**PRE 413: Software Applications for DSP and Communications**

**Fall 2022**

**Assignment 7 of Lecture 7**

**Instructor: Prof. Mohammed Abo-Zahhad**

**Student ID: 012022003**

**Student Name: Kibitok Abraham**

**15 December 2022**

1. The code below has been written to demonstrate the power of Tensorflow and Keras in solving ANN classification of handwritten digits’ problem. Execute the code and print the first digit used for training and testing. Also find the resulting training and testing accuracy and cost functions. (Hint: The MNIST dataset is a part of Keras library).

import tensorflow as tf

mnist = tf.keras.datasets.mnist

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

x\_train, x\_test = x\_train / 255.0, x\_test / 255.0

print('X\_train:', x\_train.shape)

print('y\_train:', y\_train.shape)

print('X\_test:', x\_test.shape)

print('y\_test:', y\_test.shape)

model = tf.keras.models.Sequential([ tf.keras.layers.Flatten(input\_shape=(28, 28)),

tf.keras.layers.Dense(512, activation=tf.nn.relu), tf.keras.layers.Dropout(0.2),

tf.keras.layers.Dense(10, activation=tf.nn.softmax) ])

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

model.fit(x\_train, y\_train, epochs=5)

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

**Solution**



# Importig tensorflow module

import tensorflow as tf

#Importing MNIST Datasets frm tensorflow

mnist = tf.keras.datasets.mnist

#Extracting training and testing datasets from MNIST

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

x\_train, x\_test = x\_train / 255.0, x\_test / 255.0

#Printing the size and shape of datasets

print('X\_train:', x\_train.shape)

print('y\_train:', y\_train.shape)

print('X\_test:', x\_test.shape)

print('y\_test:', y\_test.shape)

#Building the CNN network

model = tf.keras.models.Sequential([

tf.keras.layers.Flatten(input\_shape=(28, 28)),

tf.keras.layers.Dense(512, activation=tf.nn.relu),

tf.keras.layers.Dropout(0.2),

tf.keras.layers.Dense(10, activation=tf.nn.softmax)

])

# Compiling the model

model.compile(optimizer='adam',

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

# Fitting the model

model.fit(x\_train, y\_train, epochs=5)

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

X\_train: (60000, 28, 28)

y\_train: (60000,)

X\_test: (10000, 28, 28)

y\_test: (10000,)

Epoch 1/5

1875/1875 [==============================] - 6s 3ms/step - loss: 0.2159 - accuracy: 0.9355

Epoch 2/5

1875/1875 [==============================] - 5s 2ms/step - loss: 0.0956 - accuracy: 0.9707

Epoch 3/5

1875/1875 [==============================] - 5s 2ms/step - loss: 0.0671 - accuracy: 0.9796

Epoch 4/5

1875/1875 [==============================] - 5s 3ms/step - loss: 0.0548 - accuracy: 0.9825

Epoch 5/5

1875/1875 [==============================] - 5s 2ms/step - loss: 0.0437 - accuracy: 0.9863

313/313 [==============================] - 1s 2ms/step - loss: 0.0670 - accuracy: 0.9800

Out = [0.0669843852519989, 0.9800000190734863]

Testing accuracy = 0.9800

Testing loss = 0.0670

Training accuracy = 0.9863

Training loss= 0.0437

Cost function = 0.0669843852519989

1. Execute the code below for small and large ANN models and print the results (Hint: The MNIST dataset is a part of the Keras library.]

|  |  |
| --- | --- |
| 1. Small Model | 1. Large Model |

****

**Small Model**

import tensorflow as tf

mnist = tf.keras.datasets.mnist

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

x\_train, x\_test = x\_train / 255.0, x\_test / 255.0

print('X\_train:', x\_train.shape)

print('y\_train:', y\_train.shape)

print('X\_test:', x\_test.shape)

print('y\_test:', y\_test.shape)

model = tf.keras.Sequential([

tf.keras.layers.Dense(1)

])

model.compile(loss = tf.keras.losses.mae,

optimizer= tf.keras.optimizers.SGD(),

metrics=["mae"])

model.fit(x\_train, y\_train, epochs=5)

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

**Solution**

X\_train: (60000, 28, 28)

y\_train: (60000,)

X\_test: (10000, 28, 28)

y\_test: (10000,)

Epoch 1/5

1875/1875 [==============================] - 2s 785us/step - loss: 2.7746 - mae: 2.7746

Epoch 2/5

1875/1875 [==============================] - 1s 779us/step - loss: 2.5344 - mae: 2.5344

Epoch 3/5

1875/1875 [==============================] - 1s 745us/step - loss: 2.5243 - mae: 2.5243

Epoch 4/5

1875/1875 [==============================] - 2s 841us/step - loss: 2.5233 - mae: 2.5233

Epoch 5/5

1875/1875 [==============================] - 1s 766us/step - loss: 2.5226 - mae: 2.5226

313/313 [==============================] - 0s 645us/step - loss: 2.5265 - mae: 2.5265

Out[3]:

[2.5265252590179443, 2.526524066925049]

**Large Model**

import tensorflow as tf

mnist = tf.keras.datasets.mnist

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

x\_train, x\_test = x\_train / 255.0, x\_test / 255.0

print('X\_train:', x\_train.shape)

print('y\_train:', y\_train.shape)

print('X\_test:', x\_test.shape)

print('y\_test:', y\_test.shape)

model = tf.keras.Sequential([

tf.keras.layers.Dense(100, activation='relu'),

tf.keras.layers.Dense(100, activation='relu'),

tf.keras.layers.Dense(100, activation='relu'),

tf.keras.layers.Dense(1)

])

model.compile(loss = tf.keras.losses.mae,

optimizer= tf.keras.optimizers.Adam(lr=0.0001),

metrics=["mae"])

model.fit(x\_train, y\_train, epochs=100)

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

**Solution**

X\_train: (60000, 28, 28)

y\_train: (60000,)

X\_test: (10000, 28, 28)

y\_test: (10000,)

Epoch 1/100

1875/1875 [==============================] - 6s 3ms/step - loss: 2.6650 - mae: 2.6650

Epoch 2/100

1875/1875 [==============================] - 5s 2ms/step - loss: 2.5272 - mae: 2.5272

Epoch 3/100

1875/1875 [==============================] - 5s 3ms/step - loss: 2.5272 - mae: 2.5272

Epoch 4/100

1875/1875 [==============================] - 5s 3ms/step - loss: 2.5266 - mae: 2.5266

Epoch 5/100

1875/1875 [==============================] - 5s 3ms/step - loss: 2.5263 - mae: 2.5263

