**Source Code for Project Exam-2**

**Task 1:**

import pandas as pd

from keras.models import Sequential

from keras\_preprocessing.sequence import pad\_sequences

from keras\_preprocessing.text import Tokenizer

from sklearn import preprocessing

from keras.layers import Dense, Dropout, Conv1D, GlobalMaxPooling1D

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

import matplotlib.pyplot as plt

from tensorflow.python.keras.layers import Embedding

from tensorflow.python.keras.utils.np\_utils import to\_categorical

# Read file

train = pd.read\_csv("train.tsv", sep="\t")

# Assign Value

X = train['Phrase'].values

y = train['Sentiment'].values

# Tokenizing

tokenizer = Tokenizer(num\_words=2000)

tokenizer.fit\_on\_texts(X)

X = tokenizer.texts\_to\_sequences(X)

X = pad\_sequences(X)

# Encoding

le = preprocessing.LabelEncoder()

y = le.fit\_transform(y)

y = to\_categorical(y)

# Training and testing

x\_train, x\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=1000)

# CNN layers

model = Sequential()

model.add(Embedding(2000, X.shape[1]))

model.add(Dropout(0.2))

model.add(Conv1D(filters=32, kernel\_size=3, padding='same', activation='relu'))

model.add(GlobalMaxPooling1D())

model.add(Dense(64, activation='relu'))

model.add(Dropout(0.2))

model.add(Dense(y.shape[1], activation='softmax'))

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

history = model.fit(x\_train, y\_train, epochs=10, verbose=True, validation\_data=(x\_test, y\_test), batch\_size=256)

# Accuracy score

scores = model.evaluate(x\_test, y\_test)

print("Model accuracy: " + str((scores[1]\*100)))

# Plot 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='best')

plt.show()

**Task 2:**

from sklearn.feature\_extraction.text import TfidfVectorizer

from keras.preprocessing import sequence

from sklearn.feature\_extraction.text import TfidfVectorizer

import numpy as np

import matplotlib.pyplot as plt

from keras.layers import Dense, Embedding

from keras.layers import LSTM

from sklearn.datasets import fetch\_20newsgroups

from keras.models import Sequential

data = fetch\_20newsgroups(subset='train')

data.target\_names

train\_data = fetch\_20newsgroups(subset='train', categories=k, random\_state=32)

test\_data = fetch\_20newsgroups(subset='test',categories=k, random\_state=32)

train\_target = train\_data.target

test\_target = test\_data.target

max\_features = 20000

batch\_size = 32

k=['alt.atheism','rec.autos']

vectorizer = TfidfVectorizer(min\_df=0.01, max\_df=0.95)

train\_data = vectorizer.fit\_transform(train\_data.data)

test\_data = vectorizer.transform(test\_data.data)

train\_data = train\_data.todense()

test\_data = test\_data.todense()

train\_data.shape

test\_data.shape

train\_target.shape

test\_target.shape

n\_train = train\_data.shape[0]

train\_data = train\_data[:n\_train, :]

train\_target = train\_target[:n\_train]

model = Sequential()

model.add(Embedding(max\_features, 256))

model.add(LSTM(128, dropout=0.5, recurrent\_dropout=0.5))

model.add(Dense(1, activation='softmax'))

model.compile(loss='binary\_crossentropy',

optimizer='rmsprop',

metrics=['accuracy'])

# model.fit(train\_data, train\_labels,

# batch\_size=batch\_size,

# epochs=1,

# validation\_data=(test\_data, train\_labels))

history = model.fit(train\_data, train\_target, epochs=2, verbose=True, validation\_data=(train\_data, train\_target), batch\_size=32)

score, accuracy = model.evaluate(test\_data, test\_target, batch\_size=batch\_size)

print('Test score:', score)

print('Test accuracy:', accuracy)

# print(train\_data.shape)

plt.plot(history.history['accuracy'])

plt.plot(history.history['val\_accuracy'])

plt.title('Model Accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.legend(['train\_data', 'test\_data'], loc='upper left')

plt.show()

