Content

Problem Statement: -

The Lungs Disease dataset was first used in Machine Learning model with the aim to predict well. The ML model could only produce 80% Accuracy on algorithms like Logistic Regression. The ML model with use case in Healthcare Industry, it should produce the accuracy of more than 90% for better result. To overcome the problem of underperformance we apply the Neural Network to reduce the errors and provide better prediction.

Objective

- To implement the Neural Network Model
- To increase the accuracy of the model.
- To produce the better performance matrix.
- To make a comparative study to understand the reliability of different models.

Introduction

Lung diseases pose significant health risks, and early detection plays a crucial role in effective management and treatment. The project focuses on developing a predictive model to assess the risk of lung disease based on a diverse set of patient attributes. The dataset encompasses various factors, including patient demographics, medical test results, and historical health information. The objective is to create a reliable model that can aid healthcare professionals in identifying individuals at higher risk of developing lung diseases.

The Lungs Disease Prediction Model helps us to predict the risk associated with the parameters provided during the prediction and provide accurate and efficient results.

Artificial Neural Network

Artificial neural networks (ANNs, also shortened to neural networks (NNs) or neural nets) are a branch of machine learning models that are built using principles of neuronal organization discovered by connectionism in the biological neural networks constituting animal brains.

An ANN is based on a collection of connected units or nodes called artificial neurons, which loosely model the neurons in a biological brain. Each connection, like the synapses in a biological brain, can transmit a signal to other neurons. An artificial neuron receives signals then processes them and can signal neurons connected to it. The "signal" at a connection is a real number, and the output of each neuron is computed by some non-linear function of the sum of its inputs. The connections are called edges. Neurons and edges typically have a weight that adjusts as learning proceeds. The weight increases or decreases the strength of the signal at a connection. Neurons may have a threshold such that a signal is sent only if the aggregate signal crosses that threshold. Typically, neurons are aggregated into layers. Different layers may perform different transformations on their inputs. Signals travel from the first layer (the input layer), to the last

layer (the output layer), possibly after traversing the layers multiple times.

A network is typically called a deep neural network if it has at least 2 hidden layers.

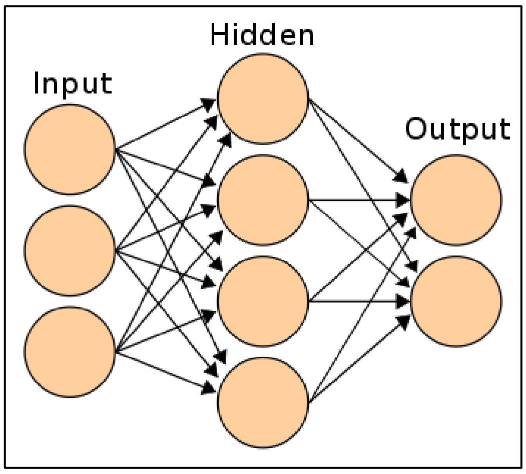


Fig1.1: Structure of ANN

Training

Neural networks are typically trained through empirical risk minimization. This method is based on the idea of optimizing the network's parameters to minimize the difference, or empirical risk, between the predicted output and the actual target values in a given dataset.[4] Gradient based methods such as backpropagation are usually used to estimate the parameters of the network.[4] During the training phase, ANNs learn from labeled training data by iteratively updating their parameters to minimize a defined loss function.[5] This method allows the network to generalize to unseen data.

Artificial neurons

ANNs are composed of artificial neurons which are conceptually derived from biological neurons. Each artificial neuron has inputs and produces a single output which can be sent to multiple other neurons. The inputs can be the feature values of a sample of external data, such as images or documents, or they can be the outputs of other neurons. The outputs of the final output neurons of the neural net accomplish the task, such as recognizing an object in an image.

To find the output of the neuron we take the weighted sum of all the inputs, weighted by the weights of the connections from the inputs to the neuron. We add a bias term to this sum. This weighted sum is sometimes called the activation. This weighted sum is then passed through a (usually nonlinear) activation function to produce the output. The initial inputs are external data, such as images and documents. The ultimate outputs accomplish the task, such as recognizing an object in an image.

Organization

The neurons are typically organized into multiple layers, especially in deep learning. Neurons of one layer connect only to neurons of the immediately preceding and immediately following layers. The layer that receives external data is the input layer. The layer that produces the ultimate result is the output layer. In between them are zero or more hidden layers. Single layer and unlayered networks are also used. Between two layers, multiple connection patterns are possible. They can be 'fully connected', with every neuron in one layer connecting to every neuron in the next layer. They can be pooling, where a group of neurons in one layer connects to a single neuron in the next layer, thereby reducing the number of neurons in that layer. Neurons with only such connections form a directed acyclic graph and are known as feedforward networks. Alternatively, networks that allow connections between neurons in the same or previous layers are known as recurrent networks.

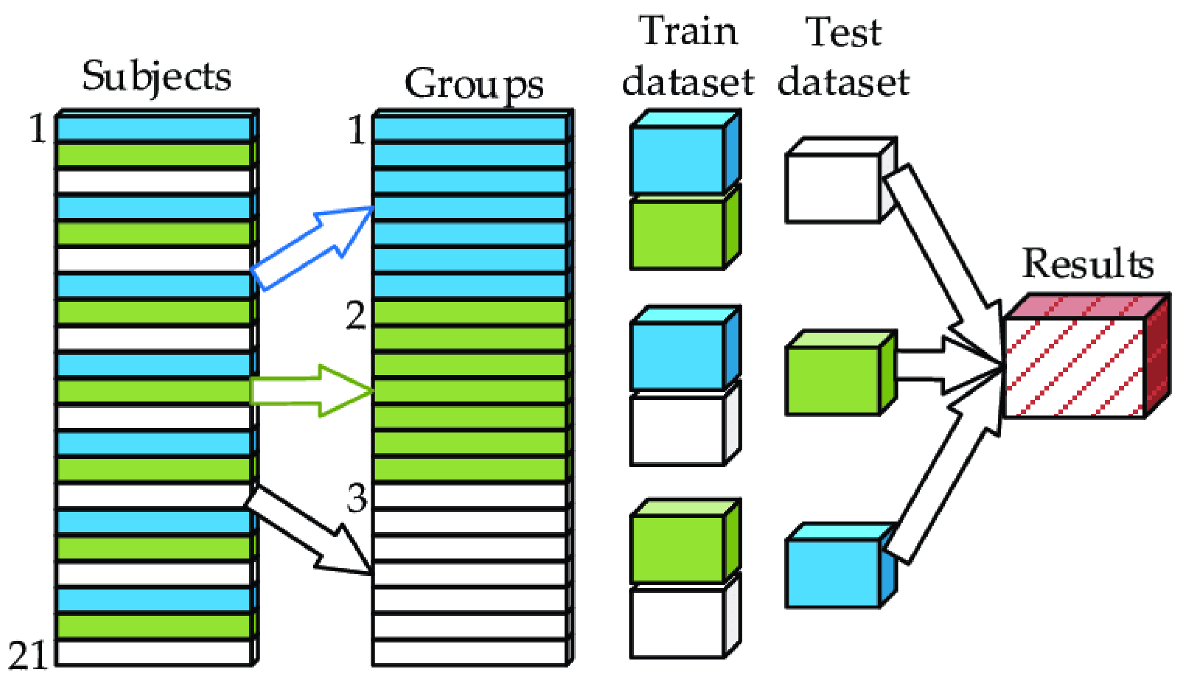


Fig1.2: - Database Organization in ANN

Hyperparameter

A hyperparameter is a constant parameter whose value is set before the learning process begins. The values of parameters are derived via learning. Examples of hyperparameters include learning rate, the number of hidden layers and batch size. The values of some hyperparameters can be dependent on those of other hyperparameters. For example, the size of some layers can depend on the overall number of layers.