Epoch 6/100

1875/1875 [==============================] - 7s 4ms/step - loss: 2.5263 - mae: 2.5263

Epoch 7/100

1875/1875 [==============================] - 6s 3ms/step - loss: 2.5262 - mae: 2.5262

Epoch 8/100

1875/1875 [==============================] - 5s 3ms/step - loss: 2.5248 - mae: 2.5248

Epoch 9/100

1875/1875 [==============================] - 5s 3ms/step - loss: 2.5254 - mae: 2.5254

Epoch 10/100

1875/1875 [==============================] - 5s 3ms/step - loss: 2.5249 - mae: 2.5249

Epoch 11/100

1875/1875 [==============================] - 5s 3ms/step - loss: 2.5253 - mae: 2.5253

Epoch 12/100

1875/1875 [==============================] - 5s 3ms/step - loss: 2.5250 - mae: 2.5250

Epoch 13/100

1875/1875 [==============================] - 4s 2ms/step - loss: 2.5246 - mae: 2.5246

Epoch 14/100

1875/1875 [==============================] - 4s 2ms/step - loss: 2.5251 - mae: 2.5251

Epoch 15/100

1875/1875 [==============================] - 4s 2ms/step - loss: 2.5243 - mae: 2.5243

Epoch 16/100

1875/1875 [==============================] - 5s 2ms/step - loss: 2.5242 - mae: 2.5242

Epoch 17/100

1875/1875 [==============================] - 4s 2ms/step - loss: 2.5240 - mae: 2.5240

Epoch 18/100

1875/1875 [==============================] - 4s 2ms/step - loss: 2.5239 - mae: 2.5239

Epoch 19/100

1875/1875 [==============================] - 4s 2ms/step - loss: 2.5237 - mae: 2.5237

Epoch 20/100

1875/1875 [==============================] - 5s 2ms/step - loss: 2.5246 - mae: 2.5246

Epoch 21/100

1875/1875 [==============================] - 4s 2ms/step - loss: 2.5237 - mae: 2.5237

Epoch 22/100

1875/1875 [==============================] - 5s 2ms/step - loss: 2.5245 - mae: 2.5245

Epoch 23/100

1875/1875 [==============================] - 5s 3ms/step - loss: 2.5234 - mae: 2.5234

Epoch 24/100

1875/1875 [==============================] - 4s 2ms/step - loss: 2.5235 - mae: 2.5235

Epoch 25/100

1875/1875 [==============================] - 5s 2ms/step - loss: 2.5232 - mae: 2.5232

Epoch 26/100

1875/1875 [==============================] - 4s 2ms/step - loss: 2.5237 - mae: 2.5237

Epoch 27/100

1875/1875 [==============================] - 4s 2ms/step - loss: 2.5229 - mae: 2.5229

Epoch 28/100

1875/1875 [==============================] - 4s 2ms/step - loss: 2.5233 - mae: 2.5233

Epoch 29/100

1875/1875 [==============================] - 4s 2ms/step - loss: 2.5236 - mae: 2.5236

Epoch 30/100

1875/1875 [==============================] - 4s 2ms/step - loss: 2.5234 - mae: 2.5234

Epoch 31/100

1875/1875 [==============================] - 5s 2ms/step - loss: 2.5235 - mae: 2.5235

Epoch 32/100

1875/1875 [==============================] - 5s 3ms/step - loss: 2.5235 - mae: 2.5235

Epoch 33/100

1875/1875 [==============================] - 5s 3ms/step - loss: 2.5232 - mae: 2.5232

Epoch 34/100

1875/1875 [==============================] - 5s 3ms/step - loss: 2.5230 - mae: 2.5230

Epoch 35/100

1875/1875 [==============================] - 5s 3ms/step - loss: 2.5225 - mae: 2.5225

Epoch 36/100

1875/1875 [==============================] - 4s 2ms/step - loss: 2.5234 - mae: 2.5234

Epoch 37/100

1875/1875 [==============================] - 5s 2ms/step - loss: 2.5233 - mae: 2.5233

Epoch 38/100

1875/1875 [==============================] - 4s 2ms/step - loss: 2.5231 - mae: 2.5231

Epoch 39/100

1875/1875 [==============================] - 4s 2ms/step - loss: 2.5226 - mae: 2.5226

Epoch 40/100

1875/1875 [==============================] - 5s 2ms/step - loss: 2.5229 - mae: 2.5229

Epoch 41/100

1875/1875 [==============================] - 5s 3ms/step - loss: 2.5224 - mae: 2.5224

Epoch 42/100

1875/1875 [==============================] - 5s 2ms/step - loss: 2.5228 - mae: 2.5228

Epoch 43/100

1875/1875 [==============================] - 4s 2ms/step - loss: 2.5224 - mae: 2.5224

Epoch 44/100

1875/1875 [==============================] - 4s 2ms/step - loss: 2.5228 - mae: 2.5228

Epoch 45/100

1875/1875 [==============================] - 4s 2ms/step - loss: 2.5228 - mae: 2.5228

Epoch 46/100

1875/1875 [==============================] - 9s 5ms/step - loss: 2.5226 - mae: 2.5226

Epoch 47/100

1875/1875 [==============================] - 5s 3ms/step - loss: 2.5226 - mae: 2.5226

Epoch 48/100

1875/1875 [==============================] - 5s 3ms/step - loss: 2.5227 - mae: 2.5227

Epoch 49/100

1875/1875 [==============================] - 5s 3ms/step - loss: 2.5219 - mae: 2.5219

Epoch 50/100

1875/1875 [==============================] - 5s 3ms/step - loss: 2.5225 - mae: 2.5225

Epoch 51/100

1875/1875 [==============================] - 5s 3ms/step - loss: 2.5221 - mae: 2.5221

Epoch 52/100

1875/1875 [==============================] - 5s 2ms/step - loss: 2.5226 - mae: 2.5226

Epoch 53/100

1875/1875 [==============================] - 5s 3ms/step - loss: 2.5228 - mae: 2.5228

Epoch 54/100

1875/1875 [==============================] - 6s 3ms/step - loss: 2.5221 - mae: 2.5221

Epoch 55/100

1875/1875 [==============================] - 5s 3ms/step - loss: 2.5225 - mae: 2.5225

Epoch 56/100

1875/1875 [==============================] - 5s 2ms/step - loss: 2.5226 - mae: 2.5226

Epoch 57/100

1875/1875 [==============================] - 5s 3ms/step - loss: 2.5229 - mae: 2.5229

Epoch 58/100

1875/1875 [==============================] - 5s 3ms/step - loss: 2.5218 - mae: 2.5218

Epoch 59/100

1875/1875 [==============================] - 6s 3ms/step - loss: 2.5229 - mae: 2.5229

