Neural Network Model for Credit Scoring

By

Amat Alrahman Amin khalif Alshoura

Course Name:

Programming in Python

Supervised by

Ruba Alkhusheiny

2022-2023

Table of Content

List of Tables

List of Figures

Abstract

This project aimed to investigate the potential of neural network models for credit scoring, a task that involves assessing an individual's creditworthiness based on their financial history and other factors. I used the (Give me some credit) dataset and developed a neural network model for credit scoring and evaluated its performance in comparison to traditional machine learning models. The results showed that the neural network model outperformed traditional models in terms of accuracy and F1 score, indicating that it was able to extract valuable features from the data and make more informed predictions. The model's ability to handle a large amount of data and capture non-linear relationships between features is particularly beneficial for credit scoring. The project also highlighted the importance of considering ethical issues and fair decision-making in any credit scoring project. Overall, the results demonstrate the potential of neural networks for credit scoring and encourage further research to improve the model's performance and generalizability.

1. Introduction

1.1 Background

Credit scoring has a long history that dates back to the 1950s, when it was initially created to impartially evaluate borrowers' creditworthiness. At first, credit scores were generated using straightforward statistical models that considered just a few factors, like a person's credit history and debt-to-income ratio.

With the development of more complex statistical models and the advancement of more data sources, the usage of credit scoring has grown both in scope and sophistication over time. With the expansion of the mortgage market in the 1980s and 1990s, the use of credit scores exploded as lenders started utilizing them to assist determine the risk of lending to property buyers.

With the rise of AI and machine learning in recent decades, credit scoring has continued to advance. Nowadays, credit scoring models frequently use a variety of data sources and cutting-edge Artificial Intelligence techniques, such neural networks, to provide more precise and trustworthy predictions about the creditworthiness of borrowers.

A credit scoring system that incorporates artificial intelligence (AI) and neural networks enables more precise and effective risk assessment. A deep learning system known as a neural network, which takes its structural cues from the human brain, is particularly effective at interpreting complicated and multivariate data. A model that can precisely predict a borrower's creditworthiness can be developed by training a neural network on a large collection of credit data. This is accomplished by giving the neural network inputs, including data on finances and credit histories, and training it to produce a default probability. Artificial intelligence (AI) and neural networks can assist lenders in making more educated and trustworthy lending decisions, ultimately resulting in enhanced loan availability and risk management for borrowers.

A neural network credit score model is a deep learning model used to forecast a person's or company's creditworthiness. Creating a model that can precisely forecast the probability of a borrower defaulting on a loan

is the goal of this project. This is a helpful tool for lenders since it enables them to choose which borrowers to lend to and under what conditions.

1.2 Artificial Neural Networks (ANN)

Neural Networks are the functional unit of Deep Learning, an artificial neuron is a computational model inspired by natural neurons. Natural neurons receive impulses via synapses on their dendrites or membrane. The neuron is engaged and produces a signal through the axon when the signals it receives are powerful enough (defy a predetermined threshold). This signal could be transferred to another synapse, activating further neurons.

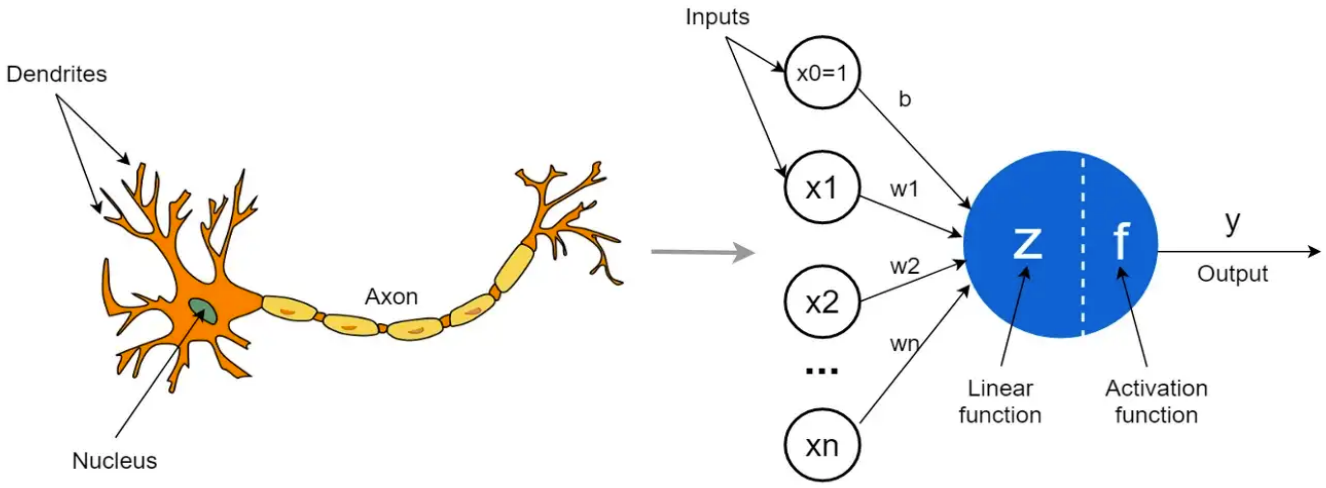


Figure 1: Brain neuron and Artificial Neurons

When modeling artificial neurons, the intricacy of natural neurons is significantly abstracted. These essentially consist of inputs (like synapses) that are multiplied by weights (the strength of the relevant signals), which are then computed by a mathematical function that determines the activity of the neuron. The identity of another function computes the output of the artificial neuron (in dependence on a certain threshold). Information processing is accomplished by combining artificial neurons in ANNs.

The strength of the input that is multiplied by an artificial neuron increases with its weight. Since weights can also be negative, we can say that the negative weight is decreasing the signal. The neuron's computation will vary depending on the weights. An artificial neuron's weights can be changed to produce the desired output for a given set of inputs. However, it would be difficult to determine all the necessary weights manually when we had an ANN with hundreds or thousands of neurons. However, there are algorithms that can change the weights of the ANN to get the desired output from the network.

There are many different ANN types and applications. Hundreds of different models that are regarded as artificial neural networks (ANNs) have been built since the first neural model by McCulloch and Pitts (1943). The distinctions between them could be found in the topology, learning techniques, accepted values, functions, etc. In many hybrid models, each neuron has additional features.

1.2.1 Key components of the neural network architecture

The neurons that make up the neural network architecture imitate the actual behavior of the brain.

The parts of a neuron are listed below

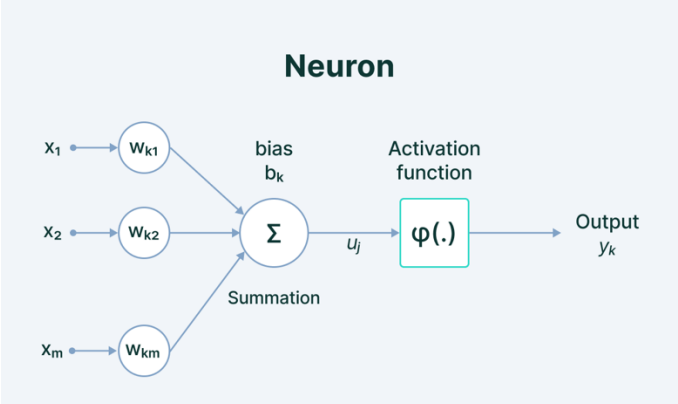


Figure 2: Neuron in artificial neural network

1. **Input** is the collection of features fed into the model throughout the learning process.
2. **Weight's** primary purpose is to prioritize the characteristics that contribute most to learning. It accomplishes this by applying scalar multiplication on the weight matrix and input value.
3. **The transfer function**'s role is to combine several inputs into a single output value so that the activation function can be used. It is accomplished by simply adding up all the transfer function's inputs.
4. **The activation function** brings non-linearity into the operation of perceptron in order to account for inputs with variable linearities. Without it, the output would simply be a linear combination of the input values, and the network would not be capable of handling non-linearity.
5. **Bias** function is to change the value that the activation function produces. Its function is equivalent to a constant in a linear function.

Layers in neural networks

The Keras API has a variety of layer types, each with a distinct purpose. Typical types of layers include:

Dense: A dense layer is one in which every node is fully connected to every other node in the layer below it. For classification problems, neural networks frequently employ this kind of layer.

Conv2D: Two-dimensional data, such as photographs, are processed using a 2D convolutional layer. In order to extract features from the input data, this layer applies a predetermined number of filters, each with a unique set of weights.

MaxPooling2D: A 2D max pooling layer is used to down-sample the spatial dimensions of the data by taking the maximum value from each pooling window. This can help to reduce the computational cost of the model and improve its performance.

Flatten: A flatten layer is used to flatten the input data into a one-dimensional array, which can then be input into a dense layer. This is often used when transitioning between convolutional and dense layers in a network.

LSTM: A recurrent layer called an LSTM (Long Short-Term Memory) layer can identify long-term dependencies in sequential data. As a result, it excels at tasks like time series prediction and language modeling.

These are just a few illustrations of the several kinds of layers that the Keras API offers. There are numerous additional layer types available, each created for particular functions and jobs.

1.2.2 Popular application of artificial neural networks

There are many applications on ANNs such as:

1. Credit scoring
2. Stock Market Prediction
3. Healthcare
4. Weather Forecasting
5. Fraud Detection

And many applications use other deep learning architectures like Convolutional Neural Networks (CNNs), Residual Networks (ResNet), Recurrent Neural Networks (RNNs), Echo State Networks (ESN), Generative Adversarial Network (GAN) and Transformer Neural Networks.

1.3 Project objective

In my project, I will be using a Multilayer perceptron MLP neural network to make a prediction of creditworthiness by predicting if the person will experience 90 days past due delinquency or worse in the next two years

2. Project overview

The process of building a neural network credit scoring model involves several steps. First, Gathering and preprocess a large dataset of credit data. This data may include information about the borrower's credit history, income, and other factors that are relevant to their creditworthiness. Next, appropriate neural network architecture must be chosen and train the model using the preprocessed data. This may involve adjusting the model's parameters and fine-tuning it to achieve optimal performance.

Once the model has been trained, it can be tested on a separate dataset to evaluate its accuracy. If the model performs well, it can be deployed in a production environment, where it can be used to make real-time credit decisions.

Overall, a neural network credit scoring model has the potential to greatly improve the efficiency and accuracy of credit decision-making, helping lenders to better manage risk and enabling more borrowers to access credit.

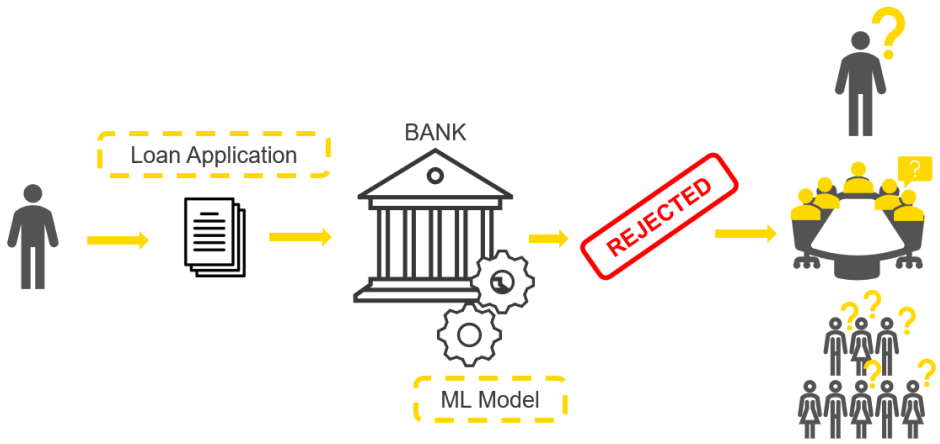


Figure 3: Credit decision using artificial intelligence

Today credit scoring is almost used by all the banks that approve credit card applications. such as Lloyds Banking Group and Security Pacific Bank (SPB).


In Jordan, CRIF Jordan was founded to offer credit information services. The Central Bank of Jordan granted it a license to begin functioning in 2015, and it began work so formally in 2016. The work of CRIF Jordan is governed by the Credit Information. Additionally, through a dedicated division inside CBJ, the Central Bank of Jordan regulates and keeps an eye on the Company's operations.

CRIF Jordan presently offers its services to "Credit Providers" operating in Jordan, as defined by the credit information law. Additionally, CRIF Jordan collects credit information from the sources listed in the Credit Information Law's articles. Banks, leasing companies, financial institutions set up under unique legislation, micro-finance businesses, and others are some of these sources. Since beginning operations in 2016, CRIF Jordan has benefited several banks, financial institutions, and telecoms companies.

