**Practical: 9**

**Aim: Design and implement neural network with Pima Indian diabetes dataset**

**Code:**

# pandas, matplotlib, seaborn, numpy

import pandas as pd

import matplotlib.pyplot as plt

import seaborn as sns

import numpy as np

# Scikit-Learn:

from sklearn.impute import SimpleImputer

from sklearn.preprocessing import MinMaxScaler

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

from sklearn.metrics import confusion\_matrix

from sklearn import metrics

# Keras and Tensorflow

import tensorflow as tf

from tensorflow import keras

from tensorflow.keras.layers import Dropout, Dense

from tensorflow.keras.callbacks import EarlyStopping

from tensorflow.keras.losses import BinaryCrossentropy

from tensorflow.keras.optimizers import Adam

# Set tensorflow and numpy random seed for reproducible results

tf.random.set\_seed(42)

np.random.seed(42)

# Loading Data:

data = pd.read\_csv('https://raw.githubusercontent.com/SanjaSrdanovic/Diabetes\_Prediction\_Neural\_Network/main/diabetes.csv')

# seting printing values to see all data in the console

pd.set\_option('display.max\_columns', None)

pd.set\_option('display.max\_rows', None)

# data info

print(data.head())

print(data.info())

"""First visualisations to get some ideas about the distribution of diabetes

and correlation of variables

"""

# count plot for the Outcome - there is more women with no diabetes than with diabetes

sns.countplot(x='Outcome', data=data)

plt.show()

plt.savefig("Distribution\_of\_diabetes.png") # save figure

# heatmap to check for the correlation

sns.heatmap(data.corr())

plt.show()

plt.savefig("Correlation\_heatmap.png") # save figure

# bar graph to check the correlation by Outcome and

# get some ideas which variables have most influence

data.corr()['Outcome'][:-1].sort\_values().plot(kind='bar')

plt.savefig("Correlation\_barplot.png") # save figure

# histogram for each variable

data.hist(figsize=(18, 12))

plt.show()

plt.savefig("Histograms\_variables.png") # save figure

# Data Inspection

print(data.isnull().sum())

print(data.isna().sum())

print(data.describe())

# min values for Glucose, Blood Pressure, SkinThickness, Insulin, BMI cannot be 0

# => these are missing values

# label them as NaN values:

conditions\_with\_nan = ['Glucose', 'BloodPressure',

'SkinThickness', 'Insulin', 'BMI']

for i in conditions\_with\_nan:

print("There are " + str(len(data.loc[(data[i] == 0), i])) +

" instances of value 0 in "+i+" variable.")

for i in conditions\_with\_nan:

data.loc[(data[i] == 0), i] = np.NaN

print('Missing Number of Observations for all Variables:' +

"\n" + str(data.isnull().sum()))

# to check that there are no more 0s as min values

print(data[conditions\_with\_nan].min())

# input values - 8 variables

# output data - outcome

X = data.iloc[:, 0:8].values # Input values.

# Output values (outcome 1-has diabetes, 0-no diabetes)

y = data.iloc[:, 8].values

# split data into train and test

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

X, y, test\_size=0.2, random\_state=42)

print('X\_train: ' + str(X\_train.shape))

print('y\_train: ' + str(y\_train.shape))

print('X\_test: ' + str(X\_test.shape))

print('y\_test: ' + str(y\_test.shape))

print(type(X\_test))

# Impute mean for the X\_train

# for median imputation replace 'mean' with 'median'

imp\_mean = SimpleImputer(strategy='mean')

imp\_mean.fit(X\_train)

X\_train = imp\_mean.transform(X\_train)

# drop the NaN values in X\_test

i = pd.isnull(X\_test).any(1).nonzero()[0]

y\_test\_fin = np.delete(y\_test, i)

X\_test = X\_test[~np.isnan(X\_test).any(axis=1)]

# print(X\_test)

# scaling the data, i.e. normalize input features with MinMaxScaler

# This estimator scales and translates each feature individually such that it is in the given range on the training set, e.g. between zero and one.

scaler = MinMaxScaler()

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

X\_test = scaler.transform(X\_test)