# For Loss Values

plt.plot(history.history['loss'])

plt.plot(history.history['val\_loss'])

plt.title('Model Loss')

plt.ylabel('Loss')

plt.xlabel('Epoch')

plt.legend(['train\_data', 'test\_data'], loc='upper left')

plt.show()

from keras.models import Sequential

from keras import layers

from keras.preprocessing.text import Tokenizer

import pandas as pd

from sklearn import preprocessing

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

import matplotlib.pyplot as plt

from sklearn.datasets import fetch\_20newsgroups

from keras.layers.embeddings import Embedding

from keras.layers import Flatten

twenty\_train = fetch\_20newsgroups(subset='train', shuffle=True)

y = twenty\_train.target

sentences = twenty\_train.data

max\_review\_len = max([len(s.split()) for s in sentences])

tokenizer = Tokenizer(num\_words=2000)

tokenizer.fit\_on\_texts(sentences)

sentences = tokenizer.texts\_to\_matrix(sentences)

le = preprocessing.LabelEncoder()

y = le.fit\_transform(y)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(sentences, y, test\_size=0.25, random\_state=1000)

# Build model

model = Sequential()

model.add(layers.Dense(300, input\_dim=max\_review\_len, activation='relu'))

model.add(layers.Dense(20,activation='softmax'))

model.compile(loss='sparse\_categorical\_crossentropy', optimizer='adam', metrics=['acc'])

history = model.fit(X\_train, y\_train, epochs=5, verbose=True, validation\_data=(X\_test, y\_test), batch\_size=256)

# Output

loss, accuracy = model.evaluate(X\_test, y\_test)

print("Loss: {}".format(loss))

print("Accuracy: {}".format(accuracy))

# For Accuracy Values

plt.plot(history.history['acc'])

plt.plot(history.history['val\_acc'])

plt.title('Model Accuracy')

plt.ylabel('Accuracy')

plt.xlabel('Epoch')

plt.legend(['train', 'test'], loc='upper left')

plt.show()

# For Loss Values

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 left')

plt.show()

**Task 3:**

1. **With Scaling and image prediction:**

import matplotlib.pylab as plt

from keras.layers import Dense, Dropout, Activation, Flatten, Conv2D, MaxPooling2D, Lambda, MaxPool2D, BatchNormalization, Input

from keras.models import Sequential

from keras.utils import to\_categorical

from keras.preprocessing.image import ImageDataGenerator

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

from pathlib import Path

from keras.optimizers import Adam,RMSprop,SGD

import pandas as pd

import random

import os

import cv2

from keras.preprocessing.image import img\_to\_array, load\_img

import numpy as np

from keras.utils import np\_utils

from keras.models import Sequential, load\_model

from keras.layers import Dense, Dropout, Flatten

from keras.layers.convolutional import Conv2D, MaxPooling2D

from keras.constraints import maxnorm

df = pd.read\_csv("/content/drive/My Drive/monkey\_labels.txt")

print(df)

#label info

cols = ['Label','Latin Name', 'Common Name','Train Images', 'Validation Images']

labels = pd.read\_csv("/content/drive/My Drive/monkey\_labels.txt", names=cols, skiprows=1)

labels = labels['Common Name']

print(labels)

height=150

width=150

channels=3

batch\_size=32

seed=1337

train\_dir = Path('/content/drive/My Drive/training/training/')

test\_dir = Path('/content/drive/My Drive/validation/validation/')

# Training generator

train\_datagen = ImageDataGenerator(rescale=1./255)

train\_generator = train\_datagen.flow\_from\_directory(train\_dir, target\_size=(height,width),batch\_size=batch\_size,seed=seed,class\_mode='categorical')

# Test generator

test\_datagen = ImageDataGenerator(rescale=1./255)

test\_generator = test\_datagen.flow\_from\_directory(test\_dir, target\_size=(height,width),batch\_size=batch\_size, seed=seed,class\_mode='categorical')

def image\_show(num\_image,label):

for i in range(num\_image):

imgdir = Path('/content/drive/My Drive/training/training/'+ label)

print(imgdir)

imgfile = random.choice(os.listdir(imgdir))

print(imgfile)

img = cv2.imread('/content/drive/My Drive/training/training/'+ label +'/'+ imgfile)

print(img.shape)

print(label)

plt.figure(i)

plt.imshow(img)

plt.title(imgfile)

plt.show()

print(labels[4])

image\_show(2,'n4')

def read\_data(path):