The Scoring Assessment System by CRIF Jordan is a tool used to forecast the likelihood that a client will be at least one payment or obligation late in the 12 months following the client's inquiry process. It is based on statistical techniques.

3. Methodology

The task of the project is to predict a person's or company's creditworthiness. Creating a Multilayer Perceptron (MLP) model that can precisely forecast the probability of a borrower defaulting on a loan.

An extremely popular approach in credit scoring systems is using classical machine learning models such as decision trees and logistic regression these models give a good prediction but with the consideration of the sensitivity of loan risks to lenders, building new approaches with higher accuracy and higher ability to learn as the bigger data is a game changer, As shown in the following figure, deep learning models are superior to classical machine learning models.

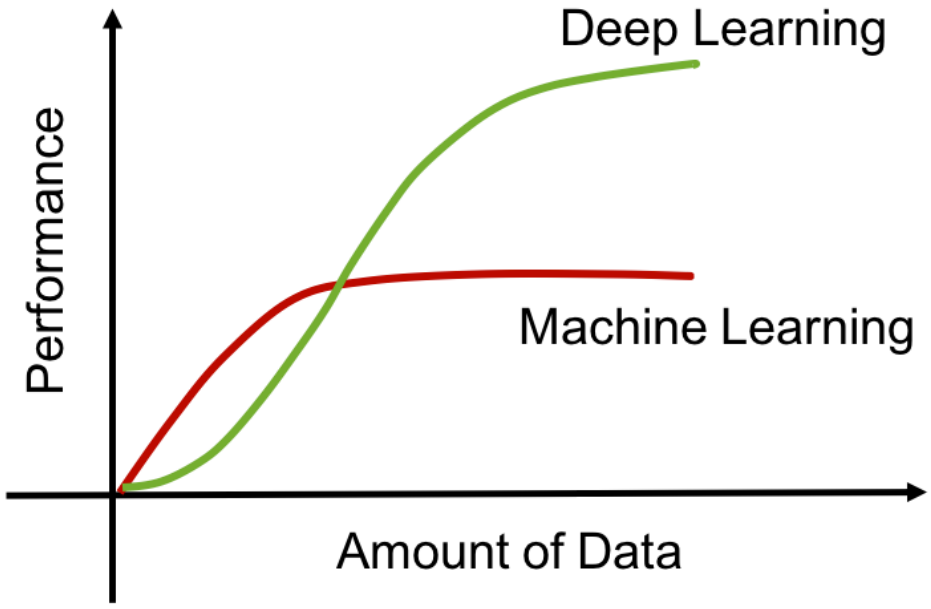


Figure 4: Machine learning vs. Deep learning

3.1 Multilayer perceptron (MLP)

A Multilayer Perceptron (MLP) is a fully connected multi-layer neural network.

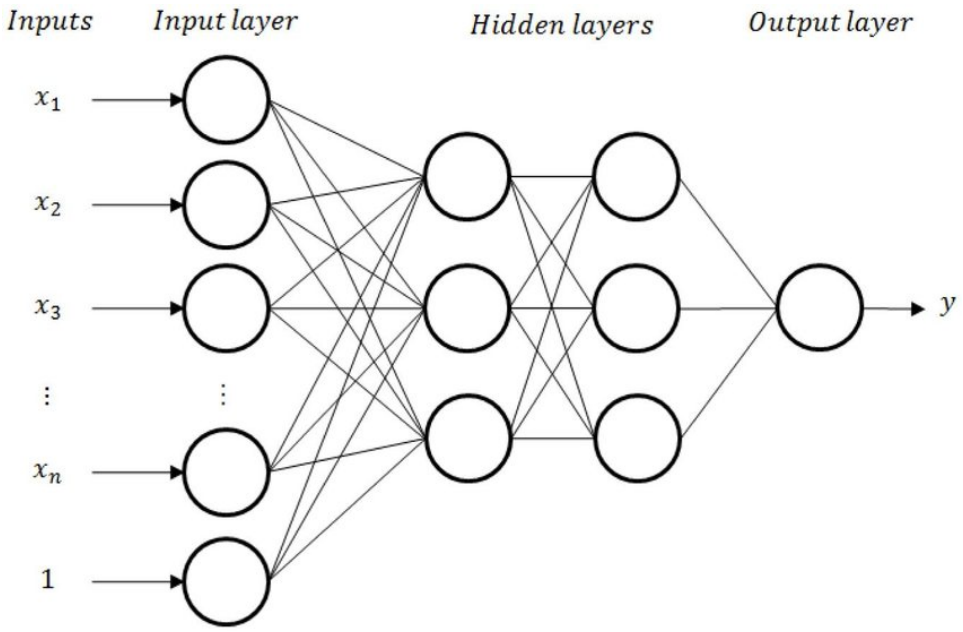


Figure 5: Multilayer Perceptron (MLP)

A layer is made up of several neurons stacked in a row, while a multi-layer neural network is made up of many layers that are stacked next to each another.

I have explained the main components of this of structure below.

Input layer

I loaded a CSV file into the input layer to provide the model with the data. It is the only layer that is visible in the entire design of a neural network that transfers all the information from the outside world without any processing, in my project the input layer size will be 10 since I have 10 features in the dataset, the dataset will be explained in the dataset section

Hidden layers

Hidden layers are intermediate layers that do all calculations and extract features from data. There can be numerous interconnected hidden layers that account for searching for various hidden features in the data, in my project I built 3 hidden layers with 100-125 –100 neurons with respect to the order.

Output layer

The output layer uses data from earlier hidden layers and the model's learnings to make a final prediction. The layer where we obtain the end outcome is the most crucial one. Classification/regression models typically have a single node in the output layer. However, it depends entirely on the nature of the problem at hand and how the model was created, In the case of my project the output layer is a single neuron since it is a binary classification problem

3.2 Used packages

NumPy

NumPy (**Numerical Python) is** an open-source library in Python used extensively in various fields of science and engineering. It is the go-to standard for working with numerical data in Python. Many popular data science and scientific Python packages, including Pandas, SciPy, Matplotlib, scikit-learn, and scikit-image, make extensive use of the NumPy API. The library offers multidimensional array and matrix data structures, with the ndarray object providing efficient methods for operations. NumPy enables a wide range of mathematical operations on arrays and provides powerful data structures that ensure efficient calculations with arrays and matrices. Additionally, it features an extensive library of high-level mathematical functions for working with these data structures.

Pandas

Pandas is a popular open-source Python package for machine learning, data science, and data analysis activities; it is designed primarily for fast and efficient operation. It offers a variety of data structures and operations for working with time series and numerical data. On top of the NumPy library, the Pandas library is constructed. Pandas is quick and offers users outstanding performance and productivity.

when it comes to tasks that involve manipulating data Pandas simplifies a lot of the time-consuming, repetitive processes, including:

1) cleaning up data
2) data saving and loading
3) Imputation of missing data
4) visualization of data
5) Statistic evaluation
6) Normalization of data
7) joins and merges

scikit-learn

Scikit-learn is a popular machine learning library for Python. It is built on top of other popular Python libraries, such as NumPy and pandas, and provides a simple and consistent interface for working with a variety of different machine learning models.

Scikit-learn includes a wide range of tools for tasks such as classification, regression, clustering, dimensionality reduction, and model selection. It also includes tools for preprocessing and feature extraction, such as data normalization and one-hot encoding.

One of the main benefits of scikit-learn is its consistency in the interface, making it easy to switch between different models and preprocessing techniques. It also provides a number of useful tools for evaluating the performance of models, including cross-validation and performance metrics.

Scikit-learn is widely used in the data science community, and is a popular choice for tasks such as classification, regression, clustering, and dimensionality reduction. It is also commonly used as a tool for prototyping and building machine learning models in industry and research.

TensorFlow

TensorFlow is an open-source software library that excels in high-performance numerical computation. Its flexible architecture enables easy deployment on a variety of platforms, including CPUs, GPUs, TPUs, and devices ranging from desktops to clusters of servers to mobile and edge devices. Initially developed by researchers and engineers from the Google Brain team within Google's AI organization, TensorFlow offers robust support for machine learning and deep learning. Additionally, its flexible numerical computation core is widely used in many other scientific fields.

Keras Keras

Keras is a high-level deep learning API created by Google and written in Python, that simplifies the process of building and training neural networks. It also allows for the use of multiple backends for neural network computation, giving you the option to switch between different frameworks such as TensorFlow, Theano, PlaidML, MXNet, and CNTK (Microsoft Cognitive Toolkit). Out of these options, TensorFlow has officially adopted Keras as its high-level API. Keras is integrated into TensorFlow, providing built-in modules for all neural network computations and enabling fast deep learning. At the same time, the TensorFlow Core API allows for custom computations involving tensors, computation graphs, sessions, and more, providing total flexibility and control over your application and allowing for quick implementation of your ideas.

matplotlib

Matplotlib is a popular Python library for creating various types of visualizations. It is known for its speed and ability to export visualizations in various image formats. With Matplotlib, one can create a wide range of plots, including line graphs, scatter plots, histograms, bar charts, and 3D charts. Additionally, it serves as the foundation for other Python libraries such as Pandas and Seaborn, which make it more convenient to access Matplotlib's features. The Matplotlib project was created by John Hunter in 2002 during his post-doctoral study in Neurobiology to display Electrocorticography data from epileptic patients. The library has become widely used and was even used to visualize data during the 2008 landing of the Phoenix spacecraft.

3.3 Project steps

Building a neural network credit scoring system typically involves the following steps.

3.3.1Collect and preprocess the data

This involves gathering the data that I will use to train and test my model and performing any necessary preprocessing steps to prepare the data for analysis. This may include tasks such as cleaning the data, imputing missing values, and normalizing numerical features.

Dataset:

the dataset is taken from a competition on Kaggle website as csv file with size of (150K,11), the purpose of this Featured Prediction Competition is to build a model to Improve on the state of the art in credit scoring by predicting the probability that somebody will experience financial distress in the next two years.

The following table illustrate the features in the dataset:

Variable Name	Description	Type	Notes
SeriousDlqin2yrs	Person experienced 90 days past due delinquency or worse	Y/N	Around 6% of samples defaulted (Y)
RevolvingUtilizationOfUnsecuredLines	Total balance on credit cards and personal lines of credit except real estate and no installment debt like car loans divided by the sum of credit limits	percentage	-
age	Age of borrower in years	integer	-
NumberOfTime30-59DaysPastDueNotWorse	The number of times borrower has been 30-59 days past due but no worse in the last 2 years.	integer	-
DebtRatio	Monthly debt payments, alimony, living costs divided by monthly gross income	percentage	-
Monthly Income	Monthly income	real	Has 29731 (19.82%) null values
NumberOfOpenCreditLinesAndLoans	Number of Open loans (installment like car loan or mortgage) and Lines of credit (e.g., credit cards)	integer	-
NumberOfTimes90DaysLate	The number of times the borrower has been 90 days or more past due.	integer	-
NumberRealEstateLoansOrLines	Number of mortgage and real estate loans including home equity lines of credit	integer	-

NumberOfTime60-89DaysPastDueNotWorse	The number of times borrower has been 60-89 days past due but no worse in the last 2 years.	integer	-
NumberOfDependents	Number of dependents in family excluding themselves (spouse, children etc.)	integer	Has 3924 (2.61%) null values

Table 1: The features in the dataset

The following is a more detailed explanation of some of the features of the dataset

personal lines of credit

A personal line of credit is a form of loan that allows you to withdraw funds as needed for a specified length of time. You can borrow up to a predetermined amount, and you can pay it back through checks or bank transfers. When you borrow money, you instantly start paying interest, and you keep doing so until the loan has been repaid. The cap goes from $1,000 to $100,000. You frequently make small monthly payments with a credit card.

Mostly, these loans are not secured by any form of property, so if you are unable to pay them back, the lender cannot seize any of your belongings. Nevertheless, lenders might agree to let you use collateral in return for a lower interest rate. The cost of having this loan also includes yearly or monthly maintenance fees, additional penalties for late or returned payments, and other expenditures. You can use a personal line of credit for everything you require if you don't go over your credit limit. For home equity loans and business expenses, there are other comparable loans that need security or are limited to certain expenses.

age

Age can be an important factor in credit scoring because it is often used as an indicator of credit experience and stability. Generally, older borrowers are considered more likely to have a longer credit history and a more stable financial situation, which can be viewed positively by lenders. Additionally, older borrowers are more likely to have established a good credit history and to have a lower risk of default. On the other hand, younger borrowers

may be viewed as having less credit experience and a higher risk of default, which can negatively impact their credit score.

