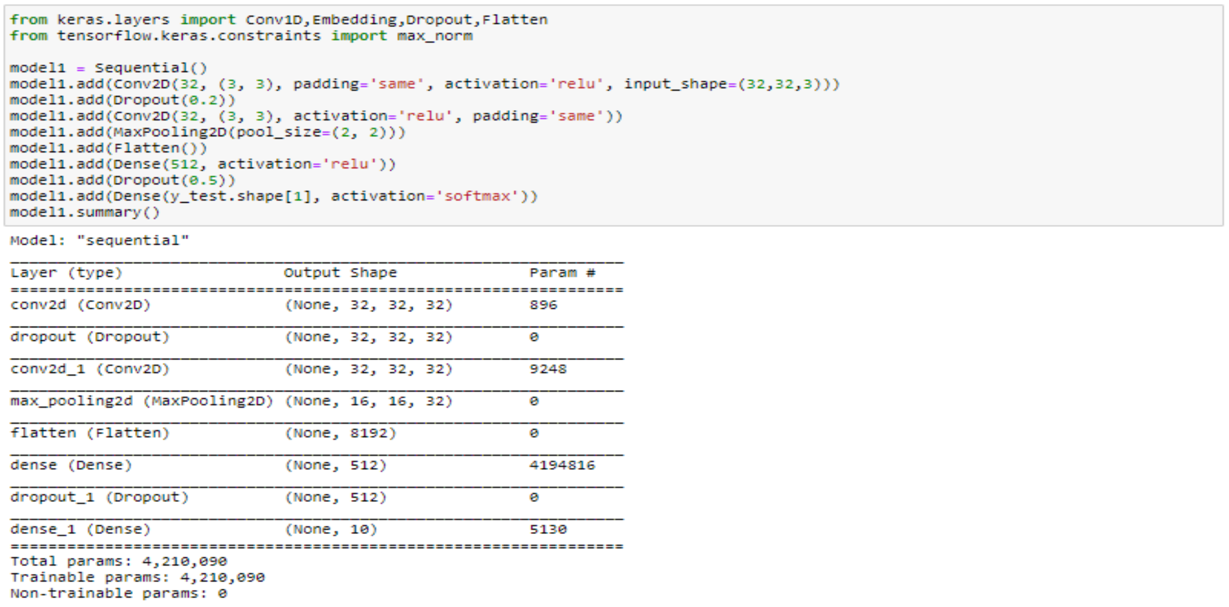


```
x_test_flat = x_test.reshape(-1,3072)
```

```
train_img_pca = pca.transform(x_train_flat)
test_img_pca = pca.transform(x_test_flat)
```

```
train_img_pca = train_img_pca.reshape(-1, 32,32,3)
test_img_pca = test_img_pca.reshape(-1, 32,32,3)
```

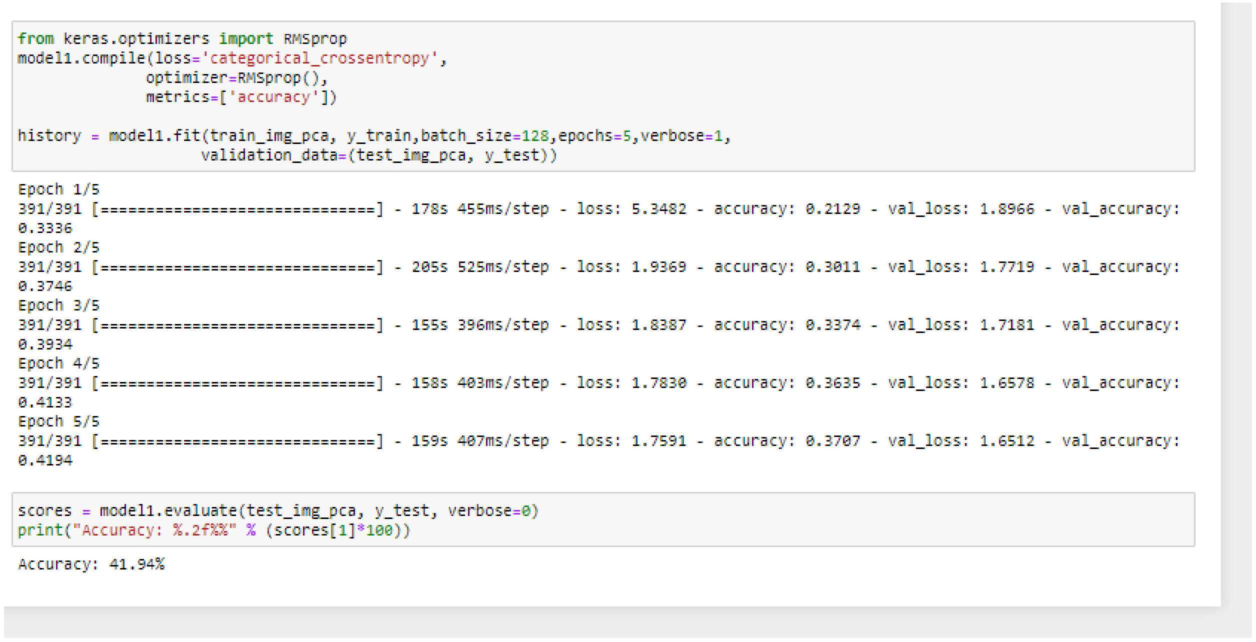
- Initialize the sequential model with the convolutional layers.



```
from keras.layers import Conv1D,Embedding,Dropout,Flatten
from tensorflow.keras.constraints import max_norm

model1 = Sequential()
model1.add(Conv2D(32, (3, 3), padding='same', activation='relu', input_shape=(32,32,3)))
model1.add(Dropout(0.2))
model1.add(Conv2D(32, (3, 3), activation='relu', padding='same'))
model1.add(MaxPooling2D(pool_size=(2, 2)))
model1.add(Flatten())
model1.add(Dense(512, activation='relu'))
model1.add(Dropout(0.5))
model1.add(Dense(y_test.shape[1], activation='softmax'))
model1.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d (Conv2D)              (None, 32, 32, 32)        896       
_________________________________________________________________
dropout (Dropout)            (None, 32, 32, 32)        0         
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 32, 32, 32)        9248      
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 16, 16, 32)        0         
_________________________________________________________________
flatten (Flatten)            (None, 8192)              0         
_________________________________________________________________
dense (Dense)                (None, 512)               4194816   
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0         
_________________________________________________________________
dense_1 (Dense)              (None, 10)                5130      
=================================================================
Total params: 4,210,090
Trainable params: 4,210,090
Non-trainable params: 0
```

