Artificial Neural Network for Binary Classification

Asala Aljazaery

11 April 2022

University of Sunderland

Contents

1 Introduction

This report is based on a dataset collected from marketing campaigns that were held by a Portuguese banking institution (Moro, Cortez and Rita, 2014). The neural network is based on a binary classification system where the output is either 0 or 1, where 1 represents clients who subscribed to the service. This report aims to provide a multi-network architecture and find the most applicable model for our dataset. The architecture of the network will be optimized based on these results until achieving the best optimal network. The network will predict the clients who will subscribe to the service.

This report is structured as follows: First, I will present data pre-processing techniques to prepare the data and make it suitable for our binary classification such as Vectorization, Data formatting, Feature selection and Handling missing values. Moreover, while presenting an overview of the model architecture and after compiling the model, the data was split into three datasets, to keep some of the data for the model evaluation. Lastly, the model was trained and tested, followed by evaluation metrics to find the optimal model by making comparisons between different types of architecture using different techniques.

2 Network architecture

Initially, start by reading the dataset (Figure 2.1).

	age	marital	education	default	balance	housing	loan	contact	duration	campaign	pdays	previous	poutcome	y
0	33	1	2	no	882	no	no	1	39	1	151	3	0	no
1	42	0	1	no	-247	yes	yes	1	519	1	166	1	2	yes
2	33	1	1	no	3444	yes	no	1	144	1	91	4	0	yes
3	36	1	2	no	2415	yes	no	1	73	1	86	4	2	no
4	36	1	2	no	0	yes	no	1	140	1	143	3	0	yes
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
7837	34	0	1	no	1475	yes	no	0	1166	3	530	12	2	no
7838	53	1	2	no	583	no	no	0	226	1	184	4	1	yes
7839	73	1	1	no	2850	no	no	0	300	1	40	8	0	yes
7840	72	1	1	no	5715	no	no	0	1127	5	184	3	1	yes
7841	37	1	1	no	2971	no	no	0	361	2	188	11	2	no

7842 rows × 14 columns

Figure 2.1: The Dataset.

2.1 Data Pre-processing

Isolating the target /output from the rest of the dataset.

2.1.1 Data formatting

Changing the format of the values by replacing yes => 1 and no => 0 because we are classifying based on binary format (Figure 2.2).

```
training_y.replace("yes", 1, inplace=True)
training_y.replace("no", 0, inplace=True)


training_x.replace("yes", 1, inplace=True)
training_x.replace("no",0, inplace=True)
```

After changing other feature formats, the dataset will be as shown (Figure 2.3).

2.2 Vectorization

Changing all features format into float type, Since all my data must be floating to enhance the model performance as follow (Figure 2.4). and (Figure 2.5).

```
7651    1
4669    0
7161    1
4210    0
3541    0
       ..
7615    1
2486    0
2930    0
6641    1
4399    0
Name: y, Length: 548, dtype: int64
```

Figure 2.2: The output format.

	age	marital	education	default	balance	housing	loan	contact	duration	campaign	pdays	previous	poutcome
0	33	1	2	0	882	0	0	1	39	1	151	3	0
1	42	0	1	0	-247	1	1	1	519	1	166	1	2
2	33	1	1	0	3444	1	0	1	144	1	91	4	0
3	36	1	2	0	2415	1	0	1	73	1	86	4	2
4	36	1	2	0	0	1	0	1	140	1	143	3	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
7837	34	0	1	0	1475	1	0	0	1166	3	530	12	2
7838	53	1	2	0	583	0	0	0	226	1	184	4	1
7839	73	1	1	0	2850	0	0	0	300	1	40	8	0
7840	72	1	1	0	5715	0	0	0	1127	5	184	3	1
7841	37	1	1	0	2971	0	0	0	361	2	188	11	2

7842 rows × 13 columns

Figure 2.3: Data formating.

	age	marital	education	default	balance	housing	loan	contact	duration	campaign	pdays	previous	poutcome
0	33.0	1	2	0	882.0	0	0	1	39.0	1	151	3	0
1	42.0	0	1	0	-247.0	1	1	1	519.0	1	166	1	2
2	33.0	1	1	0	3444.0	1	0	1	144.0	1	91	4	0
3	36.0	1	2	0	2415.0	1	0	1	73.0	1	86	4	2
4	36.0	1	2	0	0.0	1	0	1	140.0	1	143	3	0
...	...	...	...	...	...	...	...	...	...	...	...	...	...
7837	34.0	0	1	0	1475.0	1	0	0	1166.0	3	530	12	2
7838	53.0	1	2	0	583.0	0	0	0	226.0	1	184	4	1
7839	73.0	1	1	0	2850.0	0	0	0	300.0	1	40	8	0
7840	72.0	1	1	0	5715.0	0	0	0	1127.0	5	184	3	1
7841	37.0	1	1	0	2971.0	0	0	0	361.0	2	188	11	2

7842 rows × 13 columns

Figure 2.4: Data Vectorization.

```
0       0.0
1       1.0
2       1.0
3       0.0
4       1.0
       ...
7837    0.0
7838    1.0
7839    1.0
7840    1.0
7841    0.0
Name: y, Length: 7842, dtype: float64
```

Figure 2.5: Data Type.

2.2.1 Feature selection

- plotting the correlation between features in a heatmap (Figure 2.6).

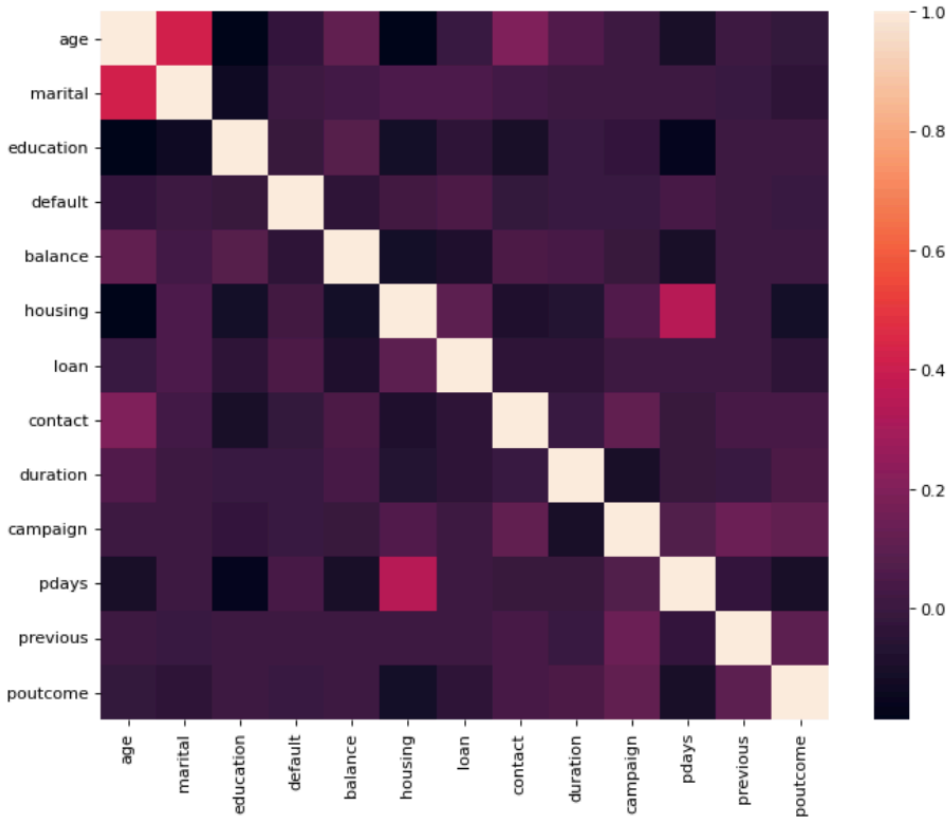


Figure 2.6: The correlation between features.