Number of dependents

The number of dependents of a borrower is considered an important factor in credit scoring because it can indicate the borrower's financial stability. People with more dependents may have higher living expenses and financial obligations, which may impact their ability to repay the loan. They may also have a lower income and less disposable income, which may make it difficult for them to repay the loan. This is not always the case, as a borrower with more dependents may have a higher income and a stable job that will help them repay the loan. As a result, credit scoring systems consider this factor as a critical characteristic in determining a borrower's creditworthiness.

Data preprocessing

Starting by importing necessary libraries as shown in figure below

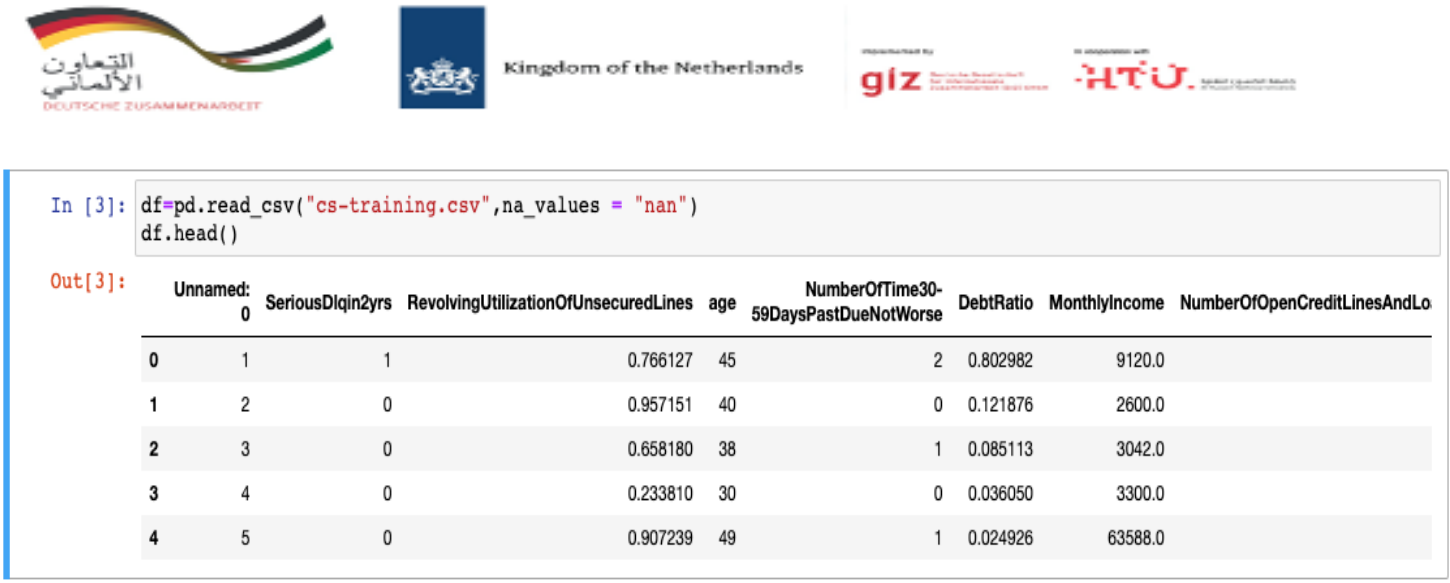
```
import pandas as pd
import numpy as np

import tensorflow as tf
import keras
from keras.models import Sequential
from keras.layers import InputLayer, Dense
from keras.callbacks import ModelCheckpoint

import matplotlib.pyplot as plt
from numpy import asarray
```

Figure 6: importing necessary libraries

The next step is reading the data using the read_csv() command to define the data frame then the head command is used to have a glence of the data frame



```
In [3]: df=pd.read_csv("cs-training.csv",na_values = "nan")
        df.head()
```

Out[3]:

	Unnamed: 0	SeriousDlqin2yrs	RevolvingUtilizationOfUnsecuredLines	age	NumberOfTime30-59DaysPastDueNotWorse	DebtRatio	MonthlyIncome	NumberOfOpenCreditLinesAndLo
0	1	1	0.766127	45	2	0.802982	9120.0	
1	2	0	0.957151	40	0	0.121876	2600.0	
2	3	0	0.658180	38	1	0.085113	3042.0	
3	4	0	0.233810	30	0	0.036050	3300.0	
4	5	0	0.907239	49	1	0.024926	63588.0	

Figure 7: reading the data

Then I want to Explore and the data, this step involves analyzing the data to understand its characteristics, as shown in the following figures.

Check for duplications and using info command to know the number of non-null data and data type for each feature (column)

```
In [4]: #check for duplicated data
        df.duplicated().sum()

Out[4]: 0

In [5]: #handeling missing values
        print(df.info())

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150000 entries, 0 to 149999
Data columns (total 12 columns):
 #   Column                                Non-Null Count   Dtype
---  ------                                --------------   -----
 0   Unnamed: 0                            150000 non-null  int64
 1   SeriousDlqin2yrs                      150000 non-null  int64
 2   RevolvingUtilizationOfUnsecuredLines  150000 non-null  float64
 3   age                                   150000 non-null  int64
 4   NumberOfTime30-59DaysPastDueNotWorse  150000 non-null  int64
 5   DebtRatio                             150000 non-null  float64
 6   MonthlyIncome                         120269 non-null  float64
 7   NumberOfOpenCreditLinesAndLoans       150000 non-null  int64
 8   NumberOfTimes90DaysLate               150000 non-null  int64
 9   NumberRealEstateLoansOrLines          150000 non-null  int64
 10  NumberOfTime60-89DaysPastDueNotWorse  150000 non-null  int64
 11  NumberOfDependents                    146076 non-null  float64
dtypes: float64(4), int64(8)
memory usage: 13.7 MB
None
```

Figure 8: analyzing the data

As seen in the previous figure I have missing values that I need to know the percentage of it to determine the best strategy to impute, the imputation process was executed after splitting the data

```
In [6]: #Null values tabel

null_val_sums = df.isnull().sum()
pd.DataFrame({"Column": null_val_sums.index, "Number of Null Values": null_val_sums.values,
              "Proportion": null_val_sums.values / len(df) })
```

Out[6]:

	Column	Number of Null Values	Proportion
0	Unnamed: 0	0	0.000000
1	SeriousDlqin2yrs	0	0.000000
2	RevolvingUtilizationOfUnsecuredLines	0	0.000000
3	age	0	0.000000
4	NumberOfTime30-59DaysPastDueNotWorse	0	0.000000
5	DebtRatio	0	0.000000
6	MonthlyIncome	29731	0.198207
7	NumberOfOpenCreditLinesAndLoans	0	0.000000
8	NumberOfTimes90DaysLate	0	0.000000
9	NumberRealEstateLoansOrLines	0	0.000000
10	NumberOfTime60-89DaysPastDueNotWorse	0	0.000000
11	NumberOfDependents	3924	0.026160

Figure 9: Missing values

Then Splitting the Data into independent (X) and dependant (y) features as showen in the following code

```
In [7]: X=df.iloc[:,2:].values
        y=df.iloc[:,1].values
```

Figure 10: Splitting the Data

Splitting the data is a very important step, it determines the independent (target) coulmn(s) in my project it's the SeriousDlqin2yrs since I want to build a Multilayer perceptron that predicts wither the person who applied for the credit will Person experience 90 days past due delinquency or wors.

And the independent features are the data I feed to the model to make predictions.

Handling missing data

As shown in the missing values figure Monthly Income column has 29731 (19.82%) null values and NumberOfDependents column has 3924 (2.61%) null values, by considering the nature of the data in the two columns the best imputing strategy for both is the median strategy as shown in the following figure.

```
In [8]: # handeling missing values
        from sklearn.impute import SimpleImputer


        imp_mean=SimpleImputer(missing_values= np.nan, strategy="median")
        X[:,[4,9]]=imp_mean.fit_transform(X[:,[4,9]])
```

Figure 11: imputing missing values

The last step in preprocessing the data is normalization, but what is data normalization? And why do I need to do this step?

Data normalization is the process of scaling input data values to a common range, typically between 0 and 1, or sometimes between -1 and 1. This is done to ensure that all input features are on a similar scale, which can help improve the performance of neural networks.

Without normalization, features with larger value ranges may dominate the learning process, while features with smaller value ranges may have little impact. This can lead to the network learning suboptimal weights for the features and can slow down the convergence of the training process.

Normalization also helps to stabilize the training process and can prevent issues such as vanishing gradients or exploding gradients that can occur when the input data is not properly scaled.

In summary, normalization is important for neural networks because it helps the network to learn more effectively by ensuring that all input features are on a similar scale, which can improve the performance of the network and stabilize the training process.

The data normalization was performed using Scikit Learns Minmax Scaler

Data Normalization

```
In [9]: #MinMax scaling
        from sklearn.preprocessing import MinMaxScaler
        # define min max scaler
        scaler = MinMaxScaler()
        X = scaler.fit_transform(X)
```

Figure 12: data normalization

3.3.2 Split the data into training and testing sets

It is common practice to split the data into a training set, which is used to train the model, and a testing set, which is used to evaluate the model's performance. this can be done by the following code

Train Test Split

```
In [10]: from sklearn.model_selection import train_test_split
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,random_state = 0)
```

Figure 13: Train-test split

As shown in the figure above the test size is equal to 20% of the dataset this leaves 80% of the data for training the model. Then I have performed the model on the test data to evaluate the performance of the model.

3.3.3 Build the model

This involves designing and implementing the neural network model using keras with TensorFlow library in the back end , The model is then trained on the training data using an optimization algorithm I used stochastic gradient descent (SGD) to adjust the model's parameters and minimize the error between the predicted and actual labels.

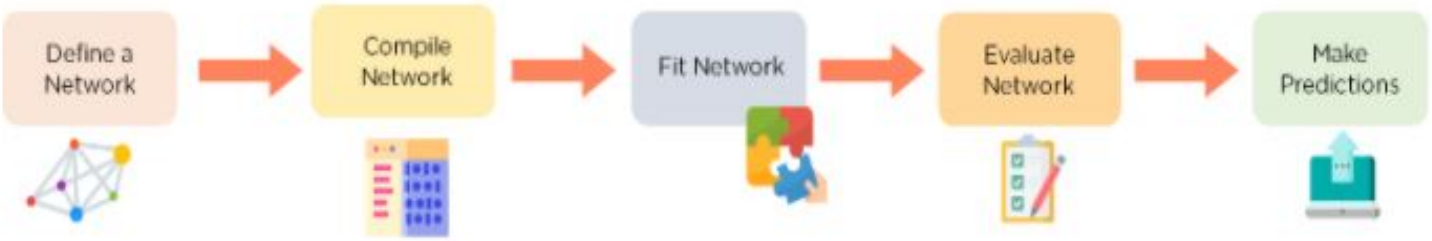


Figure 14: Building a model in keras

1. Defining the architecture of the model

In this step, I have defined the different layers in my model and the connections between them. Since Keras has two main types of models: Sequential and Functional models. I have chosen the Sequential model, then all I have done is defined each layer in the model, an input layer with the size of (None, 10) (the length of the X values) and the output layer is a single neuron layer since it is a binary classification problem, the three hidden layers are set to 100-125-100 respectively, there is no thump rule for the number of neurons in the hidden layers but by trying I found that this is the most stable structure. The following figure shows the code of the model architecture.

```
In [15]: # define hidden layers and neuron in each layer
         input_neurons = X_train.shape[1]
         output_neurons = 1
         number_of_hidden_layers = 2
         neuron_hidden_layer_1 = 100
         neuron_hidden_layer_2 = 125
         neuron_hidden_layer_3 = 100

         # activation function of different layers

         #I have picked relu as an activation function for hidden layers, And I have used sigmoid activation function in the fir

         model = Sequential()
         model.add(InputLayer(input_shape=(input_neurons,)))
         model.add(Dense(units=neuron_hidden_layer_1, activation='relu'))
         model.add(Dense(units=neuron_hidden_layer_2, activation='relu'))
         model.add(Dense(units=neuron_hidden_layer_3, activation='relu'))
         model.add(Dense(units=output_neurons, activation='sigmoid'))

         # summary of the model
         print(model.summary())
```

Figure 15: Model architecture.

