Department of Data Science & Computer Applications,

Manipal Institute of Technology, Manipal

**DSE 3159 DEEP LEARNING LAB**

OBSERVATION BOOK

**Name: Sagar Kumar**

**Registration No: 210968002**

**Batch No: B1**

**Table of Contents**

[WEEK 1- Tensorflow & Keras Tutorial 3](#_Toc145874169)

[WEEK 2- EXPERIMENTING WITH DEEP NEURAL NETWORKS 7](#_Toc145874170)

[WEEK 3- CONVOLUTIONAL NEURAL NETWORKS VS FULLY CONNECTED NEURAL NETWORKS 14](#_Toc145874171)

[WEEK 4- IMPLEMENTING CONVENTIONAL CNN ARCHITECTURES AND TRANSFER LEARNING 23](#_Toc145874172)

[WEEK 5- IMPLEMENTING RECURRENT NEURAL NETWORKS FOR TIME SERIES 32](#_Toc145874173)

[FORECASTING AND STOCK MARK PREDICTION 32](#_Toc145874174)

WEEK 1- Tensorflow & Keras Tutorial

**Q1)** Accurate measurement of body fat is inconvenient/costly and it is desirable to have easy methods of predicting Body Fat. Using the Body Fat dataset, write a Neural Network to predict body fat:

a. Number of Hidden layers = 3 and number of units are 128,64,32

b. Use RELU activation function, let learning rate be 0.1

Split the data into (80,20) split and tabulate the performance in terms of RMSE for100 epochs and comment on performance.

The attributes of the dataset are :

1. Density determined from underwater weighing

2. Percent body fat from Siri's (1956) equation

3. Age (years)

4. Weight (lbs)

5. Height (inches)

6. Neck circumference (cm)

7. Chest circumference (cm)

8. Abdomen 2 circumference (cm)

9. Hip circumference (cm)

10. Thigh circumference (cm)

11. Knee circumference (cm)

12. Ankle circumference (cm)

13. Biceps (extended) circumference (cm)

14. Forearm circumference (cm)

15. Wrist circumference (cm)

**Code:**

# Imports

import pandas as pd

import tensorflow as tf

import numpy as np

import matplotlib.pyplot as plt

keras = tf.keras

from keras.models import Sequential

from keras.layers import Dense, Input

from keras.utils import split\_dataset

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

# Read csv file and split to train and test datasets

df = pd.read\_csv('./bodyfat.csv')

X = df.drop(['BodyFat'], axis=1)

y = df.BodyFat

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, shuffle=True)

# Create, compile and train FCNN model on data

# Qa)

model = Sequential([

Input(shape=(14,)),

Dense(128, activation='relu'),

Dense(64, activation='relu'),

Dense(32, activation='relu'),

Dense(1, activation='linear'),

])

# Qb)

model.compile(optimizer=tf.keras.optimizers.Adam(learning\_rate=0.1), loss='mse', metrics=['accuracy'])

epochs = 100

batch\_size = 32

obj = model.fit(tf.convert\_to\_tensor(X\_train), tf.convert\_to\_tensor(y\_train), epochs=epochs, batch\_size=batch\_size, validation\_split=0.1)

# Performance tabulation

loss, acc = model.evaluate(tf.convert\_to\_tensor(X\_test), tf.convert\_to\_tensor(y\_test))

rmse = np.sqrt(obj.history['loss'])

pd.DataFrame({"epochs": range(1, epochs + 1), "rmse": rmse})

# Plots

fig = plt.figure()

plt.xlabel("Epochs")

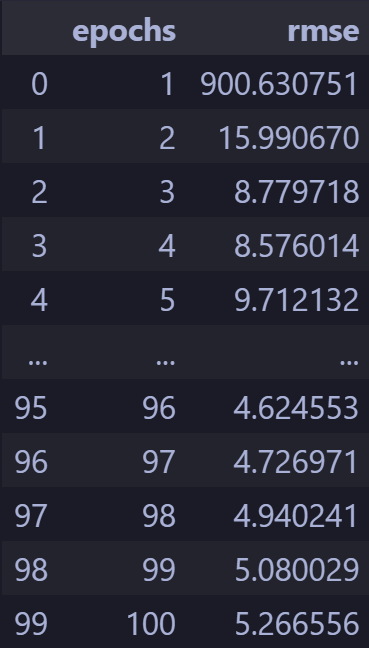
plt.ylabel("RMSE")

plt.title("RMSE vs Epochs")

plt.plot(range(1, epochs + 1), rmse)