- The mean of features in each class of the output (Figure 2.7).

y	age	marital	education	default	balance	housing	loan	contact	duration	campaign	pdays	previous	poutcome
0	40.245527	0.821910	1.173135	0.008533	1404.122764	0.708505	0.161574	0.078172	215.945500	2.159923	230.483072	3.210019	0.563446
1	42.862873	0.765858	1.316231	0.001866	2113.677239	0.328358	0.061567	0.070896	404.234142	1.794776	188.654851	3.193097	0.829291

Figure 2.7: Features Mean.

As a result, we can see the differences in the features between the two groups. The duration mean for the group who subscribed is twice the first group which clearly shows

how much the duration feature is important for model prediction. - Plotting the output classes based on loan feature (Figure 2.8), credit card holders (Figure 2.9) and the house loan (Figure 2.10).

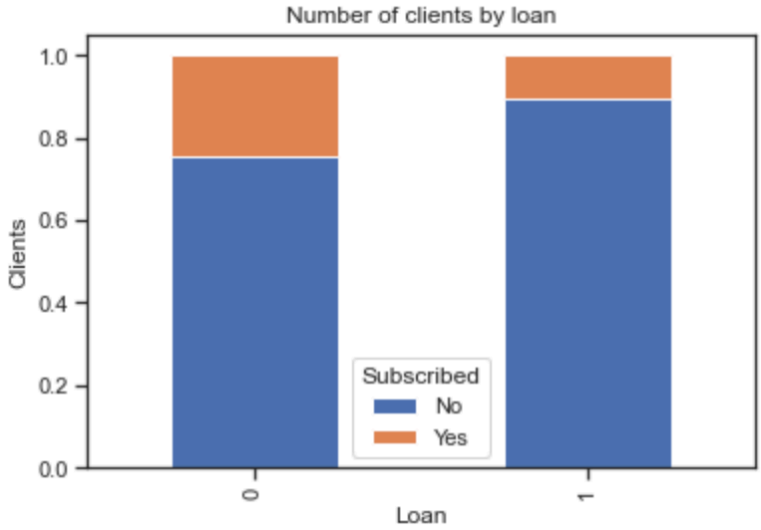


Figure 2.8: Number of clients (subscribed or not) by loan feature

As a result, the number of people who have loans and subscribed to the service is less than the people who did not subscribe. so if the client has loan, the chance of subscribing to the service will be less.

A similar percentage for both groups, it means that credit feature does not affect the model prediction. so we can drop this feature for less noise.

Clients who have house loan are less likely to subscribe than those who does not have house loan.

2.2.2 Handling missng value

Checking every feature for missing values to ensure that our data set does not have any empty values that may affect the model learning. After checking our data, no missing values were detected (Figure 2.11).

Figure 2.9: Number of clients (subscribed or not) by credit holders feature

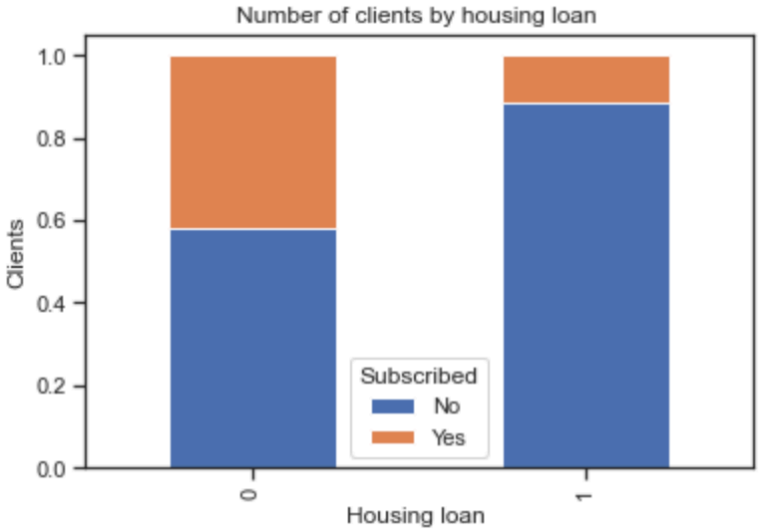


Figure 2.10: Number of clients (subscribed or not) by housing loan feature

	percent_missing
age	0.0
marital	0.0
education	0.0
default	0.0
balance	0.0
housing	0.0
loan	0.0
contact	0.0
duration	0.0
campaign	0.0
previous	0.0
poutcome	0.0
y	0.0

Figure 2.11: The percentage of missing values in each feature

2.2.3 One hot encode

Later, for neural network the datasets training_x and training_y must be into arrays so the model can learn it (Figure 2.12).

```
[[ 33.   1.   2. ... 151.   3.   0.]
 [ 42.   0.   1. ... 166.   1.   2.]
 [ 33.   1.   1. ...  91.   4.   0.]
 ...
 [ 73.   1.   1. ...  40.   8.   0.]
 [ 72.   1.   1. ... 184.   3.   1.]
 [ 37.   1.   1. ... 188.  11.   2.]]
[0 1 1 ... 1 1 0]
```

Figure 2.12: Datasets in array format

The shape of arrays (Figure 2.13).

```
(7842, 13)
(7842,)
```

Figure 2.13: The shape of arrays

2.3 Building the model

In defining the model, Sequential API was used because it is a simple model that leads directly to the output. All the features in the datasets were included, therefore, the Input shape is 13 (Features). In the first hidden layer, the activation function is Relu, and in the final layer sigmoid was applied to the output (Figure 2.14).

```
model = keras.Sequential()
model.add(layers.InputLayer(input_shape=(13)))
model.add(layers.Dense(10, activation="relu"))
model.add(layers.Dense(1, activation="sigmoid"))
model.summary()
```

```
Model: "sequential_50"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_102 (Dense)            (None, 10)                140
_________________________________________________________________
dense_103 (Dense)            (None, 1)                 11
=================================================================
Total params: 151
Trainable params: 151
Non-trainable params: 0
```

Figure 2.14: The model summary

2.4 Model Compiling

For model compiling, The binary cross-entropy was used as loss function. SGD was used for model optimization, and accuracy metrics were included as well.

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

2.4.1 Data splitting

Plotting the data distribution to build an overview of how data are distributed between the two classes (Figure 2.15).

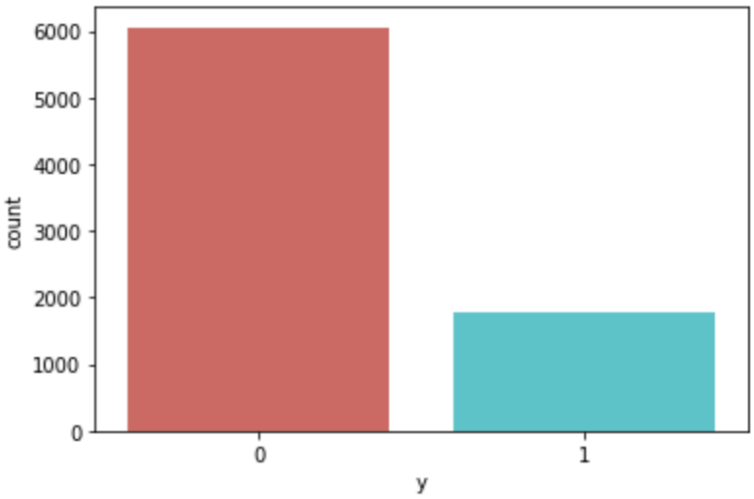


Figure 2.15: The data distribution per classes

Now the target(output) is divided into two groups: subscribed to the service or not. According to the chart, we have a higher number of clients who did not subscribe and there is not enough data from the class where clients they did subscribe. So we need to ensure when splitting our data set to cover both classes, otherwise, the network will not

learn when the client will subscribe.