# model 5

model = tf.keras.Sequential([

tf.keras.layers.Dense(8, input\_shape=(

8,), activation='relu', kernel\_initializer='he\_normal'),

tf.keras.layers.Dropout(0.25), # dropout for the input layer

tf.keras.layers.Dense(32, activation='relu',

kernel\_initializer='he\_normal'),

tf.keras.layers.Dropout(0.5), # dropout for the hidden layer

tf.keras.layers.Dense(1, activation='sigmoid',

kernel\_initializer='he\_normal')

])

model.compile(optimizer='Adam',

loss=tf.keras.losses.BinaryCrossentropy(),

metrics=['accuracy'])

# EarlyStopping to avoid overfitting

early\_stopping = EarlyStopping(

patience=30, monitor='val\_loss', restore\_best\_weights=True)

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

X\_test, y\_test\_fin), epochs=300, verbose=1, callbacks=[early\_stopping])

# Convert history of fitting to pandas Dataframe for plotting

history = pd.DataFrame(history.history)

# Plot losses and accuracy

history.plot(xlabel='epochs', ylabel='losses')

plt.show()

# Print out model architecture

print(model.summary())

# Final evaluation of generalisation error:

# Calculate accuracy on test set and print for final evaluation

acc\_test = model.evaluate(X\_test, y\_test\_fin)[1]

print(f'Accuracy Test Set : {acc\_test\*1E2:.1f}%')

# print(X\_test.shape)

# print(y\_test.shape)

# compare y\_test\_fin and y\_pred\_cat

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

y\_pred\_cat = np.around(y\_pred)

# Confusion Matrix

cm = confusion\_matrix(y\_test\_fin, y\_pred\_cat)

print(cm)