**Results & Discussion:**

# Performance tabulation



# RMSE vs Epochs plot

A graph with a line

Description automatically generated

# RMSE vs Epochs plot w/o first 15 trials

A graph of a graph

Description automatically generated

# WEEK 2- EXPERIMENTING WITH DEEP NEURAL NETWORKS

**Q1)** Consider the following dataset ‘Churn\_Modelling.csv’

https://www.kaggle.com/datasets/aakash50897/churn-modellingcsv

The data set has 14 features which are as follows:-

1. RowNumber:- Represents the number of rows

2. CustomerId:- Represents customerId

3. Surname:- Represents surname of the customer

4. CreditScore:- Represents credit score of the customer

5. Geography:- Represents the city to which customers belongs to

6. Gender:- Represents Gender of the customer

7. Age:- Represents age of the customer

8. Tenure:- Represents tenure of the customer with a bank

9. Balance:- Represents balance hold by the customer

10. NumOfProducts:- Represents the number of bank services used by the customer

11. HasCrCard:- Represents if a customer has a credit card or not

12. IsActiveMember:- Represents if a customer is an active member or not

13. EstimatedSalary:- Represents estimated salary of the customer

14. Exited:- Represents if a customer is going to exit the bank or not.

1. Perform the required pre-processing (attribute removal, encoding, feature scaling) and write

comment lines to explain the pre-processing steps.

2. Perform experiments using (70,15,15) split and tabulate the performance in terms of Accuracy,

Precision & Recall for the following experimental setup :

1. Number of Hidden Layers and Number of Units per Layer

Number of Hidden

Layers

Number of Units Activation Function

1 128, 0 ,0 relu

2 128, 64, 0 relu

3 128, 64, 32 relu

2. Epochs (10,20,30)

3. Activation function in output layer (Sigmoid )

4. Learning rate ( 0.1, 0.01,0.001)

5. Visualize the training and validation loss against the epochs using appropriate plots.

6. Comment on performance.

**Code:**

# Imports

import tensorflow as tf

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

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

keras = tf.keras

from keras.layers import Dense

from keras import Sequential

from keras.optimizers import Adam

from keras.metrics import Recall, Precision

from sklearn.preprocessing import StandardScaler

import itertools as it

from sklearn.metrics import precision\_score, accuracy\_score, recall\_score

# Q1) Read dataset and preprocess

df = pd.read\_csv("churn.csv")

df.isna().any()

df.duplicated().any()

df = df.drop(['RowNumber', 'CustomerId', 'Surname'], axis=1)

df = pd.get\_dummies(df, dtype=np.int32)

df['Balance'] = df['Balance'].mask(df['Balance'] == 0).fillna(df['Balance'].mean())

df[['CreditScore', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'EstimatedSalary']] = StandardScaler().fit\_transform(df[['CreditScore', 'Age', 'Tenure', 'Balance', 'NumOfProducts', 'EstimatedSalary']])

# Split dataset into train and test

X = df.drop(['Exited'], axis=1)

y = df['Exited']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.15, random\_state=1)

# Q2) Create, compile and train multiple models

# Repeat 3 models 9 times

models = list(it.repeat([

Sequential([

Dense(128, activation='relu', input\_shape=(dims,)),

Dense(1, activation='sigmoid', input\_shape=(dims,)),

]),

Sequential([

Dense(128, activation='relu', input\_shape=(dims,)),

Dense(64, activation='relu', input\_shape=(dims,)),

Dense(1, activation='sigmoid', input\_shape=(dims,)),

]),

Sequential([

Dense(128, activation='relu', input\_shape=(dims,)),

Dense(64, activation='relu', input\_shape=(dims,)),

Dense(32, activation='relu', input\_shape=(dims,)),

Dense(1, activation='sigmoid', input\_shape=(dims,)),

]),

], 9))

batch\_size = 32

epochs = [10, 20, 30]

learning\_rate = [0.1, 0.01, 0.001]

# Train and compile models in a loop

for i in models:

for j in range(len(i)):

i[j].compile(optimizer=Adam(learning\_rate=learning\_rate[j]), loss='binary\_crossentropy', metrics=['accuracy', Recall(), Precision()])

trains = [

i[j].fit(X\_train, y\_train, epochs=epochs[j], batch\_size=batch\_size, validation\_split=0.15) for j in range(len(i)) for i in models

]

# Plots

epoch\_index = 0

fig, axs = plt.subplots(3, 3)

for j in range(3):

for k in range(3):

axs[j, k].plot(range(epochs[epoch\_index]), tlosses[1 + j + k])

axs[j, k].plot(range(epochs[epoch\_index]), vlosses[1 + j + k])