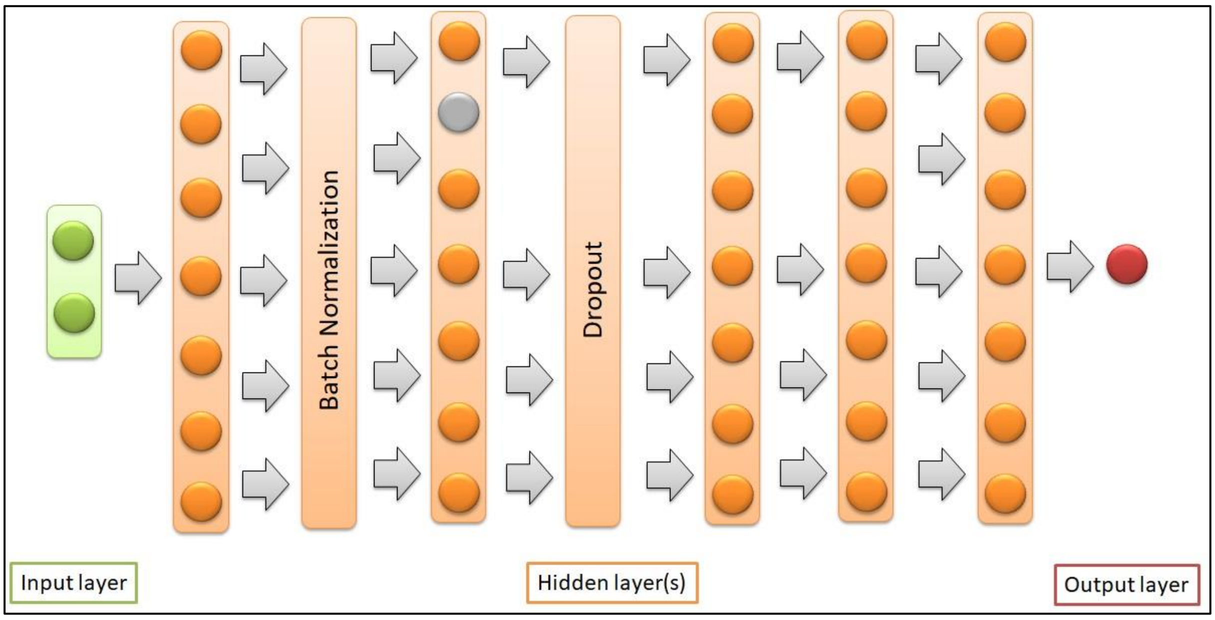


Fig1.3: - Batch Normalization in ANN

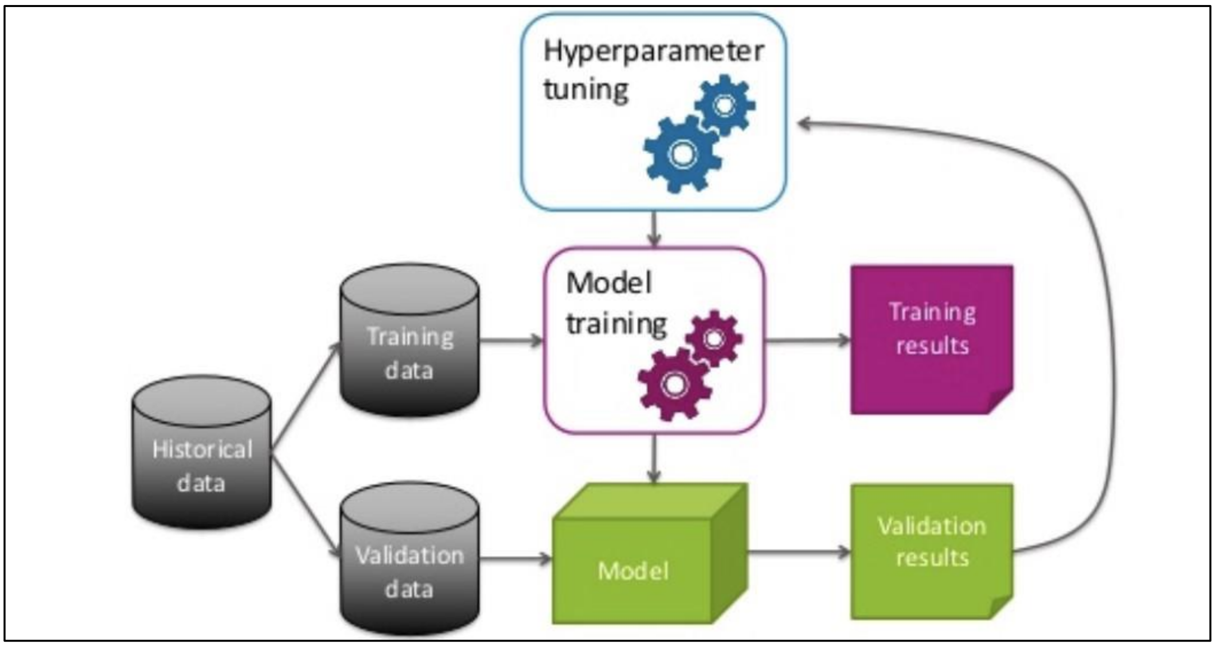


Fig1.4: - Hyperparameter Tuning in ANN

Learning

This section includes a list of references, related reading, or external links, but its sources remain unclear because it lacks inline citations. Please help to improve this section by introducing more precise citations.

Learning is the adaptation of the network to better handle a task by considering sample observations. Learning involves adjusting the weights (and optional thresholds) of the network to improve the accuracy of the result. This is done by minimizing the observed errors. Learning is complete when examining additional observations does not usefully reduce the error rate. Even after learning, the error rate typically does not reach 0. If after learning, the error rate is too high, the network typically must be redesigned. Practically this is done by defining a cost function that is evaluated periodically during learning. As long as its output continues to decline, learning continues. The cost is frequently defined as a statistic whose value can only be approximated. The outputs are actually numbers, so when the error is low, the difference between the output (almost certainly a cat) and the correct answer (cat) is small. Learning attempts to reduce the total of the differences across the observations. Most learning models can be viewed as a straightforward application of optimization theory and statistical estimation.

Learning rate

The learning rate defines the size of the corrective steps that the model takes to adjust for errors in each observation. A high learning rate shortens the training time, but with lower ultimate accuracy, while a lower learning rate takes longer, but with the potential for greater accuracy. Optimizations such as Quickprop are primarily aimed at speeding up error minimization, while other improvements mainly try to increase reliability. In order to avoid oscillation inside the network such as alternating connection weights, and to improve the rate of convergence, refinements use an adaptive learning rate that increases or decreases as appropriate. The concept of momentum allows the balance between the gradient and the previous change to be weighted such that the weight adjustment depends to some degree on the previous change. A momentum close to 0 emphasizes the gradient, while a value close to 1 emphasizes the last change.

Cost function

While it is possible to define a cost function ad hoc, frequently the choice is determined by the function's desirable properties (such as convexity) or because it arises from the model (e.g. in a probabilistic model the model's posterior probability can be used as an inverse cost).

Tools and Dataset

Fig2.1: - Numpy

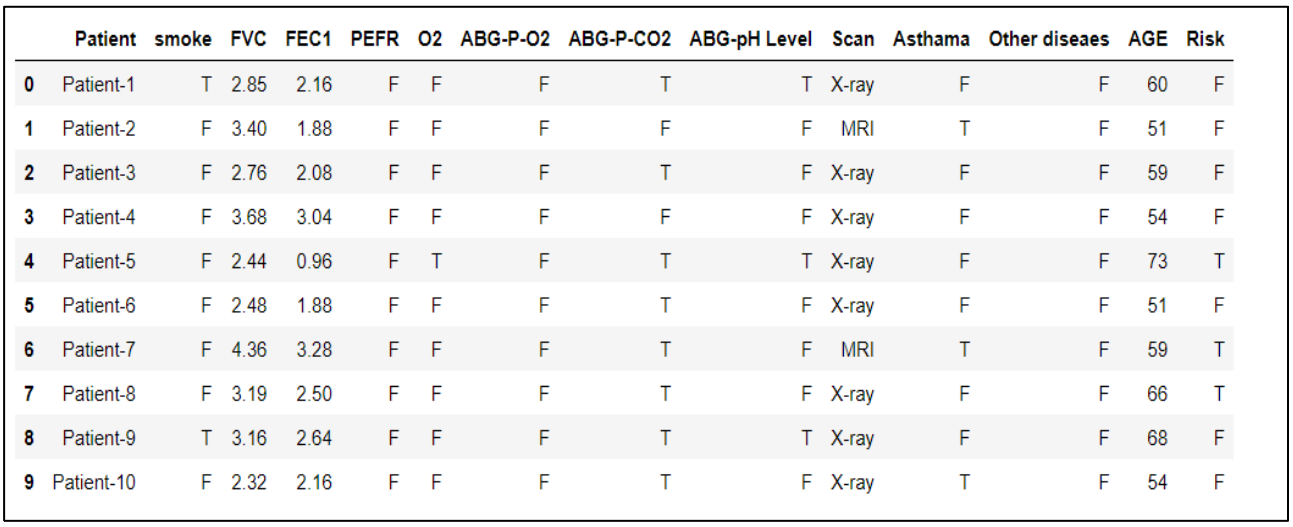
Fig2.2: - Pandas

Fig2.3: - SKlearn

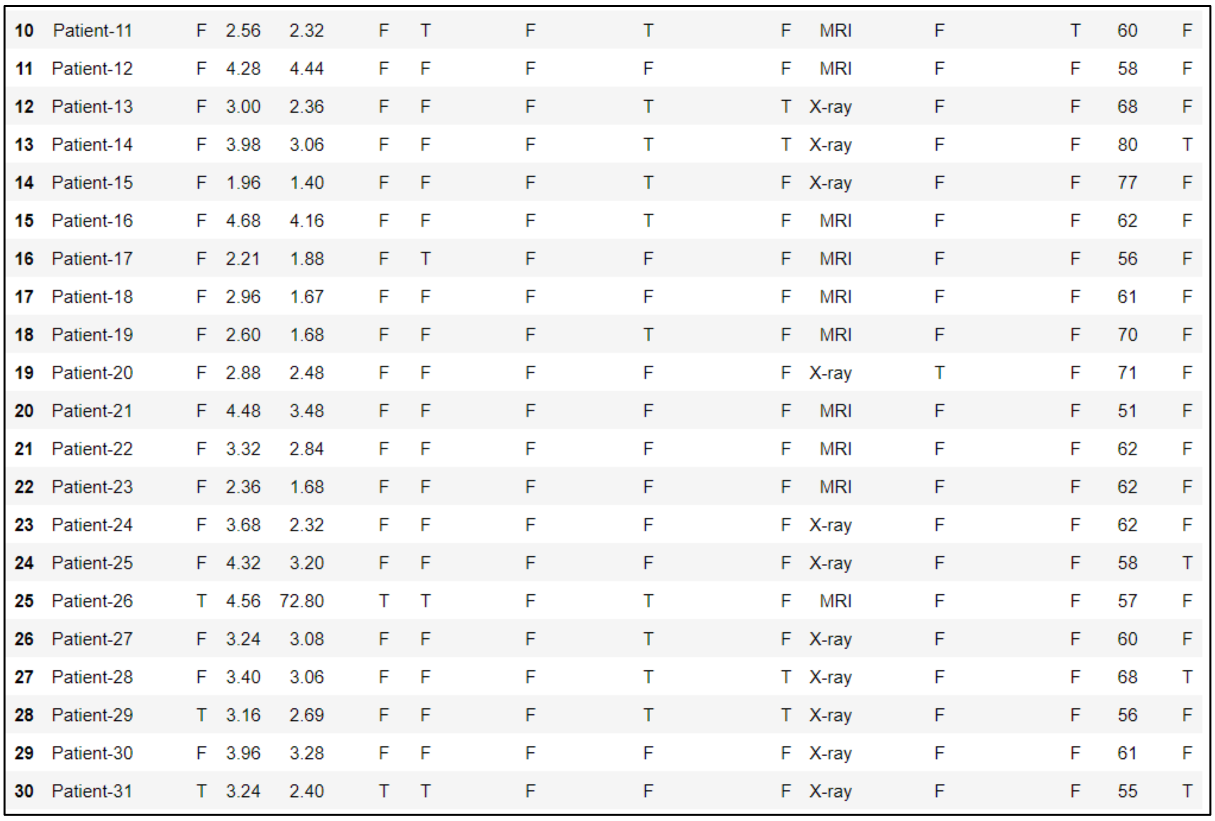
Fig2.4: - Matplotlib

Fig2.5: TensorFlow

Fig2.6: - Keras



	Patient	smoke	FVC	FEC1	PEFR	O2	ABG-P-O2	ABG-P-CO2	ABG-pH Level	Scan	Asthama	Other diseaes	AGE	Risk
0	Patient-1	T	2.85	2.16	F	F	F	T	T	X-ray	F	F	60	F
1	Patient-2	F	3.40	1.88	F	F	F	F	F	MRI	T	F	51	F
2	Patient-3	F	2.76	2.08	F	F	F	T	F	X-ray	F	F	59	F
3	Patient-4	F	3.68	3.04	F	F	F	F	F	X-ray	F	F	54	F
4	Patient-5	F	2.44	0.96	F	T	F	T	T	X-ray	F	F	73	T
5	Patient-6	F	2.48	1.88	F	F	F	T	F	X-ray	F	F	51	F
6	Patient-7	F	4.36	3.28	F	F	F	T	F	MRI	T	F	59	T
7	Patient-8	F	3.19	2.50	F	F	F	T	F	X-ray	F	F	66	T
8	Patient-9	T	3.16	2.64	F	F	F	T	T	X-ray	F	F	68	F
9	Patient-10	F	2.32	2.16	F	F	F	T	F	X-ray	T	F	54	F