The following table show the layers types, output shape, the number of parameter and Activation function, I will explain how the number of parameter is calculated

Layer (type)	Output Shape	Param #	Activation function
Input layer (InputLayer)	(None, 10)	10	relu
First hidden layer (Dense)	(None, 100)	1100	relu
Second hidden layer (Dense)	(None, 125)	12625	relu
Third hidden layer (Dense)	(None, 100)	12600	relu
Output layer (Dense)	(None, 1)	101	sigmoid

Table 2: Layer information

number of parameters between input and first hidden layer= input_neurons*neuron_hidden_layer_1
number of parameters between input and first hidden layer= input_neurons*neuron_hidden_layer_1 + neuron_hidden_layer_1 # Adding the bias for each neuron of first hidden layer
number of parameters between first and second hidden layer= neuron_hidden_layer_1*neuron_hidden_layer_2 + neuron_hidden_layer_2
number of parameters between second hidden and output layer= neuron_hidden_layer_2*output_neurons + 1

Table 3: the calculation of the number of parameters

Activation functions

Relu

The rectified linear unit (ReLU) is a type of activation function commonly used in neural networks, particularly in CNNs and MLPs. It is defined as:

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

where x is the input value.

ReLU is a piece-wise linear function that outputs the input value if it is positive and 0 if it is negative.

$$f(x) = \{x\,,\ x > 0 \ \ and\, 0\,,\ x < 0\ \}$$

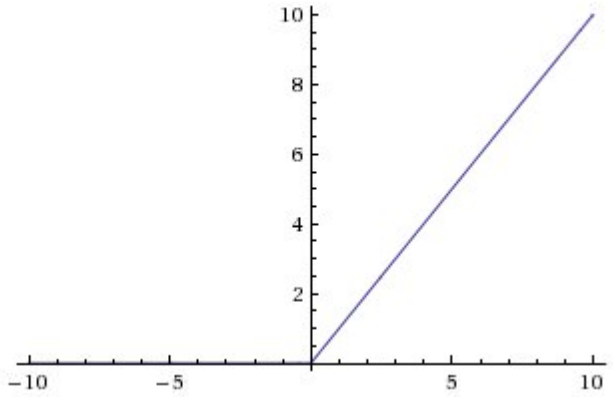


Figure 16: relu activation function

ReLU function is not fully interval-derivable, but we can take sub-gradient, as shown in the figure below.

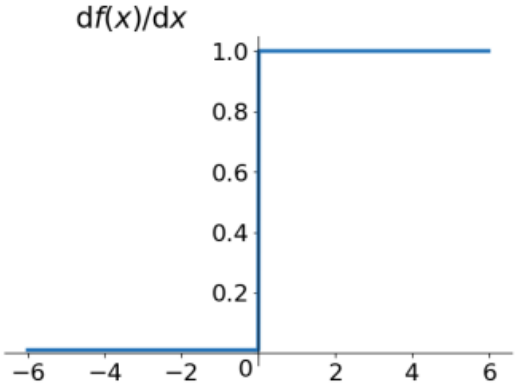


Figure 17: relu activation function derivation

ReLU is widely used in deep learning because it is computationally efficient and does not require complex mathematical operations like sigmoid or tanh functions. Additionally, ReLU can improve the training speed of a

neural network and reduce the risk of the model getting stuck in the saturated state. This function helps the model to converge faster and improve the performance of the model.

ReLU has limitations, in some cases it can produce a problem called “Dying ReLU” where the weights on the negative side of the activation function will update towards zero, resulting in a model that won’t learn from negative inputs. To overcome this problem, Leaky ReLU, PreLU, and RreLU are popular variants of ReLU, but my project is a binary classification model so this will not be a problem.

Sigmoid

A sigmoid function is a common type of activation function in neural networks. It’s a mathematical function that converts any input value to an output value between 0 and 1. The function is named “sigmoid” because it resembles a “S” curve.

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

where x is the input value and e is the mathematical constant (approximately 2.71828). The function accepts any real value as input and returns a value between 0 and 1. As a result, it may be used to describe probability or binary classification issues.

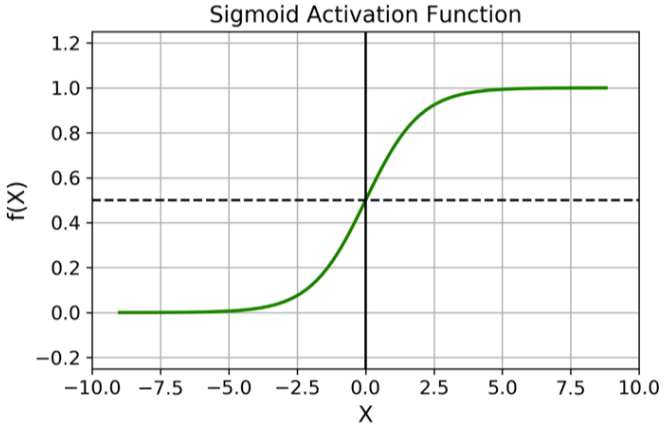


Figure 18: sigmoid function

The sigmoid function is a type of logistic function, which is popular in deep learning because it has a pleasant mathematical feature that makes it simple to optimize using gradient descent. The sigmoid function and its derivatives are simple to calculate, and it is also a smooth function, making it an excellent candidate for optimization.

The derivative of the sigmoid function is shown in the following figure:

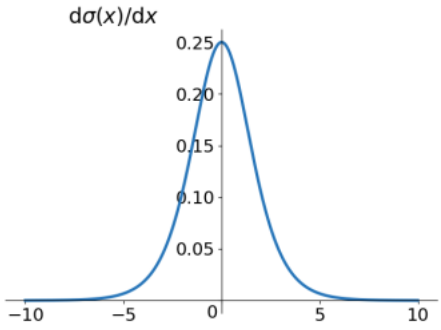


Figure 19: sigmoid function derivative

2. Compile the network:

Compiling code means converting it into a format that a machine can understand. The model.compile() method in Keras performs this function. To compile the model, I define the loss function, which calculates our model's losses, the optimizer, which reduces the loss, and the metrics, which are used to determine our model's accuracy.

Compiling the model

```
In [16]: model.compile(loss='binary_crossentropy',optimizer="SGD",metrics=['accuracy'])
```

Figure: Compiling code

Binary cross-entropy loss function

Binary cross-entropy loss is a loss function commonly used in binary classification tasks, where the goal is to predict one of two possible outcomes (e.g. true/false, 0/1). The function compares the predicted probability for the true class (labeled as 1) to the actual outcome, and calculates the loss as the negative log-likelihood of the true class. The smaller the loss, the better the model's prediction. The formula is defined as:

$$Loss = -(y * \log(p) + (1 - y) * \log(1 - p))$$

Where:

- y is the true label (0 or 1)
- p is the predicted probability of the positive class
- log is the natural logarithm

What are gradient descent and stochastic gradient descent?

Gradient descent (GD) optimization

The weights are incrementally updated after each epoch (= pass over the training dataset) using the Gradient Decent optimization algorithm.

By taking a step in the opposite direction of the cost gradient, the magnitude and direction of the weight update are computed.

$$\Delta wj = -\eta \frac{\partial J}{\partial wj},$$

where η is the learning rate. The weights are then updated after each epoch by the following update rule:

$$w := w + \Delta w,$$

where Δw is a vector that contains the weight updates of each weight coefficient w, which are computed as follows:

$$\Delta wj = -\eta \frac{\partial J}{\partial wj} = -\eta \sum^{i} \big(t \arg e\, t(i) - output(i)\big)(-x(i)j) = \eta \sum^{i} \big(t \arg e\, t(i) - output(i)\big)x(i)j.$$

Gradient Descent optimization can be visualized as a hiker (the weight coefficient) attempting to descend a mountain (cost function) into a valley (cost minimum), with each step determined by the steepness of the slope (gradient) and the hiker's leg length (learning rate). With a single weight coefficient in a cost function, we can illustrate this concept as follows:

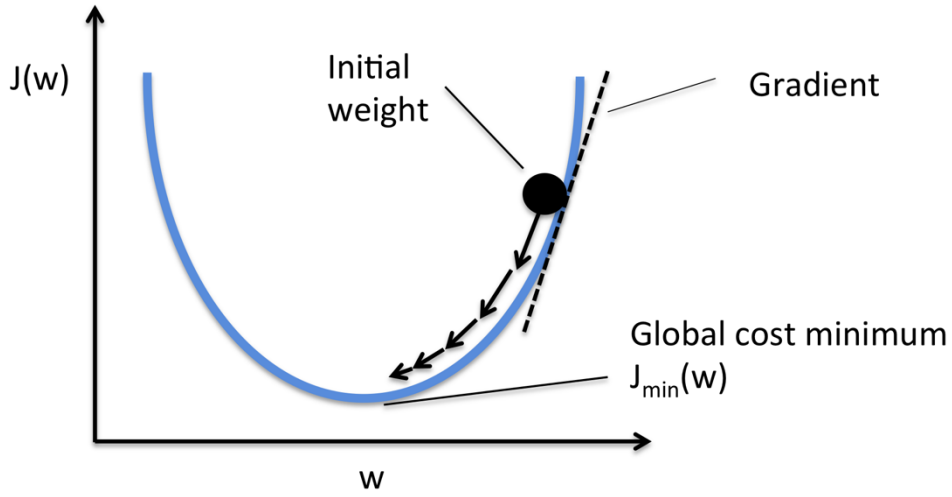


Figure 20: Gradient Descent optimization

Stochastic Gradient Descent

(SGD) is an optimization algorithm that finds the smallest value of a function. It is a gradient descent algorithm that is useful for large-scale optimization problems because it updates model parameters based on a single training example at a time (as opposed to using the entire dataset to compute the gradient). SGD can converge

faster and be more computationally efficient than other optimization algorithms because of this. The algorithm begins with a fixed set of parameters and iteratively updates them by moving in the opposite direction of the gradient of the objective function with respect to the parameters. A learning rate determines the step size, which controls how large of a step to take in the opposite direction of the gradient.

3. Fit the network:

Fitting the model to the data after compiling with model.fit(). This is used to train the model on the data.

Passing the independent and dependent features for training set for training the model as follows:

- validation data will be evaluated at the end of each epoch
- setting the epochs as 100
- storing the trained model in model_history variable which will be used to visualize the training process

ModelCheckpoint callback is used in conjunction with model.fit() training to store a model or weights (in a checkpoint file) at specified interval so that the model or weights may be loaded later to continue training from the saved state.

As shown in the following figure,

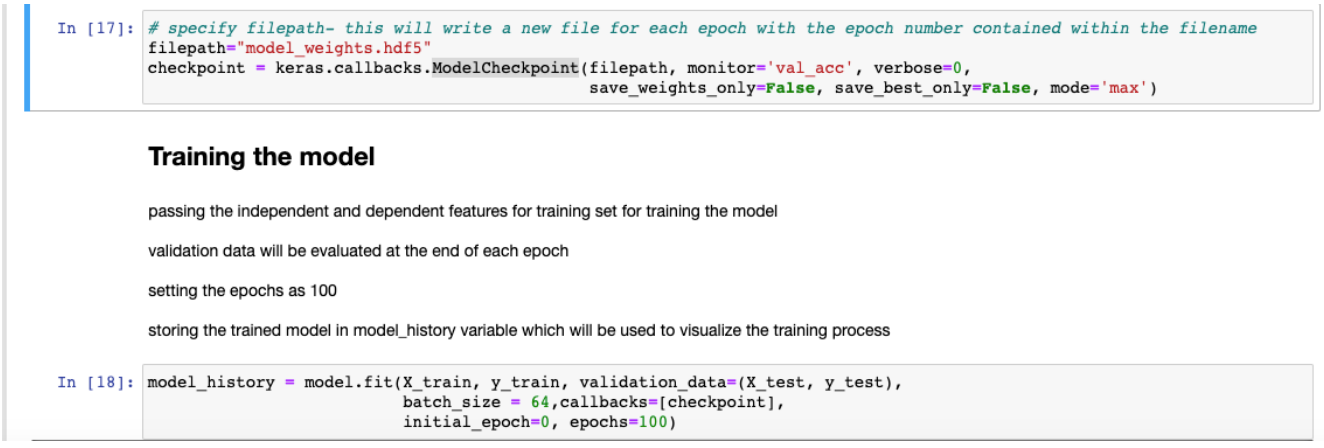


Figure 21: Fitting the model

*The training outputs are in the appendix section.

4. Evaluate the model:

Once the model is trained, it is tested on the testing data to evaluate its performance. This involves comparing the model's predictions to the actual labels and calculating metrics such as accuracy. Using the following code shown in the figure

```
In [19]: final_loss, final_accuracy = model.evaluate(X_test, y_test, verbose=0)
         print('Final Loss: {}, Final Accuracy: {}'.format(final_loss, final_accuracy))

         Final Loss: 0.19615095853805542, Final Accuracy: 0.935366690158844
```

Figure 22: Evaluate the model

The Final Accuracy= 0.935366690158844

5. Make Predictions:

Using model.predict() to make predictions using my model on test data.