Epoch 60/100

1875/1875 [==============================] - 5s 3ms/step - loss: 2.5222 - mae: 2.5222

Epoch 61/100

1875/1875 [==============================] - 5s 3ms/step - loss: 2.5220 - mae: 2.5220

Epoch 62/100

1875/1875 [==============================] - 5s 3ms/step - loss: 2.5224 - mae: 2.5224

Epoch 63/100

1875/1875 [==============================] - 5s 3ms/step - loss: 2.5225 - mae: 2.5225

Epoch 64/100

1875/1875 [==============================] - 5s 2ms/step - loss: 2.5226 - mae: 2.5226

Epoch 65/100

1875/1875 [==============================] - 5s 3ms/step - loss: 2.5224 - mae: 2.5224

Epoch 66/100

1875/1875 [==============================] - 5s 3ms/step - loss: 2.5222 - mae: 2.5222

Epoch 67/100

1875/1875 [==============================] - 4s 2ms/step - loss: 2.5221 - mae: 2.5221

Epoch 68/100

1875/1875 [==============================] - 4s 2ms/step - loss: 2.5222 - mae: 2.5222

Epoch 69/100

1875/1875 [==============================] - 4s 2ms/step - loss: 2.5223 - mae: 2.5223

Epoch 70/100

1875/1875 [==============================] - 5s 3ms/step - loss: 2.5222 - mae: 2.5222

Epoch 71/100

1875/1875 [==============================] - 4s 2ms/step - loss: 2.5226 - mae: 2.5226

Epoch 72/100

1875/1875 [==============================] - 4s 2ms/step - loss: 2.5219 - mae: 2.5219

Epoch 73/100

1875/1875 [==============================] - 5s 3ms/step - loss: 2.5222 - mae: 2.5222

Epoch 74/100

1875/1875 [==============================] - 4s 2ms/step - loss: 2.5220 - mae: 2.5220

Epoch 75/100

1875/1875 [==============================] - 5s 2ms/step - loss: 2.5222 - mae: 2.5222

Epoch 76/100

1875/1875 [==============================] - 4s 2ms/step - loss: 2.5222 - mae: 2.5222

Epoch 77/100

1875/1875 [==============================] - 5s 3ms/step - loss: 2.5217 - mae: 2.5217

Epoch 78/100

1875/1875 [==============================] - 5s 2ms/step - loss: 2.5224 - mae: 2.5224

Epoch 79/100

1875/1875 [==============================] - 4s 2ms/step - loss: 2.5219 - mae: 2.5219

Epoch 80/100

1875/1875 [==============================] - 5s 2ms/step - loss: 2.5222 - mae: 2.5222

Epoch 81/100

1875/1875 [==============================] - 4s 2ms/step - loss: 2.5216 - mae: 2.5216

Epoch 82/100

1875/1875 [==============================] - 4s 2ms/step - loss: 2.5220 - mae: 2.5220

Epoch 83/100

1875/1875 [==============================] - 4s 2ms/step - loss: 2.5220 - mae: 2.5220

Epoch 84/100

1875/1875 [==============================] - 4s 2ms/step - loss: 2.5221 - mae: 2.5221

Epoch 85/100

1875/1875 [==============================] - 4s 2ms/step - loss: 2.5217 - mae: 2.5218

Epoch 86/100

1875/1875 [==============================] - 4s 2ms/step - loss: 2.5217 - mae: 2.5217

Epoch 87/100

1875/1875 [==============================] - 4s 2ms/step - loss: 2.5216 - mae: 2.5216

Epoch 88/100

1875/1875 [==============================] - 4s 2ms/step - loss: 2.5220 - mae: 2.5220

Epoch 89/100

1875/1875 [==============================] - 4s 2ms/step - loss: 2.5216 - mae: 2.5216

Epoch 90/100

1875/1875 [==============================] - 4s 2ms/step - loss: 2.5220 - mae: 2.5220

Epoch 91/100

1875/1875 [==============================] - 4s 2ms/step - loss: 2.5217 - mae: 2.5217

Epoch 92/100

1875/1875 [==============================] - 4s 2ms/step - loss: 2.5216 - mae: 2.5216

Epoch 93/100

1875/1875 [==============================] - 4s 2ms/step - loss: 2.5215 - mae: 2.5215

Epoch 94/100

1875/1875 [==============================] - 4s 2ms/step - loss: 2.5218 - mae: 2.5218

Epoch 95/100

1875/1875 [==============================] - 4s 2ms/step - loss: 2.5214 - mae: 2.5214

Epoch 96/100

1875/1875 [==============================] - 4s 2ms/step - loss: 2.5215 - mae: 2.5215

Epoch 97/100

1875/1875 [==============================] - 4s 2ms/step - loss: 2.5218 - mae: 2.5218

Epoch 98/100

1875/1875 [==============================] - 4s 2ms/step - loss: 2.5219 - mae: 2.5219

Epoch 99/100

1875/1875 [==============================] - 4s 2ms/step - loss: 2.5212 - mae: 2.5212

Epoch 100/100

1875/1875 [==============================] - 4s 2ms/step - loss: 2.5216 - mae: 2.5216

313/313 [==============================] - 1s 1ms/step - loss: 2.5244 - mae: 2.5244

Out[4]:

**[2.524366855621338, 2.5243685245513916]**

1. Using Keras and TensorFlow, plot the linear, sigmoid, and ReLU activation functions.

**Answer**

****

#**Sigmoid**

# Example plot for the sigmoid activation function

from math import exp

from matplotlib import pyplot

# sigmoid activation function

def sigmoid(x):

return 1.0 / (1.0 + exp(-x))

# define input data

inputs = [x for x in range(-10, 10)]

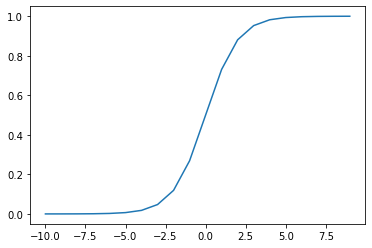
# calculate outputs

outputs = [sigmoid(x) for x in inputs]

# plot inputs vs outputs

pyplot.plot(inputs, outputs)

pyplot.show()



# **Linear**

# example for the linearactivation function

from matplotlib import pyplot

# linear activation function

def linear(x):

return x

# define input data

inputs = [x for x in range(-10, 10)]

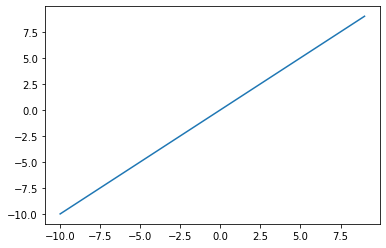
# calculate outputs

outputs = [linear(x) for x in inputs]

