

## **PYTHON PROJECT 2 REPORT**

### **Team 2**

#### **INTRODUCTION:**

We are going to perform different sequence classification and text classification models on some datasets where we have to predict the data based on the sequence and based on the category they belong to. In this report we are mainly going to focus on CNN and LSTM models which can be used to image classification and text classification and we are also going to apply some encoding processes which helps us to increase the accuracy of the predicted data.

#### **OBJECTIVE:**

The main Objective this report is to classify the test data based on the trained data in a data set and to increase the accuracy of the prediction with less loss.

#### **METHODS:**

- In this project we are going to use the sequence classification methods that are CNN that is convolutional neural network which is used to classify the images and also text in a given dataset.
- LSTM model which is known as Long short term memory model network which is used for sequence classification which is really helpful in machine learning tasks like speech recognition etc.,
- And we are going to apply auto encoder and embedded layers to the data to see how the accuracy of the predicted data changes.

#### **WORKFLOW:**

**1)**

- For this we are going to perform the text classification on the review's sentiment dataset using CNN model.
- First we have to upload all the packages needed and later import the dataset to a variable and now we have to separate the train data and test data.

```

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
tensorboard=TensorBoard(log_dir="log/.format(time())")

```

Using TensorFlow backend.

```

train_data= pd.read_csv("train.tsv", sep="\t")
print((train_data.shape))
train_data.head()
test_data = pd.read_csv("test.tsv", sep="\t")
print((test_data.shape))
test_data.head(5)

```

PhraseId	SentenceId	Phrase	Sentiment
(159866, 4)	(159866, 4)		

- Later that we have to perform the tokenization on the data to group the words that are similar.
- Later we have select the features and based on the features we are going to separate the data and predict it.

```

[ ] 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('^\w+\-\w+\-\w+\-\w+\s', '', x.lower()))
test_data['Phrase'] = test_data['Phrase'].apply(lambda x: re.sub('^\w+\-\w+\-\w+\-\w+\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)

```

- Later apply the label encoder on the data so that all the nonnumeric datasets are converted into the string dataset and later that we have to create our model.

```

[ ] max_fatures = 5000
tokenizer = Tokenizer(num_words=max_fatures, 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_fatures = 2000
tokenizer = Tokenizer(num_words=max_fatures, 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)

[ ] X_train.shape
(156060, 46)

[ ] X_test.shape
(66292, 46)

[ ] 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)

```

- We are using the CNN model here and we need to fit our data on to this model.
- We calculated the loss and accuracy of the data and plotted it.

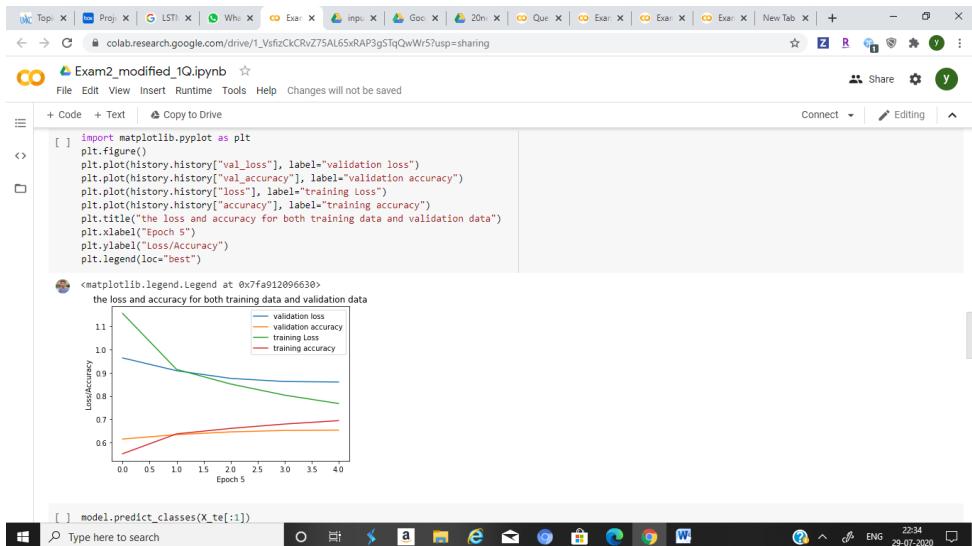
```

[ ] num_classes = Y_train.shape[1]
max_words= X_train.shape[1]
model= Sequential()
model.add(Embedding(5000,100,input_length=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=1, callbacks=[tensorboard])

```

/usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/indexed\_slices.py:434: UserWarning: Converting sparse IndexedSlices to a dense Tensor of unknown shape. "Converting sparse IndexedSlices to a dense Tensor of unknown shape."  
Train on 117045 samples, validate on 39015 samples  
Epoch 1/5  
117045/117045 [=====] - 30s 258us/step - loss: 1.1544 - accuracy: 0.5523 - val\_loss: 0.9635 - val\_accuracy: 0.6153  
Epoch 2/5  
117045/117045 [=====] - 30s 255us/step - loss: 0.9142 - accuracy: 0.6376 - val\_loss: 0.9087 - val\_accuracy: 0.6343  
Epoch 3/5  
117045/117045 [=====] - 30s 255us/step - loss: 0.8516 - accuracy: 0.6610 - val\_loss: 0.8757 - val\_accuracy: 0.6460  
Epoch 4/5  
117045/117045 [=====] - 30s 256us/step - loss: 0.8038 - accuracy: 0.6797 - val\_loss: 0.8622 - val\_accuracy: 0.6522  
Epoch 5/5  
117045/117045 [=====] - 30s 254us/step - loss: 0.7678 - accuracy: 0.6943 - val\_loss: 0.8601 - val\_accuracy: 0.6537



- Later we need to include the embedding layer to the model so if the accuracy changes and we did observe that accuracy is improved .
- Later we observed the overfitting problem in this, this overfitting can be removed by using learning rate .

```

[ ] print("PREDICTED LABEL ",y_pred[0])
[ ] print("ACTUAL LABEL ",sub_file['Sentiment'].iloc[0])

[ ] PREDICTED LABEL 2
[ ] ACTUAL LABEL 2

[ ] from keras.optimizers import adam
[ ] s=adam(lr=0.001)
[ ] model1= Sequential()
[ ] model1.add(Embedding(5000,100,input_length=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, callbacks=[tensorboard])

```

Output of the fit command:

```

/usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/indexed_slices.py:434: UserWarning: Converting sparse IndexedSlices to a dense Tensor of unknown shape.
  "Converting sparse IndexedSlices to a dense Tensor of unknown shape."
Train on 117045 samples, validate on 39015 samples
Epoch 1/5
117045/117045 [=====] - 53s 457us/step - loss: 0.3443 - accuracy: 0.8483 - val_loss: 0.3145 - val_accuracy: 0.8581
Epoch 2/5
117045/117045 [=====] - 54s 462us/step - loss: 0.3013 - accuracy: 0.8659 - val_loss: 0.3051 - val_accuracy: 0.8629
Epoch 3/5

