School of Computer Science and Engineering

Is Your Heart Healthy?

CSE3021 – Social and Information Networks Project Report

Submitted to:

Dr Punitha.K

Submitted by:

Rohan Reddy Melachervu

In partial fulfilment for the award of the degree of

Bachelor of Technology

in

Computer Science and Engineering

May 2021

TABLE OF CONTENTS

Page

1. Abstract:

Dataset will be taken from Kaggle. We will then proceed to check for the datatypes of each column and if they're all float/int values and also no Null values then we can proceed with the algorithms else there will be a requirement of pre-processing. We will then apply various ML algorithms and data visualizations to understand the data. Coronary heart disease (CHD) is one of the most frequent causes of death in adults in the industrialized countries. In Germany, one in five deaths were caused by CHD in 2003. In coronary heart disease the blood circulation of the heart muscle is inadequate; this is caused by a narrowing or blockage of one or more coronary blood vessels. A person with a healthy heart will have a lower mortality rate and a healthier and happier life. Our model will analyse the dataset and predict the deaths caused by heart failures. The most common cause of CVD is atherosclerosis. Atherosclerosis has an inflammatory nature, which was described by the Austrian pathologist Carl von Rokitansky in the 1840s and by Rudolf Virchow somewhat later. Rokitansky believed that inflammation was secondary to other disease processes and Virchow promoted atherosclerosis as a primary inflammatory disease (Frostegård, 2010; Mayerl et al., 2006). The response-to-injury hypothesis was summarized by Ross (1993). This theory postulates alteration to the endothelium and intima, due to for example mechanical injury, toxins, and oxygen radicals, as the initiating event leading to endothelial dysfunction. During the last two decades more data has linked inflammation to the occurrence of atherosclerosis and thrombosis

Heart Rate Failure Prediction

2.Introduction:

Cardiovascular diseases (CVDs) are the number 1 cause of death globally, taking an estimated 17.9 million lives each year, which accounts for 31% of all deaths worldwide. Heart failure is a common event caused by CVDs and this dataset contains 12 features that can be used to predict mortality by heart failure. Most cardiovascular diseases can be prevented by addressing behavioural risk factors such as tobacco use, unhealthy diet and obesity, physical inactivity and harmful use of alcohol using population-wide strategies. People with cardiovascular disease or who are at high cardiovascular risk (due to the presence of one or more risk factors such as hypertension, diabetes, hyperlipidaemia or already established disease) need early detection and management wherein a machine learning model can be of great help.

3.Feasibility study:

Dataset has been taken from a vast diverse group of people. The data in the dataset is very vast and diverse. It takes into consideration many factors which need a doctor appointment and result to enter. The predictor can be used by people to check whether their heart is in critical condition or not.

4.Design and Flow of modules:

Following are the modules in which we have divided are project:

- Forming the strategy for making prediction model and setting the definite approach.
- Analysing the data
- Visualizing the data using various types of graphs
- Creating a train and test dataset from the available data and making a feature extraction model.

- Implementing different ML models and algorithms and find out where the prediction is best.

4.1. Strategy:

We read the dataset, and then import various python libraries and perform various algorithms on the dataset, after pre-processing of course. Pre-processing ensures the dataset has no null values and the data will have some level of authenticity and reliability. Then we visualise the data in the form of graphs, to get a better understanding of what kind of dataset we're dealing with. Finally, we split the dataset into training and testing sets and apply algorithms on the training set and find which Machine Learning model yields the highest accuracy.

5.Risk Analysis:

Due to the vast amount of data and the diversity in it, achieving a 100% accuracy represents a significant challenge. But we have tried our best models to achieve greater accuracy.

6.About the dataset:

We'll work on a dataset which has 299 rows and 13 columns. The columns consist of information like age, sex, sodium level, etc. the last column that is the 13th column is the death event of the patient, which tells whether the patient will get a heart failure.

Link to dataset: https://www.kaggle.com/andrewmvd/heart-failure-clinical-data

Heart Rate Failure Prediction

7.Implementation:

We will first import all the necessary libraries and modules:

```
import pandas as pd
import numpy as np
import seaborn as sns
import plotly.express as px
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
```

+ Code + Text

7.1 Analysis:

The dataset has 299 rows and 13 columns.

```
[3] df.shape

    (299, 13)
```

7.2 Pre-Processing:

The dataset has no null values

```
round((df.isnull().sum()/len(df))*100)
```

```
age                         0.0
anaemia                     0.0
creatinine_phosphokinase    0.0
diabetes                    0.0
ejection_fraction           0.0
high_blood_pressure         0.0
platelets                   0.0
serum_creatinine            0.0
serum_sodium                0.0
sex                         0.0
smoking                     0.0
time                        0.0
DEATH_EVENT                 0.0
dtype: float64
```

All the columns in the dataset are of int/float data-types

```
df.dtypes
```

```
age                         float64
anaemia                       int64
creatinine_phosphokinase      int64
diabetes                      int64
ejection_fraction             int64
high_blood_pressure           int64
platelets                   float64
serum_creatinine            float64
serum_sodium                  int64
sex                           int64
smoking                       int64
time                          int64
DEATH_EVENT                   int64
dtype: object
```

Since no null values are present and all the columns are of float/int types, data pre-processing is done.