In this model, 40% percent of the dataset was held for testing , 40% for training and 20% for validation. For small training sets, stratify was added to the model while splitting the datasets, because it is an imbalanced dataset, and to ensure that we have enough observation in each dataset with Randomness state =2. The datasets shape will be training_x ,testing_x , training_y ,testing_y accordingly (Figure 2.16).

```
(2352, 13)
(1569, 13)
(2352,)
(1569,)
```

Figure 2.16: Datasets shape

2.4.2 Model training and validation

After training the model with epoch =50, the model accuracy was 0.77 and loss = 0.51, while validation accuracy 0.76 and loss =0.51. This means that our model was able to generalize and performed well in new data (Figure 2.17).

```
118/118 [==============================] - 0s 2ms/step - loss: 0.5147 -
accuracy: 0.7734 - val_loss: 0.5164 - val_accuracy: 0.7662
```

Figure 2.17: Model training and validation results

- **Experiments based on adjusting Epochs using same model:**

Epoch number	Loss	Accuracy	Validation Loss	validation accuracy
50	0.5147	0.7734	0.5164	0.7662
40	0.5061	0.7736	0.5039	0.7662
20	0.5314	0.7728	0.5281	0.7662
10	0.5152	0.7718	0.5179	0.7662

Observation: The optimal number of epochs that I found is 40, with a minimum loss of 0.5061and accuracy of 0.7736.

In the following graph, the training loss and accuracy were visualized to see how the model performed while epoch was set at 40 (Figure 2.18).

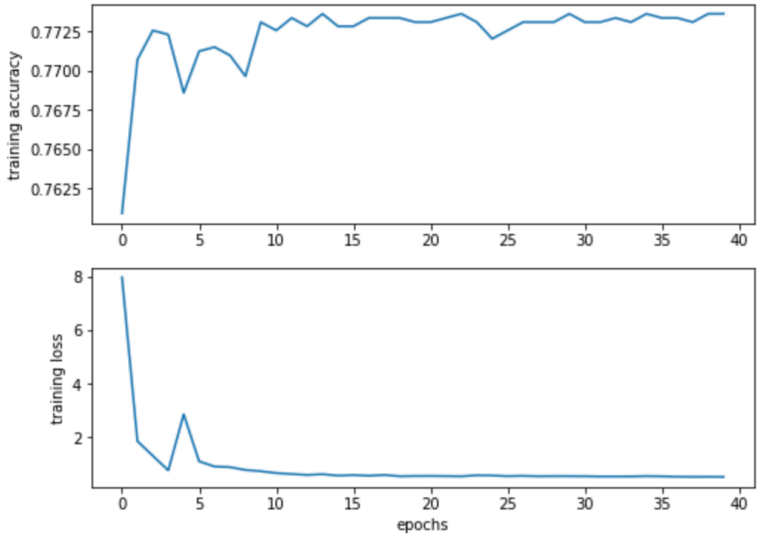


Figure 2.18: Training accuracy and loss visualisation

2.4.3 Model testing

For model testing results were:

```
99/99 [==============================] - 0s 923us/step - loss: 0.5281 - accuracy:
0.7724
test Loss = 0.5281244516372681 test Accuracy = 0.7723940014839172
```

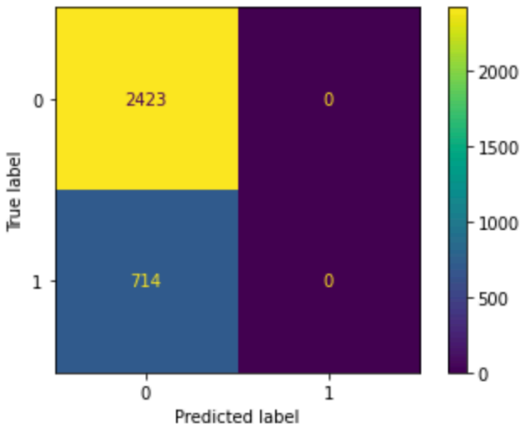
Figure 2.19: Test loss and test accuracy

The percentage of accuracy and loss in both training and test are very close, which means the model performed very well (Figure 2.19).

2.4.4 Model predictions and evaluation

- For binary classification obtaining prediction is done by using predict() function.

- Using the confusion matrix to show the model prediction (y_pred) compared to the real output(testing_y) and a heatmap to visualize how many times our model gets confused and give us a wrong prediction. Later, the evaluation metrics will be presented (Figure 2.20).




```
Using SKLEARN.
Accuracy: 0.7723940070130698
No utility for specificity alone
Sensitivity: 0.0
Precision: 0.0
F1-Score: 0.0
(3137,)
(3137, 1)
```

Figure 2.20: Confusion matrix for prediction results

The model accuracy was high but the model did not predict any samples from the other class, The true positive values were zero and the true negative was very high. This shows that the model will not perform very well in real world and the difference between the accuracy metrics and confusion matrix was happening
because we have an imbalanced dataset (Vidiyala, 2020) .

Experiments based on adjusting nodes using the same model:

By changing nodes number from 10 to16.

Number of nodes	Training Loss	Training Accuracy	Testing Loss	Testing Accuracy	Validation Loss	Validation Accuracy
10	0.50	0.77	0.52	0.77	0.51	0.76
16	0.46	0.80	12.54	0.77	0.59	0.76

Observation: Increasing the nodes results in a better performance in the training dataset, But the testing loss was higher which means the network is overfitting.

The most important metric for our dataset is the true positive, the focus is in increasing the true positive and true negative. The sensitivity is calculated only for positive cases and the specificity is for true negative rates. This means we will evaluate our model based on the sensitivity, how many positive predictions values were correct and accuracy for how many predictions the model made was correct.

3 Choosing best model hyperparameters

Model (1) Modifying the model to increase the prediction performance (Figure 3.1). - Two hidden layers with 12 nodes - Dropped 4 features ('pdays','poutcome','previous','default') - Test size=0.93 - Epochs =10

- Training and validation results (Figure 3.2) and (Figure 3.3).
- Testing results (Figure 3.4).
- Model prediction (Figure 3.5) and (Figure 3.6).
- Classification Report (Figure 3.7).

```
Model: "sequential_6"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_15 (Dense)             (None, 12)                120
_________________________________________________________________
dense_16 (Dense)             (None, 12)                156
_________________________________________________________________
dense_17 (Dense)             (None, 1)                 13
=================================================================
Total params: 289
Trainable params: 289
Non-trainable params: 0
_________________________________________________________________
```

Figure 3.1: The model summary

```
16/16 [==============================] - 0s 5ms/step - loss: 0.6006 - accuracy: 0.7728 - val_loss: 0.6559 -
val_accuracy: 0.7091
```