# plot inputs vs outputs

pyplot.plot(inputs, outputs)

pyplot.show()



# **ReLU**

# example plot for the relu activation function

from matplotlib import pyplot

# rectified linear function

def rectified(x):

return max(0.0, x)

# define input data

inputs = [x for x in range(-10, 10)]

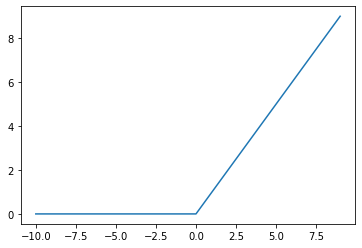
# calculate outputs

outputs = [rectified(x) for x in inputs]

# plot inputs vs outputs

pyplot.plot(inputs, outputs)

pyplot.show()



1. Find the ideal learning rate by evaluating the MNIST dataset using the following TensorFlow code.

****

**Solution**

****

import tensorflow as tf

import numpy as np

import matplotlib.pyplot as plt

mnist = tf.keras.datasets.mnist

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

x\_train, x\_test = x\_train / 255.0, x\_test / 255.0

print('X\_train:', x\_train.shape)

print('y\_train:', y\_train.shape)

print('X\_test:', x\_test.shape)

print('y\_test:', y\_test.shape)

model = tf.keras.models.Sequential([

tf.keras.layers.Flatten(input\_shape=(28, 28)),

tf.keras.layers.Dense(512, activation=tf.nn.relu),

tf.keras.layers.Dropout(0.2),

tf.keras.layers.Dense(10, activation=tf.nn.softmax)

])

model.compile(optimizer='adam',

loss='sparse\_categorical\_crossentropy',

metrics=['accuracy'])

#model.summary()

#Learning rate scheduler

lr\_scheduler = tf.keras.callbacks.LearningRateScheduler(lambda epoch: 1e-4\*10\*\*(epoch/20))

history = model.fit(x\_train,

y\_train,

epochs=100,

callbacks =[lr\_scheduler])

lrs = 1e-4\*(10\*\*(np.arange(100)/20))

plt.figure(figsize=(10,7))

plt.semilogx(lrs, history.history["loss"])

plt.xlabel("Learning rate")

plt.ylabel("Loss")

plt.title("Learning rate vs Loss")

#model.evaluate(x\_test, y\_test)

**Output**

X\_train: (60000, 28, 28)

y\_train: (60000,)

X\_test: (10000, 28, 28)

y\_test: (10000,)

Epoch 1/100

1875/1875 [==============================] - 5s 2ms/step - loss: 0.4785 - accuracy: 0.8766 - lr: 1.0000e-04

Epoch 2/100

1875/1875 [==============================] - 5s 2ms/step - loss: 0.2275 - accuracy: 0.9359 - lr: 1.1220e-04

Epoch 3/100

1875/1875 [==============================] - 4s 2ms/step - loss: 0.1708 - accuracy: 0.9510 - lr: 1.2589e-04

Epoch 4/100

1875/1875 [==============================] - 4s 2ms/step - loss: 0.1344 - accuracy: 0.9617 - lr: 1.4125e-04

Epoch 5/100

1875/1875 [==============================] - 4s 2ms/step - loss: 0.1085 - accuracy: 0.9692 - lr: 1.5849e-04

Epoch 6/100

1875/1875 [==============================] - 5s 2ms/step - loss: 0.0900 - accuracy: 0.9735 - lr: 1.7783e-04

Epoch 7/100

1875/1875 [==============================] - 4s 2ms/step - loss: 0.0755 - accuracy: 0.9782 - lr: 1.9953e-04

Epoch 8/100

1875/1875 [==============================] - 4s 2ms/step - loss: 0.0634 - accuracy: 0.9812 - lr: 2.2387e-04

Epoch 9/100

1875/1875 [==============================] - 4s 2ms/step - loss: 0.0547 - accuracy: 0.9843 - lr: 2.5119e-04

Epoch 10/100

1875/1875 [==============================] - 4s 2ms/step - loss: 0.0481 - accuracy: 0.9854 - lr: 2.8184e-04

Epoch 11/100

1875/1875 [==============================] - 4s 2ms/step - loss: 0.0404 - accuracy: 0.9880 - lr: 3.1623e-04

Epoch 12/100

1875/1875 [==============================] - 4s 2ms/step - loss: 0.0364 - accuracy: 0.9893 - lr: 3.5481e-04

Epoch 13/100

1875/1875 [==============================] - 5s 2ms/step - loss: 0.0315 - accuracy: 0.9906 - lr: 3.9811e-04

Epoch 14/100

1875/1875 [==============================] - 4s 2ms/step - loss: 0.0279 - accuracy: 0.9912 - lr: 4.4668e-04

Epoch 15/100

1875/1875 [==============================] - 4s 2ms/step - loss: 0.0274 - accuracy: 0.9915 - lr: 5.0119e-04

Epoch 16/100

1875/1875 [==============================] - 4s 2ms/step - loss: 0.0264 - accuracy: 0.9915 - lr: 5.6234e-04

Epoch 17/100

1875/1875 [==============================] - 5s 2ms/step - loss: 0.0269 - accuracy: 0.9910 - lr: 6.3096e-04

Epoch 18/100

1875/1875 [==============================] - 4s 2ms/step - loss: 0.0257 - accuracy: 0.9912 - lr: 7.0795e-04

Epoch 19/100

1875/1875 [==============================] - 4s 2ms/step - loss: 0.0275 - accuracy: 0.9903 - lr: 7.9433e-04

Epoch 20/100

1875/1875 [==============================] - 5s 3ms/step - loss: 0.0262 - accuracy: 0.9912 - lr: 8.9125e-04

Epoch 21/100

1875/1875 [==============================] - 4s 2ms/step - loss: 0.0295 - accuracy: 0.9902 - lr: 0.0010

Epoch 22/100

1875/1875 [==============================] - 4s 2ms/step - loss: 0.0322 - accuracy: 0.9895 - lr: 0.0011

Epoch 23/100

1875/1875 [==============================] - 5s 2ms/step - loss: 0.0369 - accuracy: 0.9876 - lr: 0.0013

Epoch 24/100

1875/1875 [==============================] - 5s 3ms/step - loss: 0.0409 - accuracy: 0.9871 - lr: 0.0014

Epoch 25/100

1875/1875 [==============================] - 5s 3ms/step - loss: 0.0460 - accuracy: 0.9858 - lr: 0.0016

Epoch 26/100

1875/1875 [==============================] - 5s 2ms/step - loss: 0.0507 - accuracy: 0.9852 - lr: 0.0018