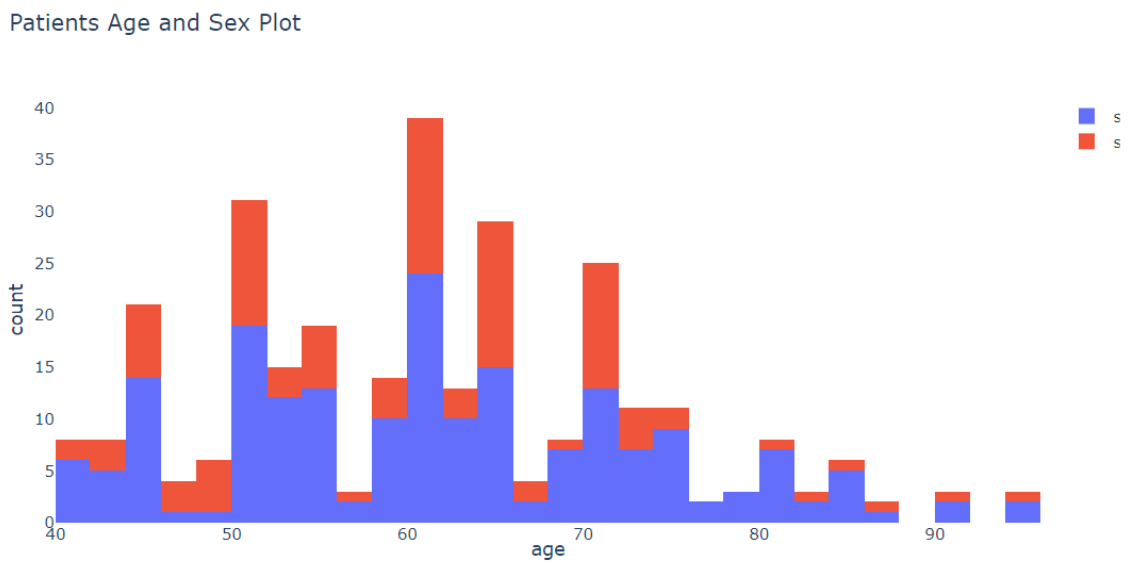
7.3 Data Visualization

Histogram

A histogram is a chart that plots the distribution of a numeric variable's values as a series of bars. It's a nice way to understand the dataset.

```
[8]  HEIGHT = 500
     WIDTH = 900
     NBINS = 50
     SCATTER_SIZE=700


     def plot_histogram(dataframe, column, color, bins, title, width=WIDTH, height=HEIGHT):
         figure = px.histogram(
             dataframe,
             column,
             color=color,
             nbins=bins,
             title=title,
             width=width,
             height=height
         )
         figure.update_layout({
                 'plot_bgcolor': 'rgba(0, 0, 0, 0)',
                 'paper_bgcolor': 'rgba(0, 0, 0, 0)',
             })
         figure.show()
```

```
[9]  #PLOTTING THE HISTOGRAM
     plot_histogram(df, 'age', 'sex', NBINS, 'Patients Age and Sex Plot')
```

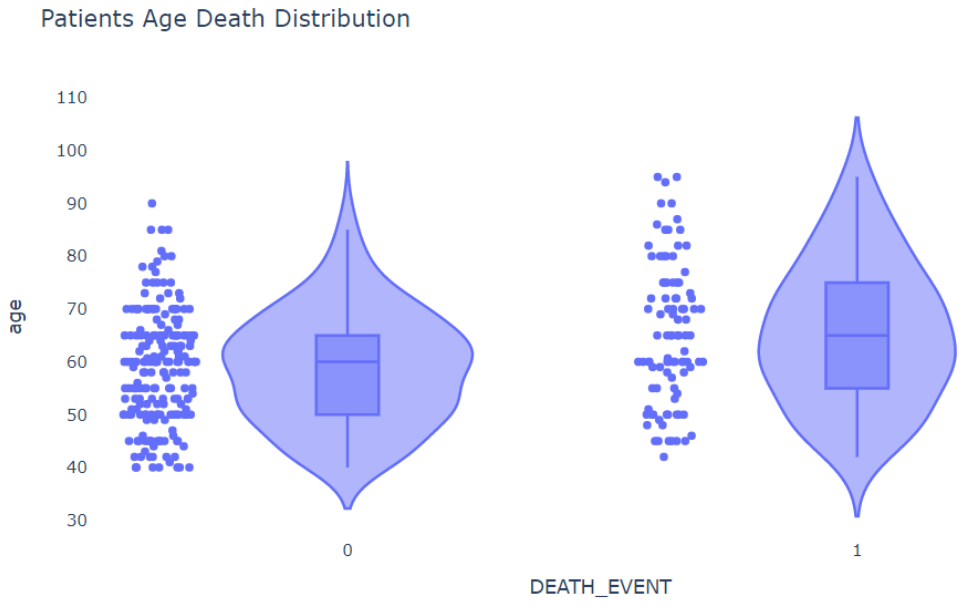


PATIENTS COUNT/SEX PLOT

Violin plot

Violin plot is a method of plotting numeric data. It is similar to a box plot. It shows the median of the data too.

```
plot_violin(df, 'DEATH_EVENT', 'age', 'Patients Age Death Distribution')
```



AGE/DEATH_EVENT VIOLIN PLOT

7.4 Splitting dataset to Training and Testing

```
[11] x = df.iloc[:, :-1].values
     y = df.iloc[:,-1].values
```

```
[12] # Splitting the dataset into training set and test set
     X_train, X_test, y_train, y_test = train_test_split(x, y, test_size = 0.2,random_state=0)
```

```
[13] print(len(X_train))

     239
```

X would be an array of all values except the last column and Y would be an array of the last column. 80% of the dataset would now be the Training dataset on which we would apply various algorithms to achieve best accuracy. The rest 20% would be Test dataset.

Here random_state=0 ensures the training and test dataset remains constant for each run.

7.5 Accuracy using different models

Using the Kernel rbf model, we've achieved an accuracy of 61.66%.

```
[15] from sklearn import svm
     from sklearn.metrics import accuracy_score
     clf = svm.SVC(kernel='rbf')
     clf.fit(X_train, y_train)
     y_pred2=clf.predict(X_test)
     print(accuracy_score(y_test,y_pred2)*100)

     61.66666666666667
```

Standard Scaler

It's a scaling technique where the values are centred around the mean with a Standard Deviation of 1 and a Mean of 0. It's performed to increase the accuracy.

Using Standard Scaler, we got an accuracy of 80%

```
[16] sc = StandardScaler()
     X_train = sc.fit_transform(X_train)
     X_test = sc.transform(X_test)
```

```
[17] from sklearn import svm
     from sklearn.metrics import accuracy_score
     clf1 = svm.SVC(kernel='rbf')
     clf1.fit(X_train, y_train)
     y_pred3=clf1.predict(X_test)
     print(accuracy_score(y_test,y_pred3)*100)

     80.0
```

Decision Tree

In this section, we have implemented Decision Tree algorithm on our Train dataset. Decision Tree is a flowchart like tree structure. It breaks down a dataset into smaller and smaller subsets while at the same time an associated decision tree is incrementally developed. The final result is a tree with decision nodes and leaf nodes. Decision trees provide an effective method of Decision making because they clearly lay out the problem so that all options can be challenged which allows us to analyse fully the possible consequences of a decision.

```
[18] from sklearn.tree import DecisionTreeClassifier
     from sklearn.tree import export_text
     decision_tree = DecisionTreeClassifier(random_state=0, max_depth=2)
```