```

```

+ Code + Text ⚡ Copy to Drive
history1=Model1.fit(X_tr, Y_tr, validation_data=(X_te, Y_te), epochs=5, batch_size=51, verbose=1, callbacks=[tensorboard])
[ ] "/usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/indexed_slices.py:434: UserWarning: Converting sparse IndexedSlices to a dense Tensor of unknown shape."
"Converting sparse IndexedSlices to a dense Tensor of unknown shape."
Train on 117045 samples, validate on 39015 samples
Epoch 1/5
117045/117045 [=====] - 53s 457us/step - loss: 0.3443 - accuracy: 0.8483 - val_loss: 0.3145 - val_accuracy: 0.8581
Epoch 2/5
117045/117045 [=====] - 54s 462us/step - loss: 0.3013 - accuracy: 0.8659 - val_loss: 0.3051 - val_accuracy: 0.8629
Epoch 3/5
117045/117045 [=====] - 53s 457us/step - loss: 0.2843 - accuracy: 0.8744 - val_loss: 0.3033 - val_accuracy: 0.8631
Epoch 4/5
117045/117045 [=====] - 54s 458us/step - loss: 0.2726 - accuracy: 0.8795 - val_loss: 0.3029 - val_accuracy: 0.8646
Epoch 5/5
117045/117045 [=====] - 53s 457us/step - loss: 0.2632 - accuracy: 0.8847 - val_loss: 0.3050 - val_accuracy: 0.8639

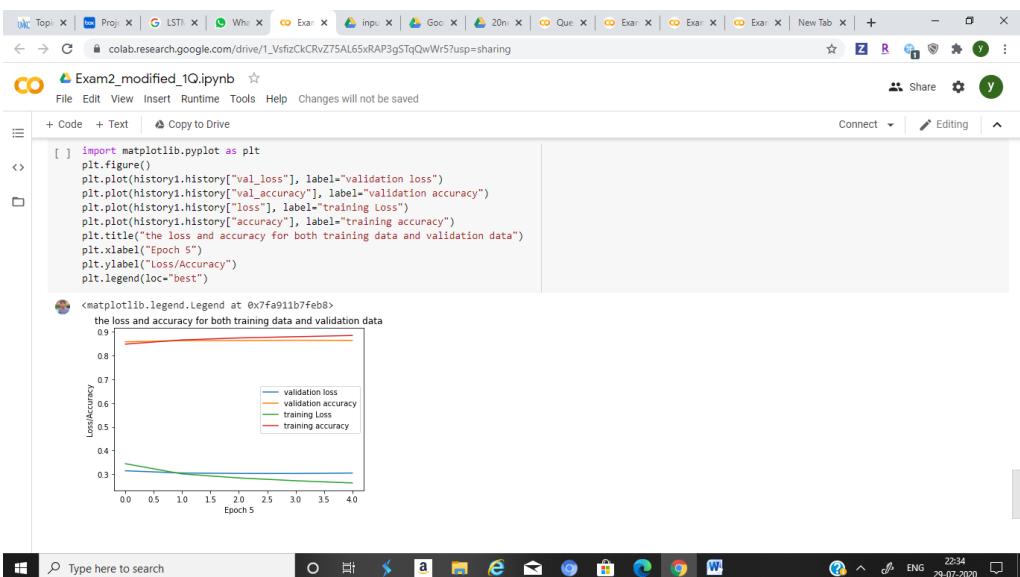
```

```

[ ] import matplotlib.pyplot as plt
plt.figure()
plt.plot(history1.history['val_loss'], label="validation loss")
plt.plot(history1.history['val_accuracy'], label="validation accuracy")
plt.plot(history1.history['loss'], label="training Loss")
plt.plot(history1.history['accuracy'], label="training accuracy")
plt.title("the loss and accuracy for both training data and validation data")
plt.xlabel("Epoch 5")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="best")

```

<matplotlib.legend.Legend at 0x7fa91b7feb8>  
the loss and accuracy for both training data and validation data



- 2) We are going to import all the packages that are needed for the program and we are going to use the 20 news groups dataset for this task, our task here is to classify the text using LSTM model.
- so first we are going to create a file and add the 20news groups data onto that file later, we are gonna create a add this file to the already existing file, the 20newsgroups data is a live data so for more information belong to same category we going to create different files and append them later.

```

[ ] files = []
for folder_name in folders:
    folder_path = join(my_path, folder_name)
    files.append([f for f in listdir(folder_path)])

[ ] #checking total no. of files gathered
sum(len(files[i]) for i in range(2))

2000

[ ] #creating a list of pathnames of all the documents
#this would serve to split our dataset into train & test later without any bias

pathname_list = []
for fo in range(len(folders)):
    for fi in files[fo]:
        pathname_list.append(join(my_path, join(folders[fo], fi)))

len(pathname_list)
2000

[ ] #making an array containing the classes each of the documents belong to

```

- Now split the data into training data and testing data using train test split model.
- Later select the stop words and separate them from the data file, we need to separate the stop words because they make the prediction go wrong sometime and these words are not really useful in the prediction.

```

[ ] doc_train, doc_test, Y_train, Y_test = train_test_split(pathname_list, Y, random_state=0, test_size=0.25)

functions for word extraction from documents

[ ] stopwords = ['a', 'about', 'above', 'after', 'again', 'against', 'all', 'am', 'ah', 'and', 'any', 'are', 'aren\'t', 'as', 'at',
'be', 'because', 'been', 'before', 'being', 'below', 'between', 'both', 'but', 'by',
'can', 'can\'t', 'cannot', 'could', 'couldn\'t', 'did', 'didn\'t', 'do', 'does', 'doesn\'t', 'doing', 'don\'t', 'down', 'during',
'each', 'few', 'for', 'from', 'further',
'had', 'hadn\'t', 'has', 'hasn\'t', 'have', 'haven\'t', 'having', 'he', 'he\'d', 'he\'ll', 'he\'s', 'hen', 'here', 'here\'s',
'hers', 'herself', 'him', 'himself', 'his', 'how', 'how\'s',
'i', 'i\'d', 'i\'ll', 'i\'m', 'i\'ve', 'if', 'in', 'into', 'is', 'isn\'t', 'it', 'it\'s', 'its', 'itself',
'let\'s', 'me', 'more', 'most', 'mustn\'t', 'my', 'myself',
'no', 'nor', 'not', 'or', 'off', 'on', 'once', 'only', 'on', 'other', 'ought', 'our', 'ours', 'ourselves', 'out', 'over', 'own',
'same', 'shan\'t', 'she', 'she\'d', 'she\'ll', 'she\'s', 'should', 'shouldn\'t', 'so', 'some', 'such',
'than', 'that', 'thats', 'the', 'their', 'theirs', 'them', 'themselves', 'then', 'there', 'theres', 'these', 'they', 'they\'d',
'they\'ll', 'they\'re', 'they\'ve', 'this', 'those', 'through', 'to', 'too', 'under', 'until', 'up', 'very',
'was', 'wasn\'t', 'we', 'we\'d', 'we\'ll', 'we\'re', 'weren\'t', 'what', 'what\'s', 'when', 'whens', 'where',
'where\'s', 'which', 'while', 'who', 'who\'s', 'whom', 'why', 'why\'s', 'will', 'with', 'won\'t', 'would', 'wouldn\'t',
'you', 'you\'d', 'you\'ll', 'you\'re', 'you\'ve', 'your', 'yours', 'yourself', 'yourselves',
'one', 'two', 'three', 'four', 'five', 'six', 'seven', 'eight', 'nine', 'ten', 'hundred', 'thousand', '1st', '2nd', '3rd',
'4th', '5th', '6th', '7th', '8th', '9th', '10th']

[ ] #function to preprocess the words list to remove punctuations

def preprocess(words):
    #we'll make use of python's translate function, that maps one set of characters to another
    #we create an empty mapping table, the third argument allows us to list all of the characters

