

**LAB RECORD**

**BACHELOR OF TECHNOLOGY**

**B.TECH CS&E 2021-2025 SEMESTER (6)**

**(Academic Session-: 2021-2025)**

**Course title : Deep Learning and Neural Network**

**Course code : AIML 302**

**Enrollment number : A7605221152**

**Name of student : Suyash Pandey**

**Faculty name : Dr. Pooja Khanna**

**Date of submission : \_ \_/\_ \_/\_ \_ \_ \_**

**Signature of student :**

**Grade/marks :**

**Faculty sign :**

**Department Of Computer Science & Engineering**

**Amity School Of Engineering &Technology**

**Amity University, Lucknow Campus**

|  |  |  |
| --- | --- | --- |
| **S.No** | **Contents** | **Page\_ No.** |
|  | **Practical 1: Intro To Kaggle** |  |
|  | **Practical 2: Basic Perceptron From Scratch** |  |
|  | **Practical 3: Customer Churn prediction on different parameters.** |  |

**Index**

**Practical-1**

**Objective:** Introduction to Kaggle and how it improves visibility.

**Description:** Kaggle is an online platform that hosts data science competitions and provides a collaborative environment for data scientists, machine learning engineers, and researchers to work on and share projects. It was founded in 2010 by Anthony Goldbloom and Ben Hamner and has since become one of the largest and most popular platforms for data science and machine learning enthusiasts.

**Key features of Kaggle include:**

1. **Competitions:** Kaggle is renowned for hosting a wide variety of data science competitions. Organizations and companies post real-world problems and datasets, challenging the Kaggle community to develop the best predictive models or solutions. Competitions cover diverse domains such as healthcare, finance, computer vision, natural language processing, and more.
2. **Datasets:**Kaggle provides a repository of datasets that users can access for free. These datasets cover a broad range of topics, allowing users to practice and experiment with different data sources.
3. **Kernels:** Kaggle Kernels are Jupyter notebooks hosted on the platform that allows users to write and execute code in a collaborative environment. Users can share their code, analyses, and visualizations with the community. This feature fosters collaboration and knowledge-sharing among data scientists.
4. **Discussion Forums:** Kaggle has discussion forums associated with competitions and datasets where users can ask questions, seek help, and discuss various aspects of data science. This community-driven approach facilitates learning and collaboration.
5. **Notebooks:** Users can create and share interactive data science notebooks using Kaggle's Notebook feature. This allows for the combination of code, visualizations, and explanations in a single document, making it easy for others to understand and replicate analyses.
6. **Courses and Learning Resources:** Kaggle provides learning resources, including courses and tutorials, to help users enhance their data science and machine learning skills. These resources cover a wide range of topics, from beginner to advanced levels.
7. **Job Board:**Kaggle has a job board where companies can post data science job openings. This feature connects data science professionals with potential employers and provides a platform for career opportunities.
8. **Datasets and Competitions for Social Impact:** Kaggle occasionally hosts competitions and provides datasets with a social impact focus, addressing issues such as public health, environmental conservation, and humanitarian aid. This enables the community to contribute to meaningful and impactful projects.
9. **APIs and Integrations:** Kaggle provides APIs that allow users to access datasets, competitions, and other platform features programmatically. This facilitates automation and integration with external tools.
10. **Rewards and Recognition:** Kaggle awards prizes to winners of competitions, fostering a competitive spirit and providing an incentive for participants to showcase their skills. Additionally, Kaggle users earn Kaggle Kernels and Dataset medals, contributing to their profile and establishing credibility within the community.

**Practical-2**

**Objective:**Program in python to implement basic perceptron model from scratch.

**Description:**The program given below is the implementation of simple artificial neural network or a perceptron in python from scratch which does not contain any inbuilt python library. First i have created a normal python function with the def keyword inside the function, i have taken all the necessary inputs for the perceptron and using these inputs i have calculated the output.

**Code:**

def neural\_network():

  for epochs in range(3-3):

        x1 = int(input("enter the 1st input: "))

        w1 = int(input("enter the 1st weight: "))

        x2 = int(input("enter the 2nd input: "))

        w2 = int(input("enter the 2nd weight: "))

        x3 = int(input("enter the 3rd input: "))

        w3 = int(input("enter the 3rd weight: "))

        x4 = int(input("enter the 4th input: "))

        w4 = int(input("enter the 4th weight: "))

        alpha = int(input("enter the value alpha: "))

        target = int(input("enter the target value: "))

        bias = int(input("enter the value of bias: "))

        y\_input = int(0)

        del\_bias = alpha\*target

        del\_w1 = alpha\*target\*x1

        del\_w2 = alpha\*target\*x2

        del\_w3 = alpha\*target\*x3

        del\_w4 = alpha\*target\*x4

        w1 = del\_w1+w1

        w2 = del\_w2+w2

        w3 = del\_w3+w3

        w4 = del\_w4+w4

        y\_input = bias+(w1\*x1)+(w2\*x2)+(w3\*x3)+(w4\*x4)

        print("y\_output: ")

        if y\_input > 0:

            print(1),

        elif y\_input == 0:

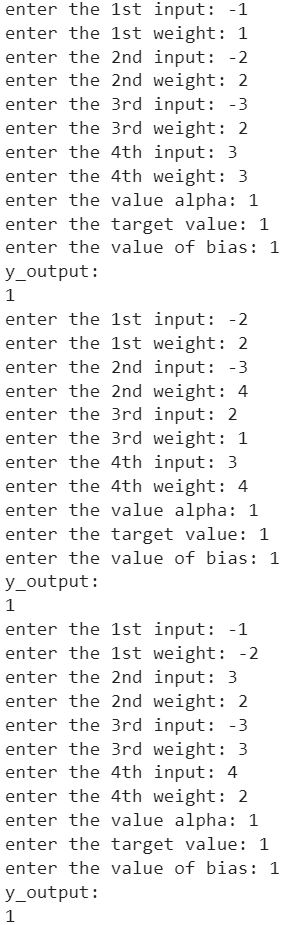
            print(0)

        else:

            print(-1),

neural\_network()