Evaluating model performance on validation set

```
In [20]: # getting predictions for the validation set
         prediction = model.predict(X_test)
         prediction = (prediction > 0.5)
         MOH=np.concatenate((prediction.reshape(len(prediction),1), y_test.reshape(len(y_test),1)),1)

         938/938 [==============================] - 2s 2ms/step
```

Figure 23: Make Predictions

Then using accuracy _score form sklearn.metrics to calculate the accuracy on validation set which is equal 0.9353666666666667, the code is as following:

```
In [23]: # calculating the accuracy on validation set
         from sklearn.metrics import accuracy_score
         accuracy= accuracy_score(y_test, prediction)
         print(accuracy)

         0.9353666666666667
```

Figure 24: calculating the accuracy on validation set

4. Results and analysis

This project offers a deep learning model using multilayer perceptron to make prediction depending on gathered information about the person who applied for a credit -like his credit history, debit ratio, Monthly Income, age and other features- whither will experience 90 days past due delinquency or worse or not, if the answer is yes (prediction= 1) then it is a "bad credit" and it should be rejected, but if the answer is no (prediction =0) then it can be considered as a "good credit" and should be accepted by lenders.

4.1 The confusion matrix and metrics

To make this model useful the results must be understood, one way to do this is using the Confusion Matrix form sklearn.metrics, by the code in the following figure.

```
In [24]: # Making the Confusion Matrix
         from sklearn.metrics import confusion_matrix
         cm = confusion_matrix(y_test, prediction)
         print(cm)

         [[27790   167]
          [ 1772   271]]
```

Figure 25: The Confusion Matrix code and result

A confusion matrix is a performance evaluation tool that is particularly useful in assessing recall, precision, specificity, accuracy, and the area under the curve (AUC) of a classification model.

Interpreting The Confusion Matrix

The terminology of The Confusion Matrix is as follows:

- True Positives (TP): The model predicted positive, and the actual label is positive
- True Negative (TN): The model predicted negative, and the actual label is negative
- False Positive (FP): The model predicted positive, and the actual label was negative
- False Negative (FN): The model predicted negative, and the actual label was positive

These terms could be presented visually as follows:

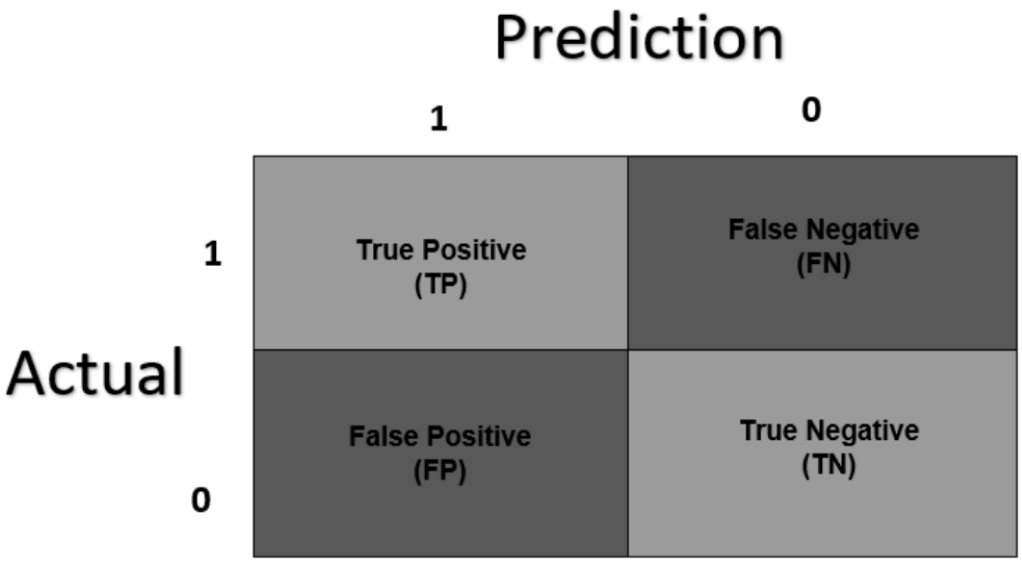


Figure 26: The terminology of The Confusion Matrix

Now I will perform the Confusion Matrix of the model in similar form

		predicted	
		1	0
Actual	1	27790	167
	0	1772	271

Table 4: The Confusion Matrix

- Accuracy

accuracy is a commonly used metric for evaluating the performance of a model on a given dataset. It is the ratio of the number of correct predictions made by the model to the total number of predictions made. It is often expressed as a percentage. However, it is important to note that accuracy alone may not always be the best metric to evaluate a model's performance, especially when the dataset is imbalanced or when the cost of false positives and false negatives are different.

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

The accuracy is 0.93536667.

- Precision

Precision is a metric used to evaluate the performance of a binary classification model. It is defined as the ratio of true positive predictions to the total number of positive predictions. In other words, it is the proportion of correct positive predictions out of all the positive predictions made by the classifier. High precision means that the classifier is good at identifying positive instances, while low precision means that it often makes false positive predictions.

$$\Pr e\,cision = \frac{TP}{TP + FP}$$

The precision is 0.94005818

- Recall

Recall is a measure of a classifier's ability to correctly identify positive instances from a dataset. It is defined as the ratio of true positive predictions to the total number of actual positive instances. It is often used in the context of binary classification problems, where it measures the proportion of actual positive instances that were correctly identified by the classifier. A high recall value indicates that the classifier has a low false negative rate, meaning that it correctly identifies most of the positive instances.

$$Recall = \frac{TP}{TP + FN}$$

The recall is 0.99402654

- F1-Score

F1-score is a measure of a model's accuracy and balance between precision and recall. It is the harmonic mean of precision and recall, where the best value is 1.0 and the worst value is 0.0. It is often used as a single number summary of the performance of a classification model, as it takes into account both the false positives and false negatives.

The F1-score is calculated using the following formula:

$$F1 = 2 * \frac{(precision * recall)}{(precision + recall)}$$

F1-score is a way to balance precision and recall to have a single number that indicates how well a model performs. It's particularly useful when the dataset is imbalanced and one class is much more frequent than the other which is the case of my project

The F1-score is 0.9662894

I must note that these metrics were calculated manually to provide familiarity to the readers with no technical background but can be easily calculated using sklearn.metrics

4.2 Visualizing the model performance

Using the matplotlib library I can plot the accuracy and the loss for both training and validation data as shown in the following figures.

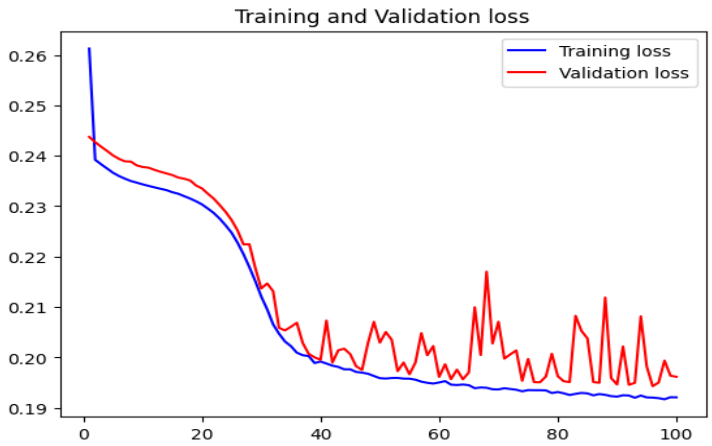


Figure 27: Training and Validation loss

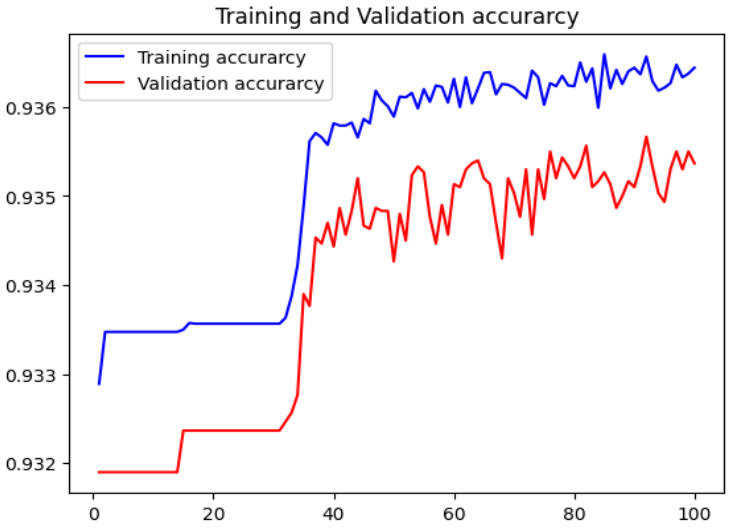


Figure 28: Training and Validation accuracy

4.3 Understanding the results

By considering the F1 score, I can say that the performance of my model is excellent and will perform well to predict the target credit scoring, and by looking at the two figures in the previous section there is some sort of instability in the validation data that can be referred to the Biased nature of the data since only Around 6% of samples defaulted (target value =1) and the rest has not defaulted- around 96% of the data -which is a big source of bias in data training leading to a slight instability in predictions, this can be solved by exploring more sensitive neural network architecture, or even using transformers neural networks such as Mixture Of Experts (MOE) this may be possible only with big size of data since these architectures will not perform well with a relatively small dataset like the one used in this project,

nevertheless, there is a necessary need for improvement by collecting more data, I have already sent a formal request to the Central Bank of Jordan with a proof letter from the HTU University requesting data to perform more training on the model, furthermore, I have tried to contact CRIF Jordan to provide me with more information about a The Scoring Assessment System, but unfortunately, there was no respond from them.

5. Conclusions and recommendations

In this project, I have developed a neural network model for credit scoring and evaluated its performance using (give me some credit) dataset. The results of the experiments indicate that the neural network model is a promising solution for credit scoring, outperforming traditional machine learning models in terms of accuracy and F1 score, by utilizing a deep learning model, I was able to extract valuable features from the raw data and make more informed predictions than traditional methods.

One of the main strengths of the neural network model is its ability to handle a large amount of data and capture non-linear relationships between features. This is particularly important in credit scoring, where a wide range of factors can influence an individual's creditworthiness.

The results also showed that the model's performance can be improved by fine-tuning the hyperparameters and using more advanced architectures. Additionally, it is important to consider the ethical implications of credit scoring and ensure that the model does not perpetuate biases or discrimination.

In conclusion, the neural network model for credit scoring has the potential to improve the accuracy and efficiency of credit decision-making. I recommend further research to explore other architectures and techniques to improve the model's performance and generalizability to other datasets. Additionally, I recommend considering ethical issues and fair decision-making in any credit scoring project.

6. References

- *Give me some credit*. Kaggle. (n.d.). Retrieved December 22, 2022, from https://www.kaggle.com/competitions/GiveMeSomeCredit/overview

- *neural network credit scoring models - researchgate*. (PDF) Neural Network Credit Scoring Models (researchgate.net) . (n.d.). Retrieved December 20, 2022.
- *CRIF*. الصفحة الرئيسية - كريف الاردن. (n.d.). Retrieved December 22, 2022, from https://www.crif.jo/EN/Pages/About_Us
- *IEEE Xplore*. (n.d.). Retrieved December 20, 2022, from https://ieeexplo
- re.ieee.org/Xplore/home.jsp
- *Deep learning techniques for credit scoring - joebm.com*. (n.d.). Retrieved December 22, 2022, from http://www.joebm.com/vol7/588-DE2009.pdf
- X. Dastile and T. Celik, "Making Deep Learning-Based Predictions for Credit Scoring Explainable," in *IEEE Access*, vol. 9, pp. 50426-50440, 2021, doi: 10.1109/ACCESS.2021.3068854.
- *Deep learning techniques for credit scoring - joebm.com*. (n.d.). Retrieved December 22, 2022, from http://www.joebm.com/vol7/588-DE2009.pdf
- *Evaluation of consumer credit in Jordanian banks: A credit scoring ...* https://www.researchgate.net/publication/303021545_Evaluation_of_Consumer_Credit_in_Jordanian_Banks_A_Credit_Scoring_Approach. (n.d.). Retrieved December 11, 2022, from https://www.researchgate.net/publication/303021545_Evaluation_of_Consumer_Credit_in_Jordanian_Banks_A_Credit_Scoring_Approach
- Hussain Ali Bekhet and, Shorouq Fathi Kamel Eletter. (n.d.). *Credit Risk Management for the Jordanian Commercial Banks: A business Intelligence Approach*. Retrieved December 13, 2022, from http://www.ajbasweb.com/old/ajbas/2012/Sep%202012/188-195.pdf