```

- Later, based on the certain features we extract the data and tokenize the data so the data falls under the same category, later that remove the stop words and after that now we have the train and test data calculate the length of the data and now we can print the data under the tokenizing.

```

#function to preprocess the words list to remove punctuations
def preprocess(words):
    #we'll make use of python's translate function, that maps one set of characters to another
    #we create an empty mapping table, the third argument allows us to list all of the characters
    #to remove during the translation process

    #first we will try to filter out some unnecessary data like tabs
    table = str.maketrans('', '', '\t')
    words = [word.translate(table) for word in words]

    punctuations = (string.punctuation).replace("'", "")
    # the character ' appears in a lot of stopwords and changes meaning of words if removed
    #hence it is removed from the list of symbols that are to be discarded from the documents
    trans_table = str.maketrans('', '', punctuations)
    stripped_words = [word.translate(trans_table) for word in words]

    #some white spaces may be added to the list of words, due to the translate function & nature of our documents
    #we remove them below
    words = [str for str in stripped_words if str]

    #some words are quoted in the documents & as we have not removed ' to maintain the integrity of some stopwords
    #we try to unquote such words below
    p_words = []
    for word in words:
        if (word[0] and word[len(word)-1] == ""):
            word = word[1:len(word)-1]
        elif(word[0] == ""):
            word = word[1:len(word)]
    return words

```

```

#function to remove stopwords
def remove_stopwords(words):
    words = [word for word in words if not word.isalpha()]
    words = [word for word in words if not len(word) == 1]

    #after removal of so many characters it may happen that some strings have become blank, we remove those
    words = [str for str in words if str]

    #we also normalize the cases of our words
    words = [word.lower() for word in words]

    #we try to remove words with only 2 characters
    words = [word for word in words if len(word) > 2]

    return words

#function to convert a sentence into list of words
def tokenize_sentence(line):
    words = line[0:len(line)-1].strip().split(" ")
    words = preprocess(words)
    words = remove_stopwords(words)

```

- Later, now encode the data so that all the datatypes of the words will be same later, and we are going to apply the embedding layer In it and then create the model and later that apply the model for the filtered data.

The screenshot shows a Google Colab notebook titled "Exam2\_2Q.ipynb". The code cell contains two functions: `tokenize` and `flatten`. The `tokenize` function reads a file at a given path, removes meta-data from the top of each document, initializes an array to hold words, and then iterates over the lines to tokenize each one using a helper function. The `flatten` function takes a list and flattens it into a single list by nested iteration. The status bar at the bottom indicates the notebook is in "Editing" mode.

```
[ ] #function to convert a document into list of words
<> def tokenize(path):
    #load document as a list of lines
    f = open(path, 'r', encoding="utf8", errors='ignore')
    text_lines = f.readlines()

    #removing the meta-data at the top of each document
    text_lines = remove_metadata(text_lines)

    #initializing an array to hold all the words in a document
    doc_words = []

    #traverse over all the lines and tokenize each one with the help of helper function: tokenize_sentence
    for line in text_lines:
        doc_words.append(tokenize_sentence(line))

    return doc_words

[ ] #a simple helper function to convert a 2D array to 1D, without using numpy

def flatten(list):
    new_list = []
    for i in list:
        for j in i:
            new_list.append(j)
    return new_list
```

The screenshot shows a Google Colab notebook titled "Exam2\_2Q.ipynb". The code cell contains a loop that iterates through a list of documents, printing each one and appending its flattened tokenized version to a list named `list_of_words`. The list of documents includes various paths from a Google Drive folder. The status bar at the bottom indicates the notebook is in "Editing" mode.

```
[ ] list_of_words = []
<> for document in doc_train:

    print(document)
    list_of_words.append(flatten(tokenize(document)))

@/content/drive/My Drive/Colab Notebooks/20newsgroup/sci.space/59848
@/content/drive/My Drive/Colab Notebooks/20newsgroup/alt.atheism/54255
@/content/drive/My Drive/Colab Notebooks/20newsgroup/sci.space/61358
@/content/drive/My Drive/Colab Notebooks/20newsgroup/alt.atheism/53322
@/content/drive/My Drive/Colab Notebooks/20newsgroup/alt.atheism/54191
@/content/drive/My Drive/Colab Notebooks/20newsgroup/sci.space/60212
@/content/drive/My Drive/Colab Notebooks/20newsgroup/sci.space/60218
@/content/drive/My Drive/Colab Notebooks/20newsgroup/alt.atheism/54250
@/content/drive/My Drive/Colab Notebooks/20newsgroup/alt.atheism/53474
@/content/drive/My Drive/Colab Notebooks/20newsgroup/alt.atheism/53126
@/content/drive/My Drive/Colab Notebooks/20newsgroup/alt.atheism/53372
@/content/drive/My Drive/Colab Notebooks/20newsgroup/sci.space/60216
@/content/drive/My Drive/Colab Notebooks/20newsgroup/alt.atheism/53055
@/content/drive/My Drive/Colab Notebooks/20newsgroup/alt.atheism/52288
@/content/drive/My Drive/Colab Notebooks/20newsgroup/sci.space/61459
@/content/drive/My Drive/Colab Notebooks/20newsgroup/alt.atheism/53335
@/content/drive/My Drive/Colab Notebooks/20newsgroup/alt.atheism/54194
@/content/drive/My Drive/Colab Notebooks/20newsgroup/alt.atheism/53070
@/content/drive/My Drive/Colab Notebooks/20newsgroup/alt.atheism/54248
@/content/drive/My Drive/Colab Notebooks/20newsgroup/sci.space/60817
```

```

n = 5000
features = wrds[0:n]
print(features)

['writes', 'article', 'space', 'people', 'just', 'like', 'god', 'think', 'know', 'say', 'also', 'time', 'see', 'get', 'even', 'may', 'many', 'way', 'system', 'muc
]

#creating a dictionary that contains each document's vocabulary and occurrence of each word of the vocabulary

dictionary = {}
doc_num = 1
for doc_words in list_of_words:
    #print(doc_words)
    np_doc_words = np.asarray(doc_words)
    w, c = np.unique(np_doc_words, return_counts=True)
    dictionary[doc_num] = {}
    for i in range(len(w)):
        dictionary[doc_num][w[i]] = c[i]
    doc_num = doc_num + 1

```

Type here to search    22:42 29-07-2020

```

dict_keys([1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50])

#now we make a 2D array having the frequency of each word of our feature set in each individual documents

X_train = []
for k in dictionary.keys():
    row = []
    for f in features:
        if(f in dictionary[k].keys()):
            #if word f is present in the dictionary of the document as a key, its value is copied
            #this gives us no. of occurrences
            row.append(dictionary[k][f])
        else:
            #if not present, the no. of occurrences is zero
            row.append(0)
    X_train.append(row)

#we convert the X and Y into np array for concatenation and conversion into dataframe

X_train = np.asarray(X_train)
Y_train = np.asarray(Y_train)

print(len(X_train))
print(len(Y_train))

```

Type here to search    22:42 29-07-2020

- Later that predict the accuracy and loss.
- And later we are plotting the accuracy and loss using matplot lib.

```

labelencoder = LabelEncoder()
integer_encoded1 = labelencoder.fit_transform(Y_test)
y1 = to_categorical(integer_encoded1)

score,acc = model.evaluate(X_test,y1,verbose=2,batch_size=32)
print(score)
print(acc)
print(model.metrics_names)

0.6929565315246582
0.8099999904632568
['loss', 'accuracy']

```

- And we observed a overfitting problem in this.