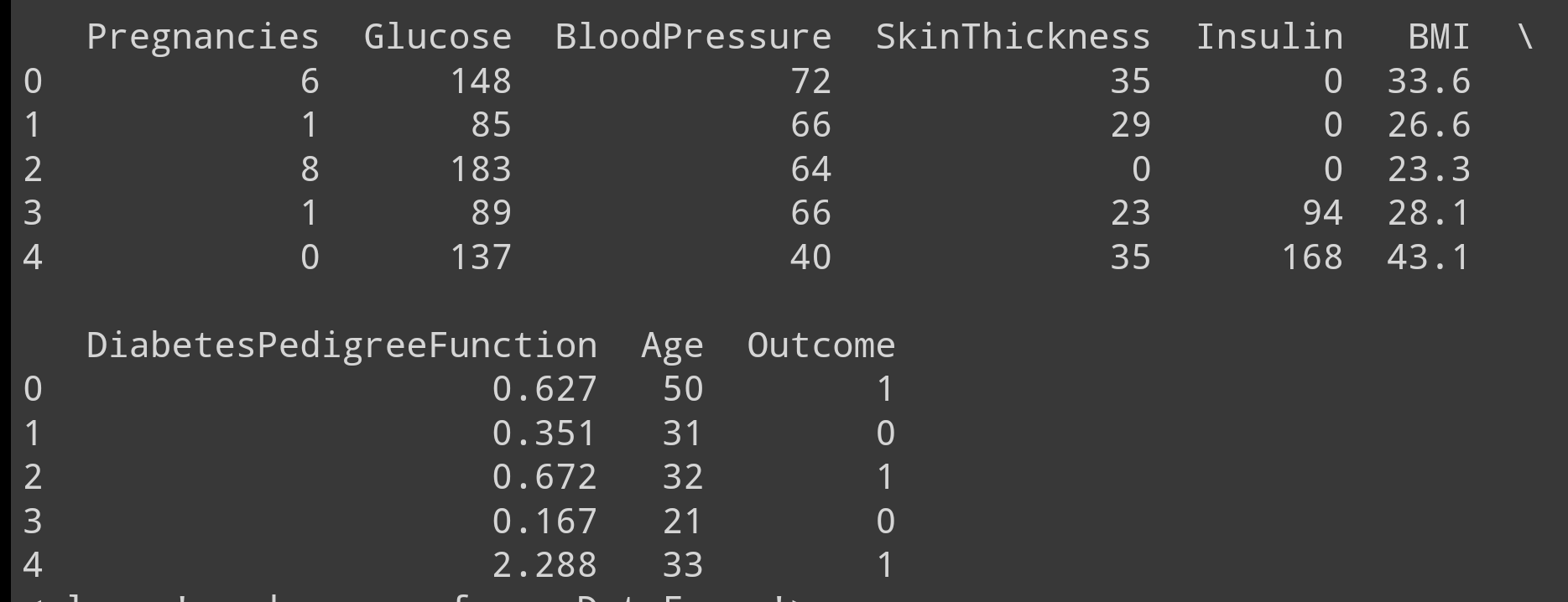
# Classification Report

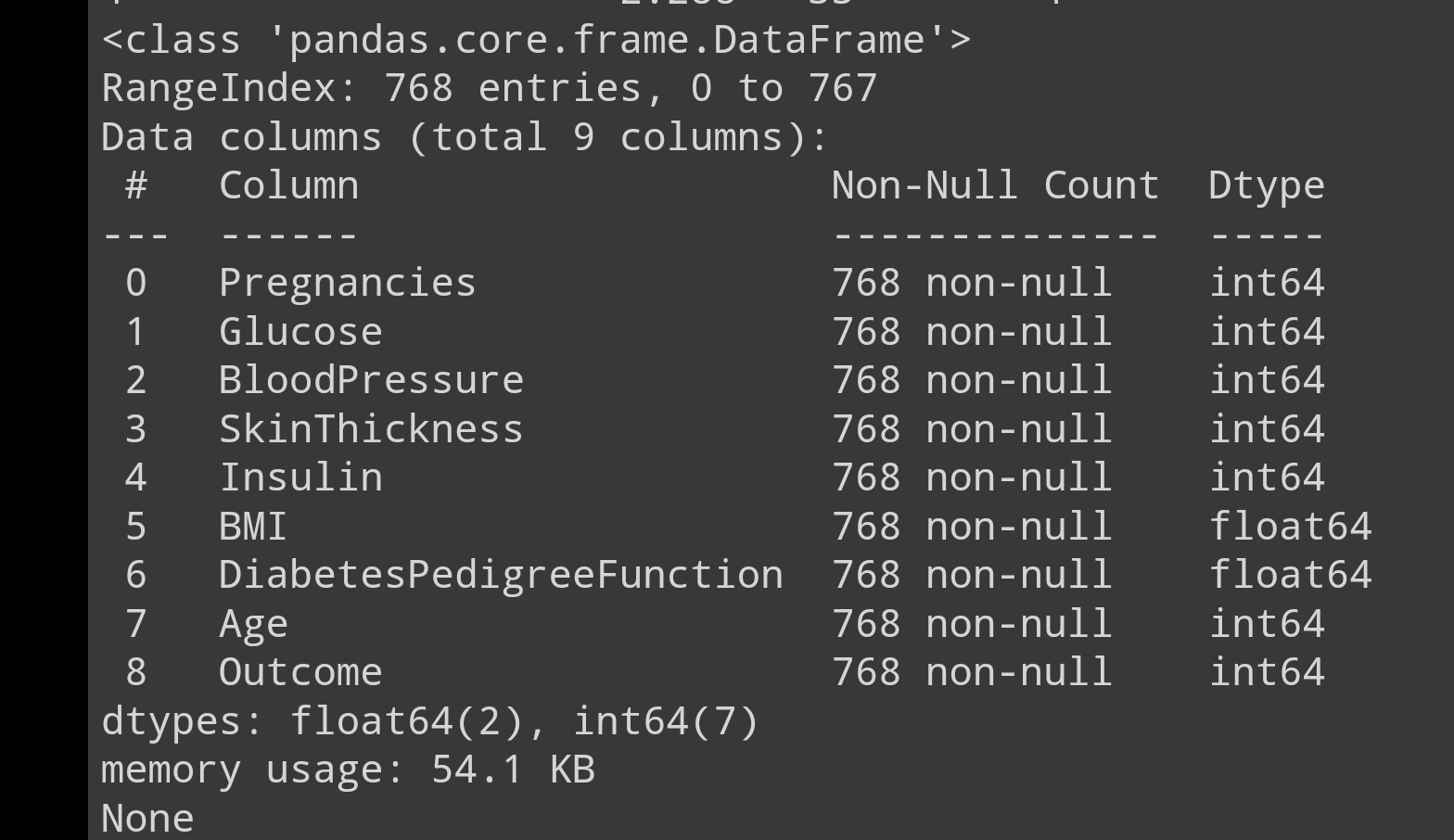
modelrep = metrics.classification\_report(y\_test\_fin, y\_pred\_cat)

