WEB OF SYNERGY:
International Interdisciplinary Research Journal
Volume 2 Issue 9, Year 2023 ISSN: 2835-3013
https://univerpubl.com/index.php/synergy

Utilising Artificial Intelligence for Disease Classification and Prediction

Bareq Zeyad kareem, Muntadher Kamel Flaih, Dr Zeyad Yousif
University of Technology/Biomedical Engineering Department

Mohammed Salim Mahmood, Ahmed Jalal Yousef
Ministry of Health of Iraq/ Diyala Health Department/Diyala- Iraq

Article Information

Received: July 02, 2023
Accepted: Aug 01, 2023
Published: Sep 23, 2023

ABSTRACT

The main objective of this research is to investigate the role of artificial intelligence in disease classification and prediction. A brief review of the techniques, algorithms, tools and terminologies that were used in this work has been conducted. Artificial Neural Networks (ANNs) are reviewed to nominate the suitable type for this work.

In this work, a real medical data set has been used. The data set includes 14 attributes, of which 13 independent diagnosis variables and one categorical dependent variable, which is the type of heart disease.

To classify heart disease, a classification model is developed by using TensorFlow in Python. It is found that the classification model is 87% accurate in classifying heart disease. The challenges to implementing this model are explained, such as the data pre-processing, which means that the medical data cannot be used directly as some of them are categorical data that requires encoding before it can be used for the model development procedures.

It is concluded that data sets cannot be directly used after the acquisition because, for example, the data sets may include missing data and faulty readings, and these represent big challenges for the real-time processing and presentation requirements. It is also found that variables of different types, such as logical variables and categorical information, require encoding before using them to build prediction or classification models.

Introduction

This chapter introduces the objectives of this work, as well as the scope of the project. In addition, the chapter briefly explains the project outline by introducing a summary of all the chapters.

The Scope of the Project

This project focused on the utilization of artificial intelligence in the health sector. The project methodology was based on Deep Neural Networks (DNNs), TensorFlow from Google, and Python programming language. All the programmes were developed and run on an ordinary personal computer (no supercomputers were used during the training and validation processes).

Project Objectives

The objectives of this project were to:

- Conduct a comprehensive review of the state of the art of the applications of AI in disease classification. (25%)
- Develop a classification (prediction) model by using Python programming language. (25%)
- Train, evaluate, and test the developed model by using different datasets. (25%)
- Explain the challenges, limitations, and conclusions. In addition, suggestions for future work must be provided at the end. (25%).

Literature Review

Introduction

This chapter expresses a brief review of the techniques, algorithms, tools and terminologies that were used in this work. ANNs are examined, focusing on three common and relevant types, before the best one for this work is indicated. TensorFlow software is introduced, and the chapter then looks at three different data normalisation techniques.

Artificial Neural Networks (ANNs)

Due to their powerful performance in complex applications such as prediction, classification, natural language processing, image processing and speech recognition, ANNs have received high levels of attention, especially after big developments in data storage and processing performance. However, the performance of the ANNs depends heavily on the quality of data sets, the power of the computer processors and training algorithms. Classical ANNs are trained by updating weights in order to minimise the difference between the ANNs output (the predictions) and the real output (observations) (Chauvin and Rumelhart, 1995). Below are three relevant neural network architectures that are commonly used in different applications:

Multilayer Perceptron (MLP)

MLPs are feedforwarded neural networks where the information is applied to the input layer and then passed through hidden layers to the output layer without any loops. In these neural network types, the training of the neural network involves selecting the weight coefficients between the layers (Orłowska-Kowalska, Blaabjerg and Rodríguez, 2014). The sigmoidal and hyperbolic tangent nonlinear functions are the commonly used activation functions for MLPs, and this is what makes this type of neural network able to map nonlinear relationships. The main challenge with MLPs is finding the right number of neurons in the hidden layer of the neural network (Orłowska-Kowalska, Blaabjerg and Rodríguez, 2014).

Radial basis function network (RBFN)

These neural networks have only one hidden layer of neurons and all the neurons have the same activation function. This structure makes RBFNs faster than the MLPs with almost the same level of accuracy (Huang, Huang and Chiou, 2003). According to Seshagiri and Khalil (2000), this type of neural network is mainly used to control nonlinear dynamic systems that have some uncertainties or parameter variations.

Deep Neural Networks (DNNs)

Deep learning algorithms are used to process massive amounts of data sets to assist in making decisions based on real-world knowledge, make useful predictions and learn complex relationships between variables (Goodfellow *et al.*, 2016). DNNs have more than one hidden layer. The deep learning algorithms help to reduce the cost function and improve the neural network's performance significantly while processing big data sets. This makes DNNs a perfect choice for forecasting, classifying and complex function approximation (Goodfellow *et al.*, 2016).

TensorFlow

The architectural components of neural networks, such as activation functions and training algorithms, may initially be represented as mathematical equations, but they must be transformed into computer programs so they can be implemented. For this reason, there are many software systems such as Scikit-learn (Pedregosa *et al.*, 2011), Theano (Al-Rfou *et al.*, 2016), Torch (Collobert, Bengio and Mariéthoz, 2002), and many more open-source software systems. In November 2015, Google released novel training and optimisation algorithms called TensorFlow (Abadi *et al.*, 2015). TensorFlow is a machine learning system that operates at a large scale, and its novel optimisations and training algorithms make it suitable for a variety of applications, with a focus on training and optimising the performance of DNNs (Pang, Nijkamp and Wu, 2020). For the purpose of this work, TensorFlow was used mainly in a Python programming environment to implement the prediction and classification models using DNNs.

Summary

The techniques, algorithms, tools, and terminologies used in this work were concisely explained in this chapter. ANNs were examined, with a focus on three common and relevant ANN types, and it was concluded that DNNs are the most suitable choice for this work. TensorFlow was introduced and defined, and the chapter then reviewed three different data normalisation techniques that were considered for use in this project.

Research Methodology

Introduction

This chapter illustrates the research methodology of this project. One data set was used in this work to fulfil the goals of the project. The data set related to heart disease, and it was used to build a classification model to identify types of heart disease based on certain parameters. The neural network structures are explained in detail as well as training algorithms. In addition, the programming language, that was used to develop both the prediction and classification models, is explained. The chapter concludes with a summary of the chapter findings.