10	Patient-11	F	2.56	2.32	F	T	F	T	F	MRI	F	T	60	F
11	Patient-12	F	4.28	4.44	F	F	F	F	F	MRI	F	F	58	F
12	Patient-13	F	3.00	2.36	F	F	F	T	T	X-ray	F	F	68	F
13	Patient-14	F	3.98	3.06	F	F	F	T	T	X-ray	F	F	80	T
14	Patient-15	F	1.96	1.40	F	F	F	T	F	X-ray	F	F	77	F
15	Patient-16	F	4.68	4.16	F	F	F	T	F	MRI	F	F	62	F
16	Patient-17	F	2.21	1.88	F	T	F	F	F	MRI	F	F	56	F
17	Patient-18	F	2.96	1.67	F	F	F	F	F	MRI	F	F	61	F
18	Patient-19	F	2.60	1.68	F	F	F	T	F	MRI	F	F	70	F
19	Patient-20	F	2.88	2.48	F	F	F	F	F	X-ray	T	F	71	F
20	Patient-21	F	4.48	3.48	F	F	F	F	F	MRI	F	F	51	F
21	Patient-22	F	3.32	2.84	F	F	F	F	F	MRI	F	F	62	F
22	Patient-23	F	2.36	1.68	F	F	F	F	F	MRI	F	F	62	F
23	Patient-24	F	3.68	2.32	F	F	F	F	F	X-ray	F	F	62	F
24	Patient-25	F	4.32	3.20	F	F	F	F	F	X-ray	F	F	58	T
25	Patient-26	T	4.56	72.80	T	T	F	T	F	MRI	F	F	57	F
26	Patient-27	F	3.24	3.08	F	F	F	T	F	X-ray	F	F	60	F
27	Patient-28	F	3.40	3.06	F	F	F	T	T	X-ray	F	F	68	T
28	Patient-29	T	3.16	2.69	F	F	F	T	T	X-ray	F	F	56	F
29	Patient-30	F	3.96	3.28	F	F	F	F	F	X-ray	F	F	61	F
30	Patient-31	T	3.24	2.40	T	T	F	F	F	X-ray	F	F	55	T

Fig2.7: - Dataset for Proposed

Features in Dataset

Patient:

Description: An identifier for each patient.

Type: Categorical or nominal.

smoke:

Description: Smoking status.

Type: Categorical (Binary - T for True/Yes, F for False/No).

Interpretation: Indicates whether the patient is a smoker (True/Yes) or a non-smoker (False/No).

FVC, FEC1, PEFR:

Description: Lung function metrics.

Type: Continuous (Numeric).

Interpretation: These metrics represent different aspects of lung function. FVC (Forced Vital Capacity), FEC1 (Forced Expiratory Capacity in 1 second), and PEFR (Peak Expiratory Flow Rate) are commonly used to assess lung health.

O2, ABG-P-O2, ABG-P-CO2, ABG-pH Level:

Description: Blood gas levels and pH.

Type: Categorical or binary (T for True/Yes, F for False/No).

Interpretation: Indicates the presence (True/Yes) or absence (False/No) of certain conditions related to blood gas levels and pH.

Scan:

Description: Type of imaging scan.

Type: Categorical or nominal.

Interpretation: Specifies the type of imaging scan performed on the patient, such as X-ray, MRI (Magnetic Resonance Imaging), or CT (Computed Tomography).

Asthama:

Description: Asthma status.

Type: Categorical (Binary - F for False/No).

Interpretation: Indicates whether the patient has asthma (False/No) or not.

Other diseases:

Description: Status of other diseases.

Type: Categorical (Binary - F for False/No).

Interpretation: Indicates whether the patient has other diseases (False/No) or not.

AGE:

Description: Age of the patient.

Type: Continuous (Numeric).

Interpretation: Represents the age of the patient in years.

Risk:

Description: Risk status.

Type: Categorical (Binary - F for False/No, T for True/Yes).

Interpretation: Indicates whether the patient is at risk (True/Yes) or not at risk (False/No) based on some criteria.

Correlation Matrix: -

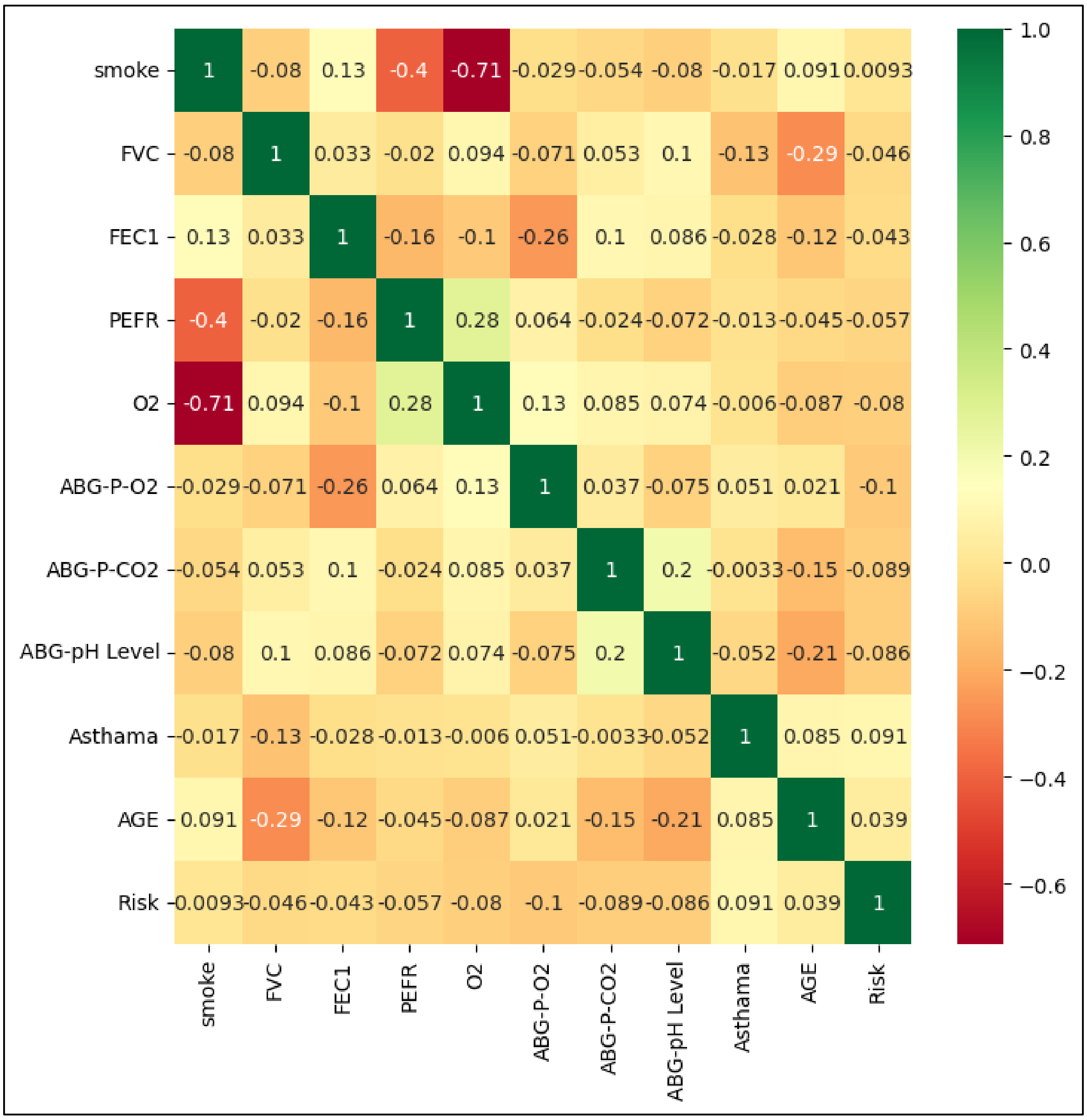


Fig2.8:- Correlation Matrix of the Features in dataset: Smoke,FVC,FEC1,PEFR,O2,ABG-P-O2,ABG-P-CO2,ABG-pH Level,Scan,Asthama,Other diseaes,AGE,Risk

Methodology

Feedforward Neural Network

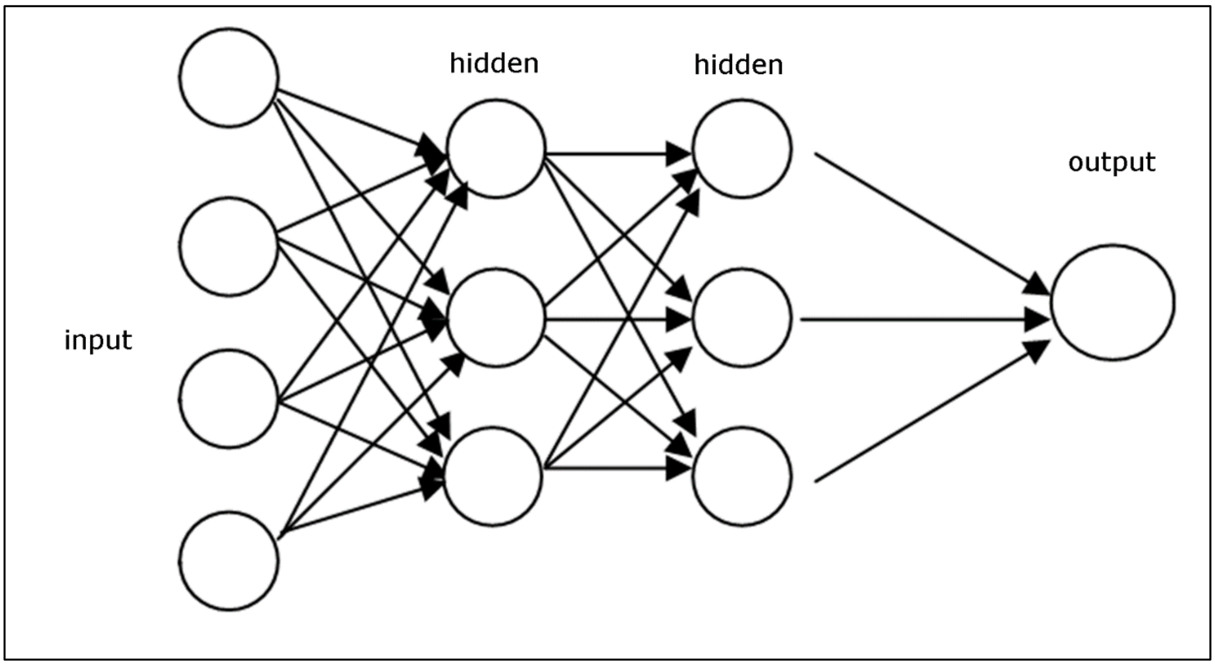


Fig3.1: - Layers in an ANN Structure

A feedforward neural network is one of the simplest types of artificial neural networks devised. In this network, the information moves in only one direction—forward—from the input nodes, through the hidden nodes (if any), and to the output nodes. There are no cycles or loops in the network. Feedforward neural networks were the first type of artificial neural network invented and are simpler than their counterparts like recurrent neural networks and convolutional neural networks.

A Feedforward Neural Network (FNN) is a type of artificial neural network where information flows in one direction, from the input layer through one or more hidden layers to the output layer. It is the simplest form of neural network architecture and is also known as a multilayer perceptron (MLP). In a feedforward neural network, there are no cycles or loops in the connections between nodes.

Here are the key components and characteristics of a feedforward neural network:

1. Basic Components:

Input Layer: Neurons in this layer represent the features or attributes of the input data.