# Performance tabulation

acc = []

rec = []

prec = []

for i in range(9):

for model in models[i]:

y\_pred = model.predict(X\_test)

labels = np.where(y\_pred > 0.5, 1, 0)

acc.append(accuracy\_score(y\_test, labels))

rec.append(recall\_score(y\_test, labels))

prec.append(precision\_score(y\_test, labels))

**Results & Discussion:**

# Performance tabulation

A screenshot of a computer screen

Description automatically generated

# Q2.5) Plots

# Losses for 10 epochs



# Losses for 20 epochs

A screenshot of a graph

Description automatically generated

# Losses for 30 epochs

A screenshot of a graph

Description automatically generated

# Comments of performance

We notice that with more epochs, the validation loss increases and training loss decreases further. This is a clear sign of overfitting that can be fixed using regularization.

WEEK 3- CONVOLUTIONAL NEURAL NETWORKS VS FULLY CONNECTED NEURAL NETWORKS

Consider the Fashion MNIST dataset [Fashion MNIST dataset, an alternative to MNIST

(keras.io)] and do the following:

**Q1)** Understanding the Dataset and Pre-processing: Implement the following:

a. Compute and display the number of classes.

b. Compute and display the dimensions of each image.

c. Display one image from each class.

d. Perform normalization.

**Q2)** Performing experiments on Fully Connected Neural Networks (FCNN):

a. Design a FCNN which is most suitable for the given dataset:

Experimentally choose the best network (the intuitions and learnings from the

experiments you have performed in Week-1 and Week-2 will help you choose the

hyperparameters for the network).

b. Train and test the network (choose the best epoch size so that there is no overfitting).

c. Plot the performance curves.

**Q3)** Performing experiments on a Convolutional Neural Networks (CNNs):

a. Design CNN-1 which contains:

• One Convolution layer which uses 32 kernels each of size 5x5, stride = 1 and,

padding =0.

• One Pooling layer which uses MAXPOOLING with stride =2.

• One hidden layer having number of neurons = 100

b. Design CNN-2 which contains:

• Two back-to-back Convolution layers which uses 32 kernels each of size 3x3, stride

= 1, and padding =0.

• One Pooling layer which uses MAXPOOLING with stride =2.

• One hidden layer having number of neurons = 100

Note: use ReLU activation function after each convolution layer.

c. Train and test the networks (choose the best epoch size so that there is no overfitting).

d. Plot the performance curves for CNN-1 and CNN-2.

e. Compare the performances of CNN-1 and CNN-2.

**Q4)** Compare the performances of FCNN and CNN.

**Q5)** Compare the number of parameters in the FCNN and the CNN.

**Q6)** Discuss the computational efficiency of both networks. Which one took longer to train and

why?

**Code:**

# Imports

import numpy as np

import tensorflow as tf

import matplotlib.pyplot as plt

keras = tf.keras

from keras import Sequential, Input

from keras.layers import Flatten, Dense, MaxPooling2D, Conv2D

from keras.optimizers import Adam

from datetime import datetime

from tensorflow.math import confusion\_matrix

import seaborn as sns

# Load dataset

(x\_train, y\_train), (x\_test, y\_test) = tf.keras.datasets.fashion\_mnist.load\_data()

# Q1a)

set(y\_train)

# Q1b)

x\_train[0].shape

# Q1c)

c = 0

y = list(y\_train)

for i in range(y\_train.shape[0]):

if c == 10: break

plt.imshow(x\_train[y.index(c)], cmap="gray")

c += 1

plt.show()

# Q1d)

x\_train = x\_train / 255.0

x\_test = x\_test / 255.0

# Q2a)

fcnn\_model = Sequential([

Flatten(input\_shape=[28, 28], name="IL"),

Dense(300, activation="relu", name="HL1"),

Dense(100, activation="relu", name="HL2"),

Dense(10, activation="softmax", name="OL"),

])

fcnn\_model.compile(optimizer=Adam(learning\_rate=0.01), loss="sparse\_categorical\_crossentropy", metrics=["accuracy"])

# Q2b)

start = datetime.now()

fcnn\_hist = fcnn\_model.fit(x\_train, y\_train, epochs=50, batch\_size=50, validation\_split=0.1)

end = datetime.now()

fcnn\_time = end – start

# Q2c)

plt.plot(fcnn\_hist.history["loss"], label="loss")

plt.plot(fcnn\_hist.history["val\_loss"], label="val loss")

plt.xlabel("Epochs")

plt.ylabel("Loss")

plt.legend()

plt.show()