- Adler, J., & Tribune, S. to the. (2021, August 22). *Age can play a role in your mortgage equation*. Chicago Tribune. Retrieved December 25, 2022, from https://www.chicagotribune.com/news/ct-xpm-2005-11-20-0511200488-story.html
- Push, A. (2022, August 15). *What is a personal line of credit? pros and cons*. ValuePenguin. Retrieved December 25, 2022, from https://www.valuepenguin.com/personal-loans/what-is-a-personal-line-of-credit#:~:text=An%20unsecured%20personal%20line%20of,in%20one%20lump%2Dsum%20payment.
- The Concept of Artificial Neurons (Perceptrons) in Neural Networks | by Rukshan Pramoditha | Towards Data Science [accessed December 18, 2022]
- xai-banking-financial-services_1.png (1139×565) (knime.com) [accessed December 22, 2022]
- Neural Networks: parameters, hyperparameters and optimization strategies | by Valentina Alto | Towards Data Science [accessed December 29, 2022]
- Lada Rudnitckaia & Paolo Tamagnini. (n.d.). *How banks can use explainable AI in credit scoring*. KNIME. Retrieved December 26, 2022, from https://www.knime.com/blog/banks-use-xai-transparent-credit-scoring
- Deep Learning Essentials | Packt (packtpub.com) [accessed December 23, 2022]
- NumPy: the absolute basics for beginners — NumPy v1.24 Manual [accessed December 24, 2022]
- tensorflow · PyPI [accessed December 24, 2022]
- About Keras [accessed December 12, 2022]
- What is Keras and Why it so Popular in 2021 | Simplilearn [accessed December 21, 2022]
- Matplotlib — Visualization with Python [accessed December 24, 2022]
- What is Matplotlib in Python? - Scaler Topics [accessed December 25, 2022]
- tf.keras.losses.BinaryCrossentropy | TensorFlow v2.11.0 [accessed December 25, 2022]

- Binary Cross Entropy/Log Loss for Binary Classification (analyticsvidhya.com) [accessed December 25, 2022]
- What are gradient descent and stochastic gradient descent? (sebastianraschka.com) [accessed December 27, 2022]
- ReLU (Rectified Linear Unit) Activation Function (opengenus.org) [accessed December 28, 2022]
- ReLU Activation Function - InsideAIML [accessed January 2, 2022]]
- Sigmoid Activation Function-InsideAIML [accessed January 5, 2022]
- ModelCheckpoint (keras.io) [accessed January 7, 2022]
- Accuracy, Precision, Recall & F1-Score - Python Examples - Data Analytics (vitalflux.com) [accessed January 8, 2022]
- What is a Confusion Matrix in Machine Learning? (datatron.com) [accessed January 10, 2022]

7. Appendix

Neural Network Credit Scoring Model

Importing Important Packages

In [2]:

```
import pandas as pd
import numpy as np

import tensorflow as tf
import keras
from keras.models import Sequential
from keras.layers import InputLayer, Dense
from keras.callbacks import ModelCheckpoint

import matplotlib.pyplot as plt
from numpy import asarray
```

Reading the Data

In [3]:

```
df=pd.read_csv("cs-training.csv",na_values = "nan")
df.head()
```

Out[3]:

	Unnamed: 0	SeriousDlqin2yrs	RevolvingUtilizationOfUnsecuredLines	age	NumberOfTime30-59DaysPastDueNotWorse	DebtRatio	MonthlyIncome	NumberOfOpenCreditLinesAndLoans	NumberOfTimes90DaysLate	NumberRealEstateLoansOrLines	NumberOfTime60-89DaysPastDueNotWorse	NumberOfDependents
0	1	1	0.766127	45	2	0.802982	9120.00	13	0	6	0	2.0
1	2	0	0.957151	40	0	0.121876	2600.00	4	0	0	0	1.0
2	3	0	0.658180	38	1	0.085113	3042.00	2	1	0	0	0.0
3	4	0	0.233810	30	0	0.036050	3300.00	5	0	0	0	0.0
4	5	0	0.907239	49	1	0.024926	63588.0	7	0	1	0	0.0

Data preprocessing

In [4]:

#check for duplicated data

df.duplicated().sum()

Out[4]:

0

In [5]:

```
#handeling missing values
print(df.info())
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 150000 entries, 0 to 149999
Data columns (total 12 columns):
 #   Column                                Non-Null Count   Dtype
---  ------                                --------------   -----
 0   Unnamed: 0                            150000 non-null  int64
 1   SeriousDlqin2yrs                      150000 non-null  int64
 2   RevolvingUtilizationOfUnsecuredLines  150000 non-null  float64
 3   age                                   150000 non-null  int64
 4   NumberOfTime30-59DaysPastDueNotWorse  150000 non-null  int64
 5   DebtRatio                             150000 non-null  float64
 6   MonthlyIncome                         120269 non-null  float64
 7   NumberOfOpenCreditLinesAndLoans       150000 non-null  int64
 8   NumberOfTimes90DaysLate               150000 non-null  int64
 9   NumberRealEstateLoansOrLines          150000 non-null  int64
 10  NumberOfTime60-89DaysPastDueNotWorse  150000 non-null  int64
 11  NumberOfDependents                    146076 non-null  float64
dtypes: float64(4), int64(8)
memory usage: 13.7 MB
None
```

In [6]:

```
#Null values tabel

null_val_sums = df.isnull().sum()
pd.DataFrame({"Column": null_val_sums.index, "Number of Null Values": null_val_sums.values,
              "Proportion": null_val_sums.values / len(df) })
```

Out[6]:

	Column	Number of Null Values	Proportion
0	Unnamed: 0	0	0.000000
1	SeriousDlqin2yrs	0	0.000000
2	RevolvingUtilizationOfUnsecuredLines	0	0.000000
3	age	0	0.000000
4	NumberOfTime30-59DaysPastDueNotWorse	0	0.000000
5	DebtRatio	0	0.000000
6	MonthlyIncome	29731	0.198207
7	NumberOfOpenCreditLinesAndLoans	0	0.000000
8	NumberOfTimes90DaysLate	0	0.000000
9	NumberRealEstateLoansOrLines	0	0.000000
10	NumberOfTime60-89DaysPastDueNotWorse	0	0.000000
11	NumberOfDependents	3924	0.026160

Splitting the Data into independent (X) and dependant (y) features

In [7]:

```
X=df.iloc[:,2:].values
y=df.iloc[:,1].values
```

In [8]:

```
# handeling missing values
from sklearn.impute import SimpleImputer


imp_mean=SimpleImputer(missing_values= np.nan, strategy="median")
X[:,[4,9]]=imp_mean.fit_transform(X[:,[4,9]])
```

Data Normalization

In [9]:

```
#MinMax scaling
from sklearn.preprocessing import MinMaxScaler
# define min max scaler
scaler = MinMaxScaler()
X = scaler.fit_transform(X)
```

Train Test Split

In [10]:

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,random_state = 0)
```

Defining the architecture of the model

In [11]:

```
# checking the version of keras
import keras
print(keras.__version__)
```

2.10.0

In [12]:

```
# checking the version of tensorflow
import tensorflow as tf
print(tf.__version__)
```

2.10.0

In [13]:

```
# number of input neurons
X_train.shape
X_train.shape[1] #number of features
```

Out[13]:

10

In [15]:

```
# define hidden layers and neuron in each layer
input_neurons = X_train.shape[1]
output_neurons = 1
number_of_hidden_layers = 2
neuron_hidden_layer_1 = 100
neuron_hidden_layer_2 = 125
neuron_hidden_layer_3 = 100

# activation function of different layers

#I have picked relu as an activation function for hidden layers, And I have used sigmoid activation function in the final layer

model = Sequential()
model.add(InputLayer(input_shape=(input_neurons,)))
model.add(Dense(units=neuron_hidden_layer_1, activation='relu'))
model.add(Dense(units=neuron_hidden_layer_2, activation='relu'))
model.add(Dense(units=neuron_hidden_layer_3, activation='relu'))
model.add(Dense(units=output_neurons, activation='sigmoid'))

# summary of the model
print(model.summary())
Model: "sequential_1"
```

Layer (type)	Output Shape	Param #
dense_4 (Dense)	(None, 100)	1100
dense_5 (Dense)	(None, 125)	12625
dense_6 (Dense)	(None, 100)	12600
dense_7 (Dense)	(None, 1)	101

Total params: 26,426
Trainable params: 26,426
Non-trainable params: 0

None

Compiling the model

In [16]:

```
model.compile(loss='binary_crossentropy',optimizer="SGD",metrics=['accuracy'])
```

In [17]:

```
# specify filepath- this will write a new file for each epoch with the epoch number contained within the filename
filepath="model_weights.hdf5"
checkpoint = keras.callbacks.ModelCheckpoint(filepath, monitor='val_acc', verbose=0,
                                             save_weights_only=False, save_best_only=False, mode='max')
```

Training the model

passing the independent and dependent features for training set for training the model

validation data will be evaluated at the end of each epoch

setting the epochs as 100

storing the trained model in model_history variable which will be used to visualize the training process

In [18]:

```
model_history = model.fit(X_train, y_train, validation_data=(X_test, y_test),
                    batch_size = 64,callbacks=[checkpoint],
                    initial_epoch=0, epochs=100)
Epoch 1/100
1875/1875 [==============================] - 7s 4ms/step - loss: 0.2613 - accuracy: 0.9329 - val_loss: 0.2438 - val_accurac
y: 0.9319
Epoch 2/100
1875/1875 [==============================] - 8s 4ms/step - loss: 0.2392 - accuracy: 0.9335 - val_loss: 0.2427 - val_accurac
y: 0.9319
Epoch 3/100
1875/1875 [==============================] - 7s 4ms/step - loss: 0.2383 - accuracy: 0.9335 - val_loss: 0.2418 - val_accurac
y: 0.9319
Epoch 4/100
1875/1875 [==============================] - 7s 4ms/step - loss: 0.2375 - accuracy: 0.9335 - val_loss: 0.2410 - val_accurac
y: 0.9319
Epoch 5/100
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2366 - accuracy: 0.9335 - val_loss: 0.2401 - val_accurac
y: 0.9319
Epoch 6/100
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2360 - accuracy: 0.9335 - val_loss: 0.2394 - val_accurac
y: 0.9319
Epoch 7/100
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2355 - accuracy: 0.9335 - val_loss: 0.2389 - val_accurac
y: 0.9319
Epoch 8/100
1875/1875 [==============================] - 7s 3ms/step - loss: 0.2350 - accuracy: 0.9335 - val_loss: 0.2389 - val_accurac
y: 0.9319
Epoch 9/100
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2347 - accuracy: 0.9335 - val_loss: 0.2381 - val_accurac
y: 0.9319
```

Epoch 10/100
1875/1875 [==========] - 6s 3ms/step - loss: 0.2343 - accuracy: 0.9335 - val_loss: 0.2378 - val_accuracy: 0.9319
Epoch 11/100
1875/1875 [==========] - 7s 3ms/step - loss: 0.2340 - accuracy: 0.9335 - val_loss: 0.2376 - val_accuracy: 0.9319
Epoch 12/100
1875/1875 [==========] - 6s 3ms/step - loss: 0.2337 - accuracy: 0.9335 - val_loss: 0.2372 - val_accuracy: 0.9319
Epoch 13/100
1875/1875 [==========] - 7s 4ms/step - loss: 0.2335 - accuracy: 0.9335 - val_loss: 0.2368 - val_accuracy: 0.9319
Epoch 14/100
1875/1875 [==========] - 6s 3ms/step - loss: 0.2332 - accuracy: 0.9335 - val_loss: 0.2365 - val_accuracy: 0.9319
Epoch 15/100
1875/1875 [==========] - 6s 3ms/step - loss: 0.2328 - accuracy: 0.9335 - val_loss: 0.2362 - val_accuracy: 0.9324
Epoch 16/100
1875/1875 [==========] - 7s 4ms/step - loss: 0.2325 - accuracy: 0.9336 - val_loss: 0.2357 - val_accuracy: 0.9324
Epoch 17/100
1875/1875 [==========] - 6s 3ms/step - loss: 0.2320 - accuracy: 0.9336 - val_loss: 0.2355 - val_accuracy: 0.9324
Epoch 18/100
1875/1875 [==========] - 7s 4ms/step - loss: 0.2315 - accuracy: 0.9336 - val_loss: 0.2351 - val_accuracy: 0.9324
Epoch 19/100
1875/1875 [==========] - 7s 4ms/step - loss: 0.2310 - accuracy: 0.9336 - val_loss: 0.2341 - val_accuracy: 0.9324
Epoch 20/100
1875/1875 [==========] - 7s 3ms/step - loss: 0.2304 - accuracy: 0.9336 - val_loss: 0.2335 - val_accuracy: 0.9324

Epoch 21/100
1875/1875 [==========] - 7s 4ms/step - loss: 0.2295 - accuracy: 0.9336 - val_loss: 0.2325 - val_accuracy: 0.9324
Epoch 22/100
1875/1875 [==========] - 6s 3ms/step - loss: 0.2286 - accuracy: 0.9336 - val_loss: 0.2315 - val_accuracy: 0.9324
Epoch 23/100
1875/1875 [==========] - 7s 4ms/step - loss: 0.2275 - accuracy: 0.9336 - val_loss: 0.2302 - val_accuracy: 0.9324
Epoch 24/100
1875/1875 [==========] - 8s 4ms/step - loss: 0.2262 - accuracy: 0.9336 - val_loss: 0.2289 - val_accuracy: 0.9324
Epoch 25/100
1875/1875 [==========] - 9s 5ms/step - loss: 0.2247 - accuracy: 0.9336 - val_loss: 0.2273 - val_accuracy: 0.9324
Epoch 26/100
1875/1875 [==========] - 7s 4ms/step - loss: 0.2228 - accuracy: 0.9336 - val_loss: 0.2252 - val_accuracy: 0.9324
Epoch 27/100
1875/1875 [==========] - 6s 3ms/step - loss: 0.2205 - accuracy: 0.9336 - val_loss: 0.2224 - val_accuracy: 0.9324
Epoch 28/100
1875/1875 [==========] - 7s 4ms/step - loss: 0.2179 - accuracy: 0.9336 - val_loss: 0.2224 - val_accuracy: 0.9324
Epoch 29/100
1875/1875 [==========] - 6s 3ms/step - loss: 0.2151 - accuracy: 0.9336 - val_loss: 0.2177 - val_accuracy: 0.9324
Epoch 30/100
1875/1875 [==========] - 7s 4ms/step - loss: 0.2121 - accuracy: 0.9336 - val_loss: 0.2137 - val_accuracy: 0.9324
Epoch 31/100
1875/1875 [==========] - 7s 3ms/step - loss: 0.2095 - accuracy: 0.9336 - val_loss: 0.2147 - val_accuracy: 0.9324