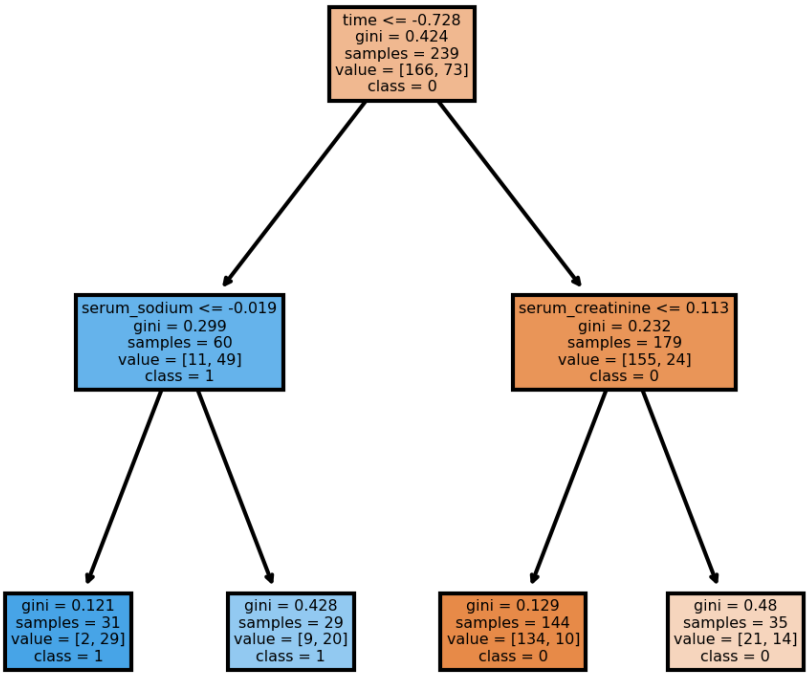
```
[19] decision_tree = decision_tree.fit(X_train, y_train)
```

```
[20] from sklearn import tree
     text_representation = tree.export_text(decision_tree)
     print(text_representation)
```

```
|--- feature_11 <= -0.73
|   |--- feature_8 <= -0.02
|   |   |--- class: 1
|   |--- feature_8 >  -0.02
|   |   |--- class: 1
|--- feature_11 >  -0.73
|   |--- feature_7 <= 0.11
|   |   |--- class: 0
|   |--- feature_7 >  0.11
|   |   |--- class: 0
```

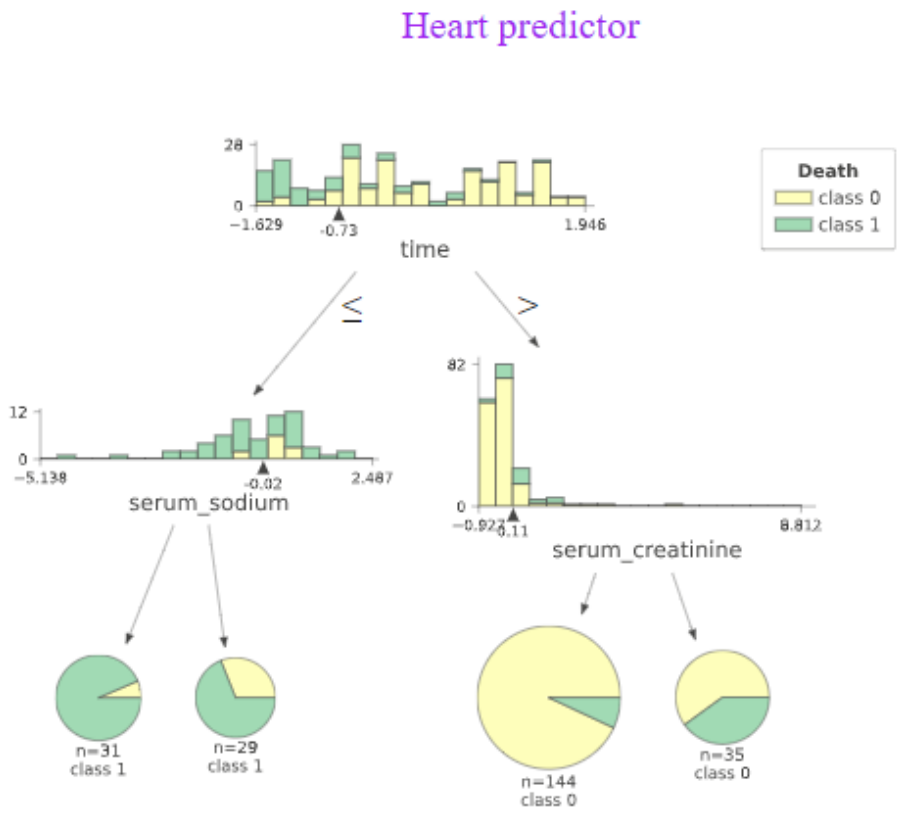
```
[21] import matplotlib.pyplot as plt
     fn = ['age', 'anaemia', 'creatinine_phosphokinase', 'diabetes',
           'ejection_fraction', 'high_blood_pressure', 'platelets',
           'serum_creatinine', 'serum_sodium', 'sex', 'smoking', 'time',
           ]
     cn=['0','1']
     fig, axes = plt.subplots(nrows = 1,ncols = 1,figsize = (4,4), dpi=300)
     tree.plot_tree(decision_tree,
                    feature_names = fn,
                    class_names=cn,
                    filled = True)
```

```
[Text(465.0, 755.0, 'time <= -0.728\ngini = 0.424\nsamples = 239\nvalue = [166, 73]\nclass = 0'),
 Text(232.5, 453.0, 'serum_sodium <= -0.019\ngini = 0.299\nsamples = 60\nvalue = [11, 49]\nclass = 1'),
 Text(116.25, 151.0, 'gini = 0.121\nsamples = 31\nvalue = [2, 29]\nclass = 1'),
 Text(348.75, 151.0, 'gini = 0.428\nsamples = 29\nvalue = [9, 20]\nclass = 1'),
 Text(697.5, 453.0, 'serum_creatinine <= 0.113\ngini = 0.232\nsamples = 179\nvalue = [155, 24]\nclass = 0'),
 Text(581.25, 151.0, 'gini = 0.129\nsamples = 144\nvalue = [134, 10]\nclass = 0'),
 Text(813.75, 151.0, 'gini = 0.48\nsamples = 35\nvalue = [21, 14]\nclass = 0')]
```



DECISION TREE

```
[23] viz = dtreeviz(decision_tree,
                    X_train,
                    y_train,
                    target_name='Death',
                    feature_names=fn,
                    title="Heart predictor",
                    fontname="DejaVu Sans",
                    title_fontsize=16,
                    colors = {"title":"purple"}
                   )
     viz
```



VIZ DECISION TREE

With Decision Tree, we achieved an accuracy of 81.66%

```
[24] from sklearn.metrics import accuracy_score
     y_pred=decision_tree.predict(X_test)
     print(accuracy_score(y_test,y_pred)*100)

     81.66666666666667
```

KNN

This algorithm is used for Classification and Regression. It takes the whole dataset into consideration for learning. It clusters various points with similar attributes with each other. It works by calculating Euclidian distance from one point to each point in the dataset and clustering it according to the results.