# Predictions

fcnn\_pred = np.argmax(fcnn\_model.predict(x\_test), axis=-1)

fcnn\_pred

# Q3a)

cnn1\_model = Sequential([

Input((x\_train.shape[1], x\_train.shape[2], 1)),

Conv2D(32, (5, 5), strides=(1, 1), padding="valid", activation="relu"),

MaxPooling2D(pool\_size=(2, 2), strides=2, padding="valid"),

Flatten(),

Dense(100, activation="relu"),

Dense(10, activation="sigmoid"),

])

cnn1\_model.compile(optimizer=Adam(learning\_rate=0.01), loss="sparse\_categorical\_crossentropy", metrics=["accuracy"])

start = datetime.now()

cnn1\_hist = cnn1\_model.fit(x\_train, y\_train, epochs=50, batch\_size=50, validation\_split=0.1)

end = datetime.now()

cnn1\_time = end – start

# Plots

plt.plot(cnn1\_hist.history["loss"], label="loss")

plt.plot(cnn1\_hist.history["val\_loss"], label="val loss")

plt.xlabel("Epochs")

plt.ylabel("Loss")

plt.legend()

plt.show()

# Q3b)

cnn2\_model = Sequential([

Input((x\_train.shape[1], x\_train.shape[2], 1)),

Conv2D(32, (3, 3), strides=(1, 1), padding="valid", activation="relu"),

Conv2D(32, (3, 3), strides=(1, 1), padding="valid", activation="relu"),

MaxPooling2D(pool\_size=(2, 2), strides=2, padding="valid"),

Flatten(),

Dense(100, activation="relu"),

Dense(10, activation="sigmoid"),

])

cnn2\_model.compile(optimizer=Adam(learning\_rate=0.01), loss="sparse\_categorical\_crossentropy", metrics=["accuracy"])

start = datetime.now()

cnn2\_hist = cnn2\_model.fit(x\_train, y\_train, epochs=25, batch\_size=50, validation\_split=0.1)

end = datetime.now()

cnn2\_time = end – start

cnn2\_loss, cnn2\_acc = cnn2\_model.evaluate(x\_test, y\_test, batch\_size=50)

print(f"Loss: {cnn2\_loss}")

print(f"Accuracy: {cnn2\_acc}")

# Plots

plt.plot(cnn2\_hist.history["loss"], label="loss")

plt.plot(cnn2\_hist.history["val\_loss"], label="val loss")

plt.xlabel("Epochs")

plt.ylabel("Loss")

plt.legend()

plt.show()

**Results & Discussion:**

# Confusion matrix for FCNN

A screenshot of a computer screen

Description automatically generated

# Q3d) Plots

# FCNN

A graph with lines and numbers

Description automatically generated

A graph with lines and text

Description automatically generated

# CNN 1

A graph with a line and a black background

Description automatically generated

A graph with lines and numbers

Description automatically generated

# CNN 2

A graph with lines and numbers on it

Description automatically generated

A graph with lines and numbers

Description automatically generated

# Q3e)

The second CNN with 2 convolutional layers seems to perform better than the CNN with 1 layer but tends to overfit much faster. Hence, I had to reduce the epochs to 25 to avoid overfitting.

# Q4)

Both FCNN and CNN achieved high accuracy with the fashion mnist dataset. However, CNN was more efficient and performed better due to its ability to extract relevant features through convolutions.

# Q5)

FCNN has lesser parameters than CNN, 266610 v/s 462742 in the above models because of the architecture that I have chosen as depicted by `model.summary()`

# Q6)

The second CNN with 2 convolutional layers takes more time than the first CNN with 1 convolutional layer due to it having more convolutional layers which implies there is more computation involved.

The FCNN techincally should take more time to train than a CNN due to it having larger number of weights to be trained due to lack of convolutions and that CNN's are designed to extract the most relevant features from the images passed to it resulting in fewer parameters

The FCNN here shows a 7 second advantage which might be due to external factors like differing GPU loads during training, processes taking variable amounts of computer power randomly, etc.

# WEEK 4- IMPLEMENTING CONVENTIONAL CNN ARCHITECTURES AND TRANSFER LEARNING

**Q1)**

A) Implement the LeNet-5, AlexNet architecture.