Data sets

The data set in this work was obtained from the Center for Machine Learning and Intelligent Systems at the University of California, Irvine, which was donated by Janosi *et al.* (1988). The data set had 14 attributes, listed as follows (Janosi *et al.*, 1988):

age	age in years
sex	sex (1 = male; 0 = female)
cp	chest pain type: ➢ Value 1: typical angina ➢ Value 2: atypical angina ➢ Value 3: non-anginal pain ➢ Value 4: asymptomatic
trestbps	resting blood pressure (in mm Hg on admission to the hospital)
chol	serum cholesterol in mg/dl
fbs	(fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)

restecg	resting electrocardiographic results: ➢ Value 0: normal ➢ Value 1: having ST-T wave abnormality (T wave inversions and/or ST elevation or depression of > 0.05 mV) ➢ Value 2: showing probable or definite left ventricular hypertrophy by Estes' criteria
thalach	maximum heart rate achieved
exang	exercise induced angina (1 = yes; 0 = no)
oldpeak	depression induced by exercise relative to rest
slope	the slope of the peak exercise: ➢ Value 1: upsloping ➢ Value 2: flat ➢ Value 3: downsloping
ca	number of major vessels (0–3) coloured by fluoroscopy
thal	3 = normal; 6 = fixed defect; 7 = reversable defect
target (to be predicted)	diagnosis of heart disease (angiographic disease status): ➢ Value 0: < 50% diameter narrowing ➢ Value 1: > 50% diameter narrowing

This data set contained no personal data because it had been removed by the donor (Janosi *et al.*, 1988). The main purpose of using this data set was to develop a classification model that predicted the 'target' variable in the data set, which was a diagnosis of heart disease, (angiographic disease status) using the following two values:

- Value 0: < 50% diameter narrowing
- Value 1: > 50% diameter narrowing

Data Exploration and Cleaning

A data cleaning process was been performed on all data sets as follows:

- Descriptive statistics were implemented to:
  - ✓ Identify impossible values that may result from faults/errors.
  - ✓ Look for outliers in the data.
- Look for patterns that may indicate errors in the data set.
- Identify duplicated data.
- Address any identified errors.

Neural Network Design

This section explains the structure of the neural network that was used in this work.

Multilayer Perceptron (MLP)

The MLP topology has been used extensively by the researchers to build prediction models. An MLP neural network model was developed by Msiza, Nelwamondo and Marwala (2007) for water demand forecasting. A wind speed prediction model was implemented by (Deo *et al.*, 2018), and an artificial intelligence approach based on MLP was developed for predicting the soil consolidation coefficient (Pham *et al.*, 2019). Figure 0-1 shows the structure of the MLP (Al-Shibaany, Hedley and Bicker, 2012).

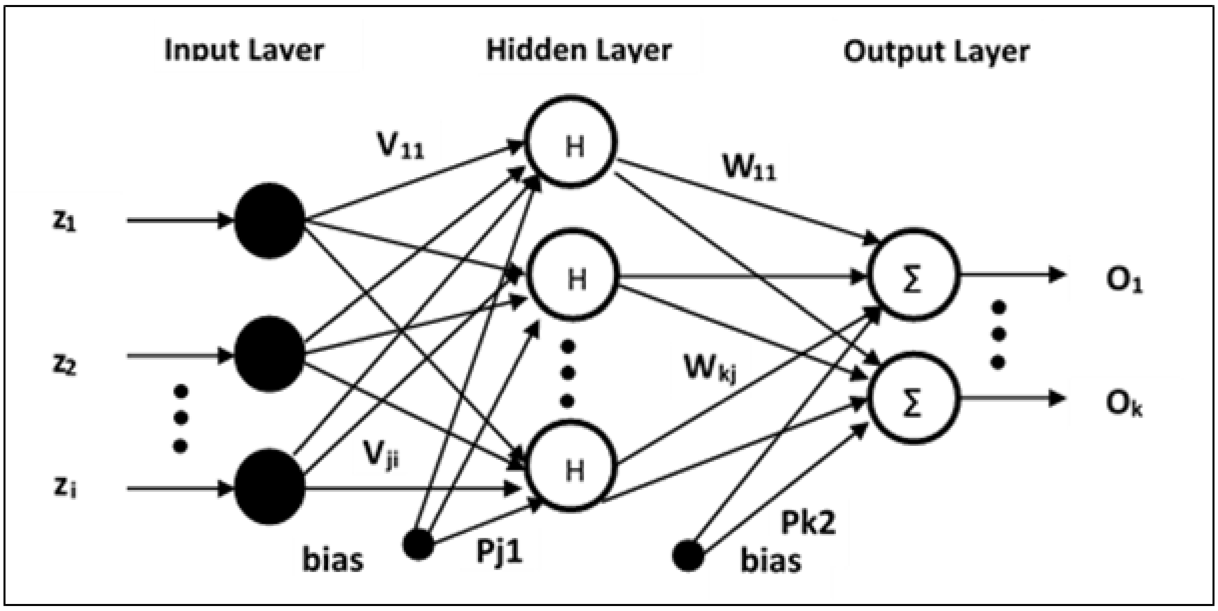


Figure 0-1: Structure of the MLP (Al-Shibaany, Hedley and Bicker, 2012)

Generally, the MLP has the following layers:

- An input layer with several nodes that are equal to the number of independent variables. The nodes (neurons) of the input layers have linear activation functions where the input is not processed at this layer.
- One hidden layer with certain number of nodes (neurons) where different activation functions are used based on the application and type of input data. In addition to the normal neurons in the hidden layer, a bias neuron may be added in certain applications to stabilise the neural network performance (Khandelwal and Singh, 2006).
- The output MLP layer is the last layer that has several neurons equal to the number of the dependent variables in the data.

The parameters of the MLP shown in Figure 0-1 are listed in Table 0-1 below:

Table 0-1: List of parameters of the MLP neural network

ni	Number nodes in the input layer
nj	Number of nodes in the hidden layer
nk	Number of nodes in the output layer
V_{ji}	Weight between (j) hidden node and (i) input node
W_{kj}	Weight between (k) output node and (j) hidden node
P_{j1}	Weight between (j) hidden node and bias node
P_{k2}	Weight between (k) output node and bias node
h	Hidden layer node
z_i	Input of (i) input node
O_k	Output of (k) output node
H	Activation function

The following steps explain the calculation of the output of MLP:

1. Apply the input data to the input layer. In this layer, the data is not processed.
2. The data is then passed to the hidden layer and the output of each neuron at the hidden layer is calculated as follows:

$$hnet_j = H(h_j) \tag{3-1}$$

Where h_j represents the jth neurons in the hidden layer, and $hnet_j$ represents the output of the jth neurons in the hidden layer.

$$h_j = \sum_{i=1}^{ni} z_i \times V_{ji} + bias \times P_{j1} \tag{3-2}$$

3. The output of the output layer is calculated then as follows:

$$O_k = \sum_{j=1}^{nj} hnet_j \times W_{kj} + bias \times P_{k2} \tag{3-3}$$

Deep Neural Networks (DNNs)

DNNs are simply neural networks with multiple hidden layers. Several research and industry fields such as computer vision, speech recognition and computational medicine have been transformed by the application of DNNs (Goodfellow *et al.*, 2016). Although single hidden layer neural networks can approximate continuous functions as accurately as possible if there are enough nodes in the hidden layer, it is almost impossible for them to approximate complex functions. Going deeper and adding more hidden layers to the neural network can help to overcome this and enable the approximation of complex functions (Lu *et al.*, 2017, Telgarsky, 2016, Cohen, Sharir and Shashua, 2016, Eldan and Shamir, 2016). Therefore, for this project, the most appropriate option was to use DNNs. Figure 0-2 shows the schematic structure of DNNs.