Figure 3.2: Training loss and training accuracy

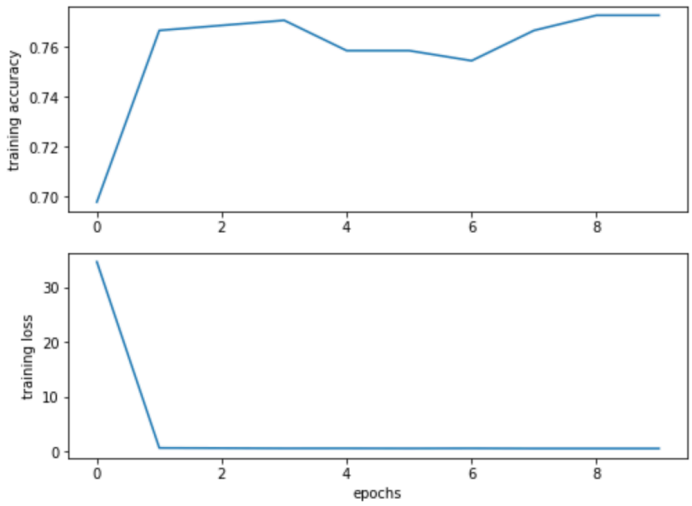


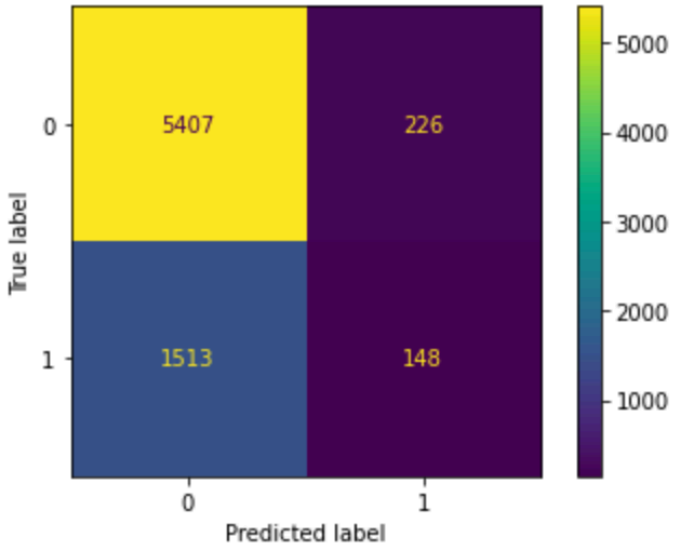
Figure 3.3: Training loss and training accuracy visualisation

```
45/228 [====>.........................] - ETA: 0s - loss: 0.6124 - accuracy: 0.7625
```

Figure 3.4: Test loss and test accuracy

```
[[5407  226]
 [1513  148]]
TP: 148
FP: 226
TN: 5407
FN: 1513
```

Figure 3.5: Prediction evaluation




```
Using SKLEARN.
Accuracy: 0.7615848642720043
No utility for specificity alone
Sensitivity: 0.08910295003010235
Precision: 0.39572192513368987
F1-Score: 0.14545454545454545
(7294,)
(7294, 1)
```

Figure 3.6: Confusion matrix for Prediction evaluation

	precision	recall	f1-score	support
0	0.78	0.96	0.86	5633
1	0.40	0.09	0.15	1661
accuracy			0.76	7294
macro avg	0.59	0.52	0.50	7294
weighted avg	0.69	0.76	0.70	7294

Figure 3.7: Confusion matrix for Prediction evaluation

- Observation: The model started to make better predictions by increasing the size of the testing dataset. Therefore, I will use this model for future experiments and make an adjustment based on the results to enhance our model performance.

4 Comparisons between architectures

Experiments based on the model number (1) from last section with some adjustments

4.1 Experiments based on modifying hidden layers

Model (2) - Increasing hidden layers from 2 to 3 (Figure 4.1).

- Training and validation results (Figure 4.2) and (Figure 4.3).

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 12)                120
_________________________________________________________________
dense_1 (Dense)              (None, 12)                156
_________________________________________________________________
dense_2 (Dense)              (None, 12)                156
_________________________________________________________________
dense_3 (Dense)              (None, 1)                 13
=================================================================
Total params: 445
Trainable params: 445
Non-trainable params: 0
```

Figure 4.1: The model summary

```
16/16 [==============================] - 0s 4ms/step - loss: 0.5769 - accuracy: 0.7688 - val_loss:
0.5805 - val_accuracy: 0.7455
```

Figure 4.2: Training loss and training accuracy

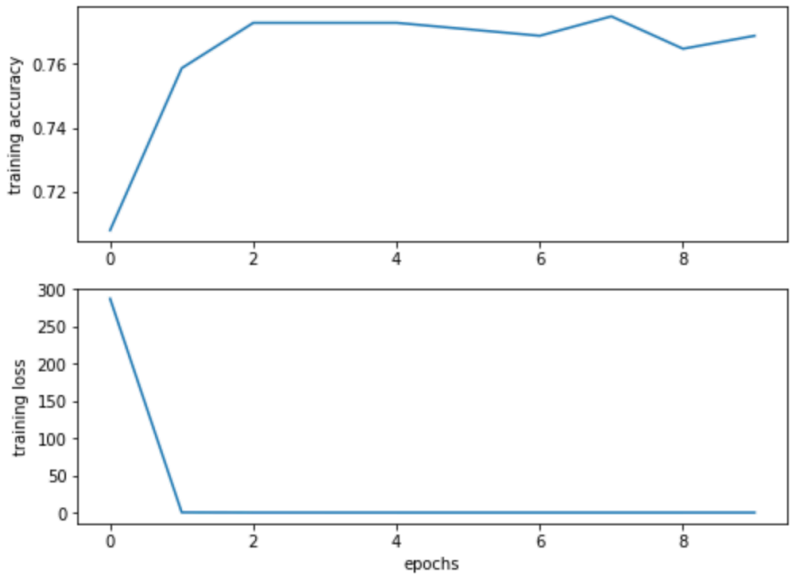


Figure 4.3: Training loss and training accuracy visualisation

```
228/228 [==============================] - 0s 1ms/step - loss: 0.5842 - accuracy: 0.7689
test Loss = 0.5841803550720215 test Accuracy = 0.768851101398468
```

Figure 4.4: Test loss and test accuracy

- Testing results (Figure 4.4).

- Evaluation metrics (Figure 4.5) and (Figure 4.6).

```
[[5579   54]
 [1632   29]]
TP: 29
FP: 54
TN: 5579
FN: 1632
```

Figure 4.5: Prediction evaluation

Model (3) - Decreasing the hidden layers from 3 to 1 (Figure 4.7).

- Training and validation results (Figure 4.8).

- Testing and evaluation (Figure 4.9).

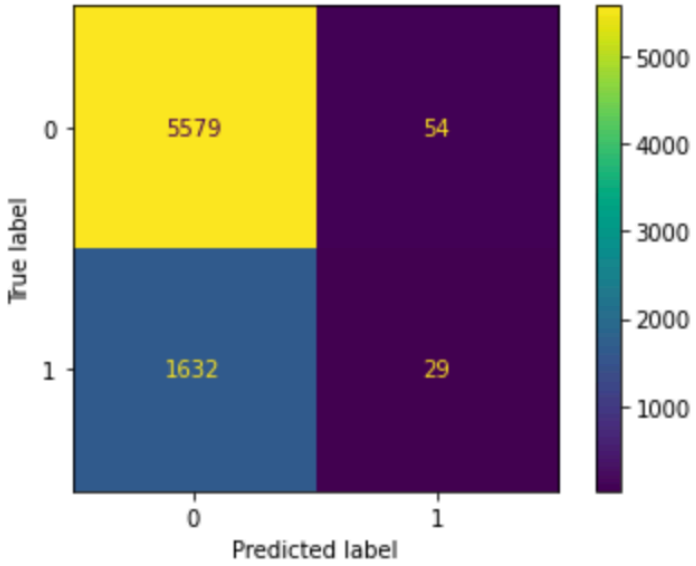
The hidden layers changes did not increase the accuracy metric. So I will start trying different numbers of nodes with one hidden layer for best sensitivity.

4.2 Experiments based on modifying nodes

Model (4) - Increasing the nodes number from 12 to 32 (Figure 4.10).

- Training and validation (Figure 4.11).

- Training results visualisation, Testing and evaluation (Figure 4.12) and (Figure 4.13).