Hidden Layers: Layers between the input and output layers where the network learns to extract and represent complex features.

Output Layer: Neurons in this layer produce the final output of the network.

2. Connection and Weights:

Each connection between neurons has an associated weight, representing the strength of the connection.

The weights are learned during the training process, where the network adjusts them to minimize the difference between predicted and actual outputs.

3. Activation Function:

Each neuron in the network (except for the input layer) has an activation function that determines its output based on the weighted sum of its inputs.

4. Feedforward Process:

During the feedforward process, input data is passed through the network layer by layer, with each layer transforming the information.

The output of one layer serves as the input to the next layer until the final output is obtained.

5. Training (Learning):

Training involves adjusting the weights of the connections based on the error between the predicted output and the actual output.

Backpropagation is the primary algorithm used for training feedforward neural networks, where the error is propagated backward through the network to update the weights.

6. Activation Functions:

Common activation functions include the sigmoid function, hyperbolic tangent (tanh), and rectified linear unit (ReLU).

ReLU Function

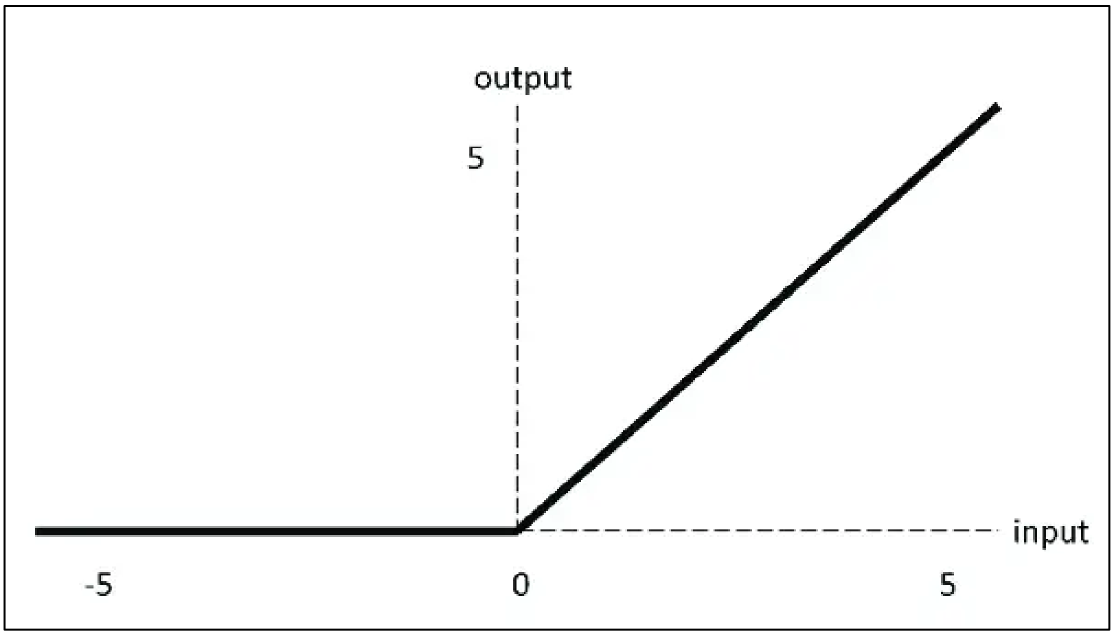


Fig3.2:- ReLU Function Graph

The ReLU function is another non-linear activation function that has gained popularity in the deep learning domain. ReLU stands for Rectified Linear Unit. The main advantage of using the ReLU function over other activation functions is that it does not activate all the neurons at the same time.

This means that the neurons will only be deactivated if the output of the linear transformation is less than 0. The plot below will help you understand this better-

$f(x)=max(0,x)$

Key Characteristics of ReLU:

Non-linearity:

ReLU introduces non-linearity to the network, which is crucial for the network to learn complex patterns and relationships in the data. Linear activation functions would result in a linear model, making it less effective for capturing intricate patterns.

Computational Efficiency:

ReLU is computationally efficient compared to other activation functions like sigmoid or hyperbolic tangent (tanh). This is because it only requires simple thresholding, making it faster to compute during both forward and backward passes.

Sparsity:

ReLU can also introduce sparsity in the network, as it outputs zero for negative values. Sparsity can be beneficial in some cases, as it helps reduce the risk of overfitting by preventing unnecessary computations.

Vanishing Gradient:

While ReLU has advantages, it can also suffer from the "dying ReLU" problem, where neurons can become inactive (always output zero) during training. If a large gradient flows through a ReLU neuron and updates its weights in a way that makes it always output negative values, the neuron will remain inactive. Techniques like leaky ReLU or parametric ReLU have been proposed to mitigate this issue.

Sigmoid Function

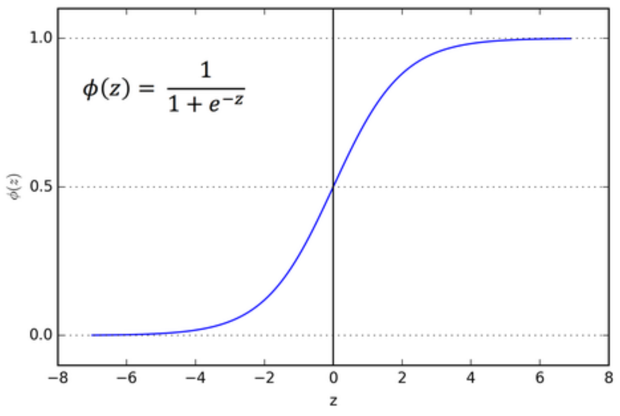


Fig3.3: - Sigmoid Function Graph

The sigmoid function, also known as the logistic function, is a common activation function used in artificial neural networks. It is especially prevalent in the output layer of binary classification models, where it maps any real-valued number to the range [0, 1]. The sigmoid function is defined as:

$$\sigma(x) = \frac{1}{1+e^{-x}}$$

Here,

e is the base of the natural logarithm (approximately 2.71828), and

x is the input to the function. The sigmoid function has an S-shaped curve, and its output ranges from 0 to 1, making it useful for models that need to produce probabilities.

Key Characteristics of the Sigmoid Function:

Output Range:

The sigmoid function squashes its input to the range [0, 1]. This is particularly useful in binary classification tasks, where the output can be interpreted as the probability of belonging to the positive class.

Smooth Gradient:

The sigmoid function has a smooth gradient, which facilitates gradient-based optimization methods like backpropagation during the training of neural networks.

Monotonicity:

The sigmoid function is monotonic, meaning that it is always increasing or always decreasing. This property can simplify the optimization process in certain scenarios.

Use Cases of the Sigmoid Function:

Binary Classification:

In binary classification problems, the sigmoid function is often used in the output layer to produce probabilities that can be thresholded to make a decision (e.g., if the output is greater than 0.5, classify as class 1; otherwise, classify as class 0).

Logistic Regression:

The sigmoid function is the activation function used in logistic regression, where it models the probability of a binary outcome.

Framework: -

TensorFlow

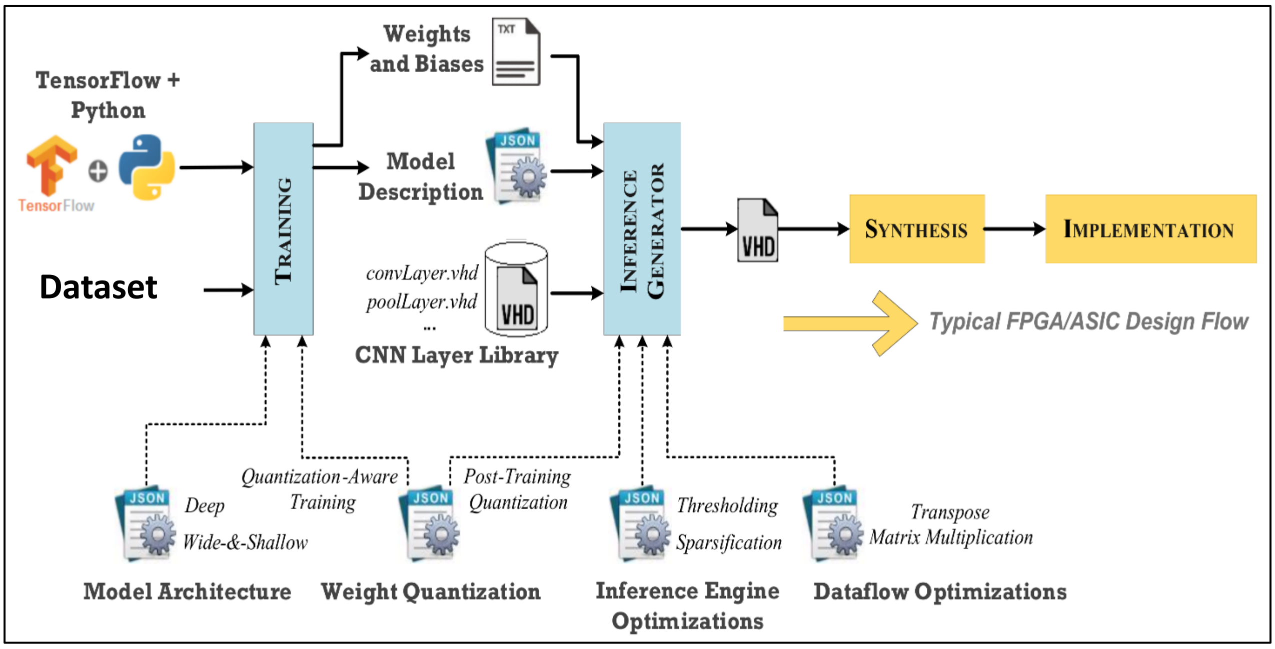


Fig:- TensorFlow Flowchart

TensorFlow allows developers to create dataflow graphs—structures that describe how data moves through a graph, or a series of processing nodes. Each node in the graph represents a mathematical operation, and each connection or edge between nodes is a multidimensional data array, or tensor.

TensorFlow applications can be run on most any target that's convenient: a local machine, a cluster in the cloud, iOS and Android devices, CPUs or GPUs. If you use Google's own cloud, you can run TensorFlow on Google's custom TensorFlow Processing Unit (TPU) silicon for further acceleration. Models created by TensorFlow can be deployed on most any device to serve predictions.

