BIRMINGHAM CITY University

Diabetes Prediction Using Deep Learning with Python

The Ultimate Guide for building Diabetes Classifier and Predicting Possibility of having Diabetes in a Patient by using Neural Networks

Blog By: Chaithanya Vamshi Sai

Student Id: 21152797

Introduction

Diabetes is a disease that occurs when the blood glucose is too high. Blood glucose is the main source of energy and comes from the food we eat.

Over time, having too much glucose in the blood can cause health problems and can lead to death. Preventing Diabetes is important and with advancements in AI, predicting the possibility of having Diabetes allows a person to take steps to manage diabetes and treat such situations at early stages making sure that more people can live healthy lives.

In this blog, I will demonstrate and show how we can harness the power of Deep Learning and apply it in healthcare. I will walk you through the entire process of how to classify and predict the possibility of having diabetes in a person using Neural Networks with Python.

Problem Statement

The objective of this task is we are given a data set of patient medical records of Pima Indians and whether they had an onset of diabetes within five years.

Using these data, we will build an effective and optimised Binary Classifier and Predict the possibility of Diabetes in a person using GridSearchCV with Neural Networks and make predictions on test data.

Dataset Description

The dataset comprises of 2 .csv files which are named “train.csv” and “test.csv”.

1. Train.csv: Training data to use with Model Training and Validation.
    - Number of entries: 668
    - Attributes: 9

2. Test.csv: Testing data to use with Predictions.
    - Number of entries: 100
    - Attributes: 8

Attribute	Description
A1	Number of times pregnant
A2	Plasma Oral Glucose tolerance test
A3	Diastolic blood pressure
A4	Triceps skinfold thickness
A5	2-Hour serum insulin
A6	Body mass index
A7	Diabetes pedigree function
A8	Age (years)
Class	0 or 1 (1= tested positive for diabetes)

Steps to Build a Neural Network Model using Keras & Optimisation with GridSearchCV

1. Importing Libraries and Loading the dataset
2. Data Exploration on all Attributes
3. Data Visualisation on all Attributes
4. Feature Scaling on all Attributes
5. Compiling the Neural Network Model
6. Hyperparameter Tuning of Neural Network Model using GridSearchCV
7. Model Building and Optimisation of Neural Network Model with Best Hyper Parameters
8. Evaluating Model Performance on Training and Validation data
9. Predictions on Test Data

1. Importing Libraries

Keras is a powerful and easy-to-use free open-source Python library for developing and evaluating deep learning models.

It wraps the efficient numerical computation libraries like TensorFlow and allows to define and train neural network models in just a few lines of code.

```
#Importing Libraries
import pandas as pd
import numpy as np
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
from numpy.random import seed
seed(100)
import tensorflow
tensorflow.random.set_seed(100)
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV, KFold
from keras.models import Sequential
from keras.layers import Dense
from keras.wrappers.scikit_learn import KerasClassifier
from keras.layers import Dropout
import tensorflow as tf
import sklearn
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, accuracy_score
```

2. Reading the Train and Test Dataset

```
# reading the Train data
df_train = pd.read_csv('/content/train.csv')

# reading the Test data
df_test = pd.read_csv('/content/test.csv')

# looking at the first five rows of thetrain data
df_train.head()

# looking at the first five rows of the test data
df_test.head()
```

3. Data Exploration

Data Exploration is an initial step to get insights from the data set to know which features have contributed more in predicting diabetes in a person.

It is always a good practice to understand the data first and try to gather as many insights from it.

a) Checking Shape and Data Types of Train & Test Data

```
# shape of the train & test data
df_train.shape, df_test.shape

# checking the train data type
df_train.info()

Output:

((668, 10), (100, 9))

Data columns (total 10 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Id      668 non-null    int64
 1   A1      668 non-null    int64
 2   A2      668 non-null    int64
 3   A3      668 non-null    int64
 4   A4      668 non-null    int64
 5   A5      668 non-null    int64
 6   A6      668 non-null    float64
 7   A7      668 non-null    float64
 8   A8      668 non-null    int64
 9   Class   668 non-null    int64
dtypes: float64(2), int64(8)
```