```
Using SKLEARN.
Accuracy: 0.7688511105017823
No utility for specificity alone
Sensitivity: 0.017459361830222758
Precision: 0.3493975903614458
F1-Score: 0.033256880733944956
(7294,)
(7294, 1)
```

Figure 4.6: Confusion matrix for Prediction evaluation

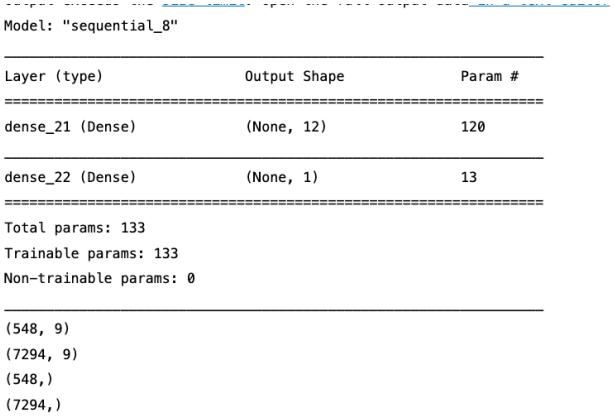


Figure 4.7: The model summary

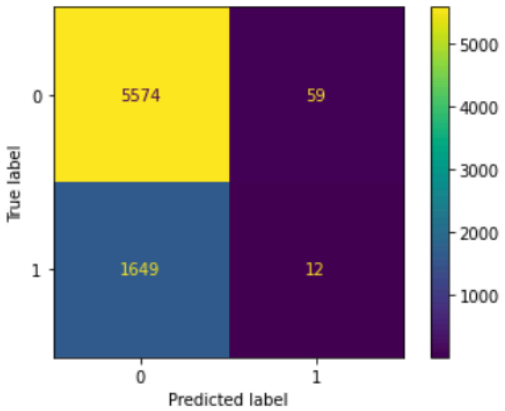
```
16/16 [==============================] - 0s 6ms/step - loss: 0.6490 - accuracy: 0.7708 - val_loss:
0.6070 - val_accuracy: 0.7455
```

Figure 4.8: Training loss and training accuracy

5 Different architectural techniques

- The model started overfitting based on the last experiments because the testing loss is higher than the training loss. The techniques in this section will help in reducing the overfitting problem.

```
===== Test Loss and Test Accuracy====
228/228 [==============================] - 0s 2ms/step - loss: 2.3731 - accuracy: 0.7658
test Loss = 2.3731143474578857 test Accuracy = 0.7658349275588989
[[0.41714686]
 [0.41714686]
 [0.41714686]
 ...
 [0.41714686]
 [0.41714686]
 [0.41714686]]
===== Evaluation metrics====
[[5574   59]
 [1649   12]]
TP: 12
FP: 59
TN: 5574
FN: 1649
===== Visualisation evaluation metrics ====
```




```
Using SKLEARN.
Accuracy: 0.7658349328214972
No utility for specificity alone
Sensitivity: 0.007224563515954244
Precision: 0.16901408450704225
F1-Score: 0.013856812933025405
(7294,)
(7294, 1)
```

Figure 4.9: The model performance

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 32)                320
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 33
=================================================================
Total params: 353
Trainable params: 353
Non-trainable params: 0
_________________________________________________________________
(548, 9)
(7294, 9)
(548,)
(7294,)
```

Figure 4.10: The model summary

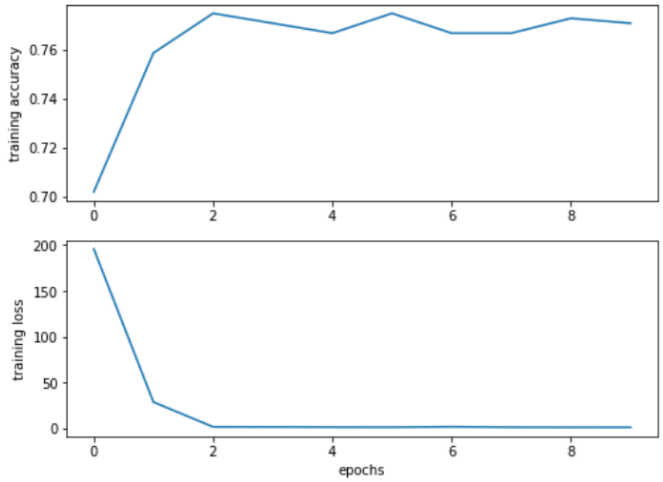
```
Epoch 10/10
16/16 [==============================] - 0s 5ms/step - loss: 0.6429 - accuracy: 0.7708 - val_loss: 0.5895 - val_accuracy: 0.7636
```

Figure 4.11: Training loss/ accuracy Validation loss/ accuracy

Summary

Model	Training Accuracy	Training Loss
1	0.77	0.60
2	0.76	0.57
3	0.77	0.64
4	0.77	0.64

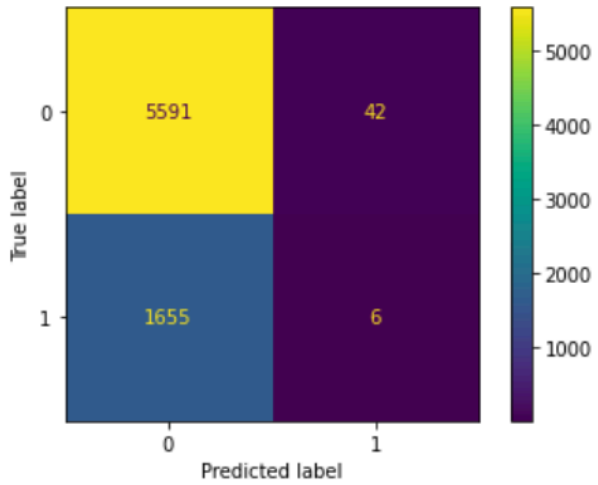
```
===== Training Loss and Training Accuracy====
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```




```
===== Test Loss and Test Accuracy====
228/228 [==============================] - 0s 2ms/step - loss: 6.0261 - accuracy: 0.7673
test Loss = 6.026131629943848 test Accuracy = 0.7673430442810059
[[0.42434245]
 [0.42434245]
 [0.42434245]
 ...
 [0.42434245]
 [0.42434245]
 [0.42434245]]
===== Evaluation metrics====
[[5591   42]
 [1655    6]]
TP: 6
FP: 42
TN: 5591
FN: 1655
```

Figure 4.12: The model performance

```
===== Visualisation evaluation metrics ====
```




```
Using SKLEARN.
Accuracy: 0.7673430216616397
No utility for specificity alone
Sensitivity: 0.003612281757977122
Precision: 0.125
F1-Score: 0.007021650087770626
(7294,)
(7294, 1)
```

Figure 4.13: Evaluation metrics

Model	Validation Accuracy	Validation Loss
1	0.70	0.65
2	0.74	0.58
3	0.74	0.60
4	0.76	0.58

Model	Testing Accuracy	Testing Loss
1	0.76	0.58
2	0.76	0.58
3	0.76	2.37
4	0.76	6.02

Based on evaluation metrics

Model	Sensitivity	Accuracy	Precision	F1-score
1	0.08	0.76	0.39	0.14
2	0.017	0.76	0.34	0.03
3	0.007	0.76	0.16	0.013
4	0.003	0.76	0.12	0.00

5.1 Reduce the network's size

Model(5) - One hidden layer (Figure 5.1). - Nodes number = 10 - epochs =10

- Training and validation results (Figure 5.2) and (Figure 5.3).

Note The model stopped before starting overfitting. - Testing and evaluation (Figure 5.4).

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_25 (Dense)             (None, 10)                100
_________________________________________________________________
dense_26 (Dense)             (None, 1)                 11
=================================================================
Total params: 111
Trainable params: 111
Non-trainable params: 0
_________________________________________________________________
(548, 9)
(7294, 9)
(548,)
(7294,)
```

Figure 5.1: The model summary

```
16/16 [==============================] - 0s 4ms/step - loss: 0.6131 - accuracy: 0.7728 - val_loss:
0.6214 - val_accuracy: 0.7636
```