B) Train, test and compare the performances of these two models on the

Cats\_and\_Dogs\_Dataset, Horse2Zebra Dataset

Note:

• The Cats\_and\_Dogs\_Dataset can downloaded from:

https://storage.googleapis.com/mledu-

datasets/cats\_and\_dogs\_filtered.zip

• Horse2Zebra Dataset can be downloaded from:

https://www.kaggle.com/datasets/balraj98/horse2zebra-dataset

**Q2)**

Train, test and report the performances using the following models on the

Cats\_and\_Dogs\_Dataset and Horse2Zebra dataset.

A) VGG-16

B) GoogleNet

C) ResNet50

D) EfficientNetB0

E) MobileNetV2

**Code:**

# Imports

import tensorflow as tf

keras = tf.keras

from keras import Sequential, Input

from keras.layers import Conv2D, Dense, AveragePooling2D, Flatten, MaxPooling2D

from keras.preprocessing.image import ImageDataGenerator

import matplotlib.pyplot as plt

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

from keras.optimizers import Adam

import os

# Load data

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

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

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

base\_dir = './catsdogs'

train\_dir = os.path.join(base\_dir, 'train')

validation\_dir = os.path.join(base\_dir, 'validation')

test\_dir = os.path.join(base\_dir, 'test')

# CatsDogs

train\_cats\_dir = os.path.join(train\_dir, 'cats')

train\_dogs\_dir = os.path.join(train\_dir, 'dogs')

validation\_cats\_dir = os.path.join(validation\_dir, 'cats')

validation\_dogs\_dir = os.path.join(validation\_dir, 'dogs')

test\_cats\_dir = os.path.join(test\_dir, 'cats')

test\_dogs\_dir = os.path.join(test\_dir, 'dogs')

train\_generator = train\_datagen.flow\_from\_directory(

train\_dir,

target\_size=(28, 28),

batch\_size=20,

class\_mode='categorical',

)

val\_generator = val\_datagen.flow\_from\_directory(

validation\_dir,

target\_size=(28, 28),

batch\_size=20,

class\_mode='categorical',

)

test\_generator = test\_datagen.flow\_from\_directory(

test\_dir,

target\_size=(28, 28),

batch\_size=20,

class\_mode='categorical',

)

# HorseZebra

base\_dir = './horse2zebra'

train\_dir = os.path.join(base\_dir, 'train')

validation\_dir = os.path.join(base\_dir, 'validation')

test\_dir = os.path.join(base\_dir, 'test')

train\_horse\_dir = os.path.join(train\_dir, 'trainA')

train\_zebra\_dir = os.path.join(train\_dir, 'trainB')

validation\_horse\_dir = os.path.join(validation\_dir, 'valA')

validation\_zebra\_dir = os.path.join(validation\_dir, 'valB')

test\_horse\_dir = os.path.join(test\_dir, 'testA')

test\_zebra\_dir = os.path.join(test\_dir, 'testB')

train\_generator = train\_datagen.flow\_from\_directory(

train\_dir,

target\_size=(28, 28),

batch\_size=20,

class\_mode='categorical',

)

test\_generator = test\_datagen.flow\_from\_directory(

test\_dir,

target\_size=(28, 28),

batch\_size=20,

class\_mode='categorical',

)

val\_generator = val\_datagen.flow\_from\_directory(

validation\_dir,

target\_size=(28, 28),

batch\_size=20,

class\_mode='categorical',

)

# Q1A,B) Create models

lenet\_model = Sequential([

Input(shape=(28, 28, 3)),

Conv2D(6, (5, 5), padding="same", activation="tanh"),

AveragePooling2D(strides=2),

Conv2D(16, (5, 5), padding="valid", activation="tanh"),

AveragePooling2D(strides=2),

Flatten(),

Dense(120, activation="sigmoid"),

Dense(84, activation="sigmoid"),

Dense(2, activation="sigmoid"),

], name="LeNet")

lenet\_model.compile(loss='binary\_crossentropy', optimizer=Adam(learning\_rate=0.001), metrics=['acc'])

lenet\_hist = lenet\_model.fit(train\_generator, epochs=20, validation\_data=val\_generator)

lenet\_model.evaluate(test\_generator, batch\_size=32)

alexnet\_model = Sequential([

Input(shape=(227, 227, 3)),

Conv2D(96, (11, 11), strides=(4, 4), padding="valid", activation="tanh"),

MaxPooling2D(pool\_size=(3, 3), strides=2),

Conv2D(256, (5, 5), padding="same", activation="tanh"),

MaxPooling2D(pool\_size=(3, 3), strides=2),

Conv2D(384, (3, 3), padding="same", activation="tanh"),

Conv2D(384, (3, 3), padding="same", activation="tanh"),

Conv2D(256, (3, 3), padding="same", activation="tanh"),

MaxPooling2D(pool\_size=(3, 3), strides=2),

Flatten(),

Dense(4096, activation="sigmoid"),

Dense(4096, activation="sigmoid"),

Dense(2, activation="sigmoid"),

], name="AlexNet")

alexnet\_model.compile(loss='binary\_crossentropy', optimizer=Adam(learning\_rate=0.001), metrics=['acc'])

alexnet\_hist = alexnet\_model.fit(train\_generator, epochs=20, validation\_data=val\_generator)

alexnet\_model.evaluate(test\_generator)