b) Checking Descriptive Statistics of Train Data

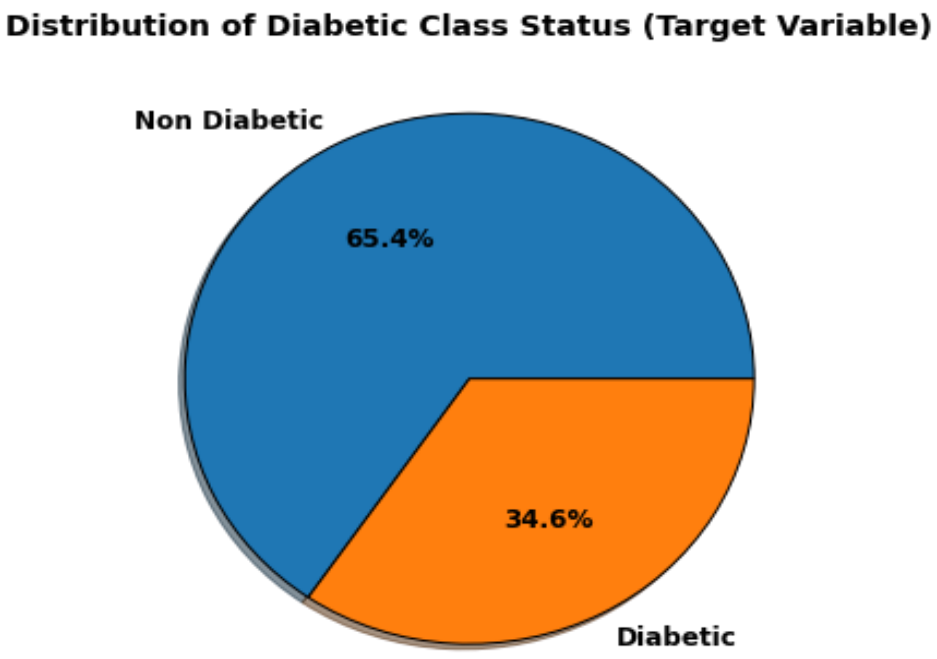
```
# Checking Descriptive Statistics of Train Data
df_train.describe()
```

	Id	A1	A2	A3	A4	A5	A6	A7	A8	Class
count	668.000000	668.000000	668.000000	668.000000	668.000000	668.000000	668.000000	668.000000	668.000000	668.000000
mean	334.500000	3.812874	120.405689	68.748503	20.567365	79.654192	31.860180	0.477329	33.091317	0.345808
std	192.979273	3.365672	32.291473	19.526392	16.020600	115.827750	7.827111	0.341398	11.711386	0.475988
min	1.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.078000	21.000000	0.000000
25%	167.750000	1.000000	99.000000	64.000000	0.000000	0.000000	27.100000	0.238750	24.000000	0.000000
50%	334.500000	3.000000	116.000000	72.000000	23.000000	36.500000	32.000000	0.377000	29.000000	0.000000
75%	501.250000	6.000000	140.000000	80.000000	32.000000	126.000000	36.500000	0.641250	40.000000	1.000000
max	668.000000	17.000000	199.000000	122.000000	99.000000	846.000000	67.100000	2.420000	81.000000	1.000000