3)

- In this task we are going to perform the image classification using CNN model, so in this we train our model with some images and later our model will predict the test image based on the training image.

```

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 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
%matplotlib inline
np.random.seed(1)

[ ] from google.colab import drive
drive.mount('/content/drive')

Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).

```

File Edit View Insert Runtime Tools Help Changes will not be saved

+ Code + Text ⌘ Copy to Drive Connect ▾ Editing

```
[ ] labels
[ ] 0  sampled_boulder
    1  patas_monkey
    2  bald_sukarai
    3  japanese_macaque
    4  pygmy_marmoset
    5  white_headed_capuchin
    6  silvery_marmoset
    7  black_faced_squirrel_monkey
    8  black_headed_night_monkey
    9  mungo_l_langur
Name: Common Name, dtype: object

[ ] def image_show(num_image,label):
    for i in range(num_image):
        imgdir = Path('/content/drive/My Drive/Colab Notebooks/Dataset/training/training/' + label)
        # print(imgdir)
        imgfile = random.choice(os.listdir(imgdir))
        # print(imgfile)
        img = cv2.imread('/content/drive/My Drive/Colab Notebooks/Dataset/training/training/' + label + '/' + imgfile)
        # print(img.shape)
        # print(label)
        plt.figure(i)
        plt.imshow(img)
        plt.title(imgfile)
    plt.show()

[ ] image show(4,'n1')
```

File Edit View Insert Runtime Tools Help Changes will not be saved

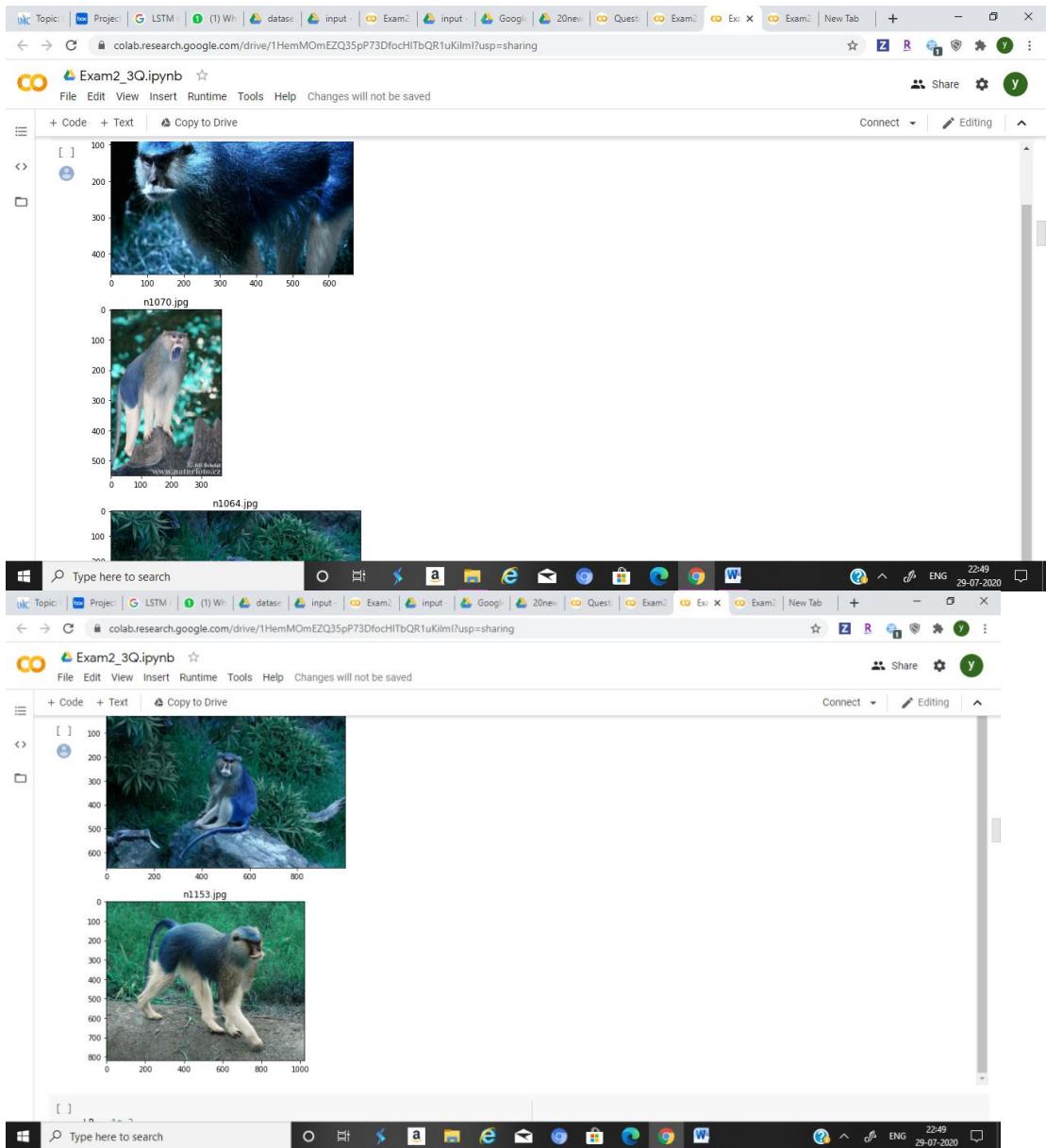
```
[ ] from keras.preprocessing.image import ImageDataGenerator  
from keras.models import Sequential  
from keras.layers import Conv2D, MaxPooling2D, Activation, Dropout, Flatten, Dense  
from keras.callbacks import ModelCheckpoint, EarlyStopping
```

Using Tensorflow backend.

```
[ ] train_dir = Path('/content/drive/My Drive/Colab Notebooks/Dataset/training/training')  
test_dir = Path('/content/drive/My Drive/Colab Notebooks/Dataset/validation/validation')
```

```
[ ] cols = ['Label', 'Latin Name', 'Common Name', 'Train Images', 'Validation Images']  
labels = pd.read_csv("/content/drive/My Drive/Colab Notebooks/Dataset/monkey_labels.txt", names=cols, skiprows=1)  
labels
```

	Label	Latin Name	Common Name	Train Images	Validation Images
0	n0	alouatta_palliata	mantled_howler	131	26
1	n1	erythrocebus_patas	patas_monkey	139	28
2	n2	cacajao_calvus	bald_uakari	137	27
3	n3	macaca_fasciata	japanese_macaque	152	30
4	n4	cebulla_pygmaea	pygmy_marmoset	131	26
5	n5	cebus_capucinus	white_headered_capuchin	141	28
6	n6	mico_argentatus	silvery_marmoset	132	26