# Q2A) VGG-16

vgg\_model = VGG16(input\_shape=(224, 224, 3), weights='imagenet', include\_top=False)

vgg\_model.trainable = False

custom\_vgg\_model = Sequential([

vgg\_model,

Flatten(),

Dense(2, activation='softmax')

])

custom\_vgg\_model.compile(loss='categorical\_crossentropy', optimizer=Adam(learning\_rate = 0.0001), metrics=['acc'])

vgg\_history = custom\_vgg\_model.fit(train\_generator, epochs=20, validation\_data=val\_generator)

custom\_vgg\_model.evaluate(test\_generator)

# Q2B) GoogleNet

google\_model = InceptionV3(input\_shape=(224, 224, 3), weights='imagenet', include\_top=False)

google\_model.trainable = False

custom\_google\_model = Sequential([

google\_model,

Flatten(),

Dense(2, activation='softmax')

])

custom\_google\_model.compile(loss='categorical\_crossentropy', optimizer=Adam(learning\_rate = 0.0001), metrics=['acc'])

google\_history = custom\_google\_model.fit(train\_generator, epochs=20, validation\_data=val\_generator)

custom\_google\_model.evaluate(test\_generator)

# Q2C) ResNet50

res\_model = ResNet50(input\_shape=(224, 224, 3), weights='imagenet', include\_top=False)

res\_model.trainable = False

custom\_res\_model = Sequential([

res\_model,

Flatten(),

Dense(2, activation='softmax')

])

custom\_res\_model.compile(loss='categorical\_crossentropy', optimizer=Adam(learning\_rate = 0.0001), metrics=['acc'])

res\_history = custom\_res\_model.fit(train\_generator, epochs=20, validation\_data=val\_generator)

custom\_res\_model.evaluate(test\_generator)

# Q2D) EfficientNetB0

eff\_model = EfficientNetB0(input\_shape=(224, 224, 3), weights='imagenet', include\_top=False)

eff\_model.trainable = False

custom\_eff\_model = Sequential([

eff\_model,

Flatten(),

Dense(2, activation='softmax')

])

custom\_eff\_model.compile(loss='categorical\_crossentropy', optimizer=Adam(learning\_rate = 0.0001), metrics=['acc'])

eff\_history = custom\_eff\_model.fit(train\_generator, epochs=20, validation\_data=val\_generator)

custom\_eff\_model.evaluate(test\_generator)

# Q2E) MobileNetV2

mobile\_model = MobileNetV2(input\_shape=(224, 224, 3), weights='imagenet', include\_top=False)

mobile\_model.trainable = False

custom\_mobile\_model = Sequential([

mobile\_model,

Flatten(),

Dense(2, activation='softmax')

])

custom\_mobile\_model.compile(loss='categorical\_crossentropy', optimizer=Adam(learning\_rate = 0.0001), metrics=['acc'])

mobile\_history = custom\_mobile\_model.fit(train\_generator, epochs=20, validation\_data=val\_generator)

custom\_mobile\_model.evaluate(test\_generator)

**Results & Discussion:**

# Catsdogs

Lenet:

Loss: 0.6820

Accuracy: 0.6007

AlexNet:

Loss: 0.7056

Accuracy: 0.5

LeNet performs better than AlexNet in this instance

VGG16:

Loss: 0.2222

Accuracy: 0.9092

GoogleNet:

Loss: 0.0510

Accuracy: 0.9876

ResNet50

Loss: 0.5743

Accuracy: 0.6927

EfficientNetB0

Loss: 0.6945

Accuracy: 0.5000

MobileNetV2

Loss: 0.0871

Accuracy: 0.9789

GoogleNet (InceptionV3 which is not actually GoogleNet – InceptionV1) performs the best overall

# HorseZebra

Lenet:

Loss: 0.4199

Accuracy: 0.8038

AlexNet:

Loss: 0.6913

Accuracy: 0.5385

LeNet yet again performs better here.