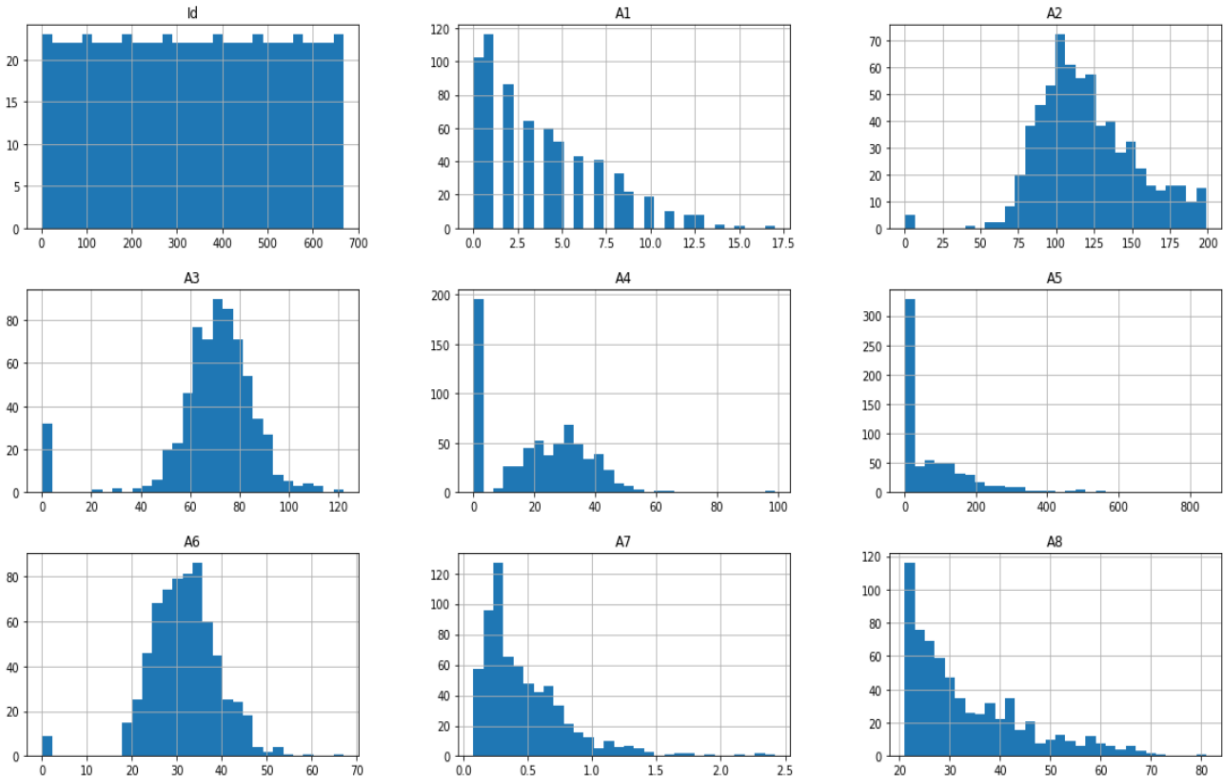
4. Data Visualisation

a) Checking Distribution of "Class" (Target Attribute) in Train Data

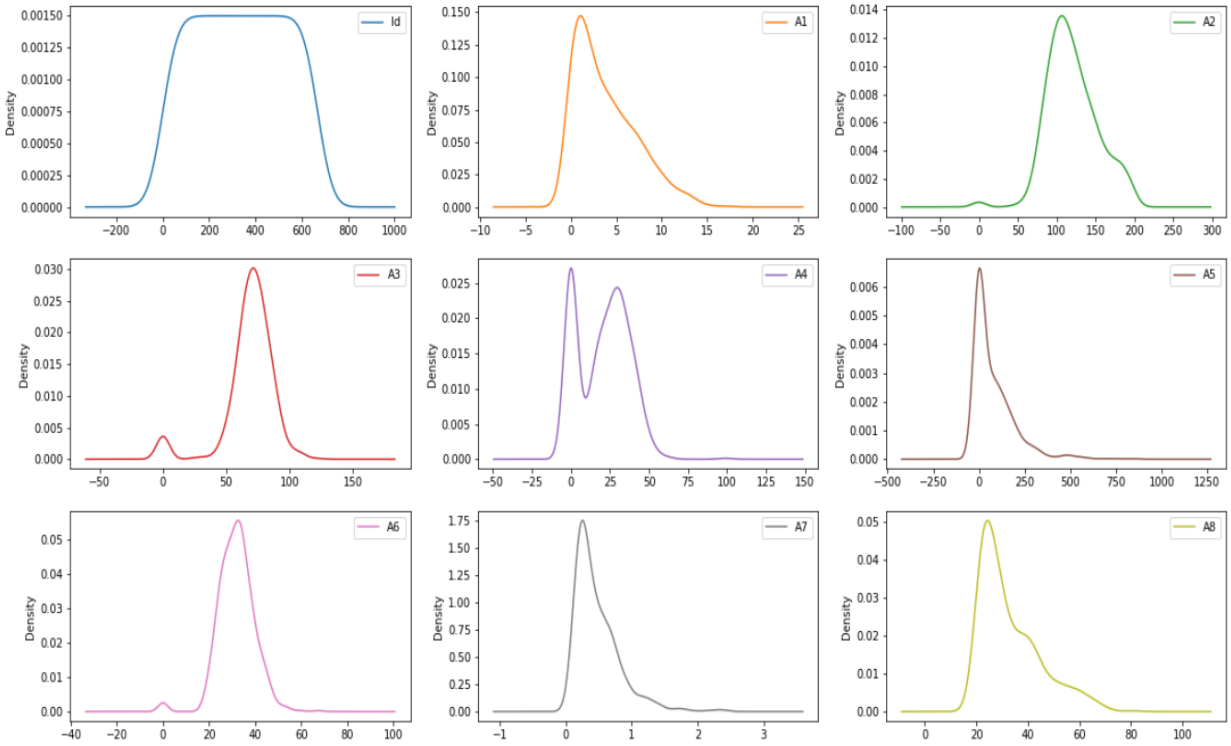
From the below pie chart, we can depict that 65.4% are non-diabetic and 34.6% are diabetic in the training data.