Using KNN we achieved and accuracy of 75%.

```
[27] from sklearn.neighbors import KNeighborsClassifier
     classifier = KNeighborsClassifier(n_neighbors=5)
     classifier.fit(X_train, y_train)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=5, p=2,
                     weights='uniform')
```

```
[28] y_pred = classifier.predict(X_test)
     from sklearn.metrics import classification_report, confusion_matrix
     print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.70      0.95      0.80        37
           1       0.80      0.35      0.48        23

    accuracy                           0.72        60
   macro avg       0.75      0.65      0.64        60
weighted avg       0.74      0.72      0.68        60
```

Gaussian Bayes

Gaussian Naive Bayes is a variant of Naive Bayes that follows Gaussian normal distribution and supports continuous data. Naive Bayes are a group of supervised machine learning classification algorithms based on the Bayes theorem. It is a simple classification technique, but has high functionality.

Using Gaussian Bayes, we got an accuracy of 69%.

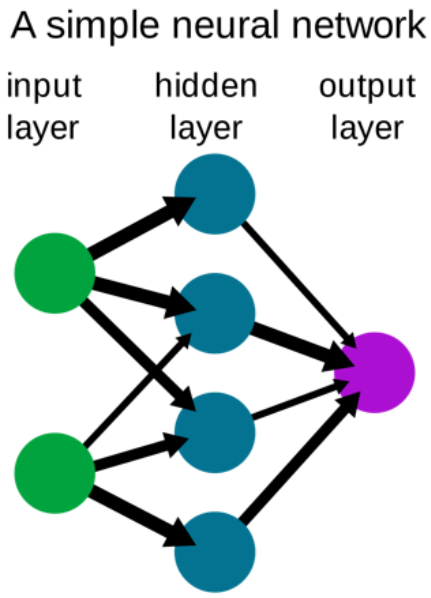
```
[29] from sklearn.naive_bayes import GaussianNB
     gnb = GaussianNB()
     gnb.fit(X_train, y_train)
     y_pred = gnb.predict(X_test)
     print(classification_report(y_test, y_pred))
```

```
              precision    recall  f1-score   support

           0       0.68      0.92      0.78        37
           1       0.70      0.30      0.42        23

    accuracy                           0.68        60
   macro avg       0.69      0.61      0.60        60
weighted avg       0.69      0.68      0.64        60
```

Neural Network

Neural Networks have the ability to learn and model non-linear and complex relationships. It has an input layer which gives input, and output layer which predicts final output. In between are hidden layers which perform computation. Each hidden layer is connected to the other via channels. Each channel has a certain weight. Each neuron will hence have a numerical value called bias. This value is passed to activation function and if the activation function is passed, it will go to the next layer, this is called forward propagation. If wrong prediction then backward propagation takes place. This continues until the network can predict properly. Each propagation from start to end is called an epoch.

A simple example of a neural network is something like this



SAMPLE NEURAL NETWORK

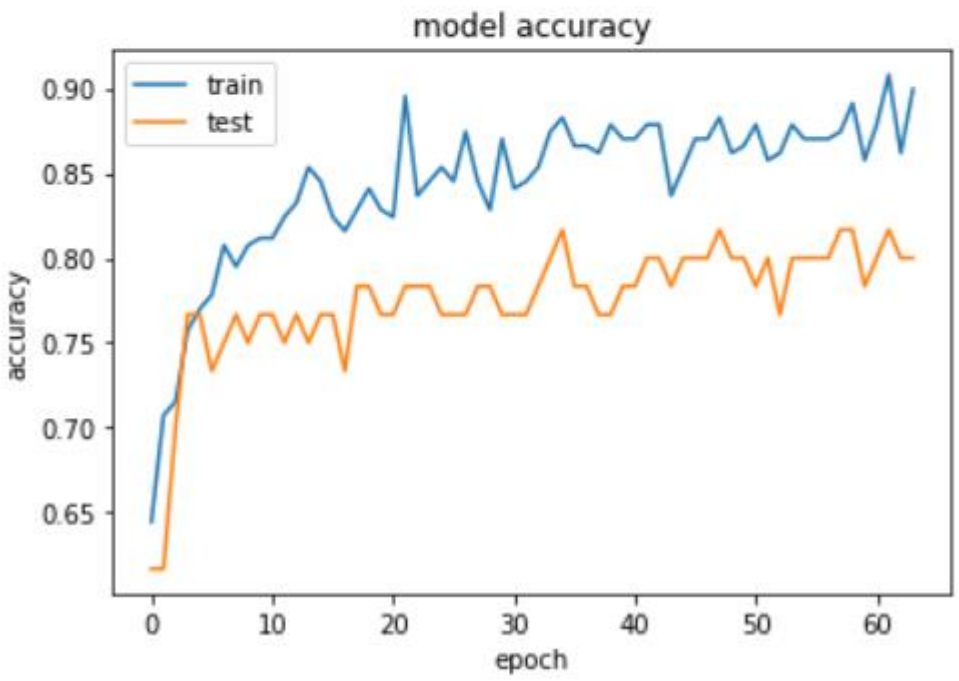
```
[30] import tensorflow as tf
     import tensorflow.keras as keras
     import tensorboard
     from tensorboard import notebook
     %load_ext tensorboard
     import datetime
     import os
     logdir = os.path.join("logs",datetime.datetime.now().strftime("%Y%m%d - %H%M%S"))
     tensorboard_callback = tf.keras.callbacks.TensorBoard(logdir,histogram_freq = 1)
     import tensorflow.keras.layers as layers
     model = keras.models.Sequential([
                        layers.Dense(100,activation = "relu"),
                        layers.Dropout(0.5),
                        layers.Dense(128,activation = "relu"),
                        layers.Dense(256,activation = "relu"),
                        layers.Dropout(0.5),
                        layers.Dense(10,activation = "relu"),
                        layers.Dense(2),
     ])

     model.compile(
         optimizer = tf.keras.optimizers.Adam(),
         loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits = True),
         metrics = ["accuracy"]
     )
     history = model.fit(X_train,y_train,batch_size = 32 , epochs = 64,validation_data=(X_test,y_test),callbacks=[tensorboard_callback])
```

