FOUNDATIONS OF INTELLIGENT SYSTEMS

PROJECT 3

Submitted to:
Prof. Leonid Reznik

By:
Shivang Bokolia

Executive Summary

This project focuses on training an Artificial Neural Network that approximates the Android Application Security Expert System (Project 2). It also includes the evaluation of the Machine Learning Model's performance on different choices of neurons for a certain number of hidden layers. The evaluation of the model is performed using parameters like time consumption, memory consumption, and accuracy of the model.

The model used in the evaluation is a multilayer perceptron (MLP) model and is trained and tested using the data obtained from the Android Application Expert System. The dataset includes a combination of all the inputs of the Android ES and the output of the security system. The total data includes 570 records and these records are all values for 6 different features of the Android System which include – Android OS Version, Developer Options, Wi-Fi, Verification of applications, Device Lock, and Third-Party Applications. The predicted value from the model is the security evaluation of the device which is categorized into 4 categories – 1(Low Risk), 2(Medium Risk), 3(High Risk), 4(Critical Risk). Most of the input data involve only binary values and the OS version has 3 values ranging from 0 to 2.

The complete project is built on Python and uses libraries like sklearn and pandas. The Machine Learning Model used in the project is MLP Classifier. The model was trained over multiple sets of neurons in the hidden layer and the outcome for all these neurons was noted down. The training was stopped when the model was able to provide an accuracy of over 99% over the testing dataset. The accuracy was calculated using the confusion matrix and the report was generated using sklearn libraries and the graphs were plotted for the loss curve of the MLP Classifier model using the matplotlib library. For the results, the increasing number of hidden layers and neurons in hidden layers is displayed using the confusion matrix, report of the model and, plotting the loss curve of the MLP Classifier model.

Requirements

The main aim of this project is to train a Machine Learning Model and make design decisions for the model to obtain the best results. This machine learning model is built to approximate the ES output built-in Project 2. ES can be computationally expensive which in today's time cannot be considered good for a mobile device. The machine learning model predicts the output by approximation, hence reducing the amount of computation required. The design decisions included in building the Neural Network involved the decision for the number of neurons in the hidden layer and the best combination that provided maximum accuracy. It also includes an evaluation of the memory and time consumption involved regarding the model.

The requirements for this project can be split into multiple parts:

1. Dataset generation
2. Creating a Neural Network model by splitting the data and training and testing the model.
3. Making changes to the number of neurons in the hidden layer to provide maximum accuracy.
4. Evaluate the memory and time consumption as the number of neurons increases.

Implementation

1. **Dataset Generation**
   The dataset for the model was generated using the inputs of the Android Application ES. All the combinations and their outputs were taken and added to the dataset for the model. The inputs consist of – Android OS Version, Developer Options, Wi-Fi, Verification of applications, Device Lock, and Third-Party Applications, and the output expected from the model is categorized into 4 categories – 1(Low Risk), 2(Medium Risk), 3(High Risk), 4(Critical Risk). The dataset generated includes a total of 570 records. Most of the data in the dataset is binary data i.e., 0 and 1 and the Android OS version has data in the range 0 to 2.

2. **Values of the Metrics**
   Screen Lock – 0 and 1
   OS Version – 0, 1 and 2
   Developer's Options – 0 and 1
   Wi-Fi – 0 and 1
   Verified Applications – 0 and 1
   Third Party Applications – 0 and 1
   Security Level – 1, 2, 3 and 4

3. **Development of the Neural Network Model**
   The complete program has been coded in python and runs on python. It uses different packages like pandas (https://pandas.pydata.org/), sklearn (https://scikit-learn.org/stable/), and matplotlib (https://matplotlib.org/). The multilayer perceptron model used in the project is "MLP Classifier" which was imported from the sklearn library.