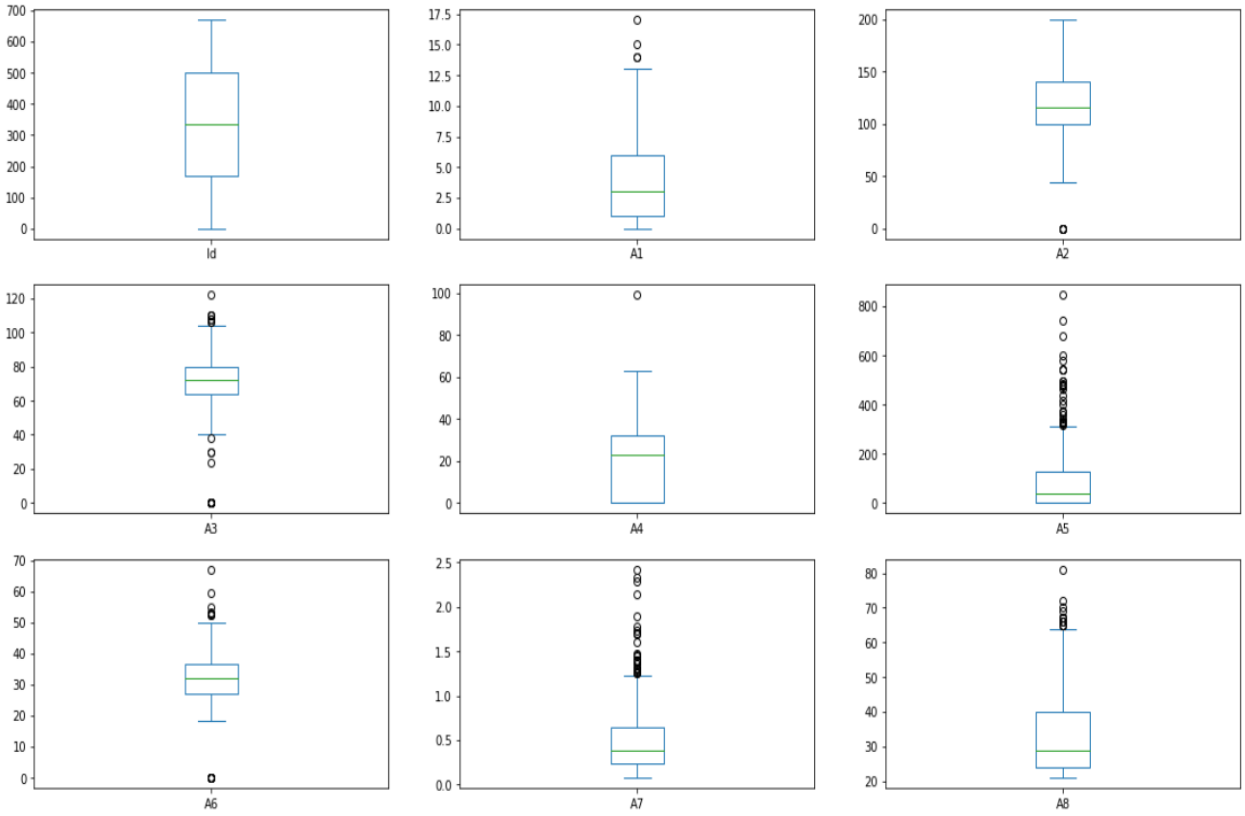
b) Plotting Histogram to Check the Distribution of all Attributes



c) Plotting Density plots to check the Distribution of all Attributes



d) Plotting Box plots to visualize the Distribution of all Attributes



e) Summary of Data Visualisation:

1. Bell shape curve: Blood Pressure
2. Right-Skewed: Age, Insulin, Pregnancies, Diabetes Pedigree Function
3. At least 75% of the people are
    - 25 years old more
    - BMI nearly 30 kg/m2
    - Insulin level 100 or more
    - 1 or more pregnancies
    - The glucose level of 100 mg/dL or more
    - Blood pressure of 60 mmHg or more

5. Feature Scaling on all Attributes

Feature Scaling is an essential step in modeling the algorithms with the datasets. Different scales of the data features affect the modeling of a dataset and it leads to a biased outcome of predictions in terms of misclassification error and accuracy rates. Thus, it is necessary to Scale the data before model building.

```
col = ['Id','A1','A2','A3','A4','A5','A6','A7','A8']
target='Class'
X=df_train[col]
Y=df_train[target].astype(int)

# Normalize the data using sklearn StandardScaler
from sklearn.preprocessing import StandardScaler

scaler=StandardScaler().fit(X)
print(scaler)

# Transform and display the training data
X_standardized = scaler.transform(X)
data = pd.DataFrame(X_standardized)
data.describe()
```

6. Hyperparameter Tuning of the Neural Network Model using GridSearchCV

Developing deep learning models is an iterative process and have lots of hyperparameters. It is very hard to tune the hyperparameter manually to get a model that can be trained efficiently in terms of time and compute resources.

So, there is a way where we can adjust the setting of the neural networks which is called hyperparameters and the process of finding a good set of hyperparameters is called hyperparameter tuning.

In this task, I have implemented hyperparameter tuning using GridSearchCV to build an optimised Neural Network model with better Accuracy and Performance.

a) Hyperparameter Tuning "Epochs" and "Batch size"
b) Hyperparameter Tuning "Learning Rate" and "Drop Out Rate"
c) Hyperparameter Tuning "Activation Function" and "Kernel Initializer"
d) Hyperparameter Tuning "Hidden Layer Neuron 1" & "Hidden Layer Neuron 2"

Python Implementation of Hyperparameter tuning of the Neural Network Model using GridSearchCV is available on **GitHub**

7. Model Building: Neural Network with the Best Hyperparameters Tuning using GridSearchCV

After performing hyperparameter tuning using GridSearchCV on various parameters like Epochs, Batch size, learning rate, Activation function, Hidden layer Neurons etc we need to build a neural network model using the best hyperparameters obtained while training to get an optimised model for better Accuracy and Performance.