Epoch 27/100

1875/1875 [==============================] - 5s 3ms/step - loss: 0.0568 - accuracy: 0.9844 - lr: 0.0020

Epoch 28/100

1875/1875 [==============================] - 5s 2ms/step - loss: 0.0620 - accuracy: 0.9840 - lr: 0.0022

Epoch 29/100

1875/1875 [==============================] - 5s 2ms/step - loss: 0.0733 - accuracy: 0.9816 - lr: 0.0025

Epoch 30/100

1875/1875 [==============================] - 5s 2ms/step - loss: 0.0822 - accuracy: 0.9802 - lr: 0.0028

Epoch 31/100

1875/1875 [==============================] - 5s 3ms/step - loss: 0.0869 - accuracy: 0.9799 - lr: 0.0032

Epoch 32/100

1875/1875 [==============================] - 5s 2ms/step - loss: 0.0972 - accuracy: 0.9791 - lr: 0.0035

Epoch 33/100

1875/1875 [==============================] - 5s 2ms/step - loss: 0.1168 - accuracy: 0.9775 - lr: 0.0040

Epoch 34/100

1875/1875 [==============================] - 5s 3ms/step - loss: 0.1315 - accuracy: 0.9756 - lr: 0.0045

Epoch 35/100

1875/1875 [==============================] - 5s 2ms/step - loss: 0.1510 - accuracy: 0.9738 - lr: 0.0050

Epoch 36/100

1875/1875 [==============================] - 5s 2ms/step - loss: 0.1749 - accuracy: 0.9714 - lr: 0.0056

Epoch 37/100

1875/1875 [==============================] - 5s 3ms/step - loss: 0.1917 - accuracy: 0.9700 - lr: 0.0063

Epoch 38/100

1875/1875 [==============================] - 5s 3ms/step - loss: 0.2327 - accuracy: 0.9669 - lr: 0.0071

Epoch 39/100

1875/1875 [==============================] - 5s 3ms/step - loss: 0.2273 - accuracy: 0.9672 - lr: 0.0079

Epoch 40/100

1875/1875 [==============================] - 5s 2ms/step - loss: 0.2722 - accuracy: 0.9639 - lr: 0.0089

Epoch 41/100

1875/1875 [==============================] - 5s 3ms/step - loss: 0.2900 - accuracy: 0.9599 - lr: 0.0100

Epoch 42/100

1875/1875 [==============================] - 5s 2ms/step - loss: 0.3474 - accuracy: 0.9590 - lr: 0.0112

Epoch 43/100

1875/1875 [==============================] - 5s 2ms/step - loss: 0.3621 - accuracy: 0.9564 - lr: 0.0126

Epoch 44/100

1875/1875 [==============================] - 5s 3ms/step - loss: 0.4577 - accuracy: 0.9473 - lr: 0.0141

Epoch 45/100

1875/1875 [==============================] - 5s 2ms/step - loss: 0.4196 - accuracy: 0.9460 - lr: 0.0158

Epoch 46/100

1875/1875 [==============================] - 5s 3ms/step - loss: 0.5425 - accuracy: 0.9352 - lr: 0.0178

Epoch 47/100

1875/1875 [==============================] - 5s 2ms/step - loss: 0.5337 - accuracy: 0.9309 - lr: 0.0200

Epoch 48/100

1875/1875 [==============================] - 5s 3ms/step - loss: 0.5230 - accuracy: 0.9226 - lr: 0.0224

Epoch 49/100

1875/1875 [==============================] - 5s 2ms/step - loss: 0.6202 - accuracy: 0.9105 - lr: 0.0251

Epoch 50/100

1875/1875 [==============================] - 5s 2ms/step - loss: 0.6088 - accuracy: 0.8990 - lr: 0.0282

Epoch 51/100

1875/1875 [==============================] - 5s 3ms/step - loss: 0.6837 - accuracy: 0.8809 - lr: 0.0316

Epoch 52/100

1875/1875 [==============================] - 5s 2ms/step - loss: 0.7607 - accuracy: 0.8608 - lr: 0.0355

Epoch 53/100

1875/1875 [==============================] - 5s 2ms/step - loss: 0.8171 - accuracy: 0.8353 - lr: 0.0398

Epoch 54/100

1875/1875 [==============================] - 5s 3ms/step - loss: 0.9265 - accuracy: 0.8159 - lr: 0.0447

Epoch 55/100

1875/1875 [==============================] - 5s 3ms/step - loss: 0.9759 - accuracy: 0.7781 - lr: 0.0501

Epoch 56/100

1875/1875 [==============================] - 5s 2ms/step - loss: 1.1386 - accuracy: 0.7578 - lr: 0.0562

Epoch 57/100

1875/1875 [==============================] - 5s 2ms/step - loss: 1.1786 - accuracy: 0.6948 - lr: 0.0631

Epoch 58/100

1875/1875 [==============================] - 5s 3ms/step - loss: 1.2977 - accuracy: 0.6694 - lr: 0.0708

Epoch 59/100

1875/1875 [==============================] - 5s 2ms/step - loss: 1.3920 - accuracy: 0.6276 - lr: 0.0794

Epoch 60/100

1875/1875 [==============================] - 5s 2ms/step - loss: 1.5785 - accuracy: 0.5918 - lr: 0.0891

Epoch 61/100

1875/1875 [==============================] - 5s 3ms/step - loss: 1.5638 - accuracy: 0.5785 - lr: 0.1000

Epoch 62/100

1875/1875 [==============================] - 5s 2ms/step - loss: 1.7108 - accuracy: 0.5013 - lr: 0.1122

Epoch 63/100

1875/1875 [==============================] - 5s 3ms/step - loss: 1.8211 - accuracy: 0.4492 - lr: 0.1259

Epoch 64/100

1875/1875 [==============================] - 5s 3ms/step - loss: 2.0732 - accuracy: 0.3509 - lr: 0.1413

Epoch 65/100

1875/1875 [==============================] - 5s 3ms/step - loss: 2.0071 - accuracy: 0.3335 - lr: 0.1585

Epoch 66/100

1875/1875 [==============================] - 5s 2ms/step - loss: 2.0764 - accuracy: 0.3143 - lr: 0.1778

Epoch 67/100

1875/1875 [==============================] - 5s 2ms/step - loss: 2.4264 - accuracy: 0.2225 - lr: 0.1995

Epoch 68/100

1875/1875 [==============================] - 5s 3ms/step - loss: 2.1905 - accuracy: 0.2141 - lr: 0.2239

Epoch 69/100

1875/1875 [==============================] - 4s 2ms/step - loss: 2.3466 - accuracy: 0.1772 - lr: 0.2512