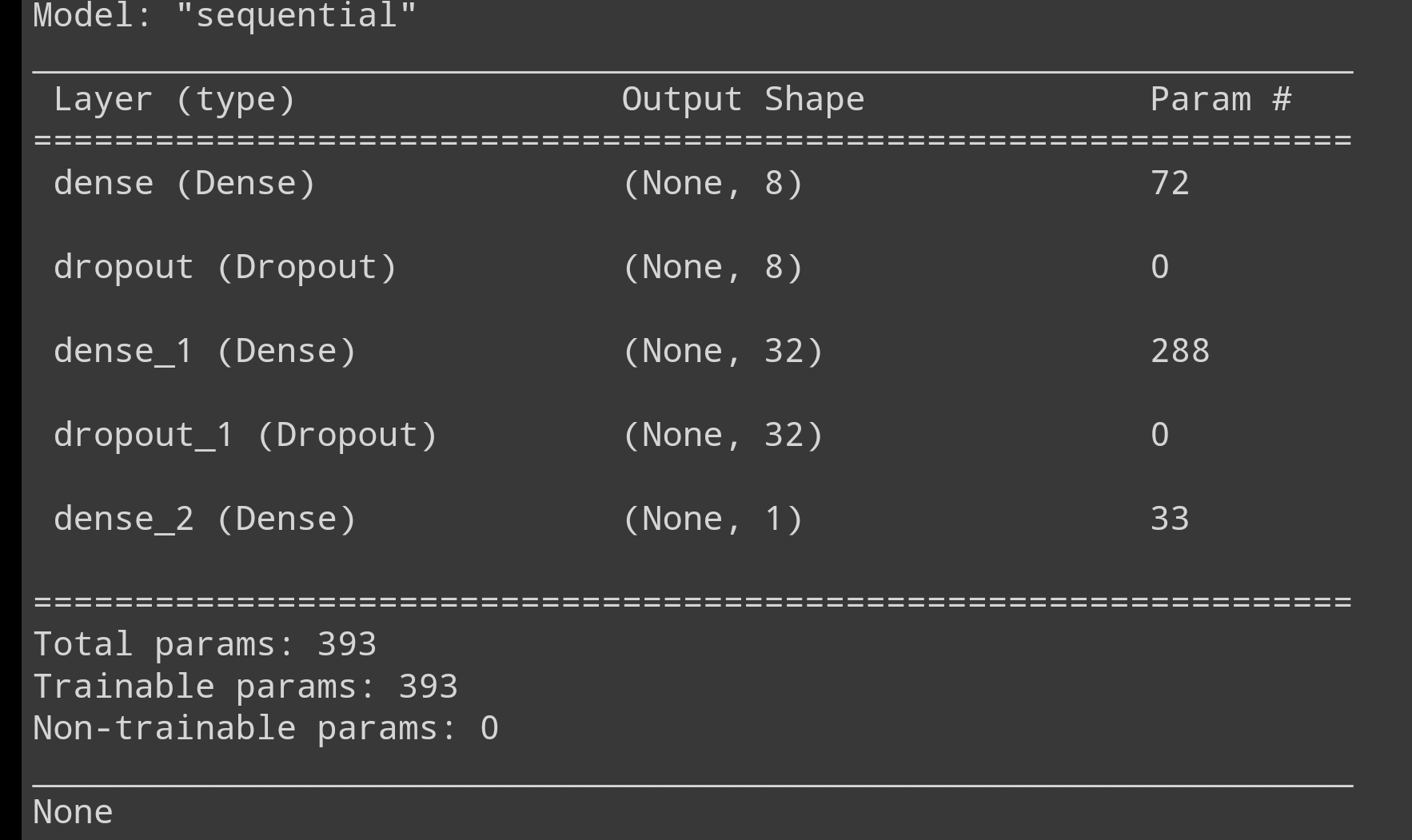
print(modelrep)