8. Evaluating Model Performance on Train & Validation Data

a) Model Accuracy and Classification Report on Train & Validation Data

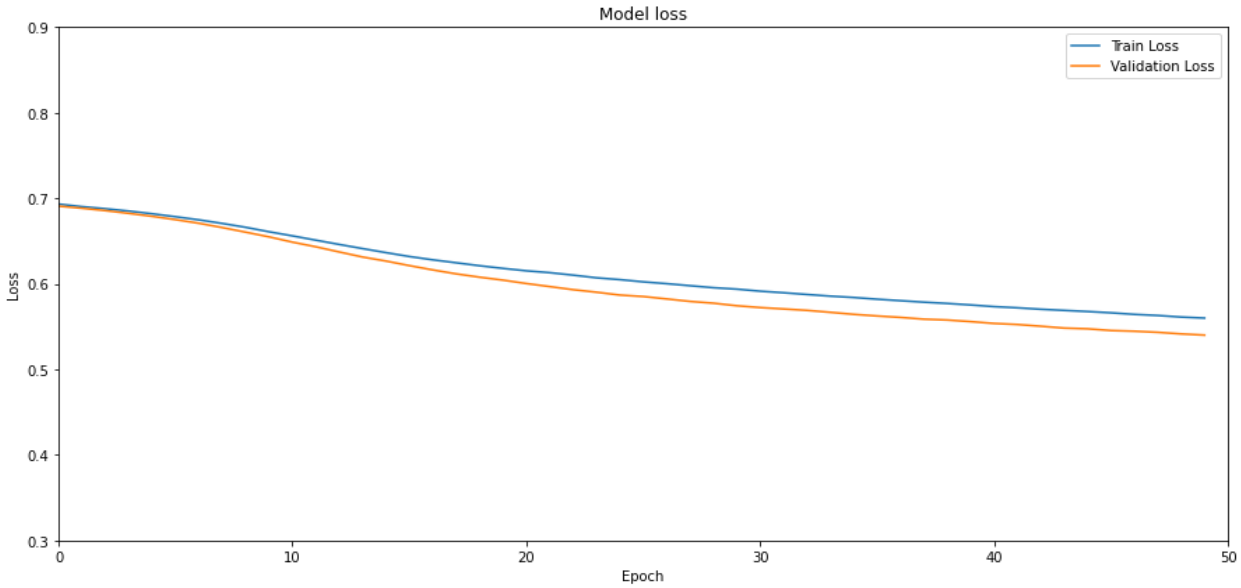
After hyperparameter tuning using GridSearchCV on the Neural Network model, we have obtained a **Training Accuracy of 78%** and a **Validation Accuracy of 80.1%** which implies a better and an optimised model.

```
#Model Accuracy Results
print("Results:")
print("--------")
scores = model_final.evaluate(X_train, y_train, verbose=False)
print("Training Accuracy: %.2f%%\n" % (scores[1] * 100))
scores = model_final.evaluate(X_val, y_val, verbose=False)
print("Validation Accuracy: %.2f%%\n" % (scores[1] * 100)

#Classification Report
from sklearn.metrics import classification_report, accuracy_score

print(accuracy_score(y_val, y_pred_categorical1))
print(classification_report(y_val,y_pred_categorical1)

Output:

Results:
--------
Training Accuracy: 77.94%

Validation Accuracy: 80.10%


0.8009950248756219
              precision    recall  f1-score   support

           0       0.85      0.85      0.85       132
           1       0.71      0.71      0.71        69

    accuracy                           0.80       201
   macro avg       0.78      0.78      0.78       201
weighted avg       0.80      0.80      0.80       201
```

b) Neural Network: Model Loss on Train and Validation Data

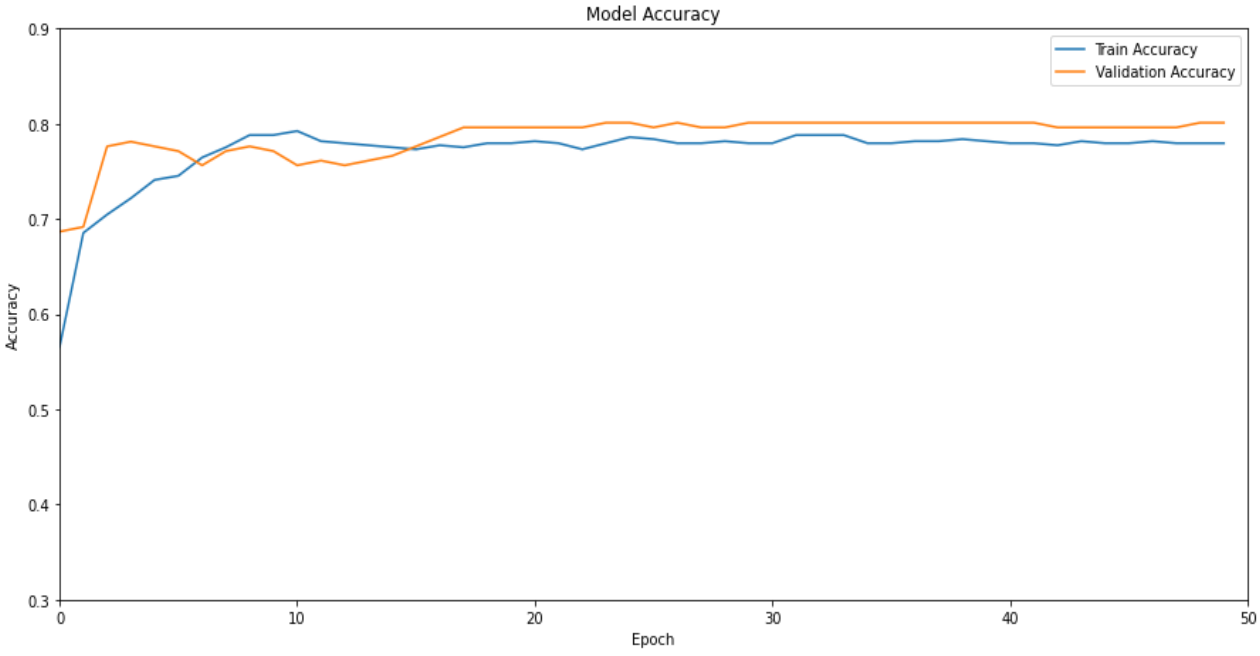
From the Model Loss chart, we can depict that as the number of epochs increases, the neural network model tends to lower the loss/cost on both training and validation data.