Figure 5.2: Training loss/ accuracy and validation loss/ accuracy

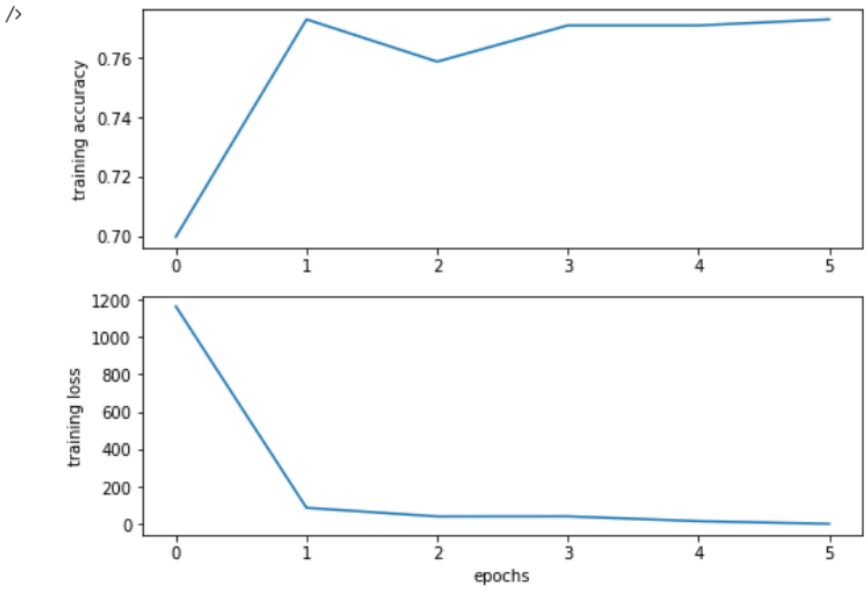
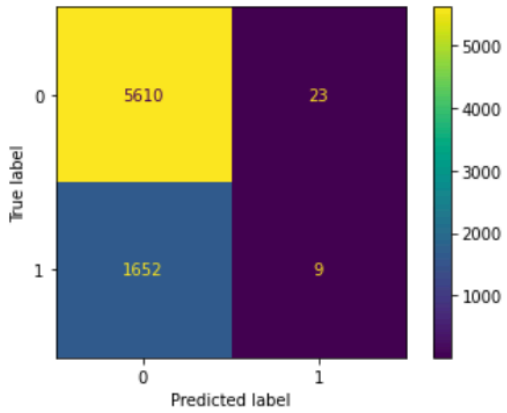


Figure 5.3: Training loss/ accuracy visualisation

```
===== Test Loss and Test Accuracy====
228/228 [==============================] - 0s 2ms/step - loss: 23.3182 - accuracy: 0.7704
test Loss = 23.318166732788086 test Accuracy = 0.770359218120575
[[0.4472854]
 [0.4472854]
 [0.4472854]
 ...
 [0.4472854]
 [0.4472854]
 [0.4472854]]
===== Evaluation metrics====
[[5610   23]
 [1652    9]]
TP: 9
FP: 23
TN: 5610
FN: 1652
===== Visualisation evaluation metrics ====
```




```
Using SKLEARN.
Accuracy: 0.7703591993419249
No utility for specificity alone
Sensitivity: 0.005418422636965683
Precision: 0.28125
F1-Score: 0.010632014176018901
(7294,)
(7294, 1)
```

Figure 5.4: The model performance

5.2 L2 regularization with batch normalisation

Model(6) - Epochs =10 - Validation split = 0.3 - Adding two hidden layer with batch normalisation - Using l2 regularization (Figure 5.5).

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_306 (Dense)            (None, 32)                320
_________________________________________________________________
batch_normalization_6 (Batch (None, 32)                128
_________________________________________________________________
re_lu_4 (ReLU)               (None, 32)                0
_________________________________________________________________
dense_307 (Dense)            (None, 32)                1056
_________________________________________________________________
batch_normalization_7 (Batch (None, 32)                128
_________________________________________________________________
re_lu_5 (ReLU)               (None, 32)                0
_________________________________________________________________
dense_308 (Dense)            (None, 1)                 33
=================================================================
Total params: 1,665
Trainable params: 1,537
Non-trainable params: 128
```

Figure 5.5: The model summary

- The model stopped before overfitting but the test loss was very high (Figure 5.6).
- Model Evaluation (Figure 5.7).

5.3 Dropout

Model(7) By adding dropout (0.5) to the model (Figure 5.8).

- Training and validation results (Figure 5.9).
- Testing and Evaluation metrics (Figure 5.10) and (Figure 5.11).

```
test Loss = 0.9172424077987671 test Accuracy = 0.7792706489562988
```

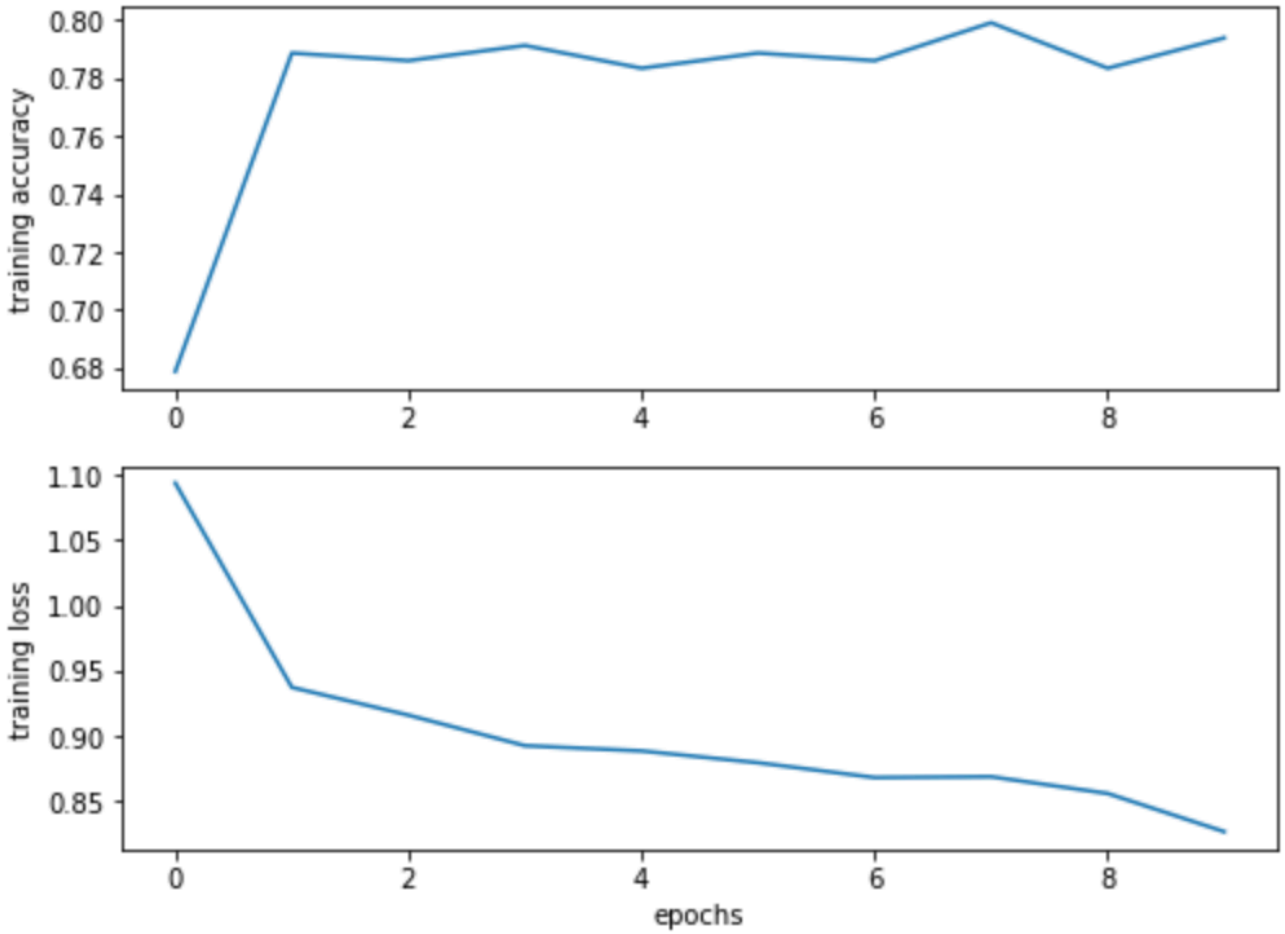
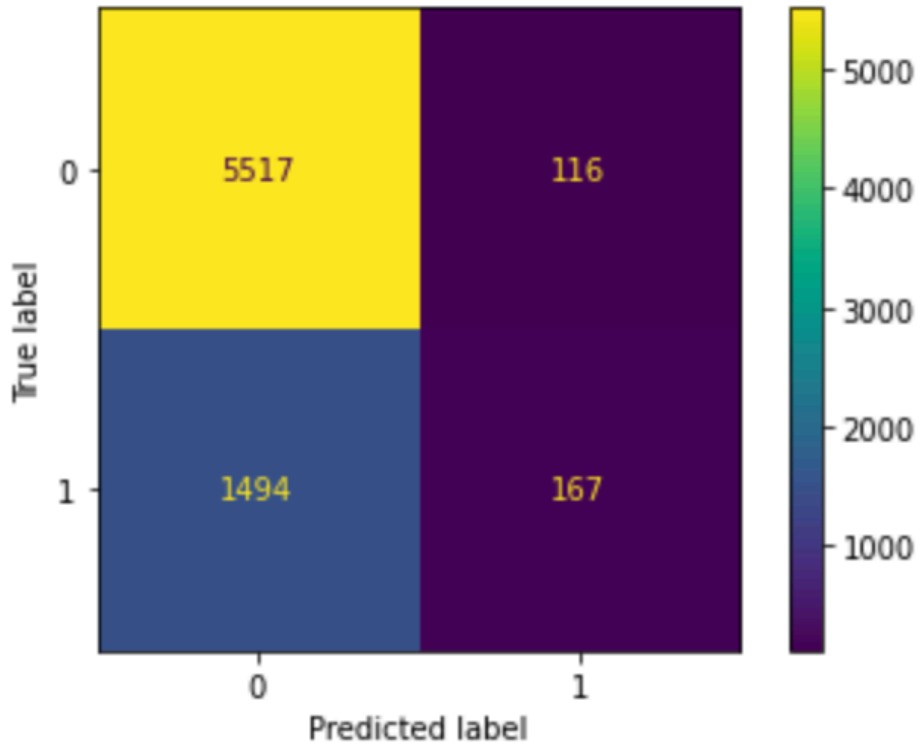


Figure 5.6: The model performance