'''This function reads images from folder'''

images = []

labels = []

count = -1

for root, folder, file in os.walk(path):

for f in file:

file\_path = os.path.join(root, f)

img = load\_img(file\_path, target\_size=(32, 32))

img = img\_to\_array(img)

img = img.reshape(img.shape)

images.append(img)

labels.append([count])

count += 1

images = np.array(images)

labels = np.array(labels)

return images, labels

#Read images from folders

x\_train, y\_train = read\_data(train\_dir)

x\_test, y\_test = read\_data(test\_dir)

#Normalize data

x\_train = x\_train.astype('float32')

x\_test = x\_test.astype('float32')

# Scaling

x\_train = x\_train / 255.0

x\_test = x\_test / 255.0

#One hot encode data

y\_train = np\_utils.to\_categorical(y\_train)

y\_test = np\_utils.to\_categorical(y\_test)

num\_classes = y\_test.shape[1]

# Creating model

model = Sequential()

model.add(Conv2D(32, (3, 3), input\_shape=x\_train.shape[1:], padding='same', activation='relu', kernel\_constraint=maxnorm(3)))

model.add(Dropout(0.2))

model.add(Conv2D(32, (3, 3), activation='relu', padding='same', kernel\_constraint=maxnorm(3)))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Conv2D(64, (3, 3), padding='same', activation='relu', kernel\_constraint=maxnorm(3)))

model.add(Dropout(0.2))

model.add(Conv2D(64, (3, 3), activation='relu', padding='same', kernel\_constraint=maxnorm(3)))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Flatten())

model.add(Dense(512, activation='relu', kernel\_constraint=maxnorm(3)))

model.add(Dropout(0.5))

model.add(Dense(num\_classes, activation='softmax'))

# Compile model

epochs = 25

lrate = 0.01

decay = lrate/epochs

sgd = SGD(lr=lrate, momentum=0.9, decay=decay, nesterov=False)

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

# Fitting model

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

print(model.summary())

print(history.history.keys())

acc = history.history['accuracy']

val\_acc = history.history['val\_accuracy']

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()

# Save model

model.save('model\_Q3.h5')

#Load model

model = load\_model('model\_Q3.h5')

print(model.summary())

#Evaluate model

score = model.evaluate(x\_test, y\_test, verbose=0)

print('Loss:', score[0])

print('Accuracy: ', score[1])

# Prediction on model

num\_to\_text = {

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"

}

test\_images = x\_test

test\_labels = y\_test

def make\_prediction(i):

test\_img = test\_images[i]

test\_data= x\_test[[i], :]

plt.imshow(test\_img, cmap=plt.get\_cmap('gray'))

plt.title("Model Prediction: {}".format(num\_to\_text[model.predict\_classes(test\_data)[0]]))

plt.show()

prediction\_idx = [0,1, 2, 3]

for idx in prediction\_idx:

make\_prediction(idx)

1. **Without Scaling:**

import matplotlib.pylab as plt

from keras.layers import Dense, Dropout, Activation, Flatten, Conv2D, MaxPooling2D, Lambda, MaxPool2D, BatchNormalization, Input

from keras.models import Sequential

from keras.utils import to\_categorical

from keras.preprocessing.image import ImageDataGenerator

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

from pathlib import Path

from keras.optimizers import Adam,RMSprop,SGD

import pandas as pd

import random

import os

import cv2

from keras.preprocessing.image import img\_to\_array, load\_img

import numpy as np

from keras.utils import np\_utils

from keras.models import Sequential, load\_model

from keras.layers import Dense, Dropout, Flatten

from keras.layers.convolutional import Conv2D, MaxPooling2D

from keras.constraints import maxnorm

df = pd.read\_csv("/content/drive/My Drive/monkey\_labels.txt")

print(df)