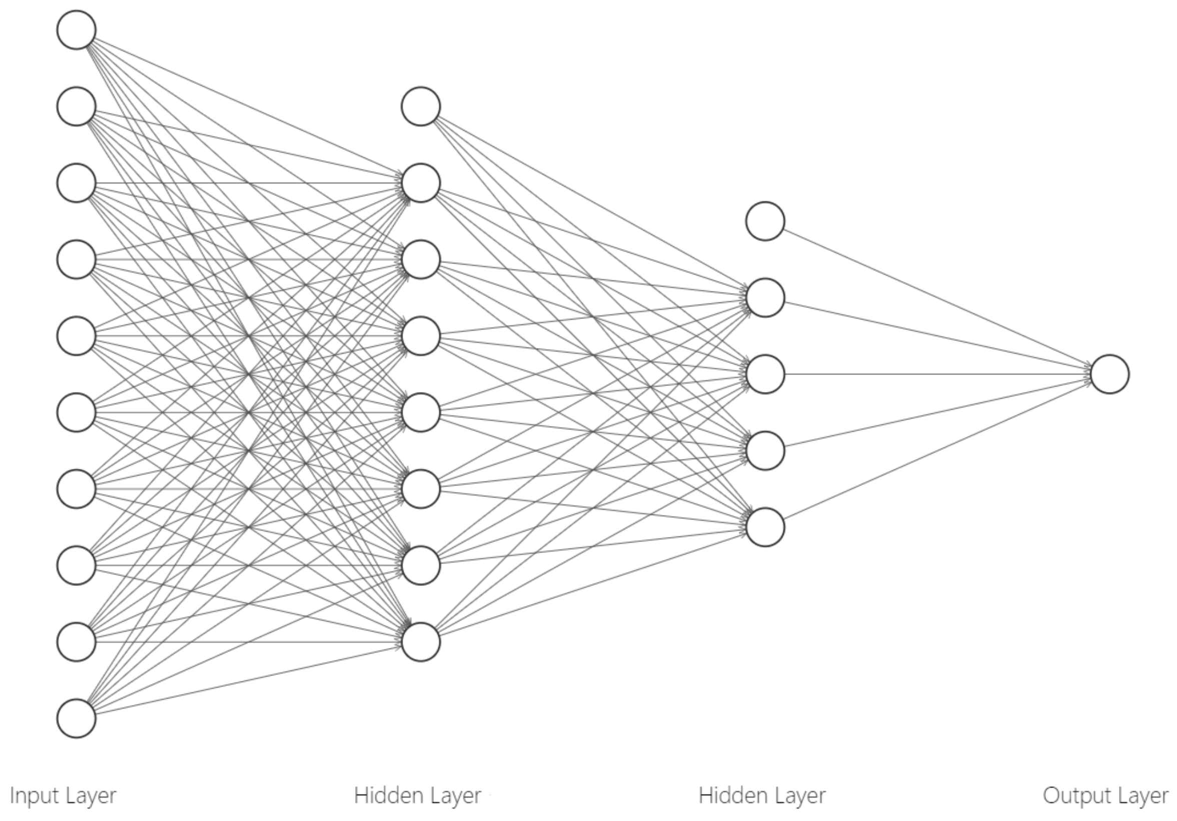


Figure 0-2: Schematic structure of DNNs

Although DNNs are the best-performing methods for many classification problems, training them to achieve high-quality performance requires high computing power and takes longer than ordinary neural networks (Yang *et al.*, 2020).

Neural Network Layers

It easy to decide how many neurons there are at the input layer of any neural network by simply counting the number of independent variables that are fed into the neural network. The same principle is applied to the output layer, where the number of neurons is equal to the number of dependent variables. However, choosing the number of neurons at the hidden layer represents a big challenge that still attracts the interest of many researchers. A binary search technique was used by Doukim, Dargham and Chekima (2010) to estimate the number of neurons in the hidden layer where the number was chosen to be 1, 2, 4, 8, 16, 32 and 64, and the accuracy of the model developed in this work did not exceed 80%. A comprehensive systematic review of the techniques used to count the number of neurons in the hidden layer was conducted by Sheela and Deepa (2013). There are many techniques detailed in the literature that have been used by researchers to count the number of neurons in the hidden layer. Table 0-1 lists the techniques that have been developed by other researchers and they were used to count the number of neurons in the hidden layer in this work as follows:

Ni: number neurons in the input layer

No: number neurons in the output layer

Nh: number neurons in the hidden layer

Nt: number of training pairs (size of the training data set)

Table 0-2: List of techniques to count the number of neurons in the hidden layer

No.	Technique	Reference
1	$Nh = \frac{\sqrt{1+8Ni}-1}{2}$	(Li, Chow and Yu, 1995)
2	$Nh = Ni - 1$	(Tamura and Tateishi, 1997)
3	$Nh = \frac{1}{2} \cdot \frac{Nt}{Ni\ logNt}$	(Xu and Chen, 2008)
4	$Nh = \frac{Nt}{Ni}$	(Xu and Chen, 2008)
5	$Nh = \sqrt{Ni\ No}$	(Shibata and Ikeda, 2009)
6	$Nh = log_2(Ni+1) - No$	(Hunter *et al.*, 2012)
7	$Nh = \frac{4\,Ni^2+3}{Ni^2-8}$	(Sheela and Deepa, 2013)

Training Algorithm

Training a neural network simply means adjusting its weights so that the difference between the neural network output and the actual output is reduced to the minimum. In this work, the error backpropagation algorithm was used to train the neural networks. This technique is used to update the weights of the neural networks by calculating the error in the output layer and then propagate back this error to update the weights as follows (Zurada, 1992):

1. Randomly initiate the weights vectors.
2. Choose an appropriate learning rate ($\eta > 0$) which is the amount of change that will be applied to the weights in order to minimise the cyclic error *E*. According to Goodfellow *et al.* (2016), the learning rate can be chosen on a logarithmic scale, e.g. a learning rate taken within the set 0.1, 0.01, 10^{-3}, 10^{-4}, and 10^{-5}. In this work, the learning rate was chosen to be $\eta = 0.1$ to start with and this was updated logarithmically to check the difference in model performance.