64 epochs have been sent; accuracy increases through each epoch.

```
Epoch 1/64
8/8 [==============================] - 1s 96ms/step - loss: 0.6742 - accuracy: 0.6018 - val_loss: 0.6285 - val_accuracy: 0.6167
Epoch 2/64
8/8 [==============================] - 0s 9ms/step - loss: 0.6508 - accuracy: 0.6460 - val_loss: 0.5983 - val_accuracy: 0.6167
Epoch 3/64
8/8 [==============================] - 0s 9ms/step - loss: 0.5411 - accuracy: 0.7289 - val_loss: 0.5601 - val_accuracy: 0.7000
Epoch 4/64
8/8 [==============================] - 0s 10ms/step - loss: 0.5208 - accuracy: 0.7344 - val_loss: 0.5269 - val_accuracy: 0.7667
Epoch 5/64
8/8 [==============================] - 0s 9ms/step - loss: 0.4786 - accuracy: 0.7834 - val_loss: 0.5163 - val_accuracy: 0.7667
Epoch 6/64
8/8 [==============================] - 0s 11ms/step - loss: 0.4944 - accuracy: 0.7653 - val_loss: 0.4975 - val_accuracy: 0.7333
Epoch 7/64
8/8 [==============================] - 0s 10ms/step - loss: 0.4050 - accuracy: 0.8146 - val_loss: 0.4795 - val_accuracy: 0.7500
Epoch 8/64
8/8 [==============================] - 0s 9ms/step - loss: 0.4088 - accuracy: 0.8102 - val_loss: 0.4646 - val_accuracy: 0.7667
Epoch 9/64
8/8 [==============================] - 0s 10ms/step - loss: 0.3981 - accuracy: 0.8181 - val_loss: 0.4683 - val_accuracy: 0.7500
Epoch 10/64
8/8 [==============================] - 0s 10ms/step - loss: 0.4968 - accuracy: 0.7754 - val_loss: 0.4663 - val_accuracy: 0.7667
Epoch 11/64
8/8 [==============================] - 0s 9ms/step - loss: 0.4145 - accuracy: 0.7848 - val_loss: 0.4712 - val_accuracy: 0.7667
Epoch 12/64
8/8 [==============================] - 0s 10ms/step - loss: 0.3906 - accuracy: 0.8287 - val_loss: 0.4463 - val_accuracy: 0.7500
Epoch 13/64
8/8 [==============================] - 0s 11ms/step - loss: 0.4417 - accuracy: 0.8263 - val_loss: 0.4558 - val_accuracy: 0.7667
Epoch 14/64
8/8 [==============================] - 0s 10ms/step - loss: 0.3710 - accuracy: 0.8360 - val_loss: 0.4851 - val_accuracy: 0.7500
Epoch 15/64
8/8 [==============================] - 0s 10ms/step - loss: 0.3476 - accuracy: 0.8327 - val_loss: 0.4792 - val_accuracy: 0.7667
Epoch 16/64
8/8 [==============================] - 0s 11ms/step - loss: 0.3569 - accuracy: 0.8273 - val_loss: 0.4558 - val_accuracy: 0.7667
Epoch 17/64
8/8 [==============================] - 0s 11ms/step - loss: 0.3614 - accuracy: 0.8366 - val_loss: 0.4607 - val_accuracy: 0.7333
Epoch 18/64
8/8 [==============================] - 0s 10ms/step - loss: 0.3464 - accuracy: 0.8330 - val_loss: 0.4907 - val_accuracy: 0.7833
Epoch 19/64
8/8 [==============================] - 0s 11ms/step - loss: 0.3614 - accuracy: 0.8338 - val_loss: 0.5113 - val_accuracy: 0.7833
Epoch 20/64
8/8 [==============================] - 0s 11ms/step - loss: 0.3770 - accuracy: 0.8213 - val_loss: 0.4728 - val_accuracy: 0.7667
Epoch 21/64
8/8 [==============================] - 0s 9ms/step - loss: 0.3402 - accuracy: 0.8497 - val_loss: 0.4609 - val_accuracy: 0.7667
Epoch 22/64
8/8 [==============================] - 0s 10ms/step - loss: 0.3127 - accuracy: 0.8792 - val_loss: 0.4863 - val_accuracy: 0.7833
Epoch 23/64
8/8 [==============================] - 0s 9ms/step - loss: 0.3758 - accuracy: 0.8122 - val_loss: 0.4822 - val_accuracy: 0.7833
Epoch 24/64
8/8 [==============================] - 0s 9ms/step - loss: 0.3351 - accuracy: 0.8459 - val_loss: 0.4685 - val_accuracy: 0.7833
Epoch 25/64
8/8 [==============================] - 0s 10ms/step - loss: 0.2940 - accuracy: 0.8794 - val_loss: 0.5062 - val_accuracy: 0.7667
Epoch 26/64
8/8 [==============================] - 0s 10ms/step - loss: 0.3489 - accuracy: 0.8359 - val_loss: 0.5007 - val_accuracy: 0.7667
Epoch 27/64
```

```
8/8 [==============================] - 0s 11ms/step - loss: 0.3005 - accuracy: 0.8326 - val_loss: 0.4727 - val_accuracy: 0.8000
Epoch 50/64
8/8 [==============================] - 0s 10ms/step - loss: 0.2883 - accuracy: 0.8594 - val_loss: 0.4971 - val_accuracy: 0.8000
Epoch 51/64
8/8 [==============================] - 0s 10ms/step - loss: 0.2865 - accuracy: 0.8784 - val_loss: 0.5257 - val_accuracy: 0.7833
Epoch 52/64
8/8 [==============================] - 0s 10ms/step - loss: 0.2865 - accuracy: 0.8499 - val_loss: 0.5039 - val_accuracy: 0.8000
Epoch 53/64
8/8 [==============================] - 0s 10ms/step - loss: 0.2755 - accuracy: 0.8694 - val_loss: 0.4478 - val_accuracy: 0.7667
Epoch 54/64
8/8 [==============================] - 0s 9ms/step - loss: 0.2316 - accuracy: 0.8918 - val_loss: 0.4738 - val_accuracy: 0.8000
Epoch 55/64
8/8 [==============================] - 0s 10ms/step - loss: 0.2477 - accuracy: 0.8769 - val_loss: 0.4767 - val_accuracy: 0.8000
Epoch 56/64
8/8 [==============================] - 0s 10ms/step - loss: 0.3108 - accuracy: 0.8514 - val_loss: 0.4477 - val_accuracy: 0.8000
Epoch 57/64
8/8 [==============================] - 0s 10ms/step - loss: 0.2333 - accuracy: 0.8888 - val_loss: 0.4554 - val_accuracy: 0.8000
Epoch 58/64
8/8 [==============================] - 0s 10ms/step - loss: 0.2286 - accuracy: 0.8954 - val_loss: 0.4971 - val_accuracy: 0.8167
Epoch 59/64
8/8 [==============================] - 0s 11ms/step - loss: 0.2377 - accuracy: 0.8933 - val_loss: 0.4816 - val_accuracy: 0.8167
Epoch 60/64
8/8 [==============================] - 0s 10ms/step - loss: 0.2873 - accuracy: 0.8530 - val_loss: 0.4577 - val_accuracy: 0.7833
Epoch 61/64
8/8 [==============================] - 0s 11ms/step - loss: 0.2697 - accuracy: 0.8677 - val_loss: 0.5239 - val_accuracy: 0.8000
Epoch 62/64
8/8 [==============================] - 0s 9ms/step - loss: 0.2256 - accuracy: 0.9111 - val_loss: 0.5135 - val_accuracy: 0.8167
Epoch 63/64
8/8 [==============================] - 0s 10ms/step - loss: 0.2536 - accuracy: 0.8602 - val_loss: 0.4709 - val_accuracy: 0.8000
Epoch 64/64
8/8 [==============================] - 0s 11ms/step - loss: 0.2345 - accuracy: 0.8935 - val_loss: 0.4630 - val_accuracy: 0.8000
```