```
Epoch 32/100
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2066 - accuracy: 0.9336 - val_loss: 0.2131 - val_accurac
y: 0.9325
Epoch 33/100
1875/1875 [==============================] - 7s 4ms/step - loss: 0.2047 - accuracy: 0.9339 - val_loss: 0.2059 - val_accurac
y: 0.9326
Epoch 34/100
1875/1875 [==============================] - 7s 4ms/step - loss: 0.2032 - accuracy: 0.9342 - val_loss: 0.2053 - val_accurac
y: 0.9328
Epoch 35/100
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2022 - accuracy: 0.9349 - val_loss: 0.2061 - val_accurac
y: 0.9339
Epoch 36/100
1875/1875 [==============================] - 7s 4ms/step - loss: 0.2009 - accuracy: 0.9356 - val_loss: 0.2069 - val_accurac
y: 0.9338
Epoch 37/100
1875/1875 [==============================] - 7s 3ms/step - loss: 0.2004 - accuracy: 0.9357 - val_loss: 0.2028 - val_accurac
y: 0.9345
Epoch 38/100
1875/1875 [==============================] - 6s 3ms/step - loss: 0.2003 - accuracy: 0.9357 - val_loss: 0.2007 - val_accurac
y: 0.9345
Epoch 39/100
1875/1875 [==============================] - 6s 3ms/step - loss: 0.1989 - accuracy: 0.9356 - val_loss: 0.2000 - val_accurac
y: 0.9347
Epoch 40/100
1875/1875 [==============================] - 6s 3ms/step - loss: 0.1992 - accuracy: 0.9358 - val_loss: 0.1995 - val_accurac
y: 0.9344
Epoch 41/100
1875/1875 [==============================] - 7s 4ms/step - loss: 0.1988 - accuracy: 0.9358 - val_loss: 0.2073 - val_accurac
y: 0.9349
Epoch 42/100
1875/1875 [==============================] - 7s 4ms/step - loss: 0.1984 - accuracy: 0.9358 - val_loss: 0.1990 - val_accurac
y: 0.9346
```

Epoch 43/100
1875/1875 [==========] - 6s 3ms/step - loss: 0.1981 - accuracy: 0.9358 - val_loss: 0.2014 - val_accuracy: 0.9348
Epoch 44/100
1875/1875 [==========] - 6s 3ms/step - loss: 0.1976 - accuracy: 0.9357 - val_loss: 0.2017 - val_accuracy: 0.9352
Epoch 45/100
1875/1875 [==========] - 7s 4ms/step - loss: 0.1976 - accuracy: 0.9359 - val_loss: 0.2007 - val_accuracy: 0.9347
Epoch 46/100
1875/1875 [==========] - 6s 3ms/step - loss: 0.1971 - accuracy: 0.9358 - val_loss: 0.1982 - val_accuracy: 0.9346
Epoch 47/100
1875/1875 [==========] - 7s 4ms/step - loss: 0.1970 - accuracy: 0.9362 - val_loss: 0.1975 - val_accuracy: 0.9349
Epoch 48/100
1875/1875 [==========] - 6s 3ms/step - loss: 0.1968 - accuracy: 0.9361 - val_loss: 0.2028 - val_accuracy: 0.9348
Epoch 49/100
1875/1875 [==========] - 6s 3ms/step - loss: 0.1963 - accuracy: 0.9360 - val_loss: 0.2071 - val_accuracy: 0.9348
Epoch 50/100
1875/1875 [==========] - 7s 4ms/step - loss: 0.1959 - accuracy: 0.9359 - val_loss: 0.2029 - val_accuracy: 0.9343
Epoch 51/100
1875/1875 [==========] - 6s 3ms/step - loss: 0.1958 - accuracy: 0.9361 - val_loss: 0.2050 - val_accuracy: 0.9348
Epoch 52/100
1875/1875 [==========] - 7s 4ms/step - loss: 0.1959 - accuracy: 0.9361 - val_loss: 0.2035 - val_accuracy: 0.9345
Epoch 53/100
1875/1875 [==========] - 6s 3ms/step - loss: 0.1959 - accuracy: 0.9362 - val_loss: 0.1973 - val_accuracy: 0.9352

Epoch 54/100
1875/1875 [==========] - 6s 3ms/step - loss: 0.1958 - accuracy: 0.9360 - val_loss: 0.1990 - val_accuracy: 0.9353
Epoch 55/100
1875/1875 [==========] - 7s 4ms/step - loss: 0.1958 - accuracy: 0.9362 - val_loss: 0.1967 - val_accuracy: 0.9353
Epoch 56/100
1875/1875 [==========] - 6s 3ms/step - loss: 0.1956 - accuracy: 0.9361 - val_loss: 0.1990 - val_accuracy: 0.9348
Epoch 57/100
1875/1875 [==========] - 7s 3ms/step - loss: 0.1952 - accuracy: 0.9362 - val_loss: 0.2048 - val_accuracy: 0.9345
Epoch 58/100
1875/1875 [==========] - 6s 3ms/step - loss: 0.1950 - accuracy: 0.9362 - val_loss: 0.2004 - val_accuracy: 0.9349
Epoch 59/100
1875/1875 [==========] - 6s 3ms/step - loss: 0.1948 - accuracy: 0.9360 - val_loss: 0.2022 - val_accuracy: 0.9346
Epoch 60/100
1875/1875 [==========] - 6s 3ms/step - loss: 0.1950 - accuracy: 0.9363 - val_loss: 0.1961 - val_accuracy: 0.9351
Epoch 61/100
1875/1875 [==========] - 6s 3ms/step - loss: 0.1953 - accuracy: 0.9360 - val_loss: 0.1986 - val_accuracy: 0.9351
Epoch 62/100
1875/1875 [==========] - 6s 3ms/step - loss: 0.1946 - accuracy: 0.9363 - val_loss: 0.1957 - val_accuracy: 0.9353
Epoch 63/100
1875/1875 [==========] - 7s 3ms/step - loss: 0.1945 - accuracy: 0.9360 - val_loss: 0.1976 - val_accuracy: 0.9354
Epoch 64/100
1875/1875 [==========] - 7s 4ms/step - loss: 0.1946 - accuracy: 0.9362 - val_loss: 0.1957 - val_accuracy: 0.9354

Epoch 65/100
1875/1875 [==========] - 7s 4ms/step - loss: 0.1945 - accuracy: 0.9364 - val_loss: 0.1970 - val_accuracy: 0.9352
Epoch 66/100
1875/1875 [==========] - 6s 3ms/step - loss: 0.1939 - accuracy: 0.9364 - val_loss: 0.2099 - val_accuracy: 0.9351
Epoch 67/100
1875/1875 [==========] - 6s 3ms/step - loss: 0.1940 - accuracy: 0.9361 - val_loss: 0.2005 - val_accuracy: 0.9347
Epoch 68/100
1875/1875 [==========] - 7s 4ms/step - loss: 0.1940 - accuracy: 0.9363 - val_loss: 0.2170 - val_accuracy: 0.9343
Epoch 69/100
1875/1875 [==========] - 6s 3ms/step - loss: 0.1937 - accuracy: 0.9362 - val_loss: 0.2028 - val_accuracy: 0.9352
Epoch 70/100
1875/1875 [==========] - 6s 3ms/step - loss: 0.1936 - accuracy: 0.9362 - val_loss: 0.2071 - val_accuracy: 0.9350
Epoch 71/100
1875/1875 [==========] - 6s 3ms/step - loss: 0.1939 - accuracy: 0.9362 - val_loss: 0.1997 - val_accuracy: 0.9348
Epoch 72/100
1875/1875 [==========] - 6s 3ms/step - loss: 0.1937 - accuracy: 0.9361 - val_loss: 0.2006 - val_accuracy: 0.9353
Epoch 73/100
1875/1875 [==========] - 6s 3ms/step - loss: 0.1936 - accuracy: 0.9364 - val_loss: 0.2014 - val_accuracy: 0.9346
Epoch 74/100
1875/1875 [==========] - 7s 4ms/step - loss: 0.1933 - accuracy: 0.9363 - val_loss: 0.1954 - val_accuracy: 0.9353
Epoch 75/100
1875/1875 [==========] - 6s 3ms/step - loss: 0.1935 - accuracy: 0.9360 - val_loss: 0.1997 - val_accuracy: 0.9350