**Output:**



**Practical-3**

**Objective:** Create a machine learning model for customer churn prediction.

**Description:** The model below is a machine learning model for predicting the customer churning from a company.The customer who are exited from the company are indicated by one and those who are there in the company are indicated by zero. Customer churn prediction is a common and crucial task in the field of data science and business analytics. Churn refers to the phenomenon where customers stop using a product or service, typically shifting to a competitor or discontinuing the usage altogether. Predicting customer churn is essential for businesses as it allows them to identify at-risk customers early on and implement strategies to retain them. Here's an overview of the process involved in customer churn prediction:

1. **Data Collection:**

Gather relevant data: Collect data related to customer interactions, behaviors, and demographics. This may include transaction history, customer support interactions, usage patterns, customer feedback, and any other information that could be indicative of churn.

1. **Data Preprocessing:**

Clean and preprocess data: Handle missing values, outliers, and inconsistencies in the dataset. Transform and normalize data to ensure it's suitable for analysis. Feature engineering may involve creating new features or transforming existing ones to extract more meaningful information.

1. **Exploratory Data Analysis (EDA):**

Explore the data: Conduct exploratory data analysis to understand patterns, correlations, and trends within the dataset. Visualization tools can be helpful in gaining insights into customer behavior and identifying potential factors contributing to churn.

1. **Feature Selection:**

Identify important features: Determine which features have the most significant impact on customer churn. Feature selection methods such as correlation analysis, feature importance from machine learning models, or domain expertise can guide this process.

1. **Model Selection:**

Choose appropriate models: Select machine learning models suitable for churn prediction. Commonly used models include logistic regression, decision trees, random forests, support vector machines, and neural networks. The choice of model depends on the nature of the data and the problem at hand.

1. **Data Splitting:**

Split the dataset: Divide the dataset into training and testing sets to evaluate the model's performance. Cross-validation techniques may also be employed to ensure robustness.

1. **Model Training:**

Train the model: Feed the training data into the chosen model and allow it to learn patterns associated with customer churn. Parameter tuning and optimization may be performed to improve the model's accuracy.

1. **Model Evaluation:**

Evaluate model performance: Use the testing dataset to assess how well the model generalizes to new, unseen data. Common evaluation metrics include accuracy, precision, recall, F1-score.

Implementing a customer churn prediction system empowers businesses to take proactive measures to retain customers, whether through targeted marketing campaigns, personalized incentives, or improved customer service. Machine learning models enhance the accuracy of predictions, allowing businesses to allocate resources more efficiently and effectively reduce churn.

**Code:**

#import all the important libraries required for developing the model.

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import tensorflow as tf

from tensorflow import keras

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

from sklearn.preprocessing import MinMaxScaler

from sklearn.linear\_model import LogisticRegression

from sklearn.ensemble import RandomForestClassifier

import xgboost as xgb

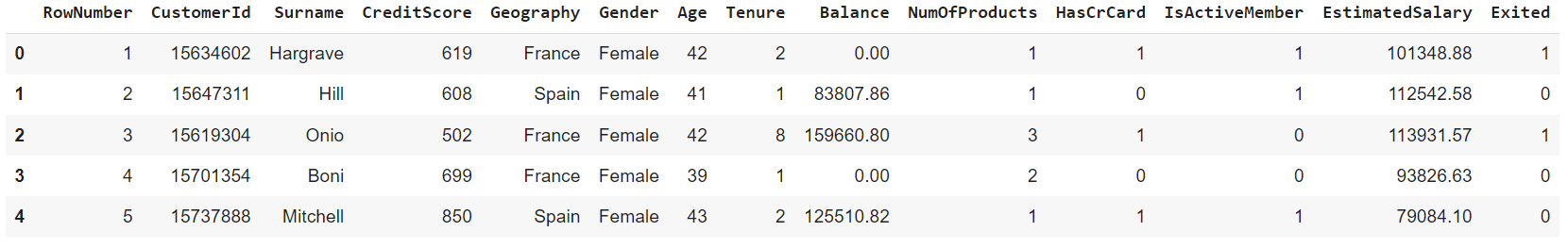
from sklearn.metrics import confusion\_matrix,classification\_report

#reading the dataset with the help of pandas library.

churn = pd.read\_csv("/content/drive/MyDrive/Colab Notebooks/Churn\_Modelling.csv")

##displaying first five rows of the dataframe using the head command

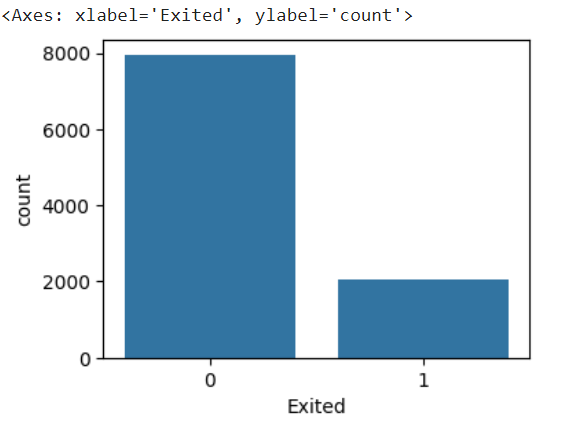
churn.head(5)