3. Apply the input variables (z_k) and the required output variable (d_k).
4. Calculate the output of the neural network (O_k) by using equations 3-1, 3-2 and 3-3.
5. Calculate the error of the output layer, which is simply the difference between the actual output from the training data set and the output of the neural network: $\delta_{ok} = d_k - O_k$
6. Calculate the error of the hidden layer by using the following equation:

$$\delta_{hj} = 0.5\,(1 - hnet_j^2)\sum_{k=1}^{j}\delta_{ok}\,W_{kj} \qquad 3\text{-}4$$

7. Update the weights of the output layer by using the following equation:

$$W_{kj} = W_{kj} + \eta\,\delta_{ok}\,hnet_j \qquad 3\text{-}5$$

8. Update the weights of the hidden layer by using the following equation:

$$V_{ji} = V_{ji} + \eta\,\delta_{hj}\,z_i \qquad 3\text{-}6$$

9. Calculate the cyclic error *E*:

$$E = \frac{1}{2}(d_k - O_k)^2 \qquad 3\text{-}7$$

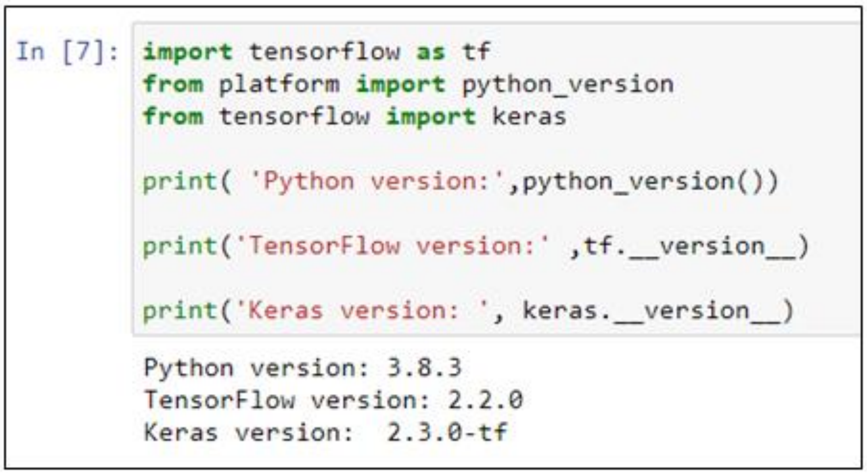
10. If the calculated *E* is less than the assumed *E*, the training process will be terminated. Otherwise, return to step 3.

Programming Language

In this study, Python programming language was used to perform all artificial intelligence-based model developments. Python is a general-purpose and high-level programming language, which was created by Guido van Rossum and first released in 1991 (Guttag, 2016). The philosophy of Python emphasises code readability to help programmers write clear, logical code for small and large-scale projects (Kuhlman, 2009).

TensorFlow was introduced in Chapter Two (section 2.5) and was used here to develop artificial intelligence-based models. TensorFlow supports a variety of applications but it particularly targets training and inference within DNNs (Abadi, 2016).

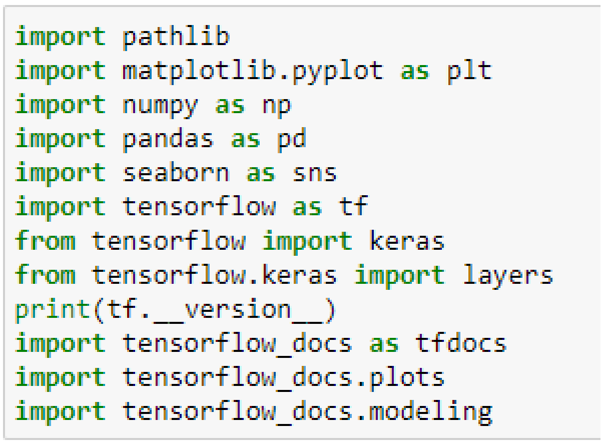
Keras is an open-source neural network library written in Python that was also used in this project. This library can run on top of TensorFlow to simplify the development process of neural network models (Gulli and Pal, 2017). Working in Python is extremely sensitive, not just to the Python version but also to the Keras and TensorFlow languages. Figure 0-3 shows the versions of Python, TensorFlow and Keras that were used in this work.



```
In [7]:  import tensorflow as tf
          from platform import python_version
          from tensorflow import keras

          print( 'Python version:',python_version())

          print('TensorFlow version:' ,tf.__version__)

          print('Keras version: ', keras.__version__)

          Python version: 3.8.3
          TensorFlow version: 2.2.0
          Keras version:  2.3.0-tf
```

Figure 0-3: The versions of Python, TensorFlow, and Keras

In addition to TensorFlow and Keras, other programming libraries and tools were imported as shown in in Figure 0-4.



```
import pathlib
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
import seaborn as sns
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
print(tf.__version__)
import tensorflow_docs as tfdocs
import tensorflow_docs.plots
import tensorflow_docs.modeling
```

Figure 0-4: Programming Tools and Libraries

Below is a brief explanation of the tools and libraries that were used in this work:

- **Pathlib**: This library offers classes that help representing filesystem paths for different operating systems (Horton and Parnin, 2018).
- **Matplotlib**: This library has many tools that help create static, animated, and interactive visualisations in Python (Tosi, 2009).
- **Pandas**: This library offers functions and operations that can help manipulate numerical tables and time series' (Chen, 2017).
- **Seaborn**: This library is based on matplotlib, but it provides a high-level interface for producing informative statistical graphics (Bisong, 2019).
- **Scikit-learn**: This is a machine learning library that is written in Python. The Scikit-learn library is designed to be simple, efficient and accessible to non-experts (Buitinck *et al.*, 2013).

Summary

The research methodology of this project has been explained in this chapter. To fulfil the project goals, one data set was used to develop a classification model. The MLP and DNNs were explained, and the decision was made to choose DNNs to build the prediction and classification models, as they have better approximation capability for complex functions and relationships. Python, TensorFlow, and Keras were also used to perform the neural network models.

Results and Discussion

Introduction

This chapter illustrates the results of the work that was done in this project. The results of developing a classification model to classify heart disease based on certain input features are explained. The data loading, exploration and preparation of all models are explained in detail in this chapter. The chapter concludes with a summary of what has been presented and discussed.