#label info

cols = ['Label','Latin Name', 'Common Name','Train Images', 'Validation Images']

labels = pd.read\_csv("/content/drive/My Drive/monkey\_labels.txt", names=cols, skiprows=1)

labels = labels['Common Name']

print(labels)

height=150

width=150

channels=3

batch\_size=32

seed=1337

train\_dir = Path('/content/drive/My Drive/training/training/')

test\_dir = Path('/content/drive/My Drive/validation/validation/')

# Training generator

train\_datagen = ImageDataGenerator(rescale=1./255)

train\_generator = train\_datagen.flow\_from\_directory(train\_dir, target\_size=(height,width),batch\_size=batch\_size,seed=seed,class\_mode='categorical')

# Test generator

test\_datagen = ImageDataGenerator(rescale=1./255)

test\_generator = test\_datagen.flow\_from\_directory(test\_dir, target\_size=(height,width),batch\_size=batch\_size,

seed=seed,class\_mode='categorical')

def image\_show(num\_image,label):

for i in range(num\_image):

imgdir = Path('/content/drive/My Drive/training/training/'+ label)

print(imgdir)

imgfile = random.choice(os.listdir(imgdir))

print(imgfile)

img = cv2.imread('/content/drive/My Drive/training/training/'+ label +'/'+ imgfile)

print(img.shape)

print(label)

plt.figure(i)

plt.imshow(img)

plt.title(imgfile)

plt.show()

print(labels[4])

image\_show(3,'n4')

def read\_data(path):

'''This function reads images from folder'''

images = []

labels = []

count = -1

for root, folder, file in os.walk(path):

for f in file:

file\_path = os.path.join(root, f)

img = load\_img(file\_path, target\_size=(32, 32))

img = img\_to\_array(img)

img = img.reshape(img.shape)

images.append(img)

labels.append([count])

count += 1

images = np.array(images)

labels = np.array(labels)

return images, labels

#Read images from folders

x\_train, y\_train = read\_data(train\_dir)

x\_test, y\_test = read\_data(test\_dir)

#Normalize data

x\_train = x\_train.astype('float32')

x\_test = x\_test.astype('float32')

#One hot encode data

y\_train = np\_utils.to\_categorical(y\_train)

y\_test = np\_utils.to\_categorical(y\_test)

num\_classes = y\_test.shape[1]

# Creating model

model = Sequential()

model.add(Conv2D(32, (3, 3), input\_shape=x\_train.shape[1:], padding='same', activation='relu', kernel\_constraint=maxnorm(3)))

model.add(Dropout(0.2))

model.add(Conv2D(32, (3, 3), activation='relu', padding='same', kernel\_constraint=maxnorm(3)))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Conv2D(64, (3, 3), padding='same', activation='relu', kernel\_constraint=maxnorm(3)))

model.add(Dropout(0.2))

model.add(Conv2D(64, (3, 3), activation='relu', padding='same', kernel\_constraint=maxnorm(3)))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Flatten())

model.add(Dense(512, activation='relu', kernel\_constraint=maxnorm(3)))

model.add(Dropout(0.5))

model.add(Dense(num\_classes, activation='softmax'))

# Compile model

epochs = 25

lrate = 0.01

decay = lrate/epochs

sgd = SGD(lr=lrate, momentum=0.9, decay=decay, nesterov=False)

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

# Fitting model

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

print(model.summary())

print(history.history.keys())

acc = history.history['accuracy']

val\_acc = history.history['val\_accuracy']

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()

# Save model

model.save('model\_Q3.h5')

#Load model

model = load\_model('model\_Q3.h5')

print(model.summary())

#Evaluate model

score = model.evaluate(x\_test, y\_test, verbose=0)

print('Loss:', score[0])

print('Accuracy: ', score[1])

**Task 4:**

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

dir = '/content/newyork\_headline.zip'

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)

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]

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]

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()

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

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))

print(generate\_text("joe biden", 20, model, max\_sequence\_len))

print(generate\_text("donald trump", 9, model, max\_sequence\_len))

**Task 5:**

**Code for part-1**

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

#Autoencoder with single encoding and decoding layers

input\_img = Input(shape=(3072,))

encoded = Dense(32, activation='relu')(input\_img)

output\_img = Dense(3072, activation='sigmoid')(encoded)