#countplot of the exited customers represented by 1 and represented by 0 those who are not exited.

plt.figure(figsize = (4,3))

sns.countplot(x = 'Exited', data = churn)

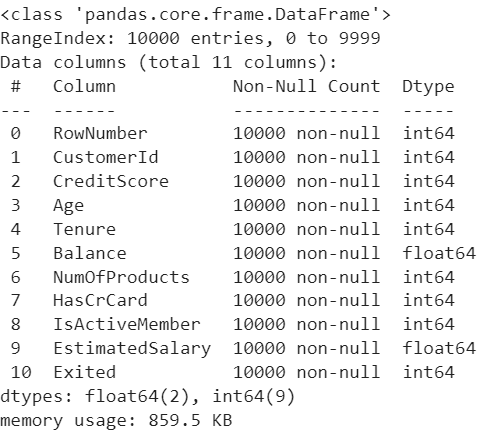


#dropping the unwanted labels

churn = churn.drop(labels = {'Gender', 'Geography', 'Surname'}, axis = 1)

#info of the dataset

churn.info()



#features to be selected as predictors

selected\_features = ['RowNumber', 'CustomerId', 'CreditScore', 'Age', 'Tenure', 'Balance',

       'NumOfProducts', 'HasCrCard', 'IsActiveMember', 'EstimatedSalary']

#taking X as predictor and y as target variable.

X = churn[selected\_features]

y = churn['Exited']

#normalizing the dataset using MinMaxScaler()

scaler = MinMaxScaler()

X\_scaled = scaler.fit\_transform(X)

#Splitting the dataset into training and testing sets

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

#imported Ann through keras API and tensorflow

#ANN is just to check how it performs on the dataset.

tf.random.set\_seed(42)

classifier\_model = tf.keras.models.Sequential()

classifier\_model.add(tf.keras.layers.Dense(units = 100, activation = 'relu', input\_shape = (10,)))

classifier\_model.add(tf.keras.layers.Dropout(0.3))

classifier\_model.add(tf.keras.layers.Dense(units = 50, activation = 'relu'))

classifier\_model.add(tf.keras.layers.Dropout(0.3))

classifier\_model.add(tf.keras.layers.Dense(units = 50, activation = 'relu'))

classifier\_model.add(tf.keras.layers.Dense(units = 1, activation = 'sigmoid'))

#compiled Ann model

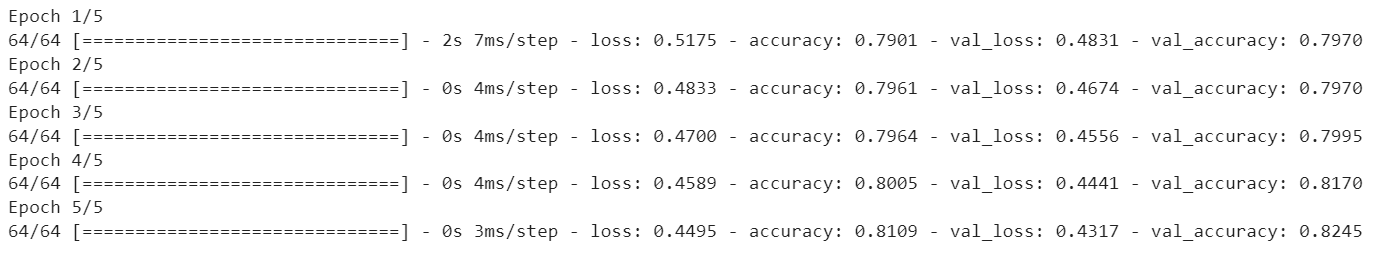
classifier\_model.compile(optimizer = 'Adam', loss = 'binary\_crossentropy', metrics = 'accuracy')

#calculating the epochs on training and validation dataset

from tensorflow.keras.callbacks import EarlyStopping

Early\_Stop = EarlyStopping()

epochs\_hist = classifier\_model.fit(X\_train,y\_train,epochs = 50, validation\_data = (X\_test, y\_test), batch\_size = 125, callbacks=[Early\_Stop])



#plotting the graph between the training loss and accuracy vs number epochs.

plt.figure(figsize = (4,4))

eh = epochs\_hist.history['loss']

eh2 = epochs\_hist.history['accuracy']

eh3 = epochs\_hist.history['val\_loss']

eh4 = epochs\_hist.history['val\_accuracy']

plt.plot(eh)

plt.plot(eh2)

plt.plot(eh3)

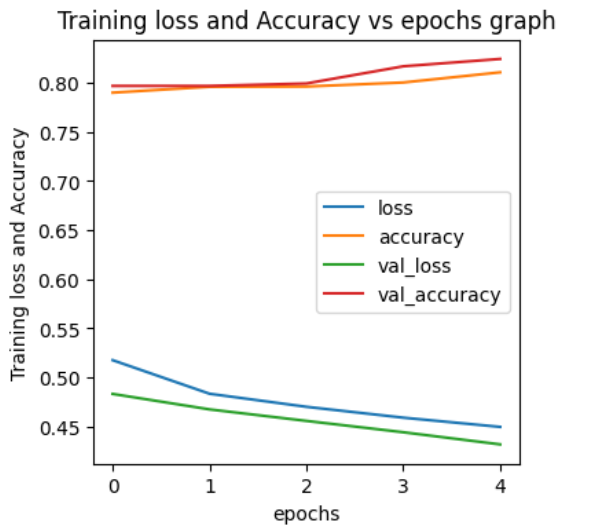
plt.plot(eh4)

plt.title('Training loss and Accuracy vs epochs graph')

plt.xlabel('epochs')

plt.ylabel('Training loss and Accuracy')

plt.legend(['loss', 'accuracy', 'val\_loss', 'val\_accuracy'])