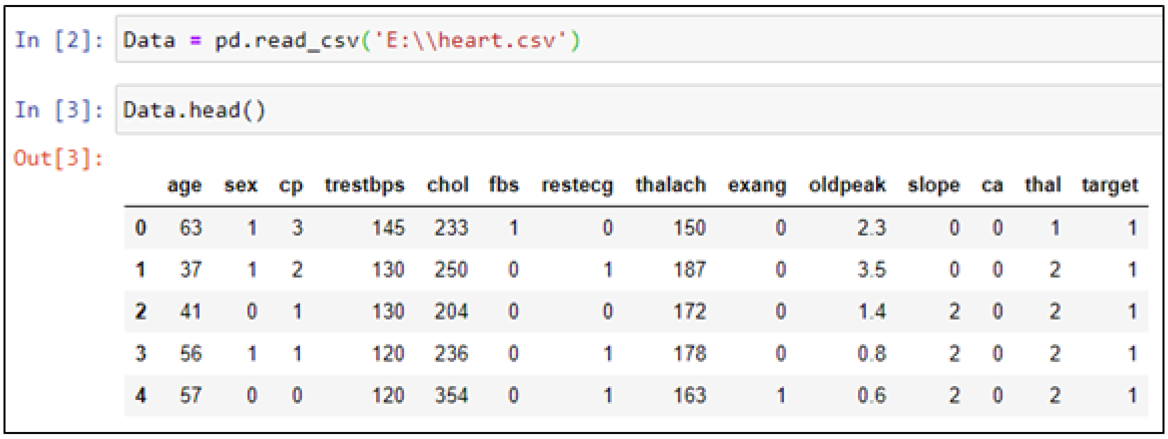
Heart Disease Classification Results

This section illustrates the results of the classification and diagnosis of heart disease (angiographic disease status) using the following two values:

- Value 0: < 50% diameter narrowing
- Value 1: > 50% diameter narrowing

Data Loading

In order to start working on the data set in this section, the required libraries (tools) were imported into the program. The Numpy, Pandas, and TensorFlow were explained in section 3.6 in Chapter 3. The data set was saved in a CSV file and loaded into Python by using the *read_csv()* function. Figure 0-1 shows the first five lines of the data sets, ensuring that the data was already uploaded to Python.



```
In [2]: Data = pd.read_csv('E:\\heart.csv')

In [3]: Data.head()
Out[3]:
```

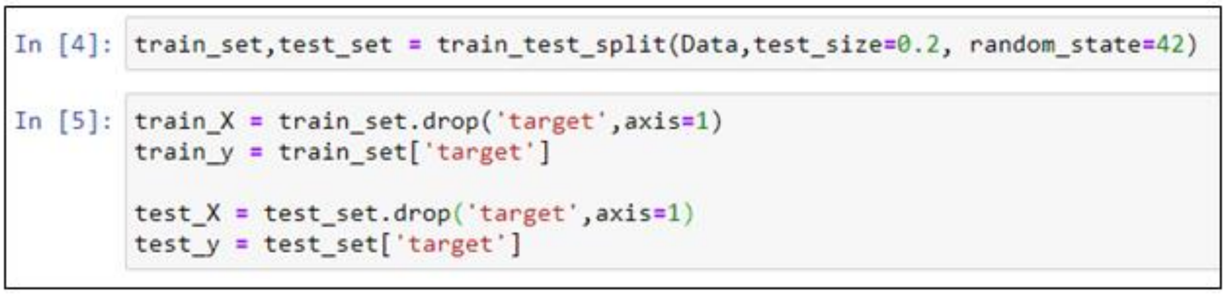
	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	ca	thal	target
0	63	1	3	145	233	1	0	150	0	2.3	0	0	1	1
1	37	1	2	130	250	0	1	187	0	3.5	0	0	2	1
2	41	0	1	130	204	0	0	172	0	1.4	2	0	2	1
3	56	1	1	120	236	0	1	178	0	0.8	2	0	2	1
4	57	0	0	120	354	0	1	163	1	0.6	2	0	2	1

Figure 0-1: Loading heart disease data set from a CSV file

Training and Testing Data Sets

The data was split into training and testing data sets using the code shown in Figure 0-2, where:

- **Train_X**: This data set contained all the input variables (features) that were used to train the classification model.
- **Train_y**: This one-column data set included the output (label). This was the 'target' variable used to train the model.
- **Test_X**: This data set contained all the input variables (features) that were used to test the classification model.
- **Test_y**: This one-column data set included the output (label). This was the 'target' variable used to test the model.



```
In [4]: train_set,test_set = train_test_split(Data,test_size=0.2, random_state=42)

In [5]: train_X = train_set.drop('target',axis=1)
        train_y = train_set['target']

        test_X = test_set.drop('target',axis=1)
        test_y = test_set['target']
```

Figure 0-2: Heart disease data split into training and testing data sets

Data Exploration

Figure 0-3 shows the histograms of the variables in the heart disease data set. These figures helped to quickly understand the nature and distribution of the variables. For example, it can be clearly seen which variables are continuous and how they are distributed, and which variables are categorical and how many categories in each variable of them. Figure 0-4 shows the distribution of age variables and it is clear that the majority of the patients are between the age of 40 and 65.