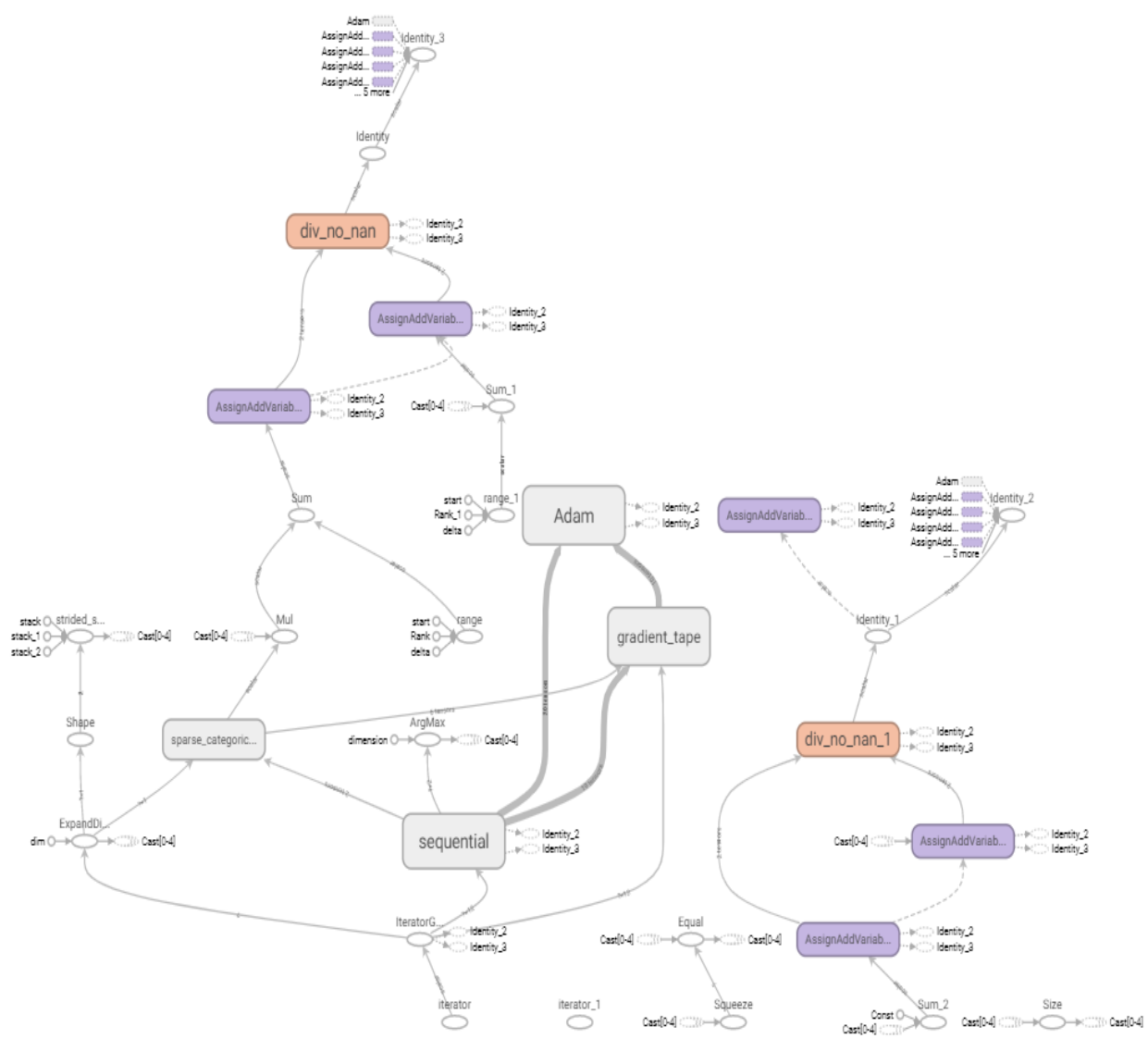
```python
import matplotlib.pyplot as plt

plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'test'], loc='upper left')
plt.show()
```



The last epoch had an accuracy of 80%. We have also plotted the accuracy of our dataset per epoch.