#evaluating the model on the testing dataset.

evaluation = classifier\_model.evaluate(X\_test,y\_test)

print('test accuracy:{}'.format(evaluation[1]))



#prediction of the model

y\_predict = classifier\_model.predict(X\_test)

#filtering those values greater than 0.5 in testing dataset.

y\_predict = (y\_predict > 0.5)

#predict the values on the training test

y\_train\_predict = classifier\_model.predict(X\_train)

#filtering those values greater than 0.5 in training dataset.

y\_train\_predict =  (y\_train\_predict > 0.5)

#plotting the confusion matrix of training dataset

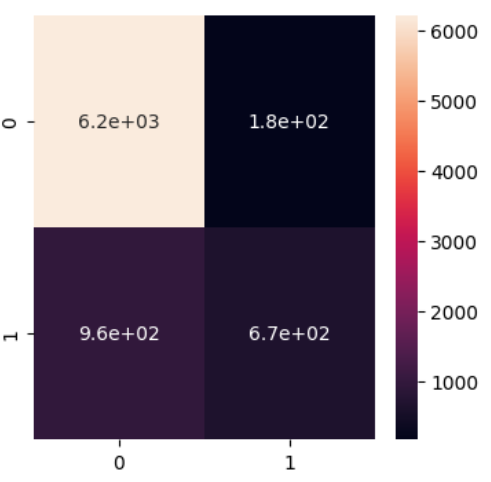
plt.figure(figsize = (4,4))

from sklearn.metrics import confusion\_matrix,classification\_report

cm = confusion\_matrix(y\_train, y\_train\_predict)

sns.heatmap(cm, annot = True)

#Training matrix:



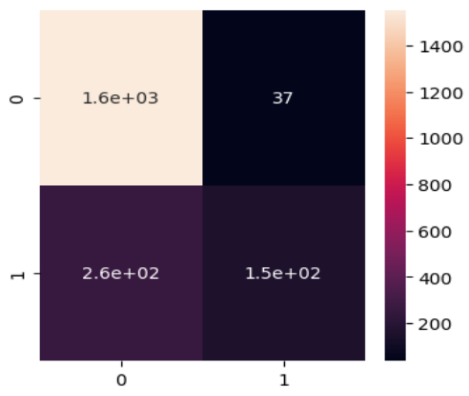
#plotting the confusion matrix of testing dataset

plt.figure(figsize = (4,4))

cm2 = confusion\_matrix(y\_test, y\_predict)

sns.heatmap(cm2, annot = True)

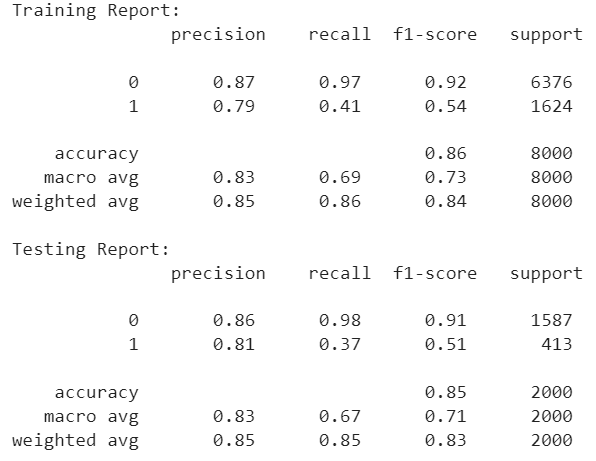
#Testing Matrix



#printing the classification report of training and testing dataset

print("Training Report:\n",classification\_report(y\_train,y\_train\_predict))

print("Testing Report:\n",classification\_report(y\_test, y\_predict))



**Now we will update the model based on various parameters**

**NOTE: Code for the accuracy graph will be same in the complete project.**

1. **Data Preprocessing**

If we are including these three'Gender', 'Geography', 'Surname' attributes then we are getting the less accuracy than the previous model in which we dropped the these attributes. These columns were object hence first converted into numbers using get\_dummies() and then concatenated into the same the original dataset by dropping the same attributes with their older values.

new\_gen = pd.get\_dummies(new['Gender'])

new\_geo = pd.get\_dummies(new['Geography'])

new\_Surname = pd.get\_dummies(new['Surname'])

df1 = pd.DataFrame(new\_Surname)

df2 = pd.DataFrame(new\_geo)

df3 = pd.DataFrame(new\_gen)

new\_data = pd.concat([new,df1, df2, df3], axis = 1)

new\_data = new\_data.drop(labels = {'Gender', 'Geography', 'Surname'}, axis = 1)

X1 = new\_data.drop("Exited", axis = 1)

y1 = new\_data['Exited']

#normalizing the dataset using MinMaxScaler()

scaler = MinMaxScaler()

X\_scaled1 = scaler.fit\_transform(X1)

X\_train,X\_test,y\_train,y\_test = train\_test\_split(X\_scaled1, y, test\_size = 0.2)

tf.random.set\_seed(42)

classifier\_model = tf.keras.models.Sequential()

classifier\_model.add(tf.keras.layers.Dense(units = 100, activation = 'relu', input\_shape = (10,)))

classifier\_model.add(tf.keras.layers.Dropout(0.3))

classifier\_model.add(tf.keras.layers.Dense(units = 50, activation = 'relu'))

classifier\_model.add(tf.keras.layers.Dropout(0.3))

classifier\_model.add(tf.keras.layers.Dense(units = 50, activation = 'relu'))

classifier\_model.add(tf.keras.layers.Dense(units = 1, activation = 'sigmoid'))