```
TP: 167
FP: 116
TN: 5517
FN: 1494
```




```
Using SKLEARN.
Accuracy: 0.7792706333973128
No utility for specificity alone
Sensitivity: 0.10054184226369657
Precision: 0.5901060070671378
F1-Score: 0.17181069958847736
```

Figure 5.7: Confusion matrix for prediction evaluation

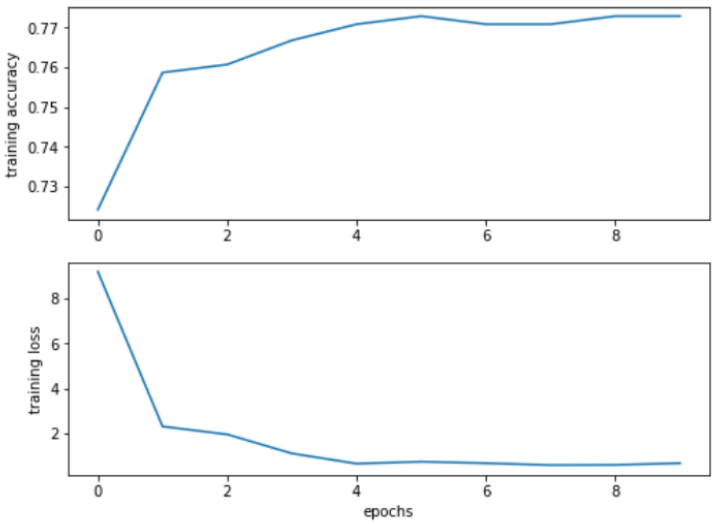
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 12)                120
_________________________________________________________________
dropout (Dropout)            (None, 12)                0
_________________________________________________________________
dense_1 (Dense)              (None, 12)                156
_________________________________________________________________
dropout_1 (Dropout)          (None, 12)                0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 13
=================================================================
Total params: 289
Trainable params: 289
Non-trainable params: 0
_________________________________________________________________
(548, 9)
(7294, 9)
(548,)
(7294,)
```

Figure 5.8: The model summary

```
16/16 [==============================] - 0s 4ms/step - loss: 0.6676 - accuracy: 0.7728 - val_loss:
0.5707 - val_accuracy: 0.7636
```

Figure 5.9: Training loss/ accuracy and validation loss/ accuracy

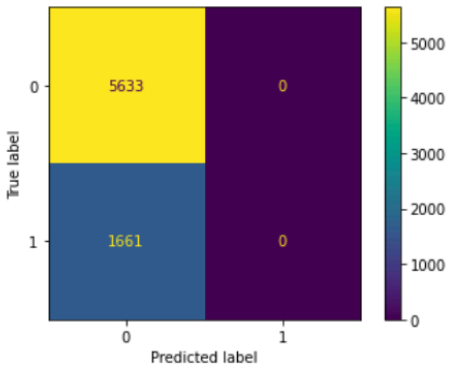
```
===== Training Loss and Training Accuracy====
dict_keys(['loss', 'accuracy', 'val_loss', 'val_accuracy'])
```




```
===== Test Loss and Test Accuracy====
228/228 [==============================] - 0s 2ms/step - loss: 0.5991 - accuracy: 0.7723
test Loss = 0.5990926027297974 test Accuracy = 0.7722786068916321
[[0.3759624]
 [0.3759624]
 [0.3759624]
 ...
 [0.3759624]
 [0.3759624]
 [0.3759624]]
===== Evaluation metrics====
[[5633    0]
 [1661    0]]
TP: 0
FP: 0
TN: 5633
FN: 1661
```

Figure 5.10: The model performance

```
===== Visualisation evaluation metrics ====
/>
```




```
Using SKLEARN.
Accuracy: 0.77227858513847
No utility for specificity alone
Sensitivity: 0.0
Precision: 0.0
F1-Score: 0.0
(7294,)
(7294, 1)
```

Figure 5.11: Evaluation metrics

5.4 Batch normalization with dropout

Model (8) - Nodes = 32 in both layers - Two layers of batch normalization (Figure 5.12). - One layer dropout = 0.5 - Epochs= 10 - Validation split = 0.3

Layer (type)	Output Shape	Param #
dense_300 (Dense)	(None, 32)	320
batch_normalization_2 (Batch	(None, 32)	128
dense_301 (Dense)	(None, 32)	1056
batch_normalization_3 (Batch	(None, 32)	128
dropout_107 (Dropout)	(None, 32)	0
dense_302 (Dense)	(None, 1)	33

Total params: 1,665
Trainable params: 1,537
Non-trainable params: 128

Figure 5.12: The model summary

- Training and testing results (Figure 5.13).

```
test Loss = 0.6064621210098267 test Accuracy = 0.7462297677993774
```