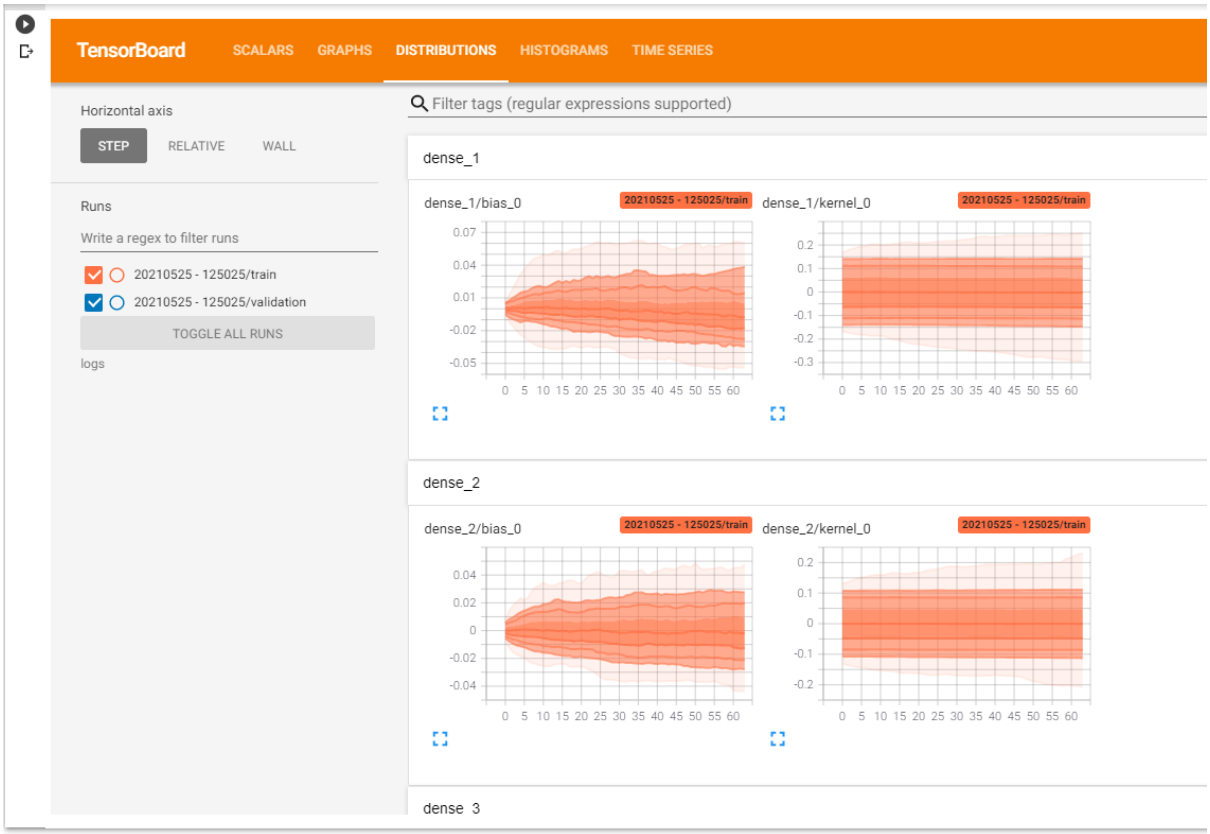
8. Tensor Board

Using Tensor Board, we could represent the whole neural network.