#compiled Ann model

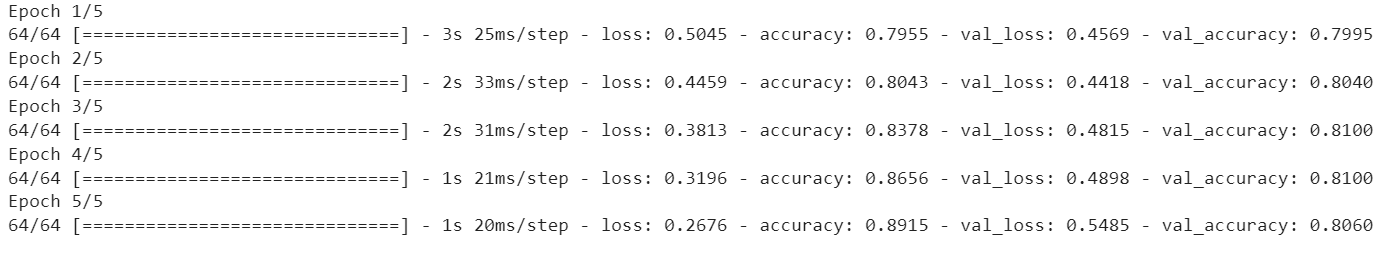
classifier\_model.compile(optimizer = 'Adam', loss = 'binary\_crossentropy', metrics = 'accuracy')

#calculating the epochs on training and validation dataset

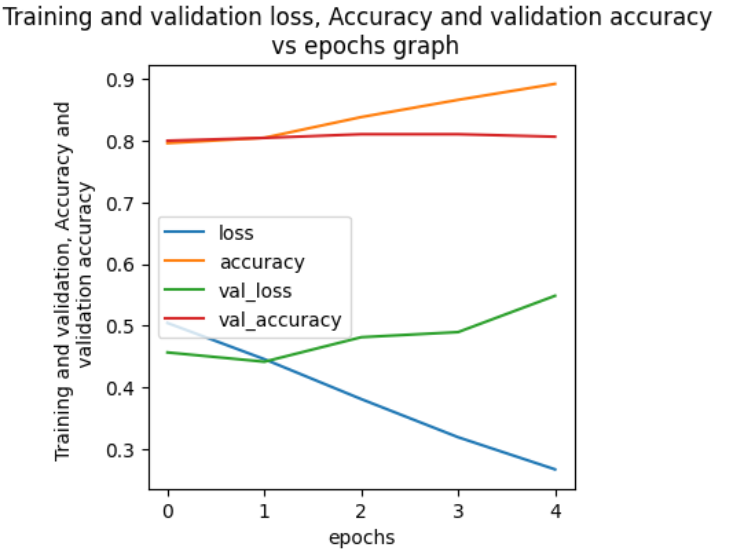
from tensorflow.keras.callbacks import EarlyStopping

Early\_Stop = EarlyStopping()

epochs\_hist = classifier\_model.fit(X\_train,y\_train,epochs = 50, validation\_data = (X\_test, y\_test), batch\_size = 125, callbacks=[Early\_Stop])



**Output:**



1. **Update the Optimizer**

When we updated the optimizer for different optimizer we got different accuracies.

1. **Optimizer=RMSprop**

classifier\_model.compile(optimizer = 'RMSprop', loss = 'binary\_crossentropy', metrics = 'accuracy')

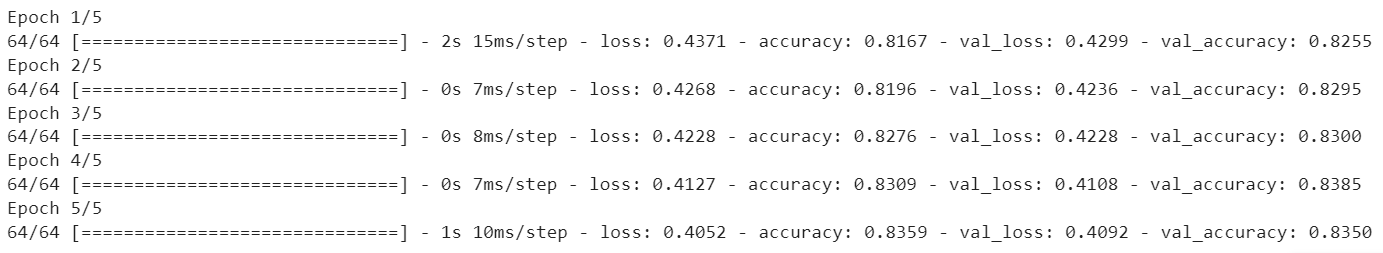
#calculating the epochs on training and validation dataset

from tensorflow.keras.callbacks import EarlyStopping

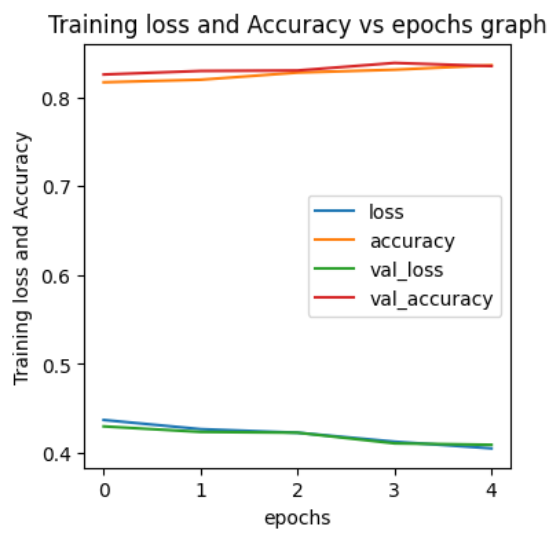
Early\_Stop = EarlyStopping()

epochs\_hist = classifier\_model.fit(X\_train,y\_train,epochs = 50, validation\_data = (X\_test, y\_test), batch\_size = 125, callbacks=[Early\_Stop])