**Output:**

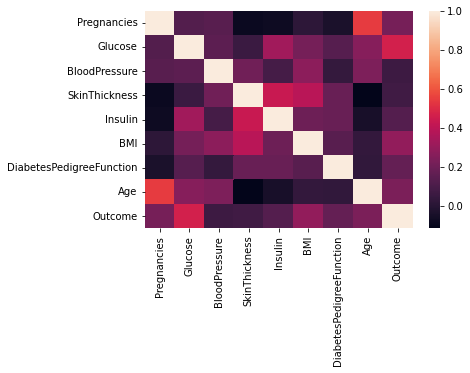
**Data information**

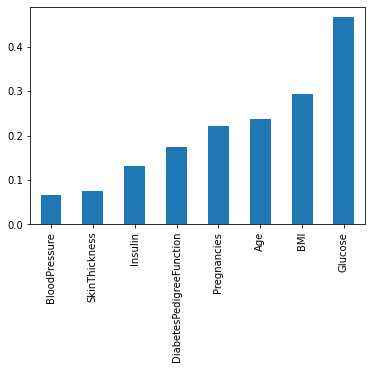
****

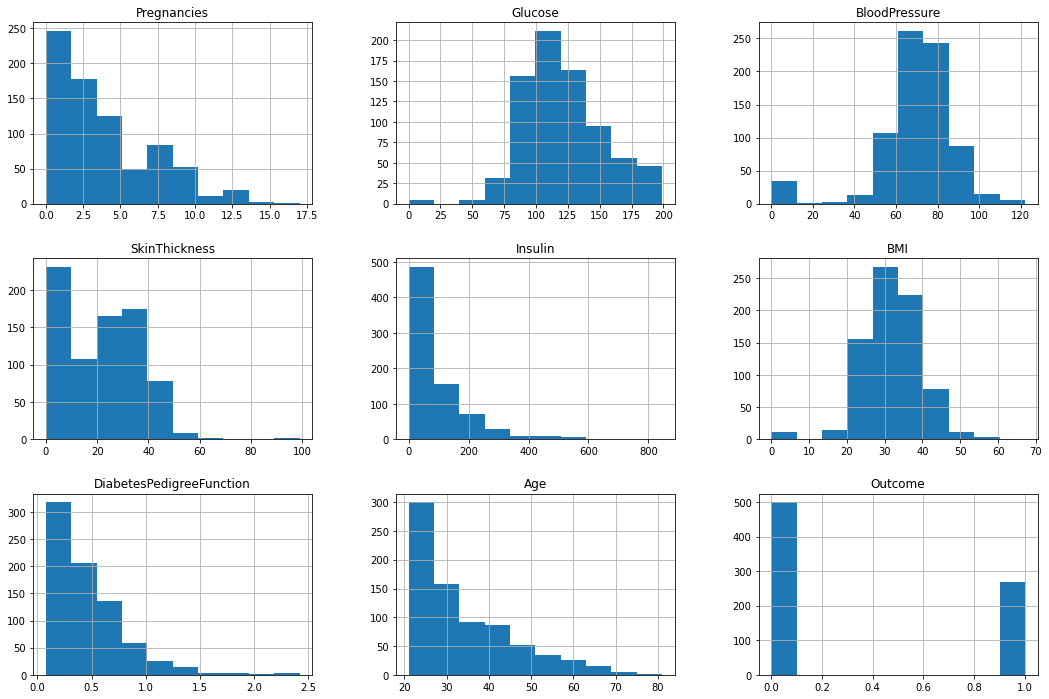