- So, first import all the required packages and later that we are going to read the dataset into a variable, now in this select the columns which you are gonna use for the prediction and select the test column and save them in different variables and print the test column, here we are taking the common name column as the test column.
- Later, now we have to train the model, for that import the images that belong to different categories and later that specify the properties for that certain category, later apply this model on the test image and check the output.

```
File Edit View Insert Runtime Tools Help Changes will not be saved
+ Code + Text Copy to Drive Connect Editing
[ ]
LR = 1e-3
height=150
width=150
channels=3
seed=1337
batch_size = 64
num_classes = 10
epochs = 100
data_augmentation = True
num_predictions = 20

# Training generator
train_datagen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest')

train_generator = train_datagen.flow_from_directory(train_dir,
    target_size=(height,width),
    batch_size=batch_size,
    seed=seed,
    shuffle=True.
```

The screenshot shows a Google Colab interface with the following details:

- Title:** Exam2\_3Q.ipynb
- File Menu:** File Edit View Insert Runtime Tools Help Changes will not be saved
- Toolbar:** Share, Connect, Editing
- Code Cell:** Contains the following Python code for a Sequential model:

```
[ ] train_num = train_generator.samples
validation_num = validation_generator.samples

[ ] Found 1098 images belonging to 10 classes.
Found 272 images belonging to 10 classes.

[ ] 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'))
```
- Output:** Shows the number of images found for training and validation.
- Bottom Bar:** Includes a search bar, file icons (Google Drive, Input, Exam2, Google Sheets, etc.), and system status (ENG, 22:49, 29-07-2020).

```

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

Model: "sequential_1"
-----  

Layer (type)      Output Shape        Param #
conv2d_1 (Conv2D) (None, 148, 148, 32) 896
activation_1 (Activation) (None, 148, 148, 32) 0
max_pooling2d_1 (MaxPooling2D) (None, 74, 74, 32) 0
conv2d_2 (Conv2D) (None, 72, 72, 32) 9248
activation_2 (Activation) (None, 72, 72, 32) 0
max_pooling2d_2 (MaxPooling2D) (None, 36, 36, 32) 0
conv2d_3 (Conv2D) (None, 36, 36, 64) 18496
activation_3 (Activation) (None, 36, 36, 64) 0
conv2d_4 (Conv2D) (None, 34, 34, 64) 36928
activation_4 (Activation) (None, 34, 34, 64) 0

```

- And calculate the accuracy and loss and predicted value to the actual value.

```

[ ] y_pred
[ ] array([[2.2458931e-04, 7.0713723e-01, 2.1596844e-03, 2.4474332e-06,
           3.7643695e-03, 4.8631078e-04, 1.2518763e-05, 9.9162191e-02,
           1.6669231e-01, 1.8358395e-02]], dtype=float32)

[ ] y_pred_label = np.argmax(y_pred, axis=1)
display(y_pred_label)

[ ] array([1])

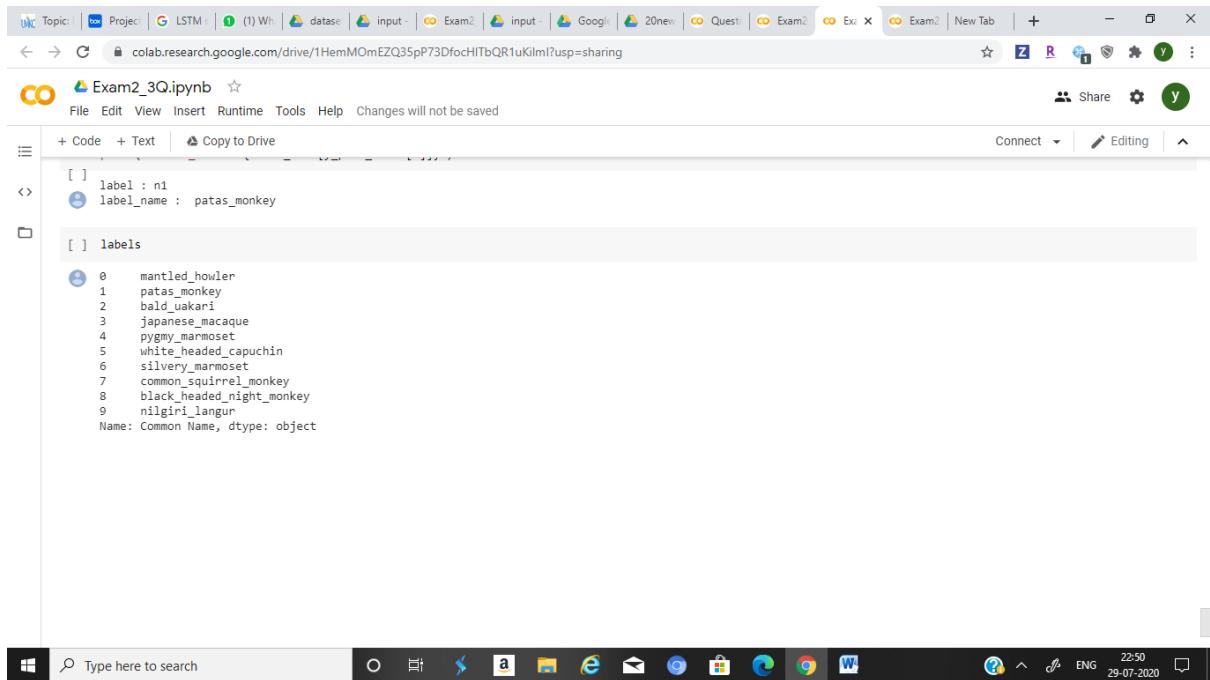
[ ] lab=train_generator.class_indices
label=list(lab.keys())
label_name=labels.tolist()

[ ] print(f"label : {label[y_pred_label[0]]}")
print(f"label_name : {label_name[y_pred_label[0]]}")

[ ] label : n1
label_name : patas_monkey

[ ] labels
[ ] 0    mantled_howler
[ ] 1    patas_monkey
[ ] 2    bald_ukari
[ ] 3    japanese_macaque
[ ] 4    ovvvn_marmoset

```



```
+ Code + Text ⌂ Copy to Drive
[ ] label : n1
↳ label_name : patas_monkey

[ ] 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
```

- We are going to plot the accuracy and loss.
- After that apply scaling on the data , so this improves the accuracy because scaling helps the model to consider all the data in an equal value.
- After that calculate the accuracy and loss and plot them.

#### 4)

- In this task we are going to train our model on the 20 news groups data headlines and comments and we are going to predict the headline of the dataset.
- For this first import all the packages needed and now we have to import the dataset.

```

# keras module for building LSTM
from keras.preprocessing.sequence import pad_sequences
from keras.layers import Embedding, LSTM, Dense
from keras.preprocessing.text import Tokenizer
from keras.callbacks import EarlyStopping
from keras.models import Sequential
import keras.utils as ku

# set seeds for reproducability
import tensorflow as tf

from numpy.random import seed
tf.random.set_seed(2)
seed(1)

import pandas as pd
import numpy as np
import string, os

import warnings
warnings.filterwarnings("ignore")
warnings.simplefilter(action='ignore', category=FutureWarning)

```

```

[ ] all_headlines = []
cwd = '/content/drive/My Drive/Colab Notebooks/Input/' # Get the current working directory (cwd)
#files = os.listdir(cwd) # Get all the files in that directory
#print("Files in %s" % (cwd, files))