Keras

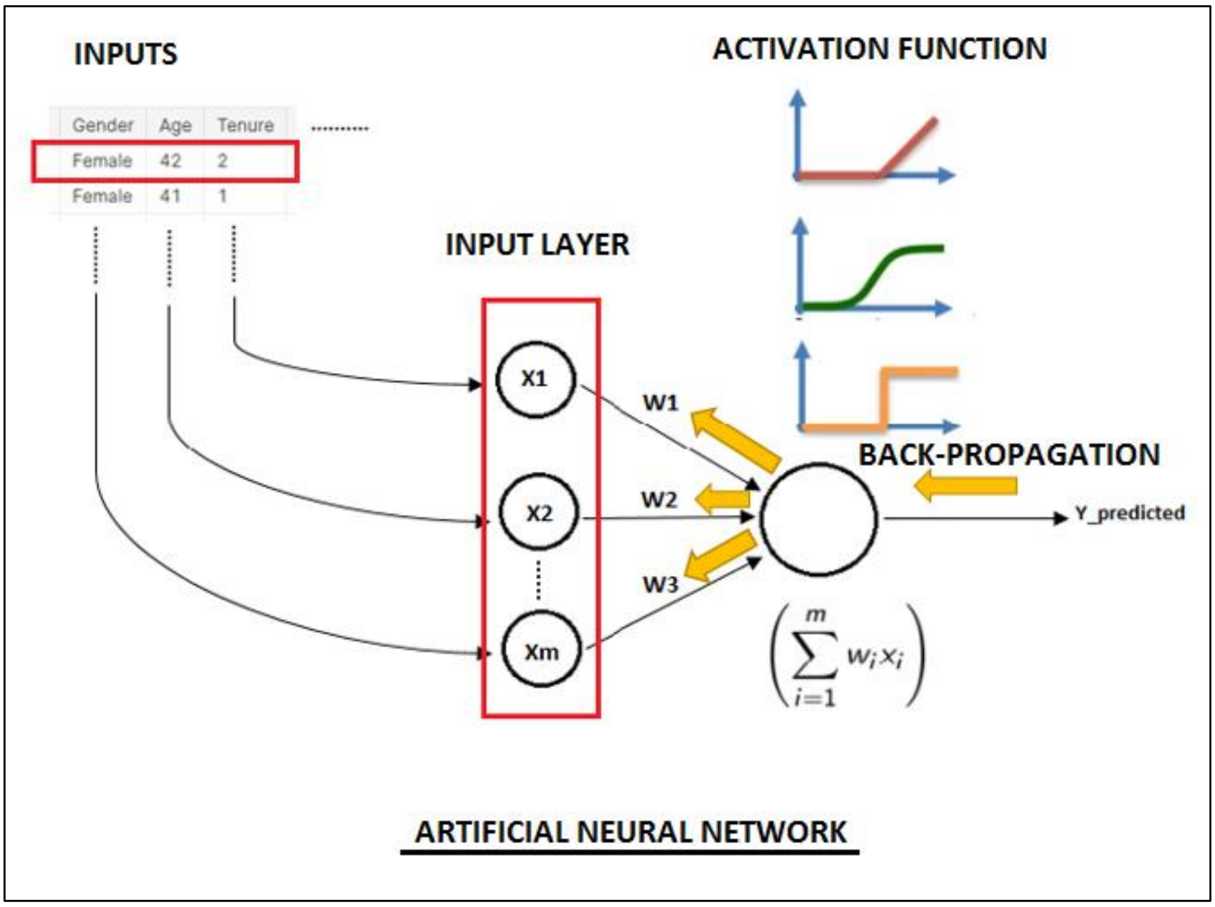
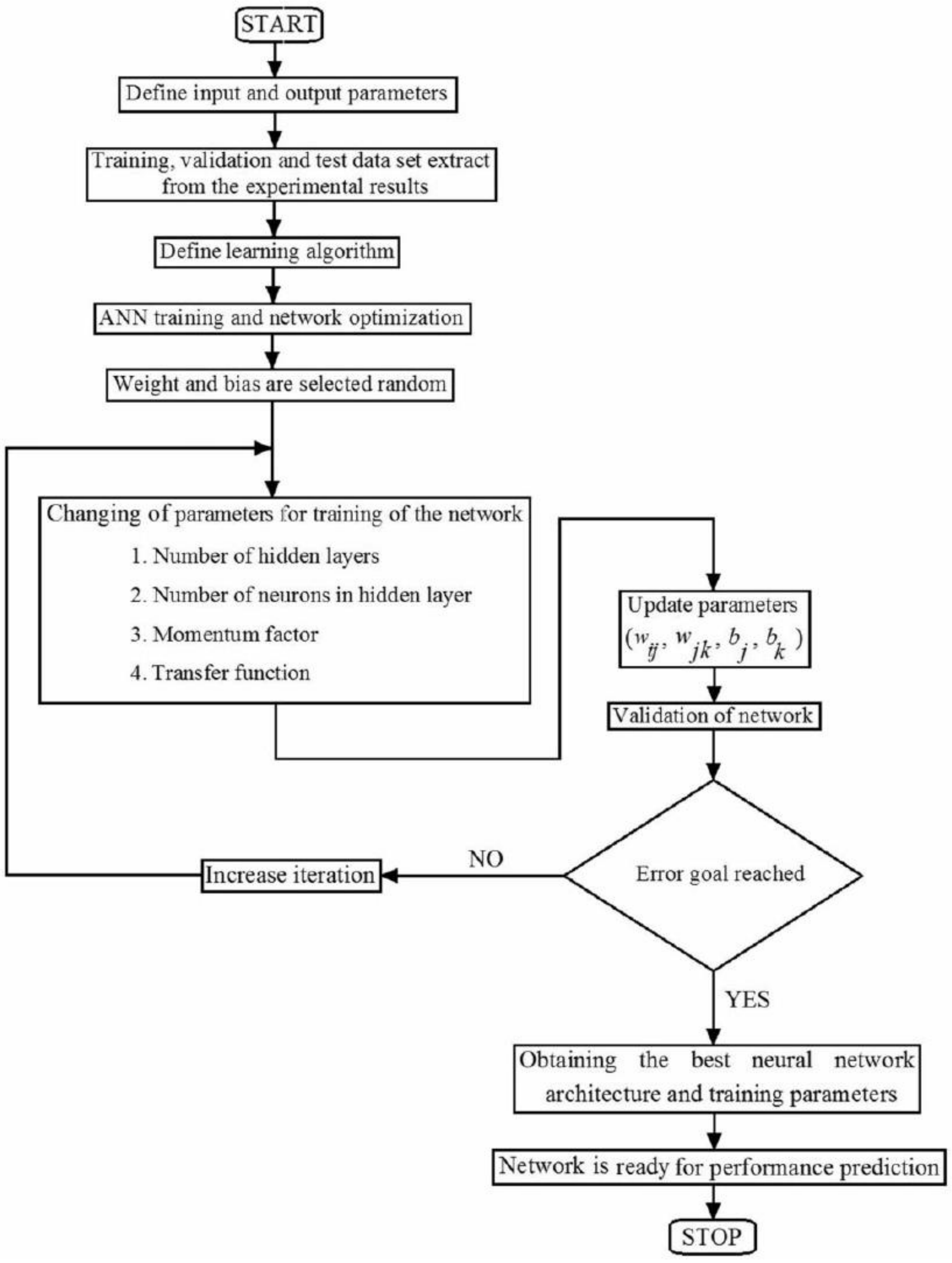


Fig: - Keras Flowchart

Architecture

C


START
Define input and output parameters
Training, validation and test data set extract from the experimental results
Define learning algorithm
ANN training and network optimization
Weight and bias are selected random
Changing of parameters for training of the network
1. Number of hidden layers
2. Number of neurons in hidden layer
3. Momentum factor
4. Transfer function
Update parameters (w_{ij}, w_{jk}, b_j, b_k)
Validation of network
Error goal reached
NO
Increase iteration
YES
Obtaining the best neural network architecture and training parameters
Network is ready for performance prediction
STOP

Project Code

```
import numpy as np

import pandas as pd

import tensorflow as tf

from tensorflow.keras import Sequential

from tensorflow.keras.layers import Dense
```

File Import

```
df=pd.read_csv('lung_disease.csv')

df.head()

df['smoke'].replace({'T': 1, 'F': 0},inplace=True)

df['PEFR'].replace({'F': 1, 'T': 0},inplace=True)

df['O2'].replace({'F': 1, 'T': 0},inplace=True)

df['PEFR'].replace({'F': 1, 'T': 0},inplace=True)

df[ 'ABG-P-O2'].replace({'F': 1, 'T': 0},inplace=True)

df[ 'ABG-P-CO2'].replace({'F': 1, 'T': 0},inplace=True)

df[ 'ABG-pH Level'].replace({'F': 1, 'T': 0},inplace=True)

df[ 'Scan'].replace({'MRI':1, 'X-ray':2 ,'CT':3},inplace=True)

df[ 'Asthama'].replace({'F': 0, 'T':1 },inplace=True)

df[ 'Other diseaes'].replace({'F': 0, 'T':1 },inplace=True)

df[ 'Risk'].replace({'F':0, 'T':1 },inplace=True)

df.head()

#Input Data

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

# Output

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

Training Data

```
from sklearn.model_selection import train_test_split
```

```
xtrain,xtest,ytrain,ytest=train_test_split(x,y,test_size=0.3,random_state=1)
from sklearn.preprocessing import StandardScaler
sc=StandardScaler()
xtrain=sc.fit_transform(xtrain)
xtest=sc.transform(xtest)
xtrain
```

Training Neural Network

```
from tensorflow.keras import layers, models
#initialize model
ann=Sequential()
#add layers
ann.add(Dense(units=13,activation="relu"))#Input Layer
ann.add(Dense(units=5,activation="relu")) #hidden layer
ann.add(Dense(units=1,activation="sigmoid"))#output Layer
ann.compile(optimizer="adam",loss="binary_crossentropy")
#compile
ann.compile(optimizer="adam",loss="binary_crossentropy",metrics=["accuracy"])
#train
ann.fit(xtrain,ytrain,batch_size=30,epochs=350)
ann.history.history
```

Loss vs Accuracy Graph

```
lossgraph=pd.DataFrame(ann.history.history)
lossgraph.plot()

ypred=ann.predict(xtest)
ypred
ypred=ann.predict(xtest)
ypred=ypred>0.5
from sklearn.metrics import classification_report
print(classification_report(ytest,ypred))
```

Regularization

Early Stopping

```
from tensorflow.keras.callbacks import EarlyStopping

earlystop=EarlyStopping(monitor="val_loss",mode="min",verbose=1,patience=250)
#train the model
ann.fit(xtrain,ytrain,epochs=350,validation_data=(xtest,ytest),callbacks=[earlystop])
ES=pd.DataFrame(ann.history.history)
ES.plot()
from tensorflow.keras.layers import Dropout
ann=Sequential()
ann.add(Dense(units=13,activation="relu"))
ann.add(Dropout(rate=0.5))
ann.add(Dense(units=5,activation="relu"))
ann.add(Dropout(rate=0.5))
ann.add(Dense(units=5,activation="relu"))
ann.add(Dropout(rate=0.5))
ann.add(Dense(units=1,activation='sigmoid'))
ann.compile(optimizer="adam",loss="binary_crossentropy")

earlystop=EarlyStopping(monitor="val_loss",mode="min",verbose=1,patience=25)
#train the model
ann.fit(xtrain,ytrain,epochs=350,validation_data=(xtest,ytest),callbacks=[earlystop])
lossdf=pd.DataFrame(ann.history.history)
lossdf.plot()
```

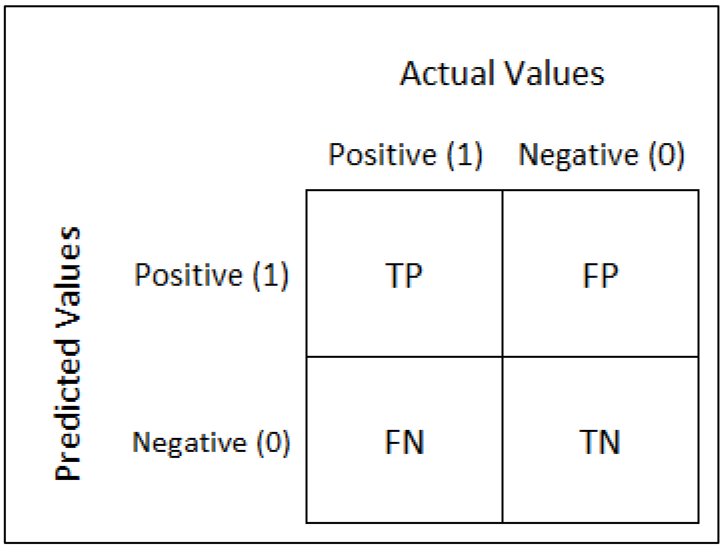
Predicting New Unknown Value