**Epochs:**



**Accuracy graph:**



1. **Optimizer=Adadelta**

#Optimizer=Adadelta

classifier\_model.compile(optimizer = 'Adadelta', loss = 'binary\_crossentropy', metrics = 'accuracy')

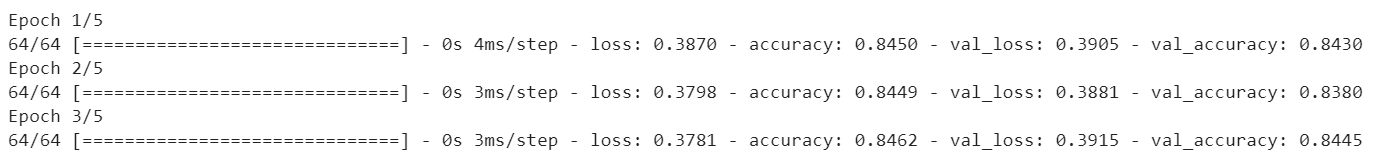
#calculating the epochs on training and validation dataset

from tensorflow.keras.callbacks import EarlyStopping

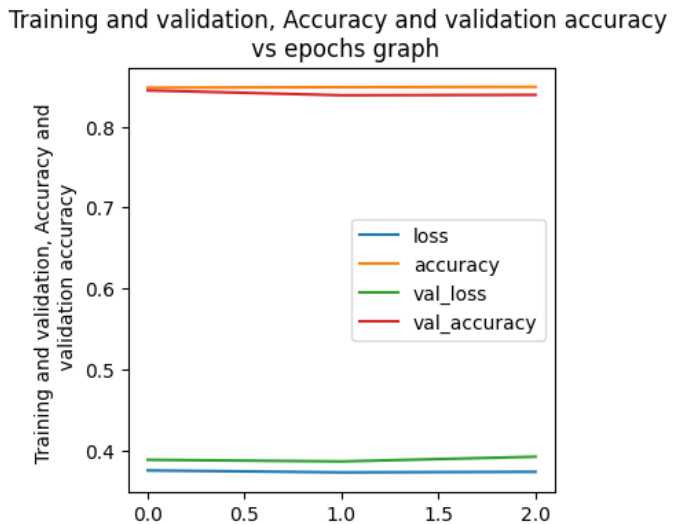
Early\_Stop = EarlyStopping()

epochs\_hist = classifier\_model.fit(X\_train,y\_train,epochs = 5, validation\_data = (X\_test, y\_test), batch\_size = 125, callbacks=[Early\_Stop])

**Epochs:**



**Accuracy graph:**



1. **Optimizer=Adagrad**

#Optimizer=Adadelta

classifier\_model.compile(optimizer = 'Adagrad', loss = 'binary\_crossentropy', metrics = 'accuracy')

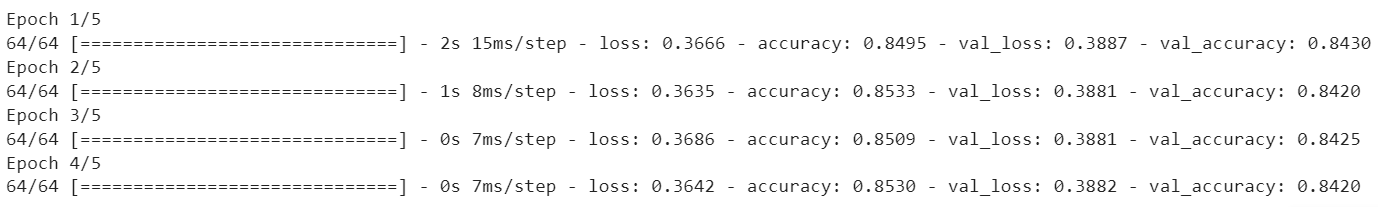
#calculating the epochs on training and validation dataset

from tensorflow.keras.callbacks import EarlyStopping

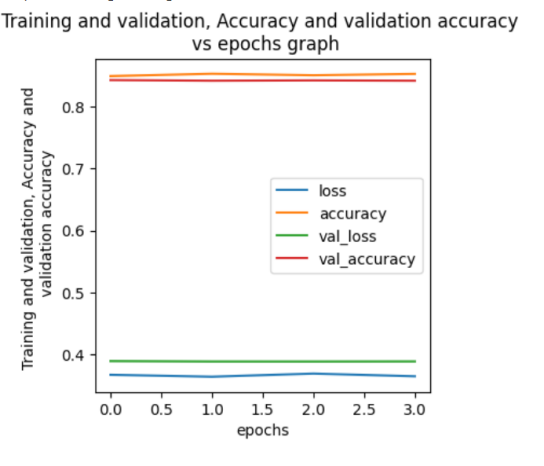
Early\_Stop = EarlyStopping()

epochs\_hist = classifier\_model.fit(X\_train,y\_train,epochs = 5, validation\_data = (X\_test, y\_test), batch\_size = 125, callbacks=[Early\_Stop])