```

- After that extract all the headlines and comments from the dataset and save them in a folder and we are going to train this data to the model.

```

all_headlines = []
cwd = '/content/drive/My Drive/Colab Notebooks/Input/' # Get the current working directory (cwd)
#files = os.listdir(cwd) # Get all the files in that directory
#print("Files in %s" % (cwd, files))

for filename in os.listdir(cwd):
    if 'Articles' in filename:
        article_df = pd.read_csv(cwd + filename)
        all_headlines.extend(list(article_df.headline.values))
        break

all_headlines[:10]

```

```

[N.F.L. vs. Politics Has Been Battle All Season Long',
'Voice, Vice, Veracity.',
'A Stand-Up's Downward Slide',
'New York Today: A Groundhog Has Her Day',
'A Swimmer's Communion With the Ocean',
'Trail Activity',
'Super Bowl',
'Trump's Mexican Shakedown',
'Pence's Presidential Pet',
'Fruit of a Poison Tree']

```

```

#Dataset preparation
#Dataset cleaning
#In dataset preparation step, we will first perform text cleaning of the data which includes removal of punctuations and :
def clean_text(txt):

```

- Now we are going to give the model a sample data and it has to predict the comment and headline for the given data.
- Now, separate the data as train and test data using train test split model.
- After that we have to apply the tokenization to the dataset and later store it, now create a model to predict the data which is a sequential model and feed the trained and test data to the model.

```

[ ] #Padding the Sequences and obtain Variables : Predictors and Target
def generate_padded_sequences(input_sequences):
    max_sequence_len = max([len(x) for x in input_sequences])
    input_sequences = np.array(pad_sequences(input_sequences, maxlen=max_sequence_len, padding='pre'))

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

predictors, label, max_sequence_len = generate_padded_sequences(inp_sequences)

[ ] #LSTMs for Text Generation
def create_model(max_sequence_len, total_words):
    input_len = max_sequence_len - 1
    model = Sequential()

    # Add Input Embedding Layer
    model.add(Embedding(total_words, 10, input_length=input_len))

    # Add Hidden Layer 1 - LSTM Layer
    model.add(LSTM(100))

    # Add Output Layer
    model.add(Dense(total_words, activation='softmax'))

    model.compile(loss='categorical_crossentropy', optimizer='adam')

    return model

```

```

[ ] #generating Sequence of n-gram tokens
tokenizer = Tokenizer()

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

    ## convert data to sequence of tokens
    input_sequences = []
    for line in corpus:
        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(corpus)
inp_sequences[:10]

```

[[661, 118],  
 [661, 118, 73],  
 [661, 118, 73, 74],  
 [661, 118, 73, 74, 662],  
 [661, 118, 73, 74, 662, 663],  
 [661, 118, 73, 74, 662, 663, 64],  
 [661, 118, 73, 74, 662, 663, 64, 30],  
 [661, 118, 73, 74, 662, 663, 64, 30, 211],  
 [212, 664],  
 [212, 664, 665]]

```

[ ] Model: "sequential_2"
-----  

Layer (type)          Output Shape         Param #  

embedding_2 (Embedding) (None, 16, 18)      22896  

lstm_2 (LSTM)          (None, 100)         44408  

dense_2 (Dense)        (None, 2289)        231189  

-----  

Total params: 298,479  

Trainable params: 298,479  

Non-trainable params: 0

[ ] #lets train our model now
model.fit(predictors, label, epochs=100, verbose=2)

Epoch 1/100
- 3s - loss: 7.3612
Epoch 2/100
- 3s - loss: 6.8417
Epoch 3/100
- 3s - loss: 6.7173
Epoch 4/100
- 3s - loss: 6.6338
Epoch 5/100
- 3s - loss: 6.5459
Epoch 6/100
- 3s - loss: 6.4481

```

```

[ ] #Generating the text
[ ] def generate_text(seed_text, next_words, model, max_sequence_len):
    for i 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("super bowl", 20, model, max_sequence_len))
print(generate_text("trail", 20, model, max_sequence_len))
print(generate_text("trumps ", 20, model, max_sequence_len))

Super Bowl Is Very Concerned Out A New Tax Leaves To Investors Odd Man Of Freely First First Help From The Common
Trail Disclosure Wall Street Zen And Investors Tower Bridge Rick Scorn Off Side Of Shrugs Kind College How Much Does It
Trumps Acrostic Rivals Wear Robes Flexibility Its I Times Are The Hell Half Premiere Door Areas Off Liberal Of Kennedy The

```

- Calculate the accuracy of the predicted data and observe the output.

## 5)

- In this we are going to apply autoencoding on the data set given and later we are going to feed this data any sequential classifying model and observe the accuracy and later that we are going to apply PCA model on the data and apply the sequential model later and observe the best one.
- First we have to import all the packages needed and later that we should create a variable and load and read the dataset into it,

```

[ ] import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import datasets, layers, models
from tensorflow.keras.models import load_model

import numpy as np
import matplotlib.pyplot as plt
from time import time
from random import randint

from keras.datasets import cifar10

from keras.utils import np_utils
from keras.constraints import maxnorm
from keras.layers import Dense, Input
from keras.layers import Dropout
from keras.layers import Flatten
from keras.optimizers import SGD
from keras.models import Model
from keras.callbacks import TensorBoard

from keras import regularizers
from keras import backend as K

from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler

Using TensorFlow backend.

```

File Edit View Insert Runtime Tools Help Changes will not be saved

```
[ ] # create a placeholder for an encoded (32-dimensional) input
encoded_input = Input(shape=(encoding_dim,))

[ ] # retrieve the last layer of the autoencoder model
decoder_layer = autoencoder.layers[-1]

[ ] # create the decoder model
decoder = Model(encoded_input, decoder_layer(encoded_input))

[ ] # Autoencoder model configuration

# this model maps an input to its encoded representation
autoencoder.compile(optimizer='adadelta', loss='binary_crossentropy', metrics=['accuracy'])

[ ] # Prepare data

# load data
(x_train, y_train), (x_test, y_test) = cifar10.load_data()

Downloading data from https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz
173ccaa0017a08a71... 25% 100000000 00:00:00 ETA
```

File Edit View Insert Runtime Tools Help Changes will not be saved

```
[ ] # apply autoencoder to the cifar_10 dataset
[ ] j = randint(0,10000)

[ ] # Single fully-connected neural layer as encoder and decoder
# this is the size of our encoded representations
encoding_dim = 128 # 128 floats -> compression of factor 24, assuming the input is 3072 floatst

[ ] # this is our input placeholder
input_img = Input(shape=(3072,))

[ ] # "encoded" is the encoded representation of the input
encoded = Dense(encoding_dim, activation='relu')(input_img)

[ ] # "decoded" is the loss reconstruction of the input
decoded = Dense(3072, activation='sigmoid')(encoded)

[ ] # this model maps an input to its reconstruction
autoencoder = Model(input_img, decoded)

[ ] # Separate encode model

# this model maps an input to its encoded representation
encoder = Model(input_img, encoded)
```