```
#unknown data prediction
newdata=[[1,2.85,2.16,1,1,1,0,0,2,0,0,60]]
newdata=sc.transform(newdata)
ann.predict(newdata)
```

Making Model from the Neural Network

```
from tensorflow.keras.models import load_model

ann.save("Lungs Disease Model")

model=load_model("Lungs Disease")
```

Performance Parameter



		Actual Values	
		Positive (1)	Negative (0)
Predicted Values	Positive (1)	TP	FP
	Negative (0)	FN	TN

Precision

$$Precision = \frac{True\ Positive(TP)}{True\ Positive(TP) + False\ Positive(FP)}$$

Recall

$$Recall = \frac{True\ Positive(TP)}{True\ Positive(TP) + False\ Negative(FN)}$$

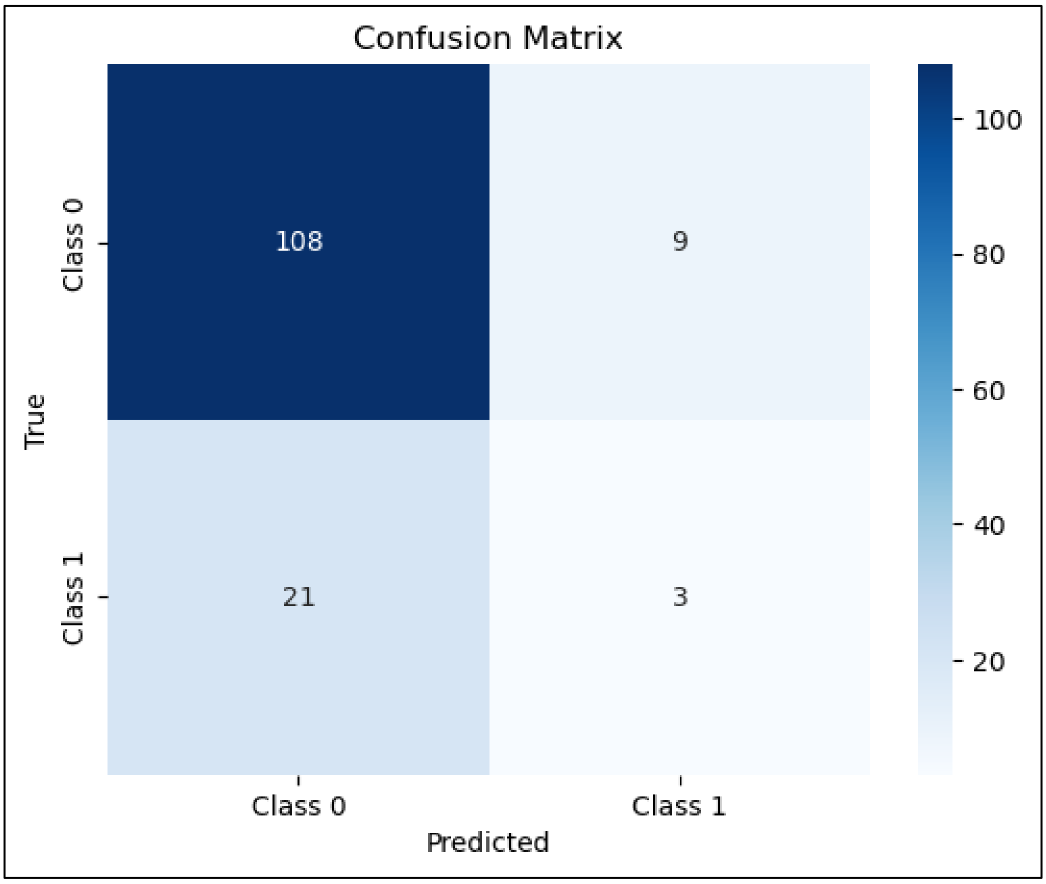
Accuracy

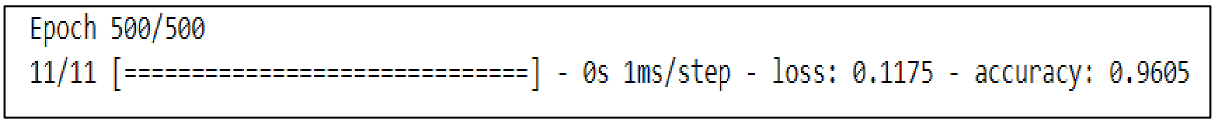
$$\text{Accuracy} = \frac{\text{True Positive} + \text{True Negative}}{(\text{True Positive} + \text{False Positive} + \text{True Negative} + \text{False Negative})}$$

F1 Score

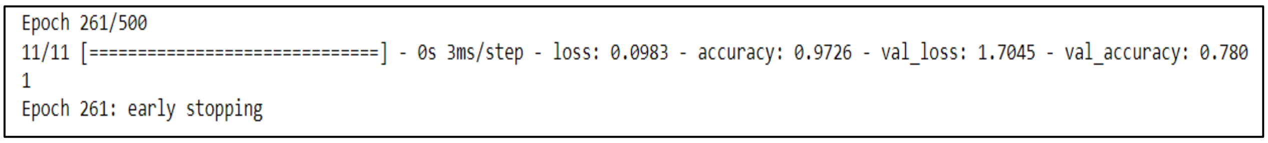
$$F1\ Score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

ANN Model Performance: -



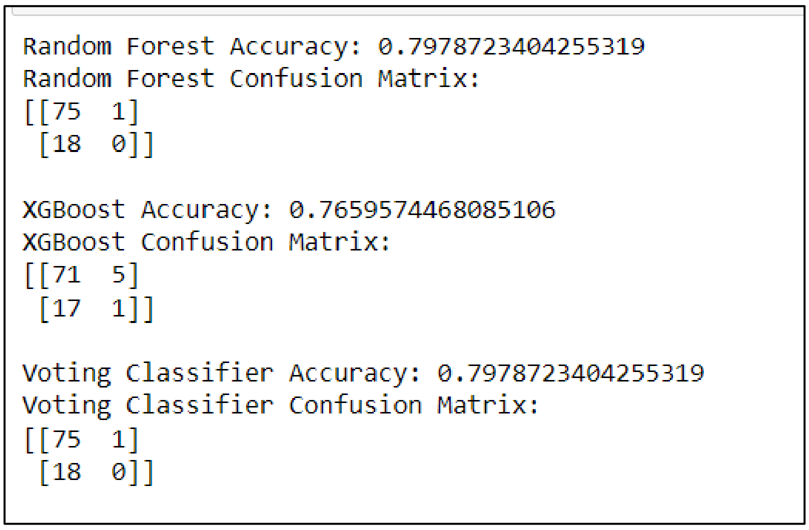


```
Epoch 500/500
11/11 [==============================] - 0s 1ms/step - loss: 0.1175 - accuracy: 0.9605
```



```
Epoch 261/500
11/11 [==============================] - 0s 3ms/step - loss: 0.0983 - accuracy: 0.9726 - val_loss: 1.7045 - val_accuracy: 0.780
1
Epoch 261: early stopping
```

Machine Learning Models Performance: -



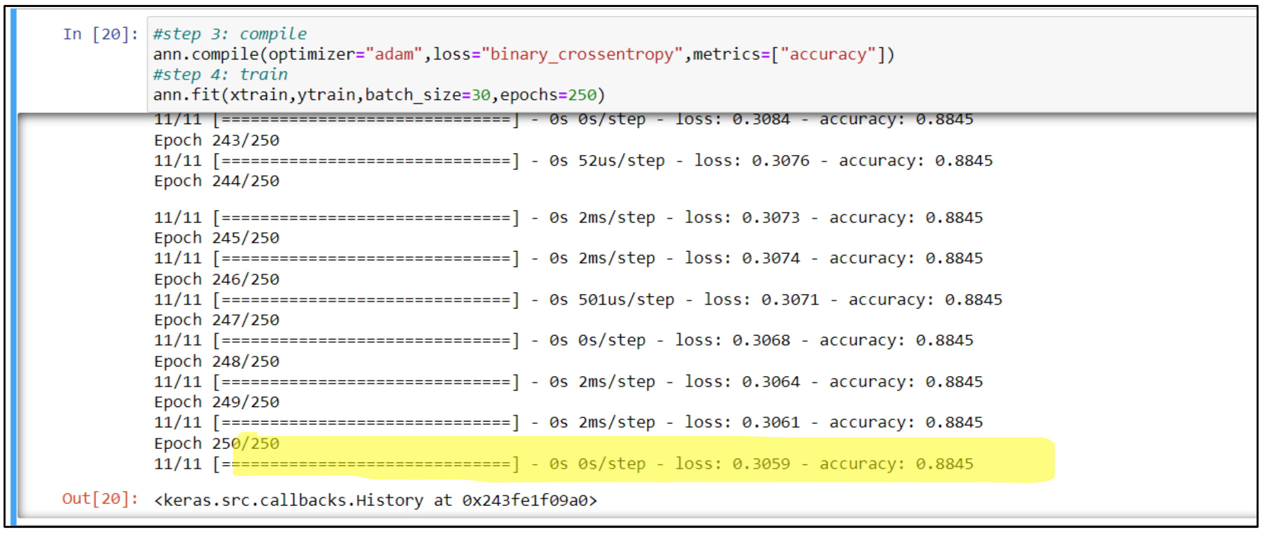
```
Random Forest Accuracy: 0.7978723404255319
Random Forest Confusion Matrix:
[[75  1]
 [18  0]]

XGBoost Accuracy: 0.7659574468085106
XGBoost Confusion Matrix:
[[71  5]
 [17  1]]