**Epochs:**



**Accuracy graph**:



1. **Changing the activation Functions**
2. **Linear Activation**

tf.random.set\_seed(42)

classifier\_model = tf.keras.models.Sequential()

classifier\_model.add(tf.keras.layers.Dense(units = 100, activation = 'linear', input\_shape = (10,)))

classifier\_model.add(tf.keras.layers.Dropout(0.3))

classifier\_model.add(tf.keras.layers.Dense(units = 50, activation = 'linear'))

classifier\_model.add(tf.keras.layers.Dropout(0.3))

classifier\_model.add(tf.keras.layers.Dense(units = 50, activation = 'linear'))

classifier\_model.add(tf.keras.layers.Dense(units = 1, activation = 'sigmoid'))

classifier\_model.compile(optimizer = 'Adam', loss = 'binary\_crossentropy', metrics = 'accuracy')

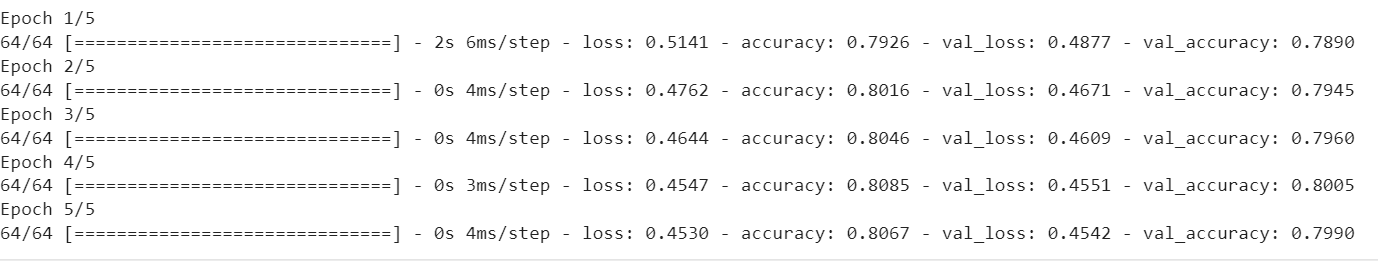
#calculating the epochs on training and validation dataset

from tensorflow.keras.callbacks import EarlyStopping

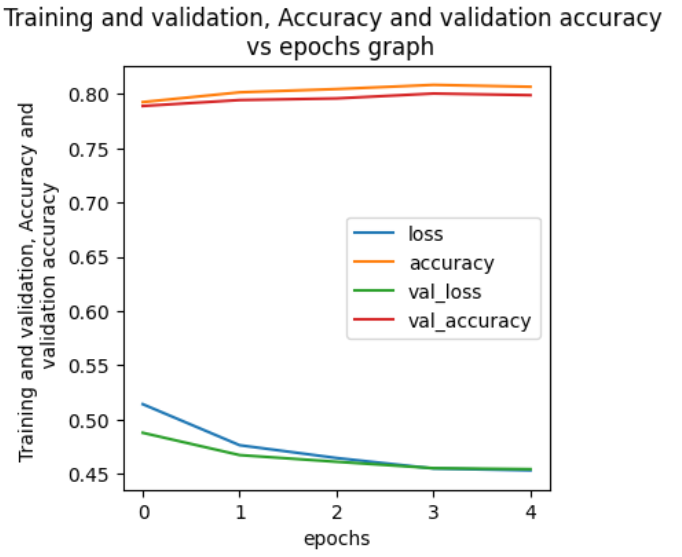
Early\_Stop = EarlyStopping()

epochs\_hist = classifier\_model.fit(X\_train,y\_train,epochs = 5, validation\_data = (X\_test, y\_test), batch\_size = 125, callbacks=[Early\_Stop])

**Epochs:**



**Accuracy graph for the Linear Activation function**:



1. **Tanh activation**

tf.random.set\_seed(42)

classifier\_model = tf.keras.models.Sequential()

classifier\_model.add(tf.keras.layers.Dense(units = 100, activation = 'tanh', input\_shape = (10,)))

classifier\_model.add(tf.keras.layers.Dropout(0.3))

classifier\_model.add(tf.keras.layers.Dense(units = 50, activation = 'tanh'))

classifier\_model.add(tf.keras.layers.Dropout(0.3))

classifier\_model.add(tf.keras.layers.Dense(units = 50, activation = 'tanh'))

classifier\_model.add(tf.keras.layers.Dense(units = 1, activation = 'sigmoid'))

classifier\_model.compile(optimizer = 'Adam', loss = 'binary\_crossentropy', metrics = 'accuracy')

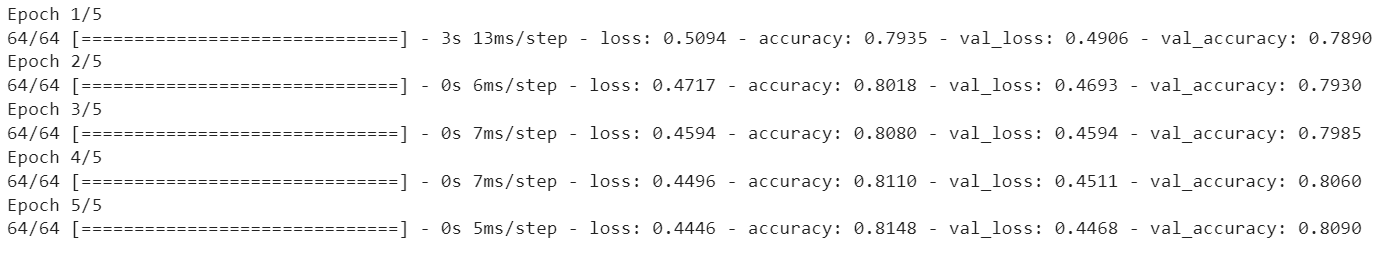
#calculating the epochs on training and validation dataset