VGG16:

Loss: 0.0904

Accuracy: 0.9692

GoogleNet:

Loss: 0.0956

Accuracy: 0.9846

ResNet50

Loss: 0.2821

Accuracy: 0.8885

EfficientNetB0

Loss: 1.1298

Accuracy: 0.5077

MobileNetV2

Loss: 0.0983

Accuracy: 0.9846

GoogleNet yet again performs the best overall.

# WEEK 5- IMPLEMENTING RECURRENT NEURAL NETWORKS FOR TIME SERIES

# FORECASTING AND STOCK MARK PREDICTION

**Q1)** Use the following code to generate a time series:

def generate\_time\_series(sample\_size, n\_steps):

freq1, freq2, offsets1, offsets2 = np.random.rand(4, sample\_size, 1)

time = np.linspace(0, 1, n\_steps)

series = 0.5 \* np.sin((time - offsets1) \* (freq1 \* 10 + 10)) #wave1+

series += 0.2 \* np.sin((time - offsets2) \* (freq2 \* 20 + 20)) #wave2+

series += 0.1 \* (np.random.rand(sample\_size, n\_steps) - 0.5) #noise

return series[..., np.newaxis].astype(np.float32)

The above code generates as many time series as requested, which can be specified using the

“sample\_size” argument. Each time series is of length “n\_steps” and there is just one value per time

step in each series.

Use the above code to do the following:

A) Create a dataset of 10,000 samples with 51 time steps each (Note: the 51st time step should be

used as the label)

B) Split the dataset in the ratio training: validation: testing = 70:20:10.

C) Design, train, test and compare the performances of the following on the prediction of the

value of 51st time step in the generated time series.

a. Fully connected neural network.

b. Simple RNN with one layer (output layer)

c. Simple RNN with one hidden layer and one output layer.

d. Simple RNN with one hidden layer and one output layer.

e. Simple RNN with two hidden layers and one output layer.

**Q2)** Consider the Google Stock Prediction dataset.

The 14 columns are:

symbol : - Name of the company (in this case Google).

date :- year and date

close:- closing of stock value

high:- highest value of stock at that day

low:- lowest value of stock at that day

open:- opening value of stock at that day

volume

adjClose

adjHigh

adjLow

adjOpen

adjVolume

divCash

splitFactor

A. Build a Simple RNN model with 5 layers (use dropouts if required) to predict the stock price for

the years 2020 and 2021.

B. Compare the accuracy using MAPE and MSE.

C. Comment on how many epochs (dropouts) is required for adequate learning.

D. Plot the actual vs predicted values using the test data for the year 2020 and 2021 .

**Code:**

# Imports

import tensorflow as tf

keras = tf.keras

from keras.layers import SimpleRNN, Dense, Flatten, Dropout

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

from sklearn.preprocessing import MinMaxScaler

from sklearn.metrics import mean\_absolute\_percentage\_error, mean\_squared\_error

from keras import Sequential, Input

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

# Q1A) Generating time series data

def gen\_time\_series(sample\_size: int, n\_steps: int):

freq1, freq2, offsets1, offsets2 = np.random.rand(4, sample\_size, 1)

time = np.linspace(0, 1, n\_steps)

series = 0.5 \* np.sin((time - offsets1) \* (freq1 \* 10 + 10))

series += 0.2 \* np.sin((time - offsets2) \* (freq2 \* 20 + 20))

series += 0.1 \* (np.random.rand(sample\_size, n\_steps) - 0.5)

return series[..., np.newaxis].astype(np.float32)

data = gen\_time\_series(10000, 51)

X = data[:, 0:50]

y = data[:, -1]

# Q1B) Split dataset into train, val and test

X\_train, X\_val, X\_test = X[0:7000, :], X[7000:9000, :], X[9000:10000, :]

y\_train, y\_val, y\_test = y[0:7000, :], y[7000:9000, :], y[9000:10000, :]

# Q1Ca) FCNN model

fcnn\_model = Sequential([

Flatten(input\_shape=[50, 1]),

Dense(1)

])

fcnn\_model.compile(loss="mse", optimizer="adam")

fcnn\_hist = fcnn\_model.fit(X\_train, y\_train, epochs=20, validation\_data=(X\_val, y\_val))

fcnn\_model.evaluate(X\_test, y\_test)

# Q1Cb) RNN 1 model

rnn1\_model = Sequential([

Input((50, 1)),

SimpleRNN(1)

])

rnn1\_model.compile(loss="mse", optimizer="adam", metrics=['accuracy'])

rnn1\_hist = rnn1\_model.fit(X\_train, y\_train, epochs=20, validation\_data=(X\_val, y\_val))

rnn1\_model.evaluate(X\_test, y\_test)