File Edit View Insert Runtime Tools Help Changes will not be saved

```
[ ] # print shape of data
print(x_train.shape)
print(x_test.shape)

(50000, 3072)
(10000, 3072)

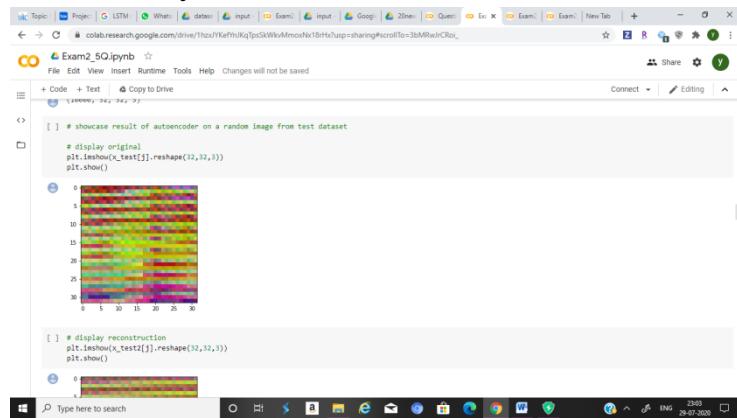
[ ] # one hot encode outputs
y_train = np_utils.to_categorical(y_train)
y_test = np_utils.to_categorical(y_test)
num_classes = y_test.shape[1]

[ ] print(num_classes)
10

[ ] tensorboard = TensorBoard(log_dir='2', histogram_freq=0, write_graph=True, write_images=False)
history = autoencoder.fit(x_train, y_train, epochs=50, batch_size=256, shuffle=True, validation_data=(x_test, y_test), callbacks=[tensorboard])
```

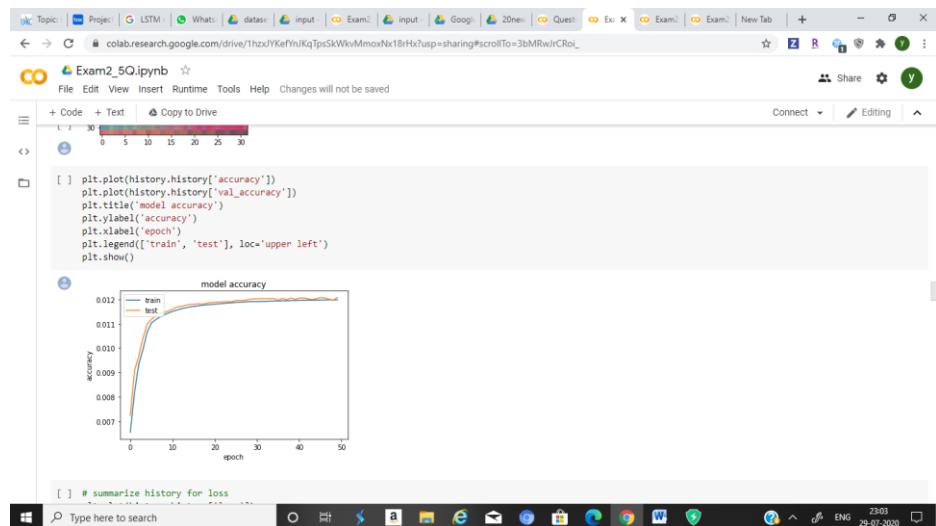
Epoch 22/50
50000/50000 [=====] - 16s 313us/step - loss: 0.6168 - accuracy: 0.0118 - val\_loss: 0.6166 - val\_accuracy: 0.0119
50000/50000 [=====] - 15s 306us/step - loss: 0.6156 - accuracy: 0.0118 - val\_loss: 0.6154 - val\_accuracy: 0.0119
Epoch 24/50
50000/50000 [=====] - 15s 308us/step - loss: 0.6142 - accuracy: 0.0118 - val\_loss: 0.6142 - val\_accuracy: 0.0119
Epoch 25/50
50000/50000 [=====] - 15s 308us/step - loss: 0.6142 - accuracy: 0.0118 - val\_loss: 0.6142 - val\_accuracy: 0.0119

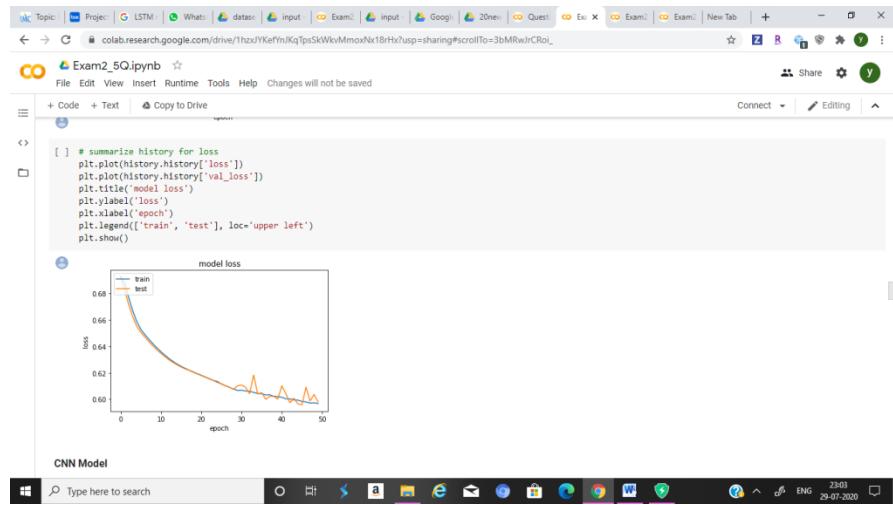
- Now we have to apply auto encoder on this data which helps to convert each datatype into a similar one so the model doesn't create preferences among the datatypes.
- Later we are going to divide the data into train data and test data, now we are going to apply CNN model on the data and we are going to predict the images and later that we are going to calculate the accuracy.



```
# showcase result of autoencoder on a random image from test dataset
# display original
plt.imshow(x_test[j].reshape(32,32,3))
plt.show()
```

The screenshot shows a Jupyter Notebook cell with the code above. It displays a 32x32 pixel image with various colors and patterns, representing a handwritten digit.





```

[ ] model = models.Sequential()
model.add(layers.Conv2D(32, (3, 3), padding='same', activation='relu', input_shape=(32,32,3), kernel_constraint=maxnorm(3)))
model.add(layers.Dropout(0.2))
model.add(layers.Conv2D(32, (3, 3), activation='relu', padding='same', kernel_constraint=maxnorm(3)))
model.add(layers.MaxPooling2D(pool_size=(2, 2)))
model.add(layers.Flatten())
model.add(layers.Dense(512, activation='relu', kernel_constraint=maxnorm(3)))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(num_classes, activation='softmax'))
model.summary()

```

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)	4104848

- And we are going to plot the accuracy and loss.
- Now we are going to apply PCA on the dataset and later we are going to feed this data onto CNN model and calculate the accuracy and loss again and compare the plots in the both cases and decide on which is a good model.