- Compile the model
- Plot the history object and then predict the loss and accuracy.



```
from keras.optimizers import RMSprop
model1.compile(loss='categorical_crossentropy',
              optimizer=RMSprop(),
              metrics=['accuracy'])

history = model1.fit(train_img_pca, y_train,batch_size=128,epochs=5,verbose=1,
                    validation_data=(test_img_pca, y_test))
```

```
Epoch 1/5
391/391 [==============================] - 178s 455ms/step - loss: 5.3482 - accuracy: 0.2129 - val_loss: 1.8966 - val_accuracy:
0.3336
Epoch 2/5
391/391 [==============================] - 205s 525ms/step - loss: 1.9369 - accuracy: 0.3011 - val_loss: 1.7719 - val_accuracy:
0.3746
Epoch 3/5
391/391 [==============================] - 155s 396ms/step - loss: 1.8387 - accuracy: 0.3374 - val_loss: 1.7181 - val_accuracy:
0.3934
Epoch 4/5
391/391 [==============================] - 158s 403ms/step - loss: 1.7830 - accuracy: 0.3635 - val_loss: 1.6578 - val_accuracy:
0.4133
Epoch 5/5
391/391 [==============================] - 159s 407ms/step - loss: 1.7591 - accuracy: 0.3707 - val_loss: 1.6512 - val_accuracy:
0.4194
```

```
scores = model1.evaluate(test_img_pca, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))
```

```
Accuracy: 41.94%
```

DATASETS:
https://drive.google.com/drive/folders/15fBFVfb5RSLEN1kqy4e8j9WiHx99Uyqx?usp=sharing

PARAMETERS:

Target columns supposed in each question are our considered parameters.

Question 1:

- Column 'Phrase' is considered parameter.
- Loss and Accuracy is also considered as a model parameter.

Question 2:

- Loss and Accuracy is also considered as a model parameter.

Question 3:

- Column 'Label' is considered parameter.
- Loss and Accuracy is also considered as a model parameter.

Question 4:

- 'Headlines' is our considered parameter.

Question 5:

- Loss and Accuracy is also considered as a model parameter.

EVALUATION AND DISCUSSION:

Question 1:

i. Applied CNN algorithm and text image classification for implementation.
ii. Obtained accuracy for this model is 64.68%.
iii. After evaluating the model, we can infer that it has an overfitting problem. To resolve this issue, add hidden embedded layers with which metrics value got raised. Final accuracy obtained is 86.37%.

Question 2:

i. Applied Text classification using LSTM model for implementation.
ii. Include embedding layers.
iii. Model is overfitted and loss has been increased.

Question 3:

i. Applied image classification with CNN algorithm.
ii. With scaling loss increased and accuracy decreased.
iii. Without scaling loss decreased and accuracy increased.

Question 4:

i. Applied LSTM model and Text generation for implementation.
ii. Used tokenizer to generate input text data sequence.
iii. Initialized softmax activation layer as we have multiclass function.

Question 5:

i. Applied CNN algorithm and Auto-encoder in the implementation part.
ii. Applied Principal Component Analysis.
iii. Using Auto-Encoder loss decreased and accuracy increased, meanwhile using PCA made difference in metrics values, loss has increased comparatively.

CONCLUSION:

This project helped us in learning and implementing Deep Learning and machine learning concepts. Implemented and analyzed CNN, LSTM, Text classification, Text generation and other ML algorithms.

Github reference: https://github.com/PallaviArikatla/Python_Exam_2

Video reference: https://umkc.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=c02dc38a-13bb-40e8-a850-ac08003f9927

TEAM MEMBERS:

ARIKATLA, PALLAVI – 4

GENTE, HARUN SAI KUMAR – 15

KOTA, SARIKA REDDY – 25

MAROJU, JAGADEESH – 28