Epoch 70/100

1875/1875 [==============================] - 4s 2ms/step - loss: 2.4466 - accuracy: 0.1428 - lr: 0.2818

Epoch 71/100

1875/1875 [==============================] - 5s 2ms/step - loss: 2.3226 - accuracy: 0.1481 - lr: 0.3162

Epoch 72/100

1875/1875 [==============================] - 5s 2ms/step - loss: 2.2821 - accuracy: 0.1408 - lr: 0.3548

Epoch 73/100

1875/1875 [==============================] - 4s 2ms/step - loss: 2.2888 - accuracy: 0.1238 - lr: 0.3981

Epoch 74/100

1875/1875 [==============================] - 4s 2ms/step - loss: 2.3332 - accuracy: 0.1210 - lr: 0.4467

Epoch 75/100

1875/1875 [==============================] - 5s 2ms/step - loss: 2.3939 - accuracy: 0.1114 - lr: 0.5012

Epoch 76/100

1875/1875 [==============================] - 4s 2ms/step - loss: 2.3588 - accuracy: 0.1088 - lr: 0.5623

Epoch 77/100

1875/1875 [==============================] - 4s 2ms/step - loss: 2.3613 - accuracy: 0.1055 - lr: 0.6310

Epoch 78/100

1875/1875 [==============================] - 5s 2ms/step - loss: 2.5559 - accuracy: 0.1085 - lr: 0.7079

Epoch 79/100

1875/1875 [==============================] - 5s 2ms/step - loss: 3.2780 - accuracy: 0.1073 - lr: 0.7943

Epoch 80/100

1875/1875 [==============================] - 4s 2ms/step - loss: 2.4101 - accuracy: 0.1037 - lr: 0.8913

Epoch 81/100

1875/1875 [==============================] - 5s 2ms/step - loss: 2.4533 - accuracy: 0.1063 - lr: 1.0000

Epoch 82/100

1875/1875 [==============================] - 5s 3ms/step - loss: 2.4128 - accuracy: 0.1016 - lr: 1.1220

Epoch 83/100

1875/1875 [==============================] - 5s 2ms/step - loss: 2.4234 - accuracy: 0.1036 - lr: 1.2589

Epoch 84/100

1875/1875 [==============================] - 5s 2ms/step - loss: 2.4450 - accuracy: 0.1032 - lr: 1.4125

Epoch 85/100

1875/1875 [==============================] - 5s 3ms/step - loss: 2.4547 - accuracy: 0.1010 - lr: 1.5849

Epoch 86/100

1875/1875 [==============================] - 5s 2ms/step - loss: 2.9143 - accuracy: 0.1020 - lr: 1.7783

Epoch 87/100

1875/1875 [==============================] - 5s 2ms/step - loss: 2.4905 - accuracy: 0.1041 - lr: 1.9953

Epoch 88/100

1875/1875 [==============================] - 5s 2ms/step - loss: 2.5138 - accuracy: 0.1030 - lr: 2.2387

Epoch 89/100

1875/1875 [==============================] - 5s 3ms/step - loss: 2.5529 - accuracy: 0.1019 - lr: 2.5119

Epoch 90/100

1875/1875 [==============================] - 5s 2ms/step - loss: 2.5860 - accuracy: 0.1023 - lr: 2.8184

Epoch 91/100

1875/1875 [==============================] - 5s 2ms/step - loss: 2.5961 - accuracy: 0.1014 - lr: 3.1623

Epoch 92/100

1875/1875 [==============================] - 5s 3ms/step - loss: 2.6184 - accuracy: 0.1035 - lr: 3.5481

Epoch 93/100

1875/1875 [==============================] - 5s 2ms/step - loss: 2.6667 - accuracy: 0.1047 - lr: 3.9811

Epoch 94/100

1875/1875 [==============================] - 5s 2ms/step - loss: 4.8517 - accuracy: 0.1034 - lr: 4.4668

Epoch 95/100

1875/1875 [==============================] - 5s 2ms/step - loss: 2.7438 - accuracy: 0.1024 - lr: 5.0119

Epoch 96/100

1875/1875 [==============================] - 5s 3ms/step - loss: 2.8162 - accuracy: 0.1026 - lr: 5.6234

Epoch 97/100

1875/1875 [==============================] - 5s 2ms/step - loss: 2.8464 - accuracy: 0.1040 - lr: 6.3096

Epoch 98/100

1875/1875 [==============================] - 5s 2ms/step - loss: 2.9374 - accuracy: 0.1031 - lr: 7.0795

Epoch 99/100

1875/1875 [==============================] - 5s 3ms/step - loss: 3.0081 - accuracy: 0.1040 - lr: 7.9433

Epoch 100/100

1875/1875 [==============================] - 5s 2ms/step - loss: 3.1236 - accuracy: 0.1029 - lr: 8.9125

Out[5]:

Text(0.5, 1.0, 'Learning rate vs Loss')

A picture containing chart

Description automatically generated

**The best Learning rate is 10-3**

1. **Project 1:** Keras and Tensorflow can be used to classify cats and dogs. Build a CNN that is trained on a few thousand images of cats and dogs. Use the dataset from the website [**https://www.kaggle.com/competitions/dogs-vs-cats/data**](https://www.kaggle.com/competitions/dogs-vs-cats/data)**.**  Explain in detail the implementation of this project. You can find more comments on the website <https://becominghuman.ai/building-an-image-classifier-using-deep-learning-in-python-totally-from-a-beginners-perspective-be8dbaf22dd8>. The complete code of this project is given as follows.