****

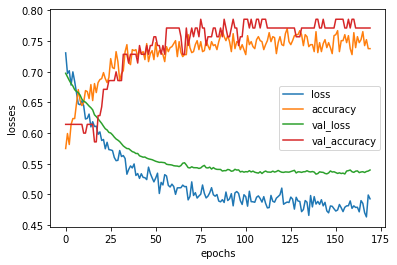
****

****

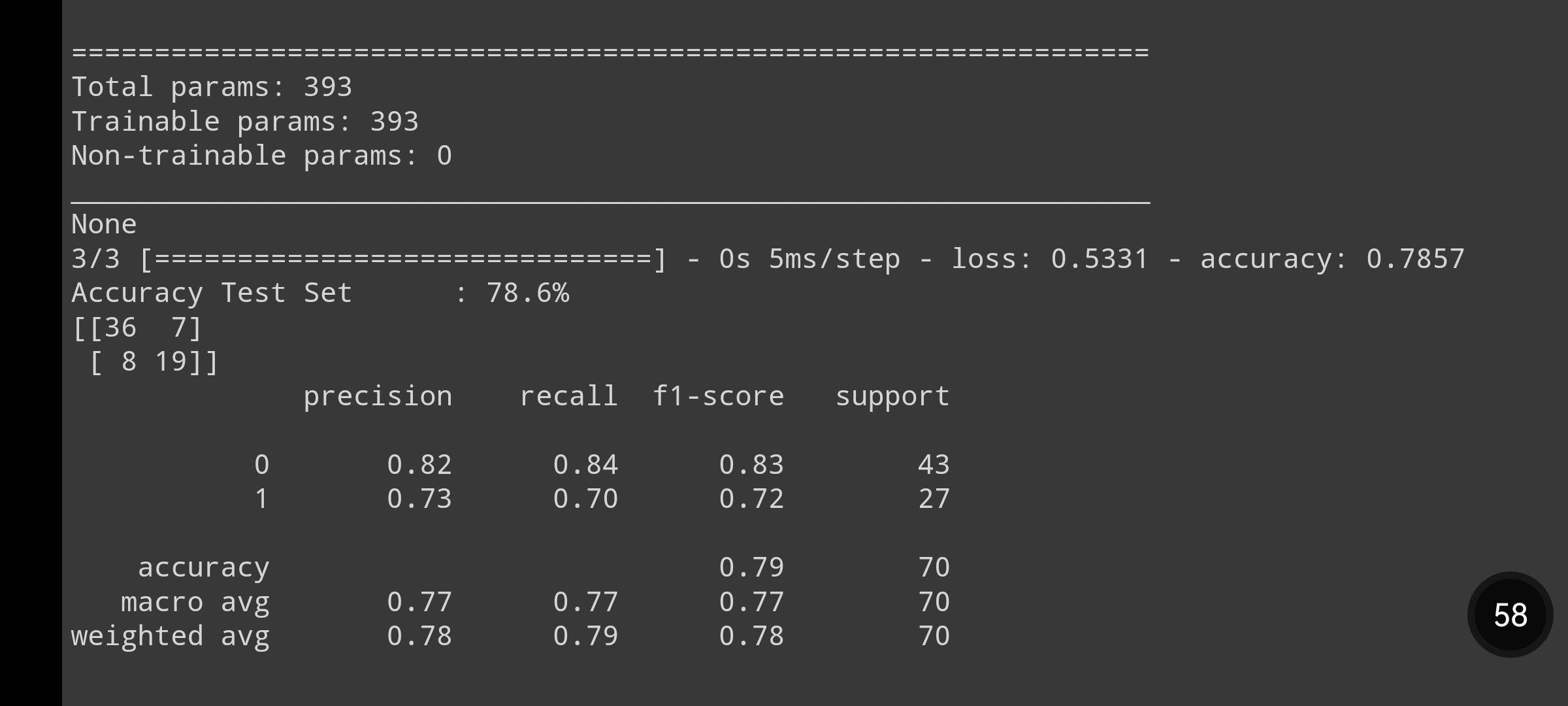
****

****

****

****

**Accuracy and Precision**

****