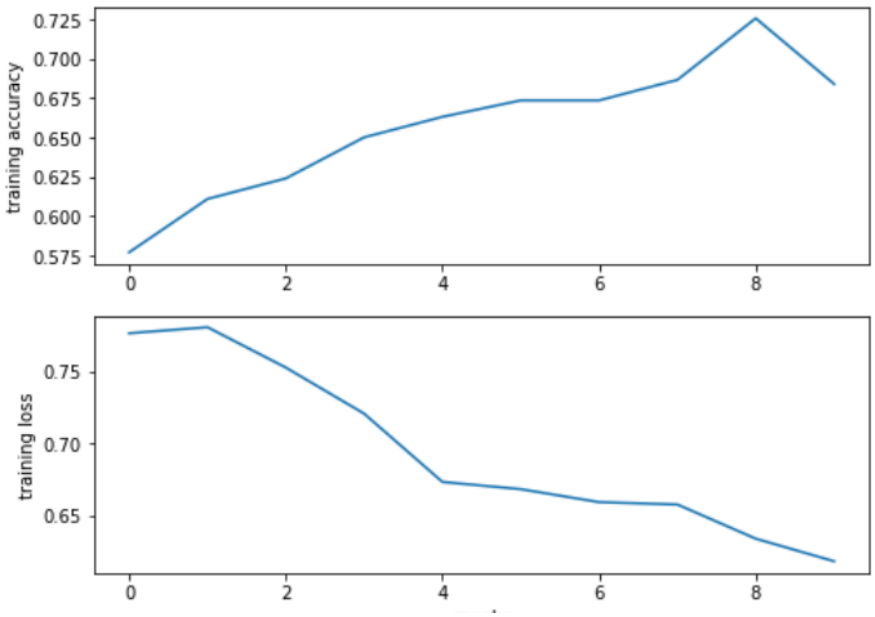


Figure 5.13: Training and Testing results

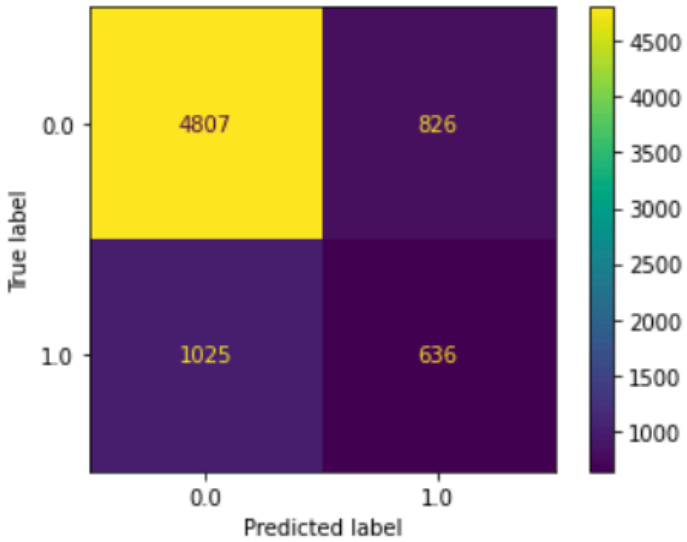
- Evaluation metrics (Figure 5.14).

Note: The model is still not overfitting.

- Comparison between the models Based on Evaluation metrics :

Model	Sensitivity	Accuracy	Precision	F1-score
5	0.00	0.77	0.28	0.10
6	0.10	0.77	0.59	0.17
7	0.0	0.77	0.0	0.0
8	0.38	0.74	0.43	0.40

```
TP: 636
FP: 826
TN: 4807
FN: 1025
```




```
Using SKLEARN.
Accuracy: 0.7462297778996435
No utility for specificity alone
Sensitivity: 0.38290186634557494
Precision: 0.43502051983584133
F1-Score: 0.4073006724303555
(7294,)
(7294, 1)
```

Figure 5.14: Evaluation metrics

- The model accuracy stayed at the same number in all models, but the sensitivity started increasing with using batch normalisation and dropout together in the model (8), and this model achieved the highest sensitivity (0.38).
- By looking at the training and validation loss in both models, model 8 stops before starting overfitting and the training and validation loss is very close, which the model generalisation became better and using batch normalisation and dropout is enhanancing the model performance (Figure 5.15).

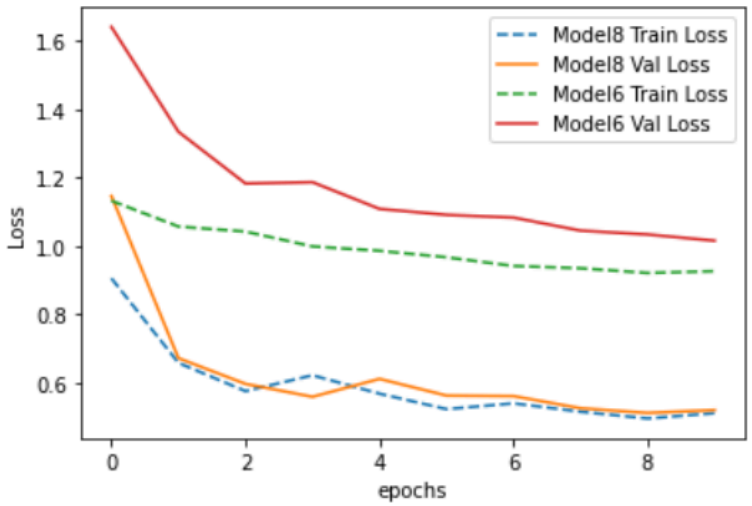


Figure 5.15: Models comparison based on training and validation loss

Training and validation loss shows better performance in model 8 than model6.

- Test accuracy comparison (Figure 5.16).

The model performance in testing data can be still modified in the future development of the network to achieve better testing accuracy. The optimal model for solving the problem is model (8) and still can do more improvment to enhance the model testing. As a result, Batch normalisation is essential when we are building a binary neural network especially if we have imbalanced datasets and we want to improve the model training (Sari, Belbahri and Nia, 2020).

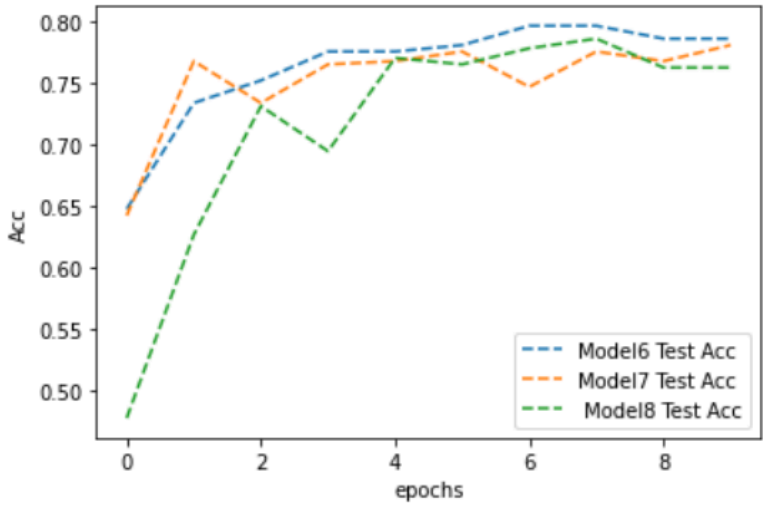


Figure 5.16: Models test accuracy

6 Conclusion

This dataset lacks balance between two classes and this has initially resulted in mode under-fitting. However, after introducing stratifying and adjusting the network, the model was over-fit. The introduction of regularization techniques was helpful in decreasing overfitting, ensuring improved generalization.

References

Moro, S., Cortez, P. and Rita, P. (2014) 'A data-driven approach to predict the success of bank telemarketing', *Decision Support Systems*, 62, pp. 22–31. doi:10.1016/j.dss.2014.03.001.

Sari, E., Belbahri, M. and Nia, V.P. (2020) 'How Does Batch Normalization Help Binary Training?' Available at: http://arxiv.org/abs/1909.09139 (Accessed: 11 April 2022).

Vidiyala, R. (2020) *Confusion Matrix in a nutshell*. Analytics Vidhya. Available at: https://medium.com/analytics-vidhya/a-z-of-confusion-matrix-under-5-mins-147c1b4467ab (Accessed: 11 April 2022).