# this model maps an input to its reconstruction

autoencoder = Model(input\_img, output\_img)

autoencoder.compile(optimizer='adadelta', loss='binary\_crossentropy', metrics=['accuracy'])

from keras.datasets import cifar10

# Downloading dataset from internet

(x\_train,y\_train), (x\_test, y\_test) = cifar10.load\_data()

#Normalizing the values from 0 to 255 == 0 to 1

x\_train = x\_train.astype('float32')

x\_test = x\_test.astype('float32')

x\_train = x\_train / 255

x\_test = x\_test / 255

#Reshaping the data into 2d

x\_train = x\_train.reshape((len(x\_train), np.prod(x\_train.shape[1:])))

x\_test = x\_test.reshape((len(x\_test), np.prod(x\_test.shape[1:])))

print(x\_train.shape)

print(x\_test.shape)

# Fitting the data to autoencoder

history = autoencoder.fit(x\_train, x\_train,

epochs=10,

batch\_size=256,

shuffle=True,

validation\_data=(x\_test, x\_test))

# Copying xtrain data and xtext data into new variables so as to use in the next step

x\_train1 = x\_train

for k in range(0, 50000):

prediction = autoencoder.predict(x\_train[k].reshape(1,3072))

x\_train1[k] = prediction

x\_test1 = x\_test.copy()

for k in range(0, 10000):

prediction = autoencoder.predict(x\_test[k].reshape(1,3072))

x\_test1[k] = prediction

x\_train1 = np.array(x\_train1).reshape([-1, 32, 32, 3]) x\_test1 = np.array(x\_test1).reshape([-1, 32, 32, 3]) print(x\_train1.shape) print(x\_test1.shape)

#Preparing model for CNN model

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(0.3))

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'))

epochs = 20

lrate = 0.01

decay = lrate/epochs

sgd = keras.optimizers.SGD(lr=lrate, momentum=0.9, decay=decay, nesterov=False)

model.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=True), optimizer=sgd, metrics=['accuracy'])

print(model.summary())

print(x\_test1.shape)

print(x\_train1.shape)

print(y\_test.shape)

print(y\_train.shape)

history1 = model.fit(x\_train1, y\_train, validation\_data=(x\_test1, y\_test), epochs=epochs, batch\_size=32)

scores = model.evaluate(x\_test1, y\_test, verbose=0)

print("Accuracy: %.2f%%" % (scores[1]\*100))

**code for part-2**

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.optimizers import RMSprop

from keras.models import Model

from keras.callbacks import TensorBoard

from keras import regularizers

from keras import backend as K

# downloading data from internet

(x\_train, y\_train), (x\_test, y\_test) = cifar10.load\_data()

# normalize inputs from 0-255 to 0.0-1.0

x\_train = x\_train.astype('uint8') / 255.

x\_test = x\_test.astype('uint8') / 255.

#Reshaping the data into 2d

x\_train = x\_train.reshape((len(x\_train), np.prod(x\_train.shape[1:])))

x\_test = x\_test.reshape((len(x\_test), np.prod(x\_test.shape[1:])))

print(x\_train.shape)

print(x\_test.shape)

from sklearn.decomposition import PCA from sklearn.preprocessing import StandardScaler standardscaler = StandardScaler() pca = PCA() pca.fit(x\_train)

x\_train\_pca = pca.transform(x\_train)

x\_test\_pca = pca.transform(x\_test)

x\_train\_pca = x\_train\_pca.reshape(-1, 32,32,3)

x\_test\_pca = x\_test\_pca.reshape(-1, 32,32,3)

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(10, activation='softmax'))

epochs = 20

lrate = 0.01

decay = lrate/epochs

sgd = keras.optimizers.SGD(lr=lrate, momentum=0.9, decay=decay, nesterov=False)

model.compile(loss=tf.keras.losses.SparseCategoricalCrossentropy(from\_logits=True), optimizer=sgd, metrics=['accuracy'])

model.summary()

history2 = model.fit(x\_train\_pca, y\_train, validation\_data=(x\_test\_pca, y\_test), epochs=epochs, batch\_size=32)

scores = model.evaluate(x\_test\_pca, y\_test, verbose=0)

print("Accuracy: %.2f%%" % (scores[1]\*100))