Topic | Project | G LSTM | WhatsApp | dataset | input | Exam2 | input | Google | 20newsgroups | Quest | Exam2 | Exam2 | New Tab | +

← → 🔍 colab.research.google.com/drive/1hzxJYKef1nJKqTpSkWkvMmoxNx18rHx?usp=sharing#scrollTo=3bMRwJrCRoi\_

### Exam2\_5Q.ipynb

File Edit View Insert Runtime Tools Help Changes will not be saved

Connect | Editing | ^

Accuracy: 18.5%

```

plt.figure()
plt.plot(historynew.history["val_loss"], label="validation loss")
plt.plot(historynew.history["val_accuracy"], label="validation accuracy")
plt.plot(historynew.history["loss"], label="training Loss")
plt.plot(historynew.history["accuracy"], label="training accuracy")
plt.title("the loss and accuracy for both training data and validation data")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="best")
# historynew.history

```

Topic | Project | G LSTM | WhatsApp | dataset | input | Exam2 | input | Google | 20newsgroups | Quest | Exam2 | Exam2 | New Tab | +

← → 🔍 colab.research.google.com/drive/1hzxJYKef1nJKqTpSkWkvMmoxNx18rHx?usp=sharing#scrollTo=3bMRwJrCRoi\_

### Exam2\_5Q.ipynb

File Edit View Insert Runtime Tools Help Changes will not be saved

Connect | Editing | ^

```

1563/1563 [=====] - 258s 165ms/step - loss: 0.9424 - accuracy: 0.6608 - val_loss: 1.5820 - val_accuracy: 0.4999
[ ] Epoch 47/50
1563/1563 [=====] - 260s 166ms/step - loss: 0.9356 - accuracy: 0.6632 - val_loss: 1.5898 - val_accuracy: 0.5030
[ ] Epoch 48/50
1563/1563 [=====] - 257s 165ms/step - loss: 0.9232 - accuracy: 0.6675 - val_loss: 1.6064 - val_accuracy: 0.5043
[ ] Epoch 49/50
1563/1563 [=====] - 260s 167ms/step - loss: 0.9139 - accuracy: 0.6674 - val_loss: 1.5987 - val_accuracy: 0.4998
[ ] Epoch 50/50
1563/1563 [=====] - 259s 166ms/step - loss: 0.9087 - accuracy: 0.6690 - val_loss: 1.6147 - val_accuracy: 0.4999

[ ] # Final evaluation of the model
scores = model.evaluate(x_test2, y_test, verbose=0)
print("Accuracy: %.2f%%" % (scores[1]*100))

```

Accuracy: 49.99%

```

[ ] plt.figure()
plt.plot(np.arange(0, 50), historynew.history["val_loss"], label="validation loss")
plt.plot(np.arange(0, 50), historynew.history["val_accuracy"], label="validation accuracy")
plt.plot(np.arange(0, 50), historynew.history["loss"], label="training Loss")
plt.plot(np.arange(0, 50), historynew.history["accuracy"], label="training accuracy")
plt.title("the loss and accuracy for both training data and validation data")
plt.xlabel("Epoch 50")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="best")

```

UNC Topic: Project LSTM Whats dataset input Exam2 input Google 20newsgroups Quest Exam2 Exam2 New Tab + - X

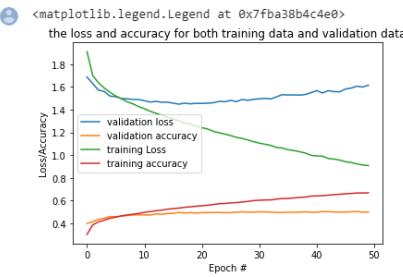
← → ⌂ colab.research.google.com/drive/1hzxJYKefyNJKqTpSkWkvMmxNx18rHx?usp=sharing#scrollTo=3bMRwJrCRoi\_

**Exam2\_5Q.ipynb** ☆

File Edit View Insert Runtime Tools Help Changes will not be saved

+ Code + Text Copy to Drive Connect Editing

```
[ ] plt.figure()
plt.plot(np.arange(0, 50), historynew.history["val_loss"], label="validation loss")
plt.plot(np.arange(0, 50), historynew.history["val_accuracy"], label="validation accuracy")
plt.plot(np.arange(0, 50), historynew.history["loss"], label="training Loss")
plt.plot(np.arange(0, 50), historynew.history["accuracy"], label="training accuracy")
plt.title("the loss and accuracy for both training data and validation data")
plt.xlabel("Epoch #")
plt.ylabel("Loss/Accuracy")
plt.legend(loc="best")
```



### PCA + CNN

Windows Type here to search ⌂ 23:04 ENG 29-07-2020

UNC Topic: Project LSTM Whats dataset input Exam2 input Google 20newsgroups Quest Exam2 Exam2 New Tab + - X

← → ⌂ colab.research.google.com/drive/1hzxJYKefyNJKqTpSkWkvMmxNx18rHx?usp=sharing#scrollTo=3bMRwJrCRoi\_

**Exam2\_5Q.ipynb** ☆

File Edit View Insert Runtime Tools Help Changes will not be saved

+ Code + Text Copy to Drive Connect Editing

```
[ ] model1.add(layers.Flatten())
[ ] model1.add(layers.Dense(32, activation='relu', kernel_constraint=maxnorm(3)))
model1.add(layers.MaxPooling2D(pool_size=(2, 2)))
model1.add(layers.Flatten())
model1.add(layers.Dense(512, activation='relu', kernel_constraint=maxnorm(3)))
model1.add(layers.Dropout(0.5))
model1.add(layers.Dense(num_classes, activation='softmax'))
model1.summary()
```

```
Model: "sequential_1"
Layer (type)          Output Shape         Param #
conv2d_2 (Conv2D)     (None, 32, 32, 32)   896
dropout_2 (Dropout)   (None, 32, 32, 32)   0
conv2d_3 (Conv2D)     (None, 32, 32, 32)   9248
max_pooling2d_1 (MaxPooling2D) (None, 16, 16, 32) 0
flatten_1 (Flatten)   (None, 8192)        0
dense_2 (Dense)       (None, 512)         4194816
dropout_3 (Dropout)   (None, 512)         0
dense_3 (Dense)       (None, 10)          5130
Total params: 4,210,096
Trainable params: 4,210,096
Non-trainable params: 0
```

Windows Type here to search ⌂ 23:04 ENG 29-07-2020

- We observed that the accuracy is more when we do apply the auto encoder and then sequential model on the dataset.

## **DATASETS:**

The datasets we are going to use in this project are review's sentiment dataset, 20news groups and 10-monkey species and Cifar\_10 dataset and in the text classification task we are going to use information from Newyork Times Headlines and comments.

1st Question:

<https://drive.google.com/drive/folders/1StULLg3upQ24LoxDirsqSBVXzg1QCij>

2nd Question:

[https://drive.google.com/drive/folders/1apbbjmYR\\_BBzl9mT2ZygSyCVPqp-6LjV](https://drive.google.com/drive/folders/1apbbjmYR_BBzl9mT2ZygSyCVPqp-6LjV)

3rd Question:

<https://drive.google.com/drive/folders/1wYGvd62UFyeoyq3kOlIspPti48KhqEP0>

4th Question

<https://drive.google.com/drive/folders/1pydNdvIxDujfb3eJqD1W5mo58gkfsA>

5th Question

From the library

## **PARAMETERS:**

The parameters we are going to take into consideration in order to evaluate the precision of our model and to figure out which model is best we are considering accuracy and loss.

## **EVALUATION AND DISCUSSION:**

- While doing this project we observed that embedding layer and autoencoding and scaling helps with better accuracy.
- We did faced with the occurrence of overfitting in the data which doesn't really help with the data prediction and we did discuss on how to remove this and we figured out that using learning rate will eliminate this

## **CONCLUSION:**

In this project we implemented sequential classification models on different data sets ,we did image classification and text classification and we observed that embedding layer, auto encoding improves the accuracy of the data prediction and when compared to CNN model LSTM has better results and in machine learning LSTM is the mostly use model in the neural networks.

## **VIDEO LINK:**

<https://umkc.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=1982343f-f8ef-4a88-9c50-ac080034d724>

## **GITHUB Source code link:**

<https://github.com/RoshiniVarada/PythonProject2>