```
Epoch 76/100
1875/1875 [==============================] - 6s 3ms/step - loss: 0.1935 - accuracy: 0.9363 - val_loss: 0.1951 - val_accurac
y: 0.9355
Epoch 77/100
1875/1875 [==============================] - 8s 4ms/step - loss: 0.1935 - accuracy: 0.9362 - val_loss: 0.1951 - val_accurac
y: 0.9352
Epoch 78/100
1875/1875 [==============================] - 7s 4ms/step - loss: 0.1934 - accuracy: 0.9363 - val_loss: 0.1962 - val_accurac
y: 0.9354
Epoch 79/100
1875/1875 [==============================] - 7s 4ms/step - loss: 0.1929 - accuracy: 0.9362 - val_loss: 0.2007 - val_accurac
y: 0.9353
Epoch 80/100
1875/1875 [==============================] - 7s 4ms/step - loss: 0.1931 - accuracy: 0.9362 - val_loss: 0.1963 - val_accurac
y: 0.9352
Epoch 81/100
1875/1875 [==============================] - 6s 3ms/step - loss: 0.1929 - accuracy: 0.9365 - val_loss: 0.1953 - val_accurac
y: 0.9353
Epoch 82/100
1875/1875 [==============================] - 8s 4ms/step - loss: 0.1925 - accuracy: 0.9363 - val_loss: 0.1951 - val_accurac
y: 0.9356
Epoch 83/100
1875/1875 [==============================] - 7s 4ms/step - loss: 0.1928 - accuracy: 0.9364 - val_loss: 0.2082 - val_accurac
y: 0.9351
Epoch 84/100
1875/1875 [==============================] - 7s 4ms/step - loss: 0.1930 - accuracy: 0.9360 - val_loss: 0.2052 - val_accurac
y: 0.9352
Epoch 85/100
1875/1875 [==============================] - 8s 4ms/step - loss: 0.1929 - accuracy: 0.9366 - val_loss: 0.2038 - val_accurac
y: 0.9353
Epoch 86/100
1875/1875 [==============================] - 8s 4ms/step - loss: 0.1925 - accuracy: 0.9362 - val_loss: 0.1951 - val_accurac
y: 0.9351
```

Epoch 87/100
1875/1875 [==========] - 8s 4ms/step - loss: 0.1927 - accuracy: 0.9364 - val_loss: 0.1950 - val_accuracy: 0.9349
Epoch 88/100
1875/1875 [==========] - 8s 4ms/step - loss: 0.1926 - accuracy: 0.9363 - val_loss: 0.2119 - val_accuracy: 0.9350
Epoch 89/100
1875/1875 [==========] - 8s 4ms/step - loss: 0.1923 - accuracy: 0.9364 - val_loss: 0.1959 - val_accuracy: 0.9352
Epoch 90/100
1875/1875 [==========] - 10s 6ms/step - loss: 0.1922 - accuracy: 0.9364 - val_loss: 0.1946 - val_accuracy: 0.9351
Epoch 91/100
1875/1875 [==========] - 7s 4ms/step - loss: 0.1925 - accuracy: 0.9364 - val_loss: 0.2022 - val_accuracy: 0.9353
Epoch 92/100
1875/1875 [==========] - 9s 5ms/step - loss: 0.1924 - accuracy: 0.9366 - val_loss: 0.1946 - val_accuracy: 0.9357
Epoch 93/100
1875/1875 [==========] - 7s 4ms/step - loss: 0.1920 - accuracy: 0.9363 - val_loss: 0.1949 - val_accuracy: 0.9353
Epoch 94/100
1875/1875 [==========] - 8s 4ms/step - loss: 0.1924 - accuracy: 0.9362 - val_loss: 0.2081 - val_accuracy: 0.9350
Epoch 95/100
1875/1875 [==========] - 7s 4ms/step - loss: 0.1921 - accuracy: 0.9362 - val_loss: 0.1982 - val_accuracy: 0.9349
Epoch 96/100
1875/1875 [==========] - 8s 4ms/step - loss: 0.1920 - accuracy: 0.9363 - val_loss: 0.1943 - val_accuracy: 0.9353
Epoch 97/100
1875/1875 [==========] - 8s 4ms/step - loss: 0.1919 - accuracy: 0.9365 - val_loss: 0.1950 - val_accuracy: 0.9355

Epoch 98/100

1875/1875 [==========] - 8s 4ms/step - loss: 0.1917 - accuracy: 0.9363 - val_loss: 0.1994 - val_accuracy: 0.9353

Epoch 99/100

1875/1875 [==========] - 8s 4ms/step - loss: 0.1921 - accuracy: 0.9364 - val_loss: 0.1964 - val_accuracy: 0.9355

Epoch 100/100

1875/1875 [==========] - 9s 5ms/step - loss: 0.1921 - accuracy: 0.9364 - val_loss: 0.1962 - val_accuracy: 0.9354

In [19]:

```
final_loss, final_accuracy = model.evaluate(X_test, y_test, verbose=0)
print('Final Loss: {}, Final Accuracy: {}'.format(final_loss, final_accuracy))
```

Final Loss: 0.19615095853805542, Final Accuracy: 0.935366690158844

Evaluating model performance on validation set

In [20]:

```
# getting predictions for the validation set
prediction = model.predict(X_test)
prediction = (prediction > 0.5)
MOH=np.concatenate((prediction.reshape(len(prediction),1), y_test.reshape(len(y_test),1)),1)
```

938/938 [==========] - 2s 2ms/step

In [21]:

```
df1 = pd.DataFrame(MOH, columns = ['prediction','actual'])

print(df1)
print(type(df1))
```

	prediction	actual
0	0	0
1	0	0
2	0	0
3	0	0

4 0 0
... ...
29995 0 0
29996 0 0
29997 0 0
29998 0 0
29999 0 0

[30000 rows x 2 columns]
<class 'pandas.core.frame.DataFrame'>

In [22]:

```
df1.to_csv('out.csv')
```

In [23]:

```
# calculating the accuracy on validation set
from sklearn.metrics import accuracy_score
accuracy= accuracy_score(y_test, prediction)
print(accuracy)
```

0.9353666666666667

In [24]:

```
# Making the Confusion Matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, prediction)
print(cm)
```

[[27790 167]
[1772 271]]

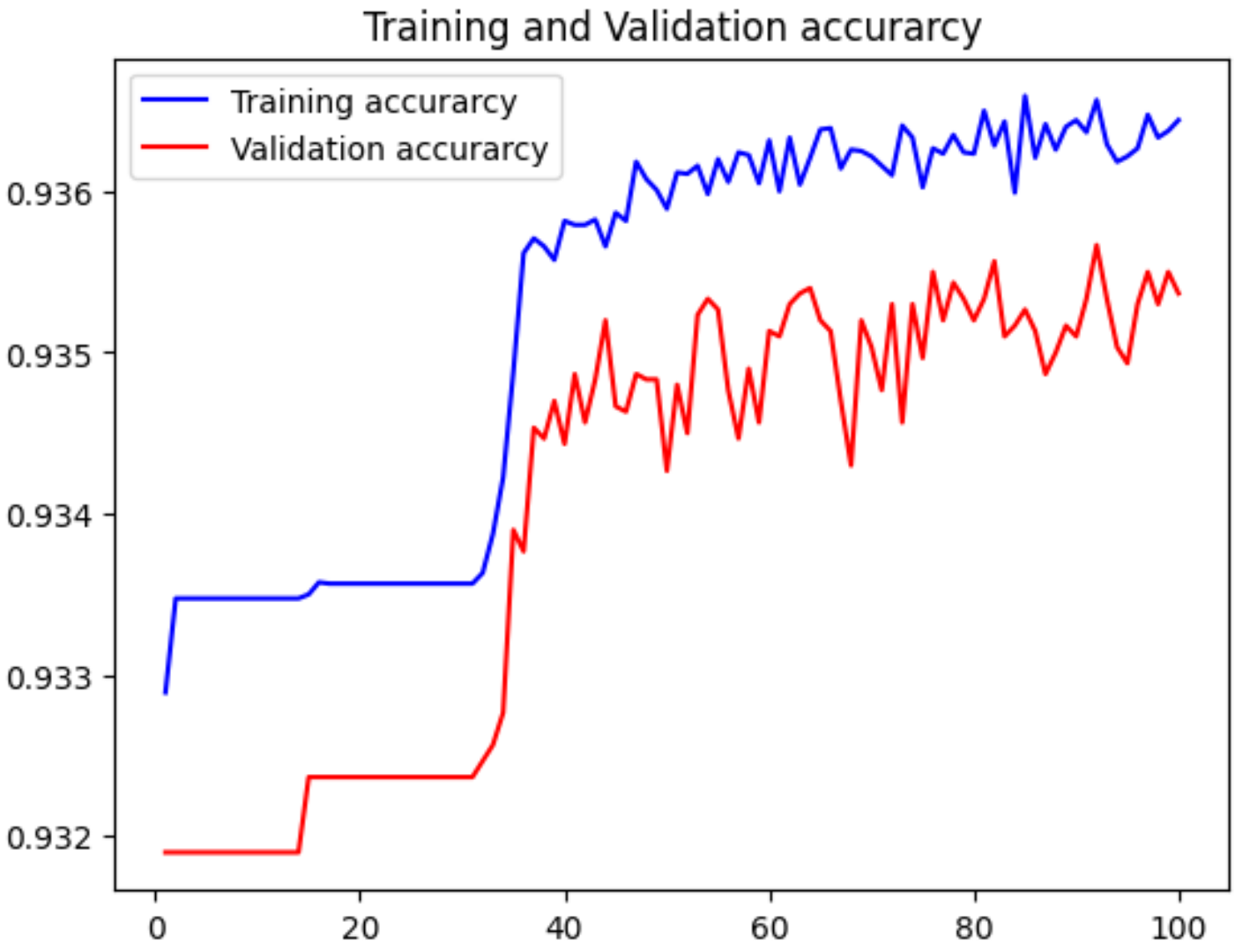
Visualizing the model performance

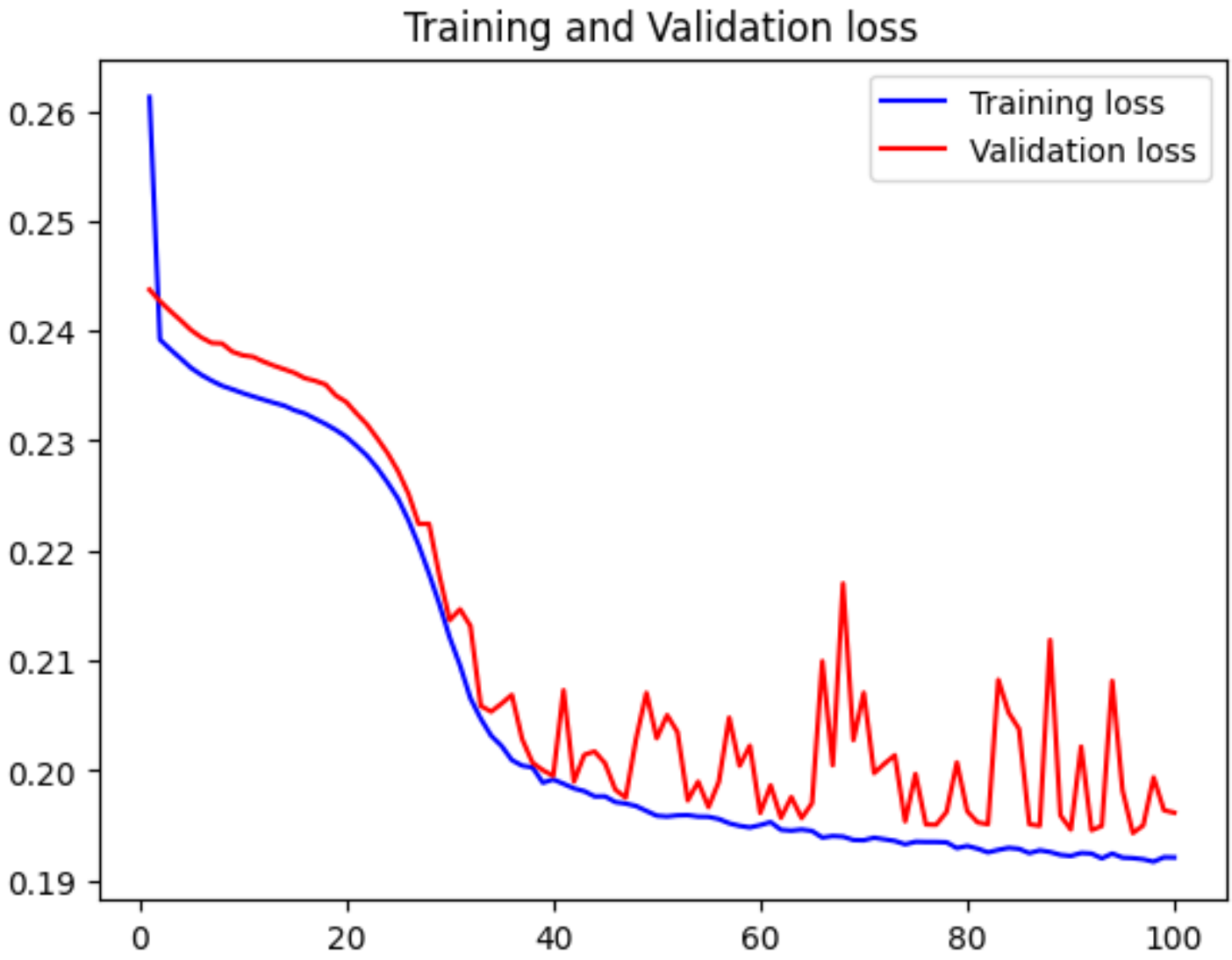
In [27]:

```
acc = model_history.history['accuracy']
val_acc = model_history.history['val_accuracy']
loss = model_history.history['loss']
```

```
val_loss = model_history.history['val_loss']
epochs = range(1, len(acc) + 1)
#Train and validation accuracy
plt.plot(epochs, acc, 'b', label='Training accurarcy')
plt.plot(epochs, val_acc, 'r', label='Validation accurarcy')
plt.title('Training and Validation accurarcy')
plt.legend()

plt.figure()
#Train and validation loss
plt.plot(epochs, loss, 'b', label='Training loss')
plt.plot(epochs, val_loss, 'r', label='Validation loss')
plt.title('Training and Validation loss')
plt.legend()
plt.show()
```


Training and Validation accurarcy
Training accurarcy
Validation accurarcy
0.936
0.935
0.934
0.933
0.932
0
20
40
60
80
100


Training and Validation loss
Training loss
Validation loss
0.26
0.25
0.24
0.23
0.22
0.21
0.20
0.19
0
20
40
60
80
100