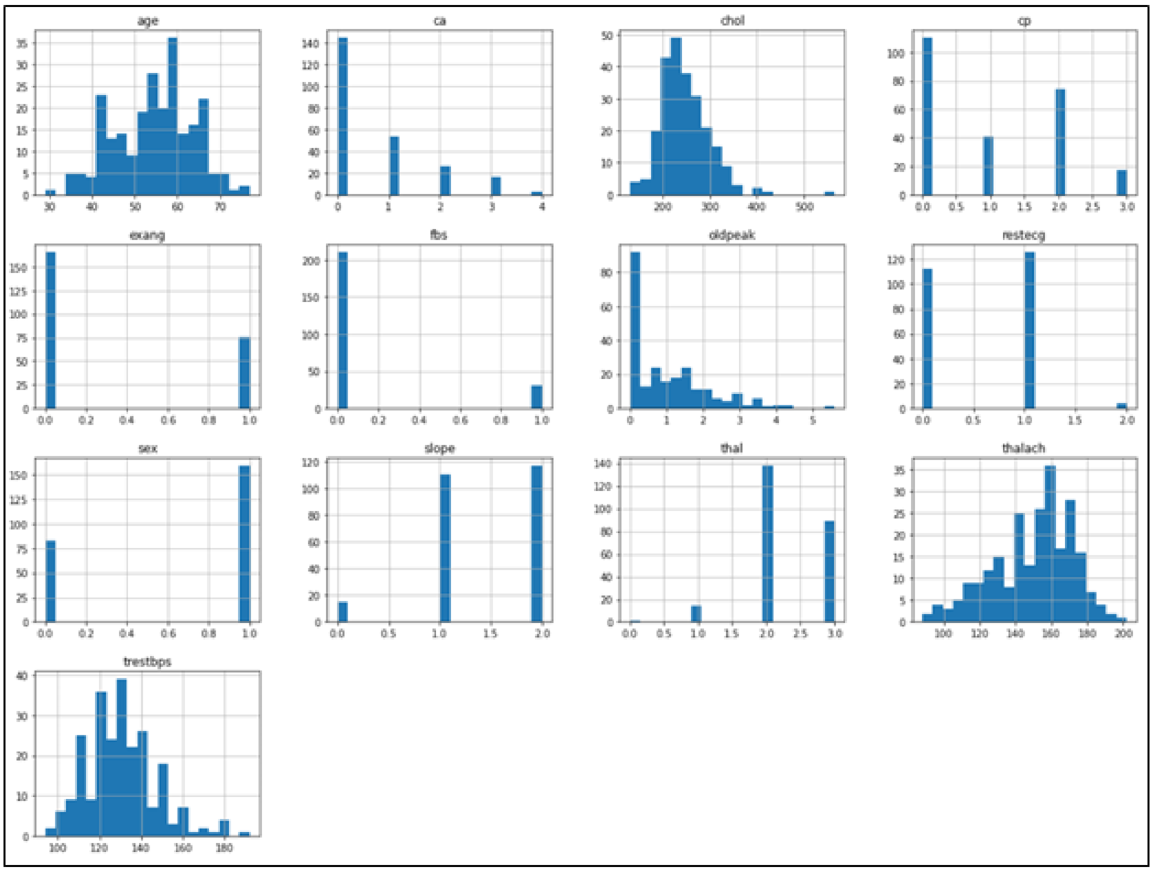
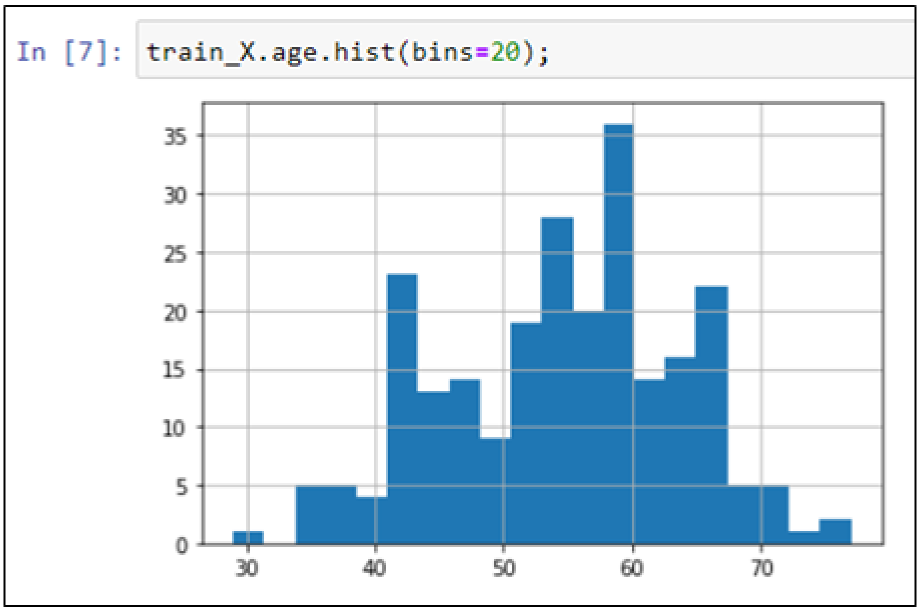


Figure 0-3: Histograms of heart disease data set variables



```
In [7]: train_X.age.hist(bins=20);
```

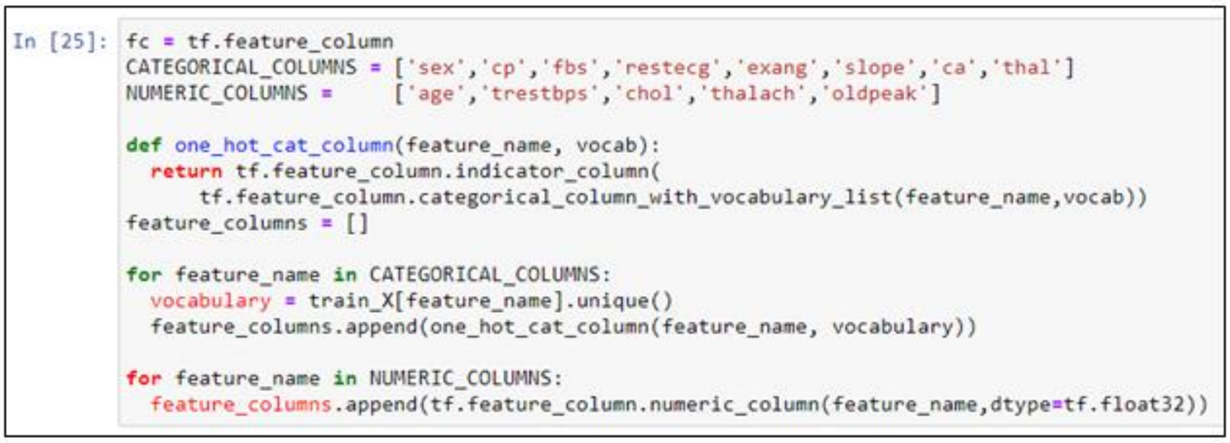
Figure 0-4: Age variable distribution

Data Preparation

Following subsection 3.2.2 in Chapter 3, the columns in Figure 0-1 contained two types of information as follows:

- Categorical information (*sex*, *cp*, *fbs*, *restecg*, *exang*, *slope*, *ca*, and *thal*)
- Numerical information (*age*, *trestbps*, *chol*, *thalach*, and *oldpeak*)

TensorFlow has built-in features that were used to encode the categorical and numerical data to help enhance and accelerate the training process, as shown in Figure 0-5.

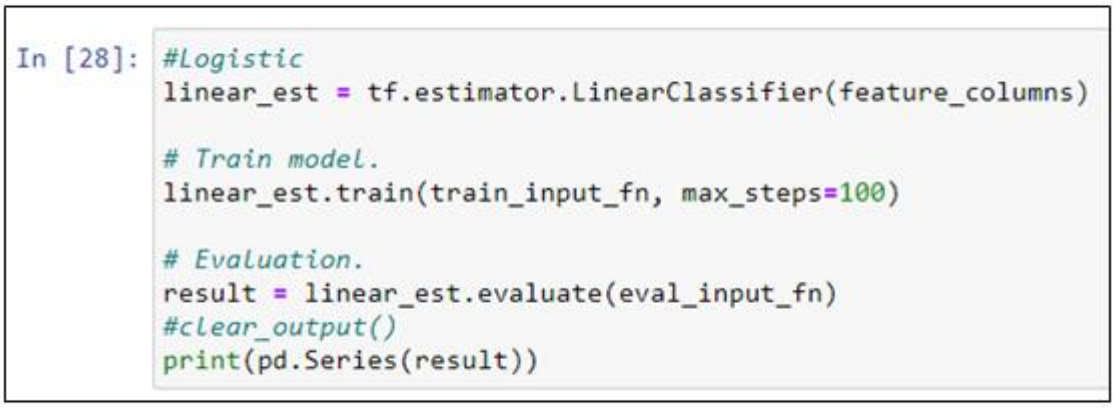


```
In [25]: fc = tf.feature_column
         CATEGORICAL_COLUMNS = ['sex','cp','fbs','restecg','exang','slope','ca','thal']
         NUMERIC_COLUMNS =     ['age','trestbps','chol','thalach','oldpeak']

         def one_hot_cat_column(feature_name, vocab):
           return tf.feature_column.indicator_column(
               tf.feature_column.categorical_column_with_vocabulary_list(feature_name,vocab))
         feature_columns = []

         for feature_name in CATEGORICAL_COLUMNS:
           vocabulary = train_X[feature_name].unique()
           feature_columns.append(one_hot_cat_column(feature_name, vocabulary))

         for feature_name in NUMERIC_COLUMNS:
           feature_columns.append(tf.feature_column.numeric_column(feature_name,dtype=tf.float32))
```