Voting Classifier Accuracy: 0.7978723404255319
Voting Classifier Confusion Matrix:
[[75  1]
 [18  0]]
```

Model Comparison

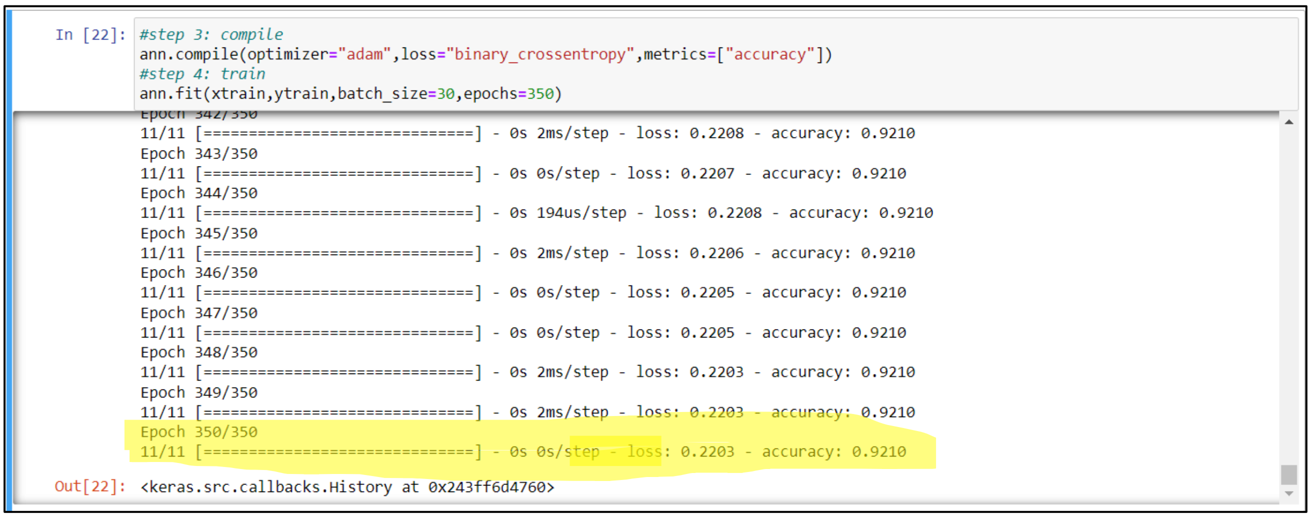
Model	XGBoost	ANN
Accuracy	76%	78%
Precision (Positive Predictive Value)	16%	25%
Recall True Positive Rate)	0.5%	12.5%
Specificity True Negative Rate	93%	92%
F1 Score	0.8%	16%

Conclusion: -

1. **The Accuracy of the Neural Network depends on the No of Iterations.**



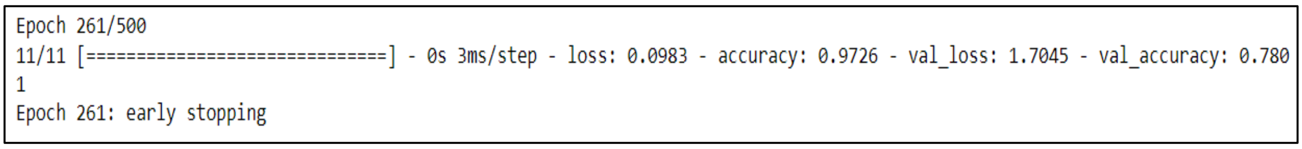
```
In [20]: #step 3: compile
         ann.compile(optimizer="adam",loss="binary_crossentropy",metrics=["accuracy"])
         #step 4: train
         ann.fit(xtrain,ytrain,batch_size=30,epochs=250)

         11/11 [==============================] - 0s 0s/step - loss: 0.3084 - accuracy: 0.8845
         Epoch 243/250
         11/11 [==============================] - 0s 52us/step - loss: 0.3076 - accuracy: 0.8845
         Epoch 244/250

         11/11 [==============================] - 0s 2ms/step - loss: 0.3073 - accuracy: 0.8845
         Epoch 245/250
         11/11 [==============================] - 0s 2ms/step - loss: 0.3074 - accuracy: 0.8845
         Epoch 246/250
         11/11 [==============================] - 0s 501us/step - loss: 0.3071 - accuracy: 0.8845
         Epoch 247/250
         11/11 [==============================] - 0s 0s/step - loss: 0.3068 - accuracy: 0.8845
         Epoch 248/250
         11/11 [==============================] - 0s 2ms/step - loss: 0.3064 - accuracy: 0.8845
         Epoch 249/250
         11/11 [==============================] - 0s 2ms/step - loss: 0.3061 - accuracy: 0.8845
         Epoch 250/250
         11/11 [==============================] - 0s 0s/step - loss: 0.3059 - accuracy: 0.8845

Out[20]: <keras.src.callbacks.History at 0x243fe1f09a0>
```



```
In [22]: #step 3: compile
         ann.compile(optimizer="adam",loss="binary_crossentropy",metrics=["accuracy"])
         #step 4: train
         ann.fit(xtrain,ytrain,batch_size=30,epochs=350)

         Epoch 342/350
         11/11 [==============================] - 0s 2ms/step - loss: 0.2208 - accuracy: 0.9210
         Epoch 343/350
         11/11 [==============================] - 0s 0s/step - loss: 0.2207 - accuracy: 0.9210
         Epoch 344/350
         11/11 [==============================] - 0s 194us/step - loss: 0.2208 - accuracy: 0.9210
         Epoch 345/350
         11/11 [==============================] - 0s 2ms/step - loss: 0.2206 - accuracy: 0.9210
         Epoch 346/350
         11/11 [==============================] - 0s 0s/step - loss: 0.2205 - accuracy: 0.9210
         Epoch 347/350
         11/11 [==============================] - 0s 0s/step - loss: 0.2205 - accuracy: 0.9210
         Epoch 348/350
         11/11 [==============================] - 0s 2ms/step - loss: 0.2203 - accuracy: 0.9210
         Epoch 349/350
         11/11 [==============================] - 0s 2ms/step - loss: 0.2203 - accuracy: 0.9210
         Epoch 350/350
         11/11 [==============================] - 0s 0s/step - loss: 0.2203 - accuracy: 0.9210

Out[22]: <keras.src.callbacks.History at 0x243ff6d4760>
```

2. The Early Stopping Function

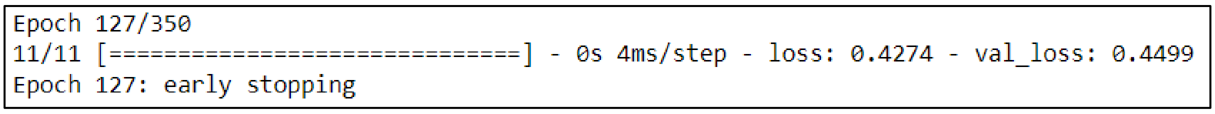
early stopping is a form of regularization used to avoid overfitting when training a learner with an iterative method, such as gradient descent. Such methods update the learner so as to make it better fit the training data with each iterate



```
Epoch 261/500
11/11 [==============================] - 0s 3ms/step - loss: 0.0983 - accuracy: 0.9726 - val_loss: 1.7045 - val_accuracy: 0.780
1
Epoch 261: early stopping
```

3. ## DropOut Function

Dropout is a regularization technique used in deep learning models to prevent overfitting. It's a method that simulates the concurrent training of multiple neural networks with different designs.

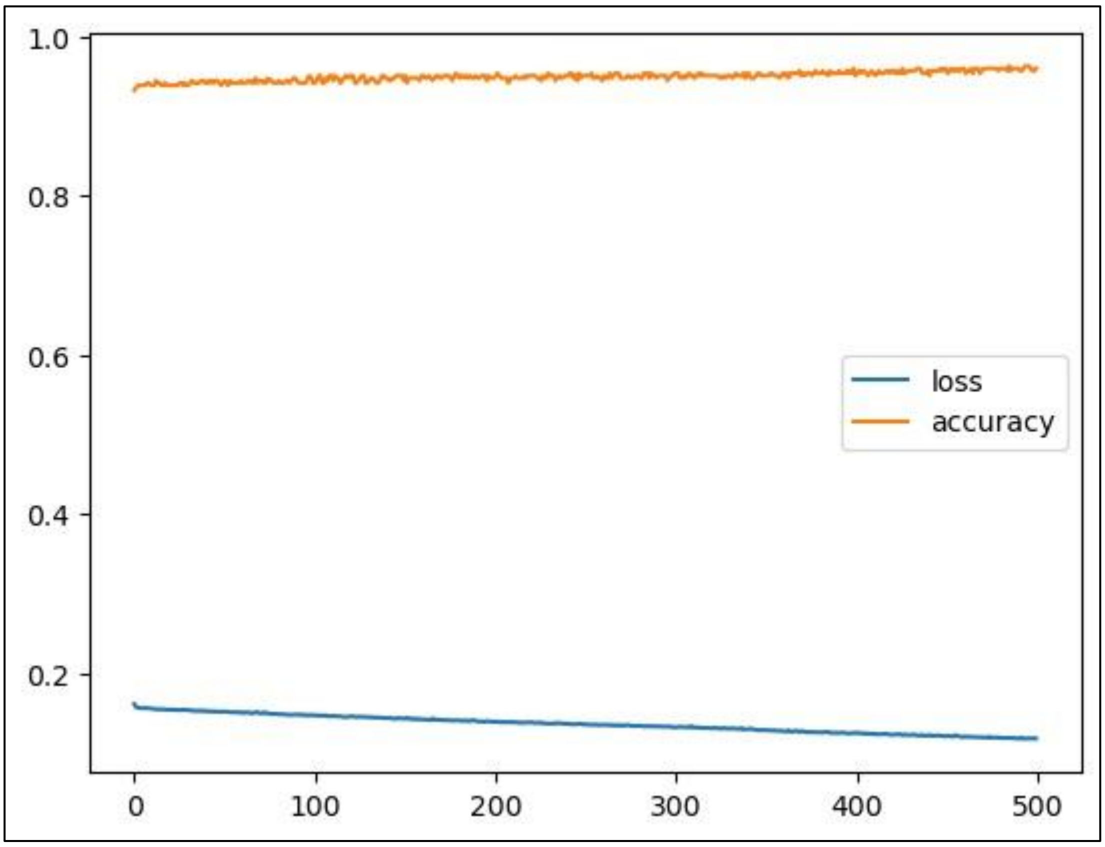