from tensorflow.keras.callbacks import EarlyStopping

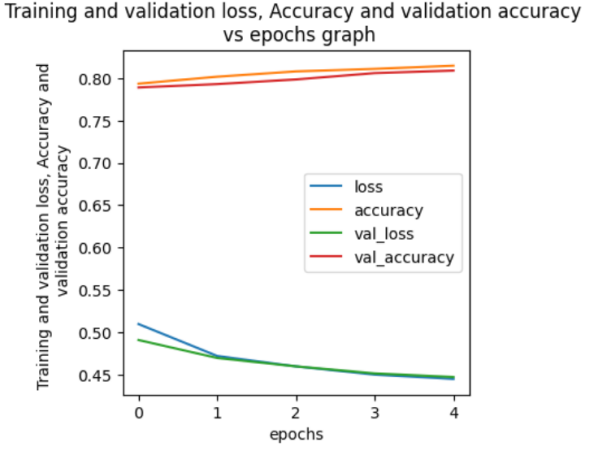
Early\_Stop = EarlyStopping()

epochs\_hist = classifier\_model.fit(X\_train,y\_train,epochs = 5, validation\_data = (X\_test, y\_test), batch\_size = 125, callbacks=[Early\_Stop])

**Epochs:**



**Accuracy graph:**



1. **Increase Number of Neurons:**

#imported Ann through keras API and tensorflow

#ANN is just to check how it performs on the dataset.

tf.random.set\_seed(42)

classifier\_model = tf.keras.models.Sequential()

classifier\_model.add(tf.keras.layers.Dense(units = 256, activation = 'relu', input\_shape = (10,)))

classifier\_model.add(tf.keras.layers.Dropout(0.3))

classifier\_model.add(tf.keras.layers.Dense(units = 128, activation = 'relu'))

classifier\_model.add(tf.keras.layers.Dropout(0.3))

classifier\_model.add(tf.keras.layers.Dense(units = 64, activation = 'relu'))

classifier\_model.add(tf.keras.layers.Dense(units = 1, activation = 'sigmoid'))

classifier\_model.compile(optimizer = 'Adam', loss = 'binary\_crossentropy', metrics = 'accuracy')

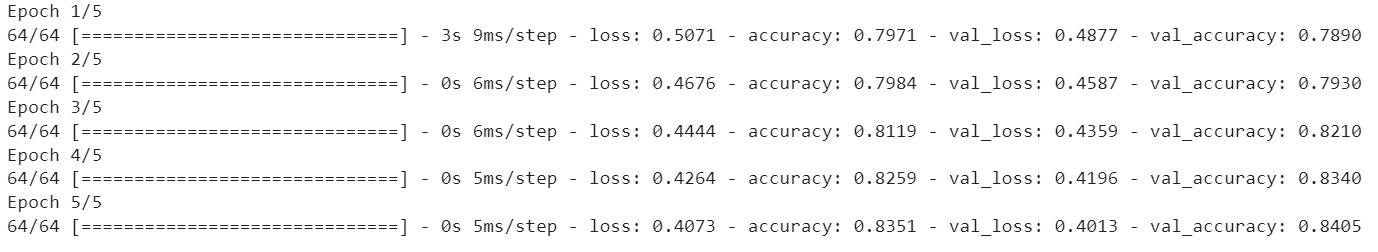
#calculating the epochs on training and validation dataset

from tensorflow.keras.callbacks import EarlyStopping

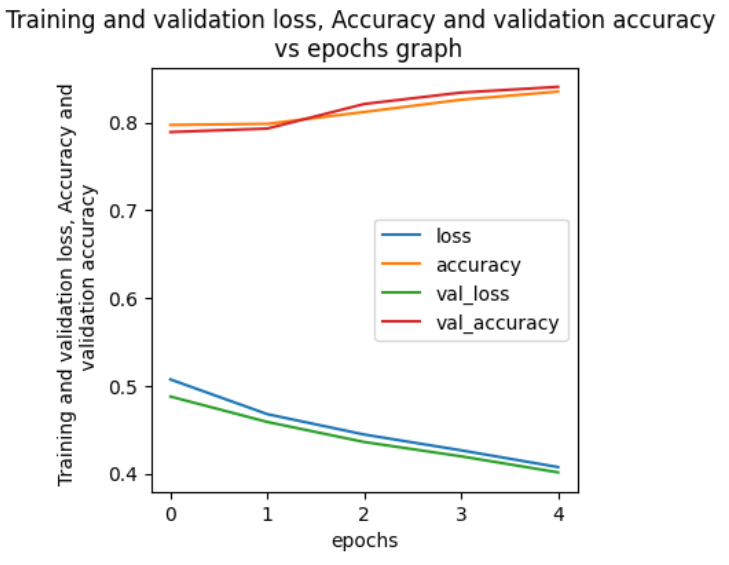
Early\_Stop = EarlyStopping()

epochs\_hist = classifier\_model.fit(X\_train,y\_train,epochs = 5, validation\_data = (X\_test, y\_test), batch\_size = 125, callbacks=[Early\_Stop])

**Epochs:**



**Accuracy graph after changing the number of neurons:**



**Result:**

|  |  |
| --- | --- |
| **Parameters** | **Accuracy** |
| 1. Data Preprocessing | 82.45%(old)  80.60% (After Preprocessing) |
| 1. Optimizers 2. Adam 3. RMSprop 4. Adadelta 5. Adagrad | 82.45%  83.50%  84.45%  84.20% |
| 1. Activation Functions 2. Relu 3. Linear 4. Tanh | 82.45%  79.90%  89.90% |
| 1. Increasing the number of neurons | 82.45% (old)  84.05% (new) |