# Q1Cc,d) RNN 2

rnn2\_model = Sequential([

Input((50, 1)),

SimpleRNN(1, return\_sequences=True),

SimpleRNN(1)

])

rnn2\_model.compile(loss="mse", optimizer="adam")

rnn2\_hist = rnn2\_model.fit(X\_train, y\_train, epochs=20, validation\_data=(X\_val, y\_val))

rnn2\_model.evaluate(X\_test, y\_test)

# Q1Ce) RNN 3

rnn3\_model = Sequential([

Input((50, 1)),

SimpleRNN(1, return\_sequences=True),

SimpleRNN(1, return\_sequences=True),

SimpleRNN(1)

])

rnn3\_model.compile(loss="mse", optimizer="adam")

rnn3\_hist = rnn3\_model.fit(X\_train, y\_train, epochs=20, validation\_data=(X\_val, y\_val))

rnn3\_model.evaluate(X\_test, y\_test)

# Q2)

# Read data

df = pd.read\_csv('GOOG.csv')

# Preprocess

df['date'] = pd.to\_datetime(df['date'])

df\_2020\_2021 = df[(df['date'].dt.year >= 2020) & (df['date'].dt.year <= 2021)]

features = ['close', 'high', 'low', 'open', 'volume', 'adjClose', 'adjHigh', 'adjLow', 'adjOpen', 'adjVolume']

df\_selected = df\_2020\_2021[features]

scaler = MinMaxScaler()

df\_norm = scaler.fit\_transform(df\_selected)

window\_size = 30

sequences = []

target = []

for i in range(len(df\_norm) - window\_size):

sequences.append(df\_norm[i:i+window\_size])

target.append(df\_norm[i+window\_size])

sequences = np.array(sequences)

target = np.array(target)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(sequences, target, test\_size=0.2, random\_state=42)

# Q2A) RNN model

model = Sequential([

SimpleRNN(units=64, activation='relu', return\_sequences=True, input\_shape=(window\_size, len(features))),

Dropout(0.2),

SimpleRNN(units=64, activation='relu', return\_sequences=True),

Dropout(0.2),

SimpleRNN(units=64, activation='relu', return\_sequences=True),

Dropout(0.2),

SimpleRNN(units=64, activation='relu'),

Dropout(0.2),

Dense(units=len(features))

])

model.compile(optimizer='adam', loss='mean\_squared\_error')

hist = model.fit(X\_train, y\_train, epochs=50, batch\_size=32,validation\_split=0.1)

# Plots

training = hist.history['loss']

validation = hist.history['val\_loss']

plt.plot(training, label='Training')

plt.plot(validation, label='Validation')

plt.legend()

plt.show()

# Q2B) MSE and MAPE

predicted\_prices = model.predict(X\_test)

predicted\_prices\_actual = scaler.inverse\_transform(predicted\_prices)

mape = mean\_absolute\_percentage\_error(y\_test, predicted\_prices)

print(f"MAPE: {mape:.2f}%")

mse = mean\_squared\_error(y\_test, predicted\_prices)

print(f"MSE: {mse:.2f}")

# Q2D) Actual vs Predicted prices

actual\_price = scaler.inverse\_transform(y\_test)

plt.plot(actual\_price[:, 0], label='Actual')

plt.plot(predicted\_prices\_actual[:, 0], label='Predicted')

plt.xlabel('Day')

plt.ylabel('Normalized Close Price')

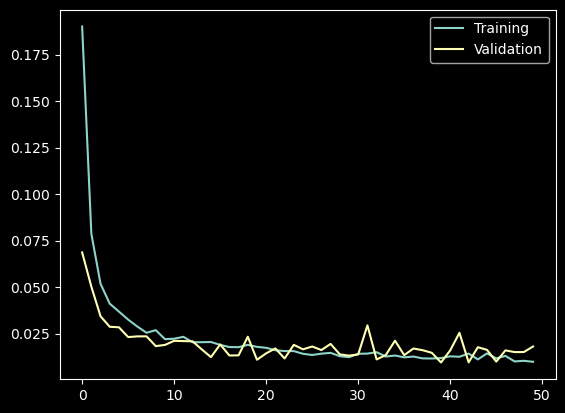
plt.title('Actual vs Predicted')

plt.legend()

plt.show()

**Results & Discussion:**

# Loss plot



# Q2C) Results

Good results were seen with 50 epochs and 4 dropouts

# Q2D) Actual vs Predicted price plot