Figure 0-5: Heart disease data preparation

Classification Model

In this work, a linear classifier from TensorFlow was used to classify the targeted variable into one of multiple possible classes. In this case, there were two classes, 0 and 1, so it was considered as a binary (logistic) classification. Figure 0-6 shows the Python code used to build and train the classification model.



```
In [28]: #Logistic
         linear_est = tf.estimator.LinearClassifier(feature_columns)

         # Train model.
         linear_est.train(train_input_fn, max_steps=100)

         # Evaluation.
         result = linear_est.evaluate(eval_input_fn)
         #clear_output()
         print(pd.Series(result))
```

Figure 0-6: Building and training the classification model

Classification Results

In order to evaluate the model, the following parameters were calculated:

- True Positives (TP): the correctly predicted positive ‘target’ that meant the value of the actual variable (the one in the testing data set) was the same as the value of the predicted variable.
- True Negatives (TN): the correctly predicted negative ‘target’ that meant the value of the actual variable was not correct, and the value of the predicted variable was also not correct.
- False Positives (FP): the values when the actual ‘target’ was not correct, but the predicted ‘target’ was correct.
- False Negative (FN): the values when the actual ‘target’ was correct, but the predicted ‘target’ was not.

Once the TP, TN, FP, and FN values were calculated, the following model performance measures such as accuracy, precision, recall, and F1 score, that represent the most popular adopted metrics in classification tasks, were able to be calculated (Chicco and Jurman, 2020). Accuracy was the most important performance measure of the classification model. It was simply a ratio of correctly predicted observation to the total observations and was calculated as follows:

Accuracy = (TP+TN) / (TP+FP+FN+TN).

Model precision represented the ratio of correctly predicted positive observations to the total predicted positive observations and was calculated as follows:

Precision = TP / (TP+FP).

Recall, which is also called Model Sensitivity, represented the ratio of correctly predicted positive observations to all observations in actual class and was calculated using the following formula:

Recall = TP / (TP+FN).

Finally, the F1 score, which was a weighted average of Precision and Recall, was calculated as follows: *F1 = 2 × (Recall × Precision) / (Recall + Precision).*

Table 0-1 shows the classification model performance. The model achieved 87% classification accuracy which was considered successful compared to the 81.96% heart disease classification using DNNs that was recently published by Sharma, Rasool and Hajela (2020).

Table 0-1: Classification model performance

Target	Precision	Recall	F1 Score
0	86%	86%	86%
1	88%	88%	88%

Summary

In this chapter, the results of developing a model to classify heart disease based on certain input features were explained. The classification model was evaluated in terms of accuracy and precision. The data loading, exploration, and preparation for all models were illustrated in detail in this chapter.

Conclusion and Future Work

Introduction

This chapter explains the main conclusions that have been drawn from this work, as well as suggesting future research ideas for further work.

Project Objective Review

This work started with a list of research objectives, and this section will briefly explain how these objectives were conducted and reflect on the challenges that were faced where possible. Below is the list of the objectives with a brief discussion under each one:

- *Conduct a comprehensive review of the state of the art of the applications of AI in disease classification:*

A review was conducted for the techniques, algorithms, tools and terminologies that were used in this work. ANNs were examined, focusing on three common and relevant types, before the best one for this work is indicated. TensorFlow software was introduced, and the chapter then looked at three different data normalisation techniques.

- *Develop a classification (prediction) model by using Python programming language:*
- *Train, evaluate, and test the developed model by using different datasets.*

These research objectives were completed as a classification model was built using TensorFlow in Python. The classification model was 87% accurate in classifying heart disease. The challenges to implementing this model related to data pre-processing, as the medical data could not be directly used because some of it was categorical data that required encoding before it could be used for the model.

- ➢ *Explain the challenges, limitations, and conclusions. In addition, suggestions for future work must be provided at the end:*

The project ends with a list of conclusions as well as some recommendations for future work if there is a possibility for that.

- ➢ *Use real data sets to train and test the neural network-based and classification models, and examine their performance in terms of prediction and classification:*

The data set in this work was obtained from the Center for Machine Learning and Intelligent Systems at the University of California, Irvine, which was donated by Janosi *et al.* (1988).

Project Conclusions

Following the work that was conducted in this project, It has been concluded that:

- ➢ Data cannot be used directly and there must be a transformation process prior to any data processing phase.
- ➢ Data presentation is also a challenge as it requires high levels of integration between systems where all the data needs to be transformed into a unified data frame so it can be understood and presented across all terminals.
- ➢ Working with DNNs involves estimating the right number of neurons in the hidden layers. It is not correct to simply pick a high number randomly, as the results showed that going higher may reduce the prediction accuracy.

Future Work

The work that has been done in this project can be taken further to:

- ➢ Apply different optimisers that are available in TensorFlow and examine the performance of each optimiser. For example, apply "RMSprop" optimiser which is one of the common optimisers in TensorFlow.
- ➢ Gather data from national data repositories if there are any in order to make the project more applicable nationally.

Summary

This chapter summarised the main conclusions of the work that has been conducted, and suggested research ideas for further work.

References

1. Abadi, M. (2016) 'TensorFlow: learning functions at scale', *Proceedings of the 21st ACM SIGPLAN International Conference on Functional Programming*, Nara, Japan, Association for Computing Machinery [Online]. Available at: https://doi.org/10.1145/2951913.2976746.
2. Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G.S., Davis, A., Dean, J. and Devin, M. (2015) 'TensorFlow: large-scale machine learning on heterogeneous systems. Software available from tensorflow. org. 2015', *URL https://www. tensorflow. org*.