Hence, the Neural network model built is more effective and an optimised model.



c) Neural Network: Model Accuracy on Train and Validation Data

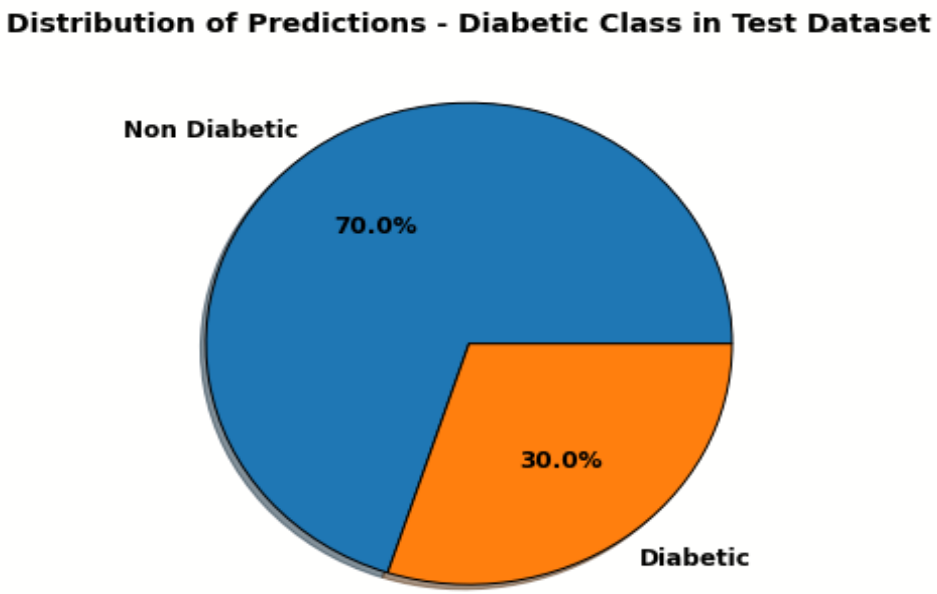
From the Model Accuracy chart, we can depict that as the number of epochs increases, the neural network model tends to increase the accuracy on both training and validation data. Hence, the Neural network model built is more effective and an optimised model.



9. Predictions on Test Data

Using the Optimised Neural Network Model with the best hyperparameters and Accuracy, we will make predictions on test data and save predictions on .csv file name “test-predictions.csv”

```
# make predictions using the best model
test_pred = model_final.predict(X_standardized_test)

# convert predictions to the required label format (0 or 1)
final_predictions = [1 if pred > 0.5 else 0 for pred in test_pred]
print(final_predictions)

# create data frame for submission
df_submission = pd.DataFrame(columns = ['Class'])
df_submission['Class'] = final_predictions

df_submission.insert(1, 'Id', range(1, 1 + len(df_submission)))
df_submission

# save data frame to .csv file
df_submission.to_csv('/content/test-predictions.csv', index=False)
```



10. Conclusion

In this blog, we discussed how to approach the classification problem and predict the possibility of diabetes by implementing a Neural networks model using Keras and GridSearchCV.

We can explore this work further by trying to improve the accuracy by using advanced Deep Learning algorithms.

11. References

https://machinelearningmastery.com/tutorial-first-neural-network-python-keras/

https://medium.datadriveninvestor.com/hyperparameter-tuning-with-deep-learning-grid-search-8630aa45b2da