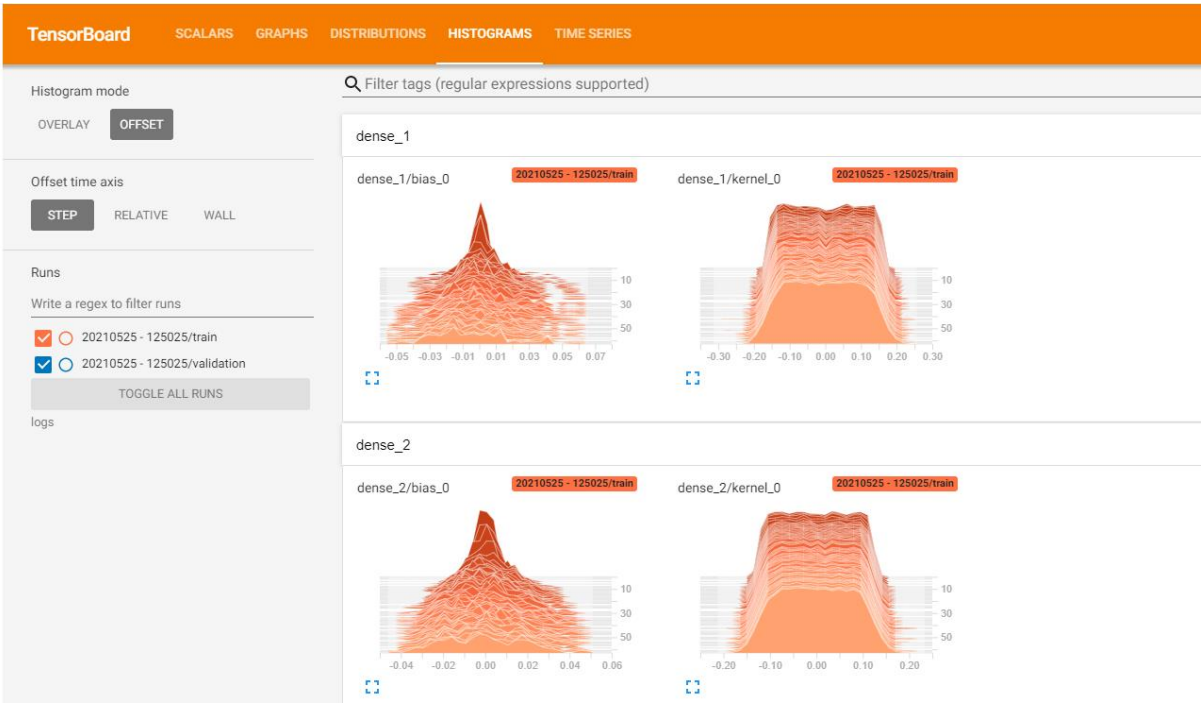


TENSOR BOARD MODEL

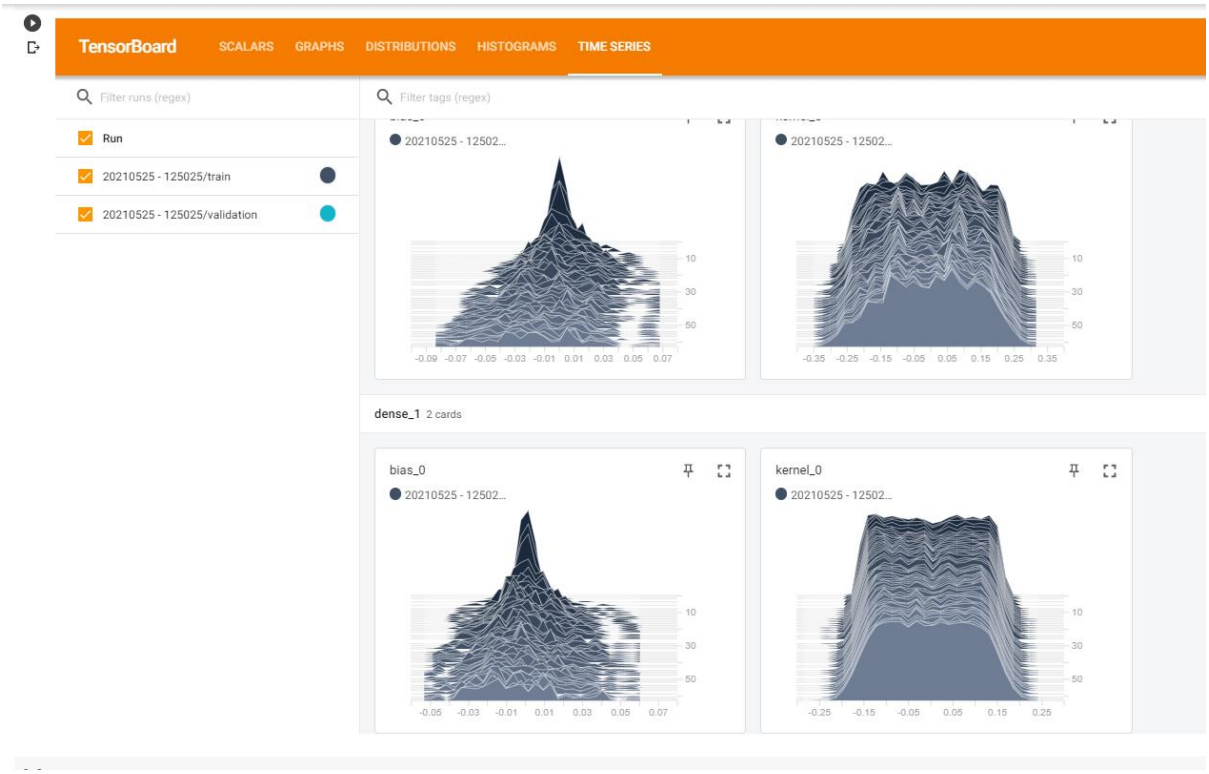
We can also notice the various Density distributions per epoch.

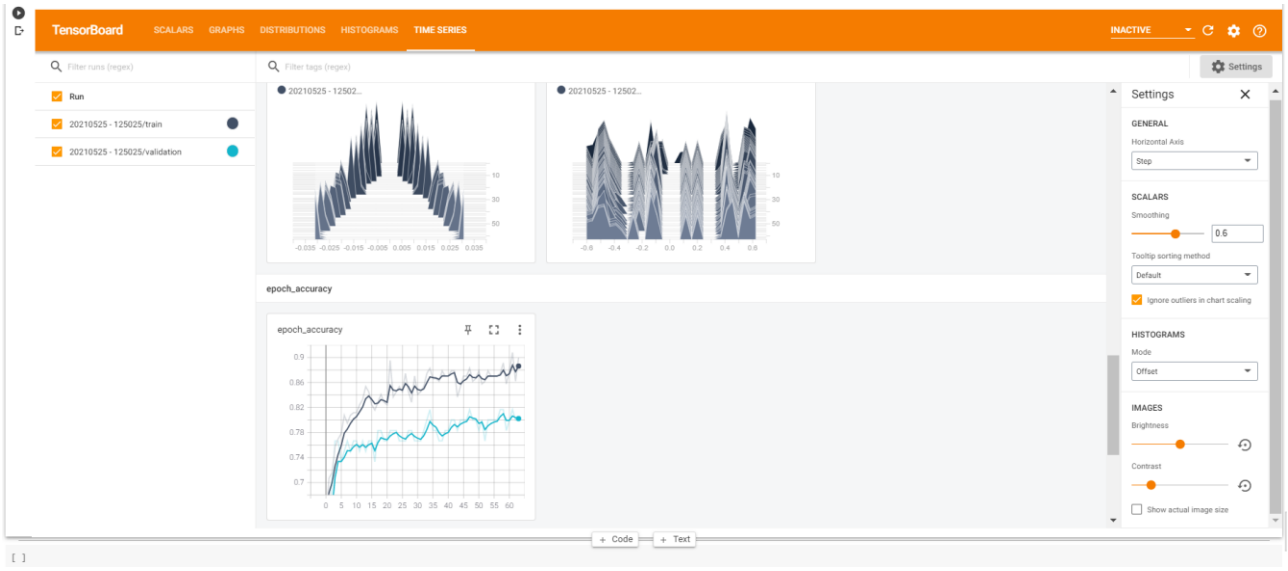


Histograms per epoch:



And lastly the Time Series per epoch.





9. Conclusion:

In this project, we have taken a dataset and performed various Machine Learning algorithms and Data Visualization algorithms to visually depict the data and also try to achieve high accuracy for our training dataset. The main difficulty with the project is that it is almost impossible to achieve 100% accuracy. But achieving a high accuracy was our main goal. The highest accuracy we got was with the Decision Tree Model which yielded 81.66%.

10. References:

[1] https://www.geeksforgeeks.org/decision-tree-implementation-python/

[2] https://towardsdatascience.com/how-to-impliment-a-gaussian-naive-bayes-classifier-in-python-from-scratch-11e0b80faf5a

[3] Marjia Sultana, Afrin Haider; Mohammad Shorif Uddin Research paper on "Analysis of data mining techniques for heart disease prediction" Published in: 2016 3rd International Conference on Electrical Engineering and Information Communication Technology (ICEEICT)