# Importing the Keras libraries and packages  
from keras.models import Sequential  
from keras.layers import Conv2D  
from keras.layers import MaxPooling2D  
from keras.layers import Flatten  
from keras.layers import Dense  
# Initialising the CNN  
classifier = Sequential()  
# Step 1 - Convolution  
classifier.add(Conv2D(32, (3, 3), input\_shape = (64, 64, 3), activation = 'relu'))  
# Step 2 - Pooling  
classifier.add(MaxPooling2D(pool\_size = (2, 2)))  
# Adding a second convolutional layer  
classifier.add(Conv2D(32, (3, 3), activation = 'relu'))  
classifier.add(MaxPooling2D(pool\_size = (2, 2)))  
# Step 3 - Flattening  
classifier.add(Flatten())  
# Step 4 - Full connection  
classifier.add(Dense(units = 128, activation = 'relu'))  
classifier.add(Dense(units = 1, activation = 'sigmoid'))  
# Compiling the CNN  
classifier.compile(optimizer = 'adam', loss = 'binary\_crossentropy', metrics = ['accuracy'])  
# Part 2 - Fitting the CNN to the images  
from keras.preprocessing.image import ImageDataGenerator  
train\_datagen = ImageDataGenerator(rescale = 1./255,  
shear\_range = 0.2,  
zoom\_range = 0.2,  
horizontal\_flip = True)  
test\_datagen = ImageDataGenerator(rescale = 1./255)  
training\_set = train\_datagen.flow\_from\_directory('dataset/training\_set',  
target\_size = (64, 64),  
batch\_size = 32,  
class\_mode = 'binary')  
test\_set = test\_datagen.flow\_from\_directory('dataset/test\_set',  
target\_size = (64, 64),  
batch\_size = 32,  
class\_mode = 'binary')  
classifier.fit\_generator(training\_set,  
steps\_per\_epoch = 8000,  
epochs = 25,  
validation\_data = test\_set,  
validation\_steps = 2000)  
# Part 3 - Making new predictions  
import numpy as np  
from keras.preprocessing import image  
test\_image = image.load\_img('dataset/single\_prediction/cat\_or\_dog\_1.jpg', target\_size = (64, 64))  
test\_image = image.img\_to\_array(test\_image)  
test\_image = np.expand\_dims(test\_image, axis = 0)  
result = classifier.predict(test\_image)  
training\_set.class\_indices  
if result[0][0] == 1:  
prediction = 'dog'  
else:  
prediction = 'cat'

**Solution**

****

# Convolutional Neural Network

# Installing Theano

# pip install --upgrade --no-deps git+git://github.com/Theano/Theano.git

# Installing Tensorflow

# pip install tensorflow

# Installing Keras

# pip install --upgrade keras

# Part 1 - Building the CNN

# Importing the Keras libraries and packages

from keras.models import Sequential

from keras.layers import Conv2D

from keras.layers import MaxPooling2D

from keras.layers import Flatten

from keras.layers import Dense

# Initialising the CNN

classifier = Sequential()

# Step 1 - Convolution

classifier.add(Conv2D(32, (3, 3), input\_shape = (64, 64, 3), activation = 'relu'))

# Step 2 - Pooling

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

# Adding a second convolutional layer

classifier.add(Conv2D(32, (3, 3), activation = 'relu'))

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

# Step 3 - Flattening

classifier.add(Flatten())

# Step 4 - Full connection

classifier.add(Dense(units = 128, activation = 'relu'))

classifier.add(Dense(units = 1, activation = 'sigmoid'))

# Compiling the CNN

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

# Part 2 - Fitting the CNN to the images

from keras.preprocessing.image import ImageDataGenerator

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

shear\_range = 0.2,

zoom\_range = 0.2,

horizontal\_flip = True)

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

training\_set = train\_datagen.flow\_from\_directory('dataset/training\_set',

target\_size = (64, 64),

batch\_size = 32,

class\_mode = 'binary')

test\_set = test\_datagen.flow\_from\_directory('dataset/test\_set',

target\_size = (64, 64),

batch\_size = 32,

class\_mode = 'binary')

batch\_size = 32

steps\_per\_epoch = len(training\_set)//batch\_size

validation\_steps = len(test\_set)//batch\_size # if you have validation data

#Training the model

classifier.fit\_generator(training\_set,

steps\_per\_epoch = steps\_per\_epoch,

epochs = 25,

validation\_data = test\_set,

validation\_steps = validation\_steps)

#Testing with a new image

import numpy as np

import keras.utils as image

test\_image = image.load\_img('dataset/single\_prediction/dog\_cat1.jpg', target\_size = (64, 64))

test\_image = image.img\_to\_array(test\_image)

test\_image = np.expand\_dims(test\_image, axis = 0)

result = classifier.predict(test\_image)

training\_set.class\_indices

if result[0][0] == 1:

prediction = 'dog'

else:

prediction = 'cat'

print(prediction)

Found 8005 images belonging to 2 classes.

Found 2023 images belonging to 2 classes.

Epoch 1/25

C:\Users\bettk\AppData\Local\Temp\ipykernel\_12132\1640425861.py:70: UserWarning: `Model.fit\_generator` is deprecated and will be removed in a future version. Please use `Model.fit`, which supports generators.

classifier.fit\_generator(training\_set,

7/7 [==============================] - 2s 152ms/step - loss: 0.7310 - accuracy: 0.4911 - val\_loss: 0.6958 - val\_accuracy: 0.4219

Epoch 2/25

7/7 [==============================] - 1s 118ms/step - loss: 0.6943 - accuracy: 0.4732 - val\_loss: 0.7007 - val\_accuracy: 0.4688

Epoch 3/25

7/7 [==============================] - 1s 119ms/step - loss: 0.6932 - accuracy: 0.5179 - val\_loss: 0.6880 - val\_accuracy: 0.6094

Epoch 4/25

7/7 [==============================] - 1s 119ms/step - loss: 0.6921 - accuracy: 0.5357 - val\_loss: 0.6893 - val\_accuracy: 0.5625

Epoch 5/25

7/7 [==============================] - 1s 122ms/step - loss: 0.6928 - accuracy: 0.5000 - val\_loss: 0.6870 - val\_accuracy: 0.5312

Epoch 6/25

7/7 [==============================] - 1s 123ms/step - loss: 0.6920 - accuracy: 0.5089 - val\_loss: 0.6879 - val\_accuracy: 0.5156

Epoch 7/25

7/7 [==============================] - 1s 123ms/step - loss: 0.6904 - accuracy: 0.5536 - val\_loss: 0.6848 - val\_accuracy: 0.7344

Epoch 8/25

7/7 [==============================] - 1s 118ms/step - loss: 0.6890 - accuracy: 0.5670 - val\_loss: 0.6876 - val\_accuracy: 0.5469

Epoch 9/25

7/7 [==============================] - 1s 119ms/step - loss: 0.6957 - accuracy: 0.5402 - val\_loss: 0.6774 - val\_accuracy: 0.7031

Epoch 10/25

7/7 [==============================] - 1s 118ms/step - loss: 0.6852 - accuracy: 0.5357 - val\_loss: 0.6907 - val\_accuracy: 0.5156

Epoch 11/25

7/7 [==============================] - 1s 122ms/step - loss: 0.6867 - accuracy: 0.5446 - val\_loss: 0.6823 - val\_accuracy: 0.5781

Epoch 12/25

7/7 [==============================] - 1s 118ms/step - loss: 0.6778 - accuracy: 0.5804 - val\_loss: 0.6695 - val\_accuracy: 0.5625

Epoch 13/25

7/7 [==============================] - 1s 142ms/step - loss: 0.6874 - accuracy: 0.5402 - val\_loss: 0.6727 - val\_accuracy: 0.5781