```
Epoch 127/350
11/11 [==============================] - 0s 4ms/step - loss: 0.4274 - val_loss: 0.4499
Epoch 127: early stopping
```

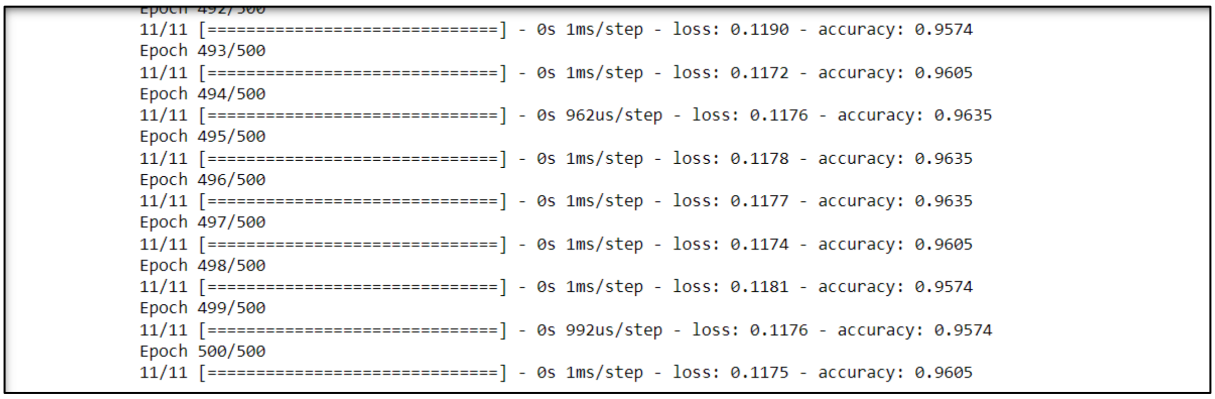
4. 108 instances were correctly classified as negative (True Negatives)
5. 9 instances were incorrectly classified as positive when they were actually negative (False Positives).
6. 21 instances were incorrectly classified as negative when they were actually positive (False Negatives).
7. 3 instances were correctly classified as positive (True Positives).

Results: -

1. Loss Vs Accuracy Graph



2. The accuracy of the Neural Network with single Hidden Layer and epoch=500 is 96%



```
Epoch 492/500
11/11 [==============================] - 0s 1ms/step - loss: 0.1190 - accuracy: 0.9574
Epoch 493/500
11/11 [==============================] - 0s 1ms/step - loss: 0.1172 - accuracy: 0.9605
Epoch 494/500
11/11 [==============================] - 0s 962us/step - loss: 0.1176 - accuracy: 0.9635
Epoch 495/500
11/11 [==============================] - 0s 1ms/step - loss: 0.1178 - accuracy: 0.9635
Epoch 496/500
11/11 [==============================] - 0s 1ms/step - loss: 0.1177 - accuracy: 0.9635
Epoch 497/500
11/11 [==============================] - 0s 1ms/step - loss: 0.1174 - accuracy: 0.9605
Epoch 498/500
11/11 [==============================] - 0s 1ms/step - loss: 0.1181 - accuracy: 0.9574
Epoch 499/500
11/11 [==============================] - 0s 992us/step - loss: 0.1176 - accuracy: 0.9574
Epoch 500/500
11/11 [==============================] - 0s 1ms/step - loss: 0.1175 - accuracy: 0.9605
```

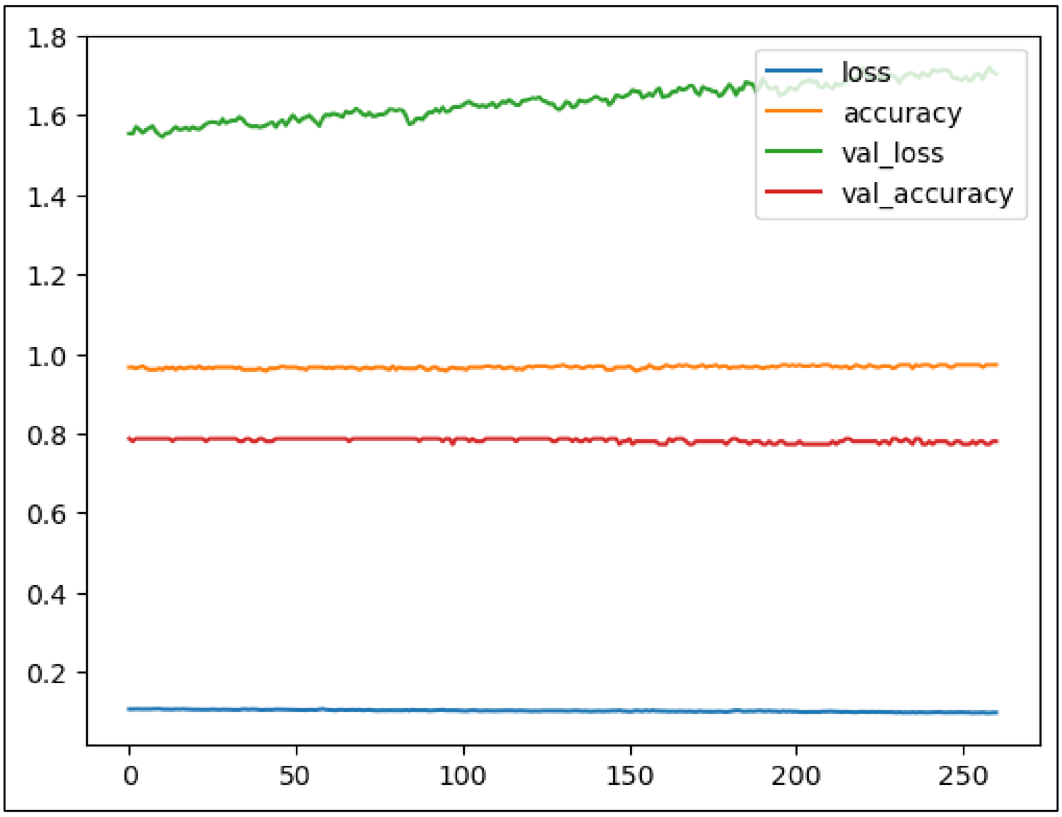
3. Classification Report of Neural Network

```
              precision    recall  f1-score   support

           0       0.83      0.94      0.88       117
           1       0.22      0.08      0.12        24

    accuracy                           0.79       141
   macro avg       0.53      0.51      0.50       141
weighted avg       0.73      0.79      0.75       141
```

4. Early Stopping

```
Epoch 261/500
11/11 [==============================] - 0s 3ms/step - loss: 0.0983 - accuracy: 0.9726 - val_loss: 1.7045 - val_accuracy: 0.7
801
Epoch 261: early stopping
```

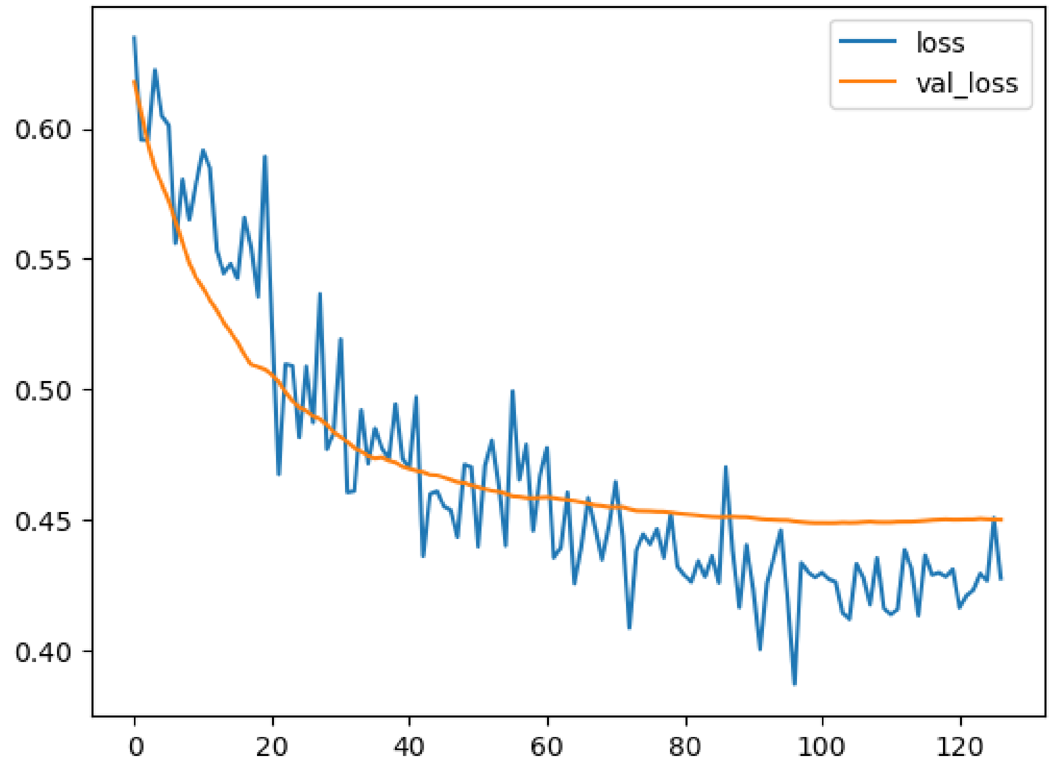
5. Test Loss vs Test Accuracy Graph



6. DropOut Rate

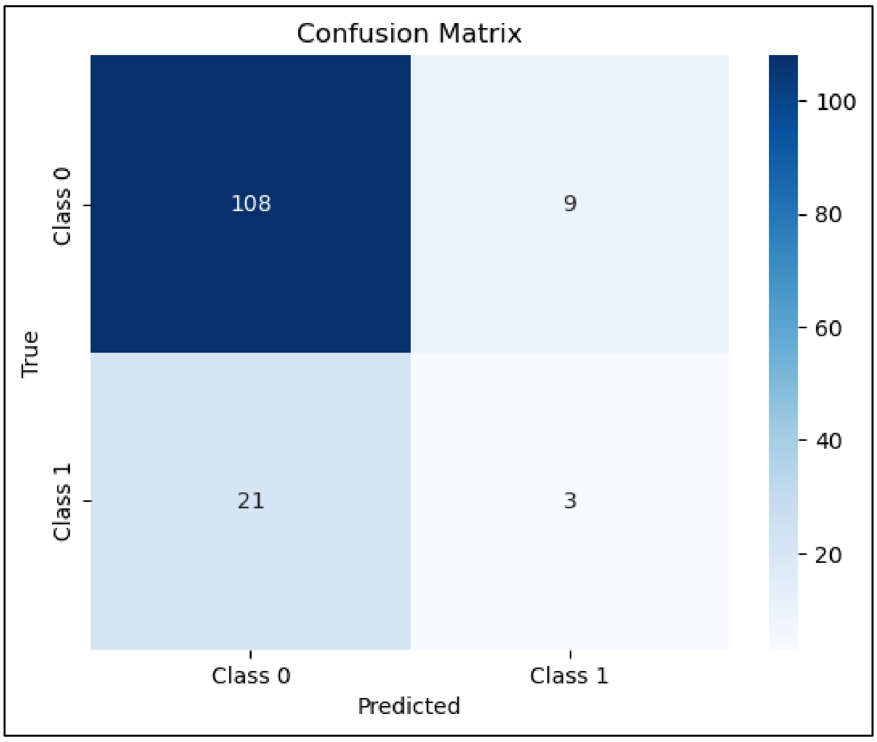
```
Epoch 127/350
11/11 [==============================] - 0s 4ms/step - loss: 0.4274 - val_loss: 0.4499
Epoch 127: early stopping
```

7. **Actual Loss Vs Predicted Loss Graph**



8. **Accuracy: 0.787**
9. **Precision (Positive Predictive Value): 0.25**
10. **Recall (Sensitivity, True Positive Rate): 0.125**
11. **Specificity (True Negative Rate): Specificity=0.923**
12. **F1 Score: F1 Score=0.1667**

Confusion Matrix



Project Repository: -

https://github.com/26ro/Lungs-Disease_Neural-Network-Model.git