3. Al-Rfou, R., Alain, G., Almahairi, A., Angermueller, C., Bahdanau, D., Ballas, N., Bastien, F., Bayer, J., Belikov, A. and Belopolsky, A. (2016) 'Theano: A Python framework for fast computation of mathematical expressions', *arXiv*, p. arXiv: 1605.02688.
4. Al-Shibaany, Z.Y., Hedley, J. and Bicker, R. (2012) 'Design of an adaptive neural kinematic controller for a National Instrument mobile robot system', *2012 IEEE International Conference on Control System, Computing and Engineering*, 23-25 Nov. 2012.
5. Bisong, E. (2019) 'Matplotlib and Seaborn', *Building Machine Learning and Deep Learning Models on Google Cloud Platform.* Springer, pp. 151-165.
6. Buitinck, L., Louppe, G., Blondel, M., Pedregosa, F., Mueller, A., Grisel, O., Niculae, V., Prettenhofer, P., Gramfort, A. and Grobler, J. (2013) 'API design for machine learning software: experiences from the scikit-learn project', *arXiv preprint arXiv:1309.0238*.
7. Chauvin, Y. and Rumelhart, D.E. (1995) *Backpropagation: theory, architectures, and applications.* Psychology press.
8. Chen, D.Y. (2017) *Pandas for everyone: Python data analysis.* Addison-Wesley Professional.
9. Chicco, D. and Jurman, G. (2020) 'The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation', *BMC Genomics*, 21(1), p. 6.
10. Cohen, N., Sharir, O. and Shashua, A. *On the expressive power of deep learning: A tensor analysis.*
11. Collobert, R., Bengio, S. and Mariéthoz, J. 2002. Torch: a modular machine learning software library. Idiap.
12. Deo, R.C., Ghorbani, M.A., Samadianfard, S., Maraseni, T., Bilgili, M. and Biazar, M. (2018) 'Multi-layer perceptron hybrid model integrated with the firefly optimizer algorithm for windspeed prediction of target site using a limited set of neighboring reference station data', *Renewable Energy*, 116, pp. 309-323.
13. Doukim, C.A., Dargham, J.A. and Chekima, A. *Finding the number of hidden neurons for an MLP neural network using coarse to fine search technique.* IEEE.
14. Eldan, R. and Shamir, O. *The power of depth for feedforward neural networks.*
15. Goodfellow, I., Bengio, Y., Courville, A. and Bengio, Y. (2016) *Deep learning.* MIT press Cambridge.
16. Gulli, A. and Pal, S. (2017) *Deep learning with Keras.* Packt Publishing Ltd.
17. Guttag, J. (2016) *Introduction to computation and programming using Python: With application to understanding data.* MIT Press.
18. Horton, E. and Parnin, C. *Gistable: Evaluating the executability of python code snippets on github.* IEEE.
19. Huang, S.-J., Huang, K.-S. and Chiou, K.-C. (2003) 'Development and application of a novel radial basis function sliding mode controller', *Mechatronics*, 13(4), pp. 313-329.
20. Hunter, D., Yu, H., Pukish III, M.S., Kolbusz, J. and Wilamowski, B.M. (2012) 'Selection of proper neural network sizes and architectures—A comparative study', *IEEE Transactions on Industrial Informatics*, 8(2), pp. 228-240.
21. Janosi, A., Steinbrunn, W., Pfisterer, M. and Detrano, R. 1988. Heart Disease Data Set. Center for Machine Learning and Intelligent Systems at the University of California, Irvine.
22. Khandelwal, M. and Singh, T.N. (2006) 'Prediction of blast induced ground vibrations and frequency in opencast mine: A neural network approach', *Journal of Sound and Vibration*, 289(4), pp. 711-725.

23. Kuhlman, D. (2009) *A python book: Beginning python, advanced python, and python exercises*. Dave Kuhlman Lutz.
24. Li, J.-Y., Chow, T.W. and Yu, Y.-L. *The estimation theory and optimization algorithm for the number of hidden units in the higher-order feedforward neural network*. IEEE.
25. Lu, Z., Pu, H., Wang, F., Hu, Z. and Wang, L. *The expressive power of neural networks: A view from the width.*
26. Msiza, I.S., Nelwamondo, F.V. and Marwala, T. *Water Demand Forecasting Using Multi-layer Perceptron and Radial Basis Functions*. 12-17 Aug. 2007.
27. Orłowska-Kowalska, T., Blaabjerg, F. and Rodríguez, J. (2014) *Advanced and intelligent control in power electronics and drives*. Springer.
28. Pang, B., Nijkamp, E. and Wu, Y.N. (2020) 'Deep Learning With TensorFlow: A Review', *Journal of Educational and Behavioral Statistics*, 45(2), pp. 227-248.
29. Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M. and Duchesnay, É. (2011) 'Scikit-learn: Machine Learning in Python', *J. Mach. Learn. Res.*, 12(null), pp. 2825–2830.
30. Pham, B.T., Nguyen, M.D., Bui, K.-T.T., Prakash, I., Chapi, K. and Bui, D.T. (2019) 'A novel artificial intelligence approach based on Multi-layer Perceptron Neural Network and Biogeography-based Optimization for predicting coefficient of consolidation of soil', *CATENA*, 173, pp. 302-311.
31. Seshagiri, S. and Khalil, H.K. (2000) 'Output feedback control of nonlinear systems using RBF neural networks', *IEEE Transactions on Neural Networks*, 11(1), pp. 69-79.
32. Sharma, V., Rasool, A. and Hajela, G. (2020) 'Prediction of Heart disease using DNN', *2020 Second International Conference on Inventive Research in Computing Applications (ICIRCA)*, 15-17 July 2020.
33. Sheela, K.G. and Deepa, S.N. (2013) 'Review on methods to fix number of hidden neurons in neural networks', *Mathematical Problems in Engineering*, 2013.
34. Shibata, K. and Ikeda, Y. *Effect of number of hidden neurons on learning in large-scale layered neural networks*. IEEE.
35. Tamura, S.i. and Tateishi, M. (1997) 'Capabilities of a four-layered feedforward neural network: four layers versus three', *IEEE Transactions on Neural Networks*, 8(2), pp. 251-255.
36. Telgarsky, M. (2016) 'Benefits of depth in neural networks', *arXiv preprint arXiv:1602.04485*.
37. Tosi, S. (2009) *Matplotlib for Python developers*. Packt Publishing Ltd.
38. Xu, S. and Chen, L. (2008) *A novel approach for determining the optimal number of hidden layer neurons for FNN's and its application in data mining*. 23-26 June.
39. Yang, H., Liu, J., Sun, H. and Zhang, H. (2020) 'PACL: Piecewise Arc Cotangent Decay Learning Rate for Deep Neural Network Training', *IEEE Access*, 8, pp. 112805-112813.
40. Zurada, J.M. (1992) *Introduction to artificial neural systems*. West St. Paul.