Epoch 14/25

7/7 [==============================] - 1s 129ms/step - loss: 0.6851 - accuracy: 0.5357 - val\_loss: 0.6582 - val\_accuracy: 0.5781

Epoch 15/25

7/7 [==============================] - 1s 133ms/step - loss: 0.6765 - accuracy: 0.5670 - val\_loss: 0.6629 - val\_accuracy: 0.7031

Epoch 16/25

7/7 [==============================] - 1s 119ms/step - loss: 0.6566 - accuracy: 0.6786 - val\_loss: 0.6679 - val\_accuracy: 0.5625

Epoch 17/25

7/7 [==============================] - 1s 117ms/step - loss: 0.6766 - accuracy: 0.6027 - val\_loss: 0.6689 - val\_accuracy: 0.5938

Epoch 18/25

7/7 [==============================] - 1s 124ms/step - loss: 0.6561 - accuracy: 0.6741 - val\_loss: 0.6096 - val\_accuracy: 0.7031

Epoch 19/25

7/7 [==============================] - 1s 124ms/step - loss: 0.6601 - accuracy: 0.6473 - val\_loss: 0.6462 - val\_accuracy: 0.6719

Epoch 20/25

7/7 [==============================] - 1s 122ms/step - loss: 0.6776 - accuracy: 0.6027 - val\_loss: 0.7375 - val\_accuracy: 0.5312

Epoch 21/25

7/7 [==============================] - 1s 126ms/step - loss: 0.6427 - accuracy: 0.5938 - val\_loss: 0.6066 - val\_accuracy: 0.7031

Epoch 22/25

7/7 [==============================] - 1s 132ms/step - loss: 0.6742 - accuracy: 0.5759 - val\_loss: 0.6349 - val\_accuracy: 0.6562

Epoch 23/25

7/7 [==============================] - 1s 120ms/step - loss: 0.6479 - accuracy: 0.6607 - val\_loss: 0.5969 - val\_accuracy: 0.6719

Epoch 24/25

7/7 [==============================] - 1s 124ms/step - loss: 0.6390 - accuracy: 0.6250 - val\_loss: 0.6283 - val\_accuracy: 0.6719

Epoch 25/25

7/7 [==============================] - 1s 120ms/step - loss: 0.6498 - accuracy: 0.6384 - val\_loss: 0.7159 - val\_accuracy: 0.4688

1/1 [==============================] - 0s 54ms/step

Prediction of the image = **dog**

​

**6) Project 2:** The website <https://realpython.com/python-keras-text-classification/> includes the practical project for text classification with python and Keras. Implement the project of text classification using the dataset[**https://www.kaggle.com/code/matleonard/text-classification/data**](https://www.kaggle.com/code/matleonard/text-classification/data)**.**



# Load the libraries

from keras.models import Sequential

from keras import layers

from keras.utils import pad\_sequences

from keras.preprocessing.text import Tokenizer

import pandas as pd

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

#Load the path for the dataset

filepath\_dict = {'yelp': 'data/sentiment\_analysis/yelp\_labelled.txt',

'amazon': 'data/sentiment\_analysis/amazon\_cells\_labelled.txt',

'imdb': 'data/sentiment\_analysis/imdb\_labelled.txt'}

# Create an empty list to loop through the dataset

df\_list = []

for source, filepath in filepath\_dict.items():

df = pd.read\_csv(filepath, names=['sentence', 'label'], sep='\t')

df['source'] = source # Add another column filled with the source name

df\_list.append(df)

#Concatenate the list to a dataframe

df = pd.concat(df\_list)

print(df)

#Assigning values to labels and spliting the dataset into train, test

df\_yelp = df[df['source']=='yelp']

sentences = df\_yelp['sentence'].values

y = df\_yelp['label'].values

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

# Generating the dataset

tokenizer = Tokenizer(num\_words=5000)

tokenizer.fit\_on\_texts(sentences\_train)

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

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

vocab\_size = len(tokenizer.word\_index) + 1 # Adding 1 because of reserved 0 index

# Checking the dataset

print(sentences\_train[2])

print(X\_train[2])

embedding\_dim = 100

maxlen = 100

X\_train = pad\_sequences(X\_train, padding = 'post', maxlen= maxlen)

X\_test = pad\_sequences(X\_test, padding = 'post', maxlen= maxlen)

# Developing the model with Convolution of 1d

model = Sequential()

model.add(layers.Embedding(vocab\_size, embedding\_dim, input\_length=maxlen))

model.add(layers.Conv1D(128, 5, activation='relu'))

model.add(layers.GlobalMaxPooling1D())

model.add(layers.Dense(10, activation='relu'))

model.add(layers.Dense(1, activation='sigmoid'))

model.compile(optimizer='adam',

loss='binary\_crossentropy',

metrics=['accuracy'])

model.summary()

# Training the model with dataset

history= model.fit(X\_train, y\_train,

epochs = 10,

verbose = False,

validation\_data = (X\_test, y\_test),

batch\_size = 10)

# Evaluating the model to obtain the accuracy and loss for the testing and training

loss, accuracy = model.evaluate(X\_train, y\_train, verbose=False)

print("Training Accuracy: {:.4f}".format(accuracy))

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

print("Testing Accuracy: {:.4f}".format(accuracy))

**Solution**

sentence label source

0 Wow... Loved this place. 1 yelp

1 Crust is not good. 0 yelp

2 Not tasty and the texture was just nasty. 0 yelp

3 Stopped by during the late May bank holiday of... 1 yelp

4 The selection on the menu was great and so wer... 1 yelp

.. ... ... ...

743 I just got bored watching Jessice Lange take h... 0 imdb

744 Unfortunately, any virtue in this film's produ... 0 imdb

745 In a word, it is embarrassing. 0 imdb

746 Exceptionally bad! 0 imdb

747 All in all its an insult to one's intelligence... 0 imdb

[2748 rows x 3 columns]

Of all the dishes, the salmon was the best, but all were great.

[11, 43, 1, 171, 1, 283, 3, 1, 47, 26, 43, 24, 22]

Model: "sequential\_1"

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Layer (type) Output Shape Param #

=================================================================

embedding\_1 (Embedding) (None, 100, 100) 174700

conv1d\_1 (Conv1D) (None, 96, 128) 64128

global\_max\_pooling1d\_1 (Glo (None, 128) 0

balMaxPooling1D)

dense\_2 (Dense) (None, 10) 1290

dense\_3 (Dense) (None, 1) 11

=================================================================

Total params: 240,129

Trainable params: 240,129

Non-trainable params: 0

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Training Accuracy: 1.0000

Testing Accuracy: 0.7920