   a. Installing all the packages:
      i. Pandas – "pip install pandas" (https://pandas.pydata.org/pandas-docs/stable/getting_started/install.html)
      ii. MatplotLib – "pip install -U matplotlib" (https://matplotlib.org/stable/users/installing.html)
      iii. Scikit-Learn – "pip install -U scikit-learn" (https://scikit-learn.org/stable/install.html)
      iv. Openpyxl – "pip install openpyxl" (https://pypi.org/project/openpyxl/)
      v. Memory_profiler – "pip install -U memory_profiler" (https://pypi.org/project/memory-profiler/)

   b. Splitting of data:
      The dataset generated above was split into the testing set and the training set with a ratio of 1:3 i.e., 33% testing set and 67% training set. This was done using the train_test_split that was imported from the sklearn library.

c. Training the model:
   After splitting the data, the model was then trained using the training set. The training started with 3 neurons in the hidden layer. The activation function that was being used by the model was set to be "tanh" and the learning rate was set at 0.001. "cross_val_score" was used for training the data and checking the accuracy of the training data.
   i. Cross_val_score:
      cross-validation is the statistical method to estimate the skill of the machine learning model. It shuffles the dataset provided to it and splits it into k groups. The first group is taken as the test dataset and the remaining is taken as the training dataset. It then fits the model on the training dataset and evaluates it on the testing dataset. It then holds on to the evaluation score and discards the model. It does the same process multiple times to remove any sort of bias towards the dataset splitting. The score provided by cross-validation is unbiased and is an average of about 5 different evaluations in this project.

d. Testing the model:
   After the data has been trained on the training dataset, it is then fit to the model and predicts the values for the testing dataset. The predictions are compared to the testing dataset outputs and a confusion matrix is created. The accuracy of the model is then calculated using the confusion matrix generated along with a report that includes the precision, recall, f1 – score, and accuracy values.

e. Repetition until maximum accuracy:
   Step 'b' and 'c' are performed multiple times with different number of neurons in the hidden layer and are repeated until maximum accuracy is obtained for both the cross-evaluation score and the ANN model.

f. Calculating Accuracy:
   The accuracy for the ANN model is calculated using the confusion matrix. The diagonal sum of the matrix is taken and is divided by the total sum of all the elements in the matrix. This provides with the accuracy of the ANN model.

g. Displaying the output:
   The output for the code is displayed through the following:
   i. Cross-Evaluation Score: The score obtained on training the dataset multiple times with unbiased data.
   ii. Confusion Matrix: The confusion matrix shows the accuracies and inaccuracies performed by the model.
   iii. ANN Model Accuracy: The accuracy of the model while running the testing dataset.

iv. ANN Model Report: The report consists of the precision, recall, f1 – score, and accuracy of the model used.
v. Loss-Curve Graph: The graph that represents the loss curve depending on the number of neurons used in the hidden layer.
vi. Time and memory consumption: This is displayed using the time (https://docs.python.org/3/library/time.html) and memory_profiler (https://pypi.org/project/memory-profiler/) libraries in python.

4. User Guide

As the python code is opened in an IDE, the data from the dataset will be obtained in the "# Read the excel file and collect data" section.
The number of neurons in the code can be changed at this line:

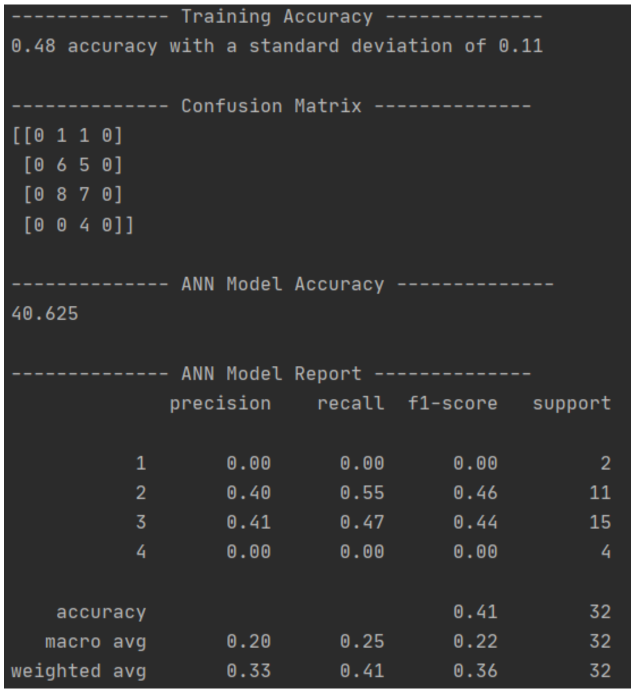
```
mlp = MLPClassifier(hidden_layer_sizes=(192), max_iter=1000, activation='tanh', solver='adam', learning_rate='constant', learning_rate_init=0.001, random_state=0)
```

The hidden_layer_sizes will change the number of neurons in the hidden layer and if more hidden layers are to be added it can be done so using parathesis and comma i.e., (first_layer_neurons, second_layer_neurons,…).

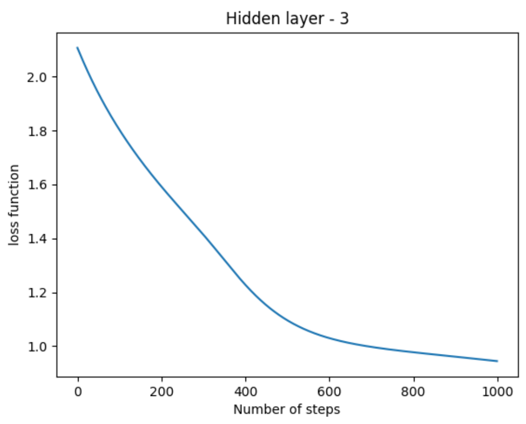
The code can be run with the following command "python -m memory_profiler main.py".

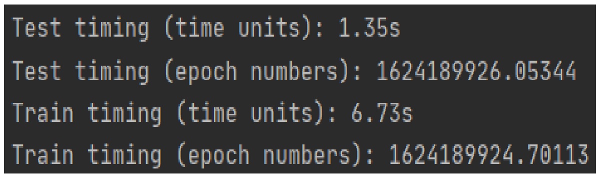
Results

The first experiment was performed with a single hidden layer and with **3 neurons** as mentioned in the project description. The following output was obtained when the code was run:

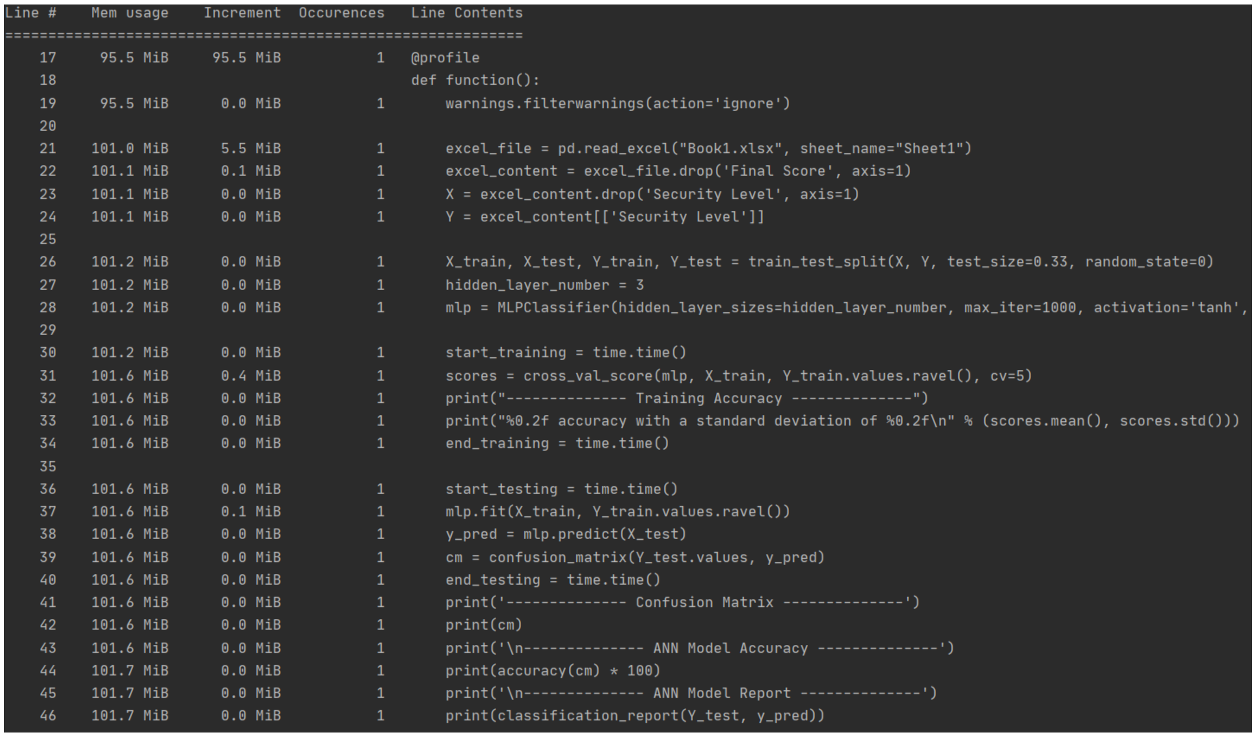


```
-------------- Training Accuracy --------------
0.48 accuracy with a standard deviation of 0.11

-------------- Confusion Matrix --------------
[[0 1 1 0]
 [0 6 5 0]
 [0 8 7 0]
 [0 0 4 0]]

-------------- ANN Model Accuracy --------------
40.625

-------------- ANN Model Report --------------
              precision    recall  f1-score   support

           1       0.00      0.00      0.00         2
           2       0.40      0.55      0.46        11
           3       0.41      0.47      0.44        15
           4       0.00      0.00      0.00         4

    accuracy                           0.41        32
   macro avg       0.20      0.25      0.22        32
weighted avg       0.33      0.41      0.36        32
```





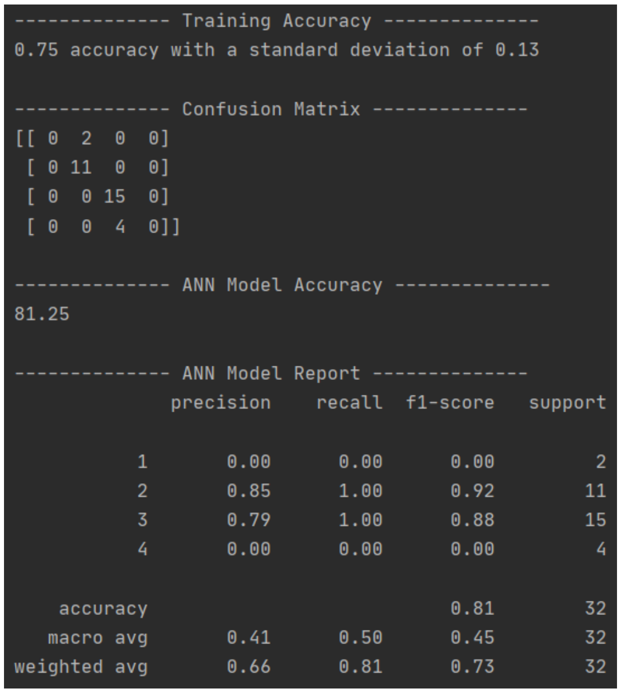
```
Test timing (time units): 1.35s
Test timing (epoch numbers): 1624189926.05344
Train timing (time units): 6.73s
Train timing (epoch numbers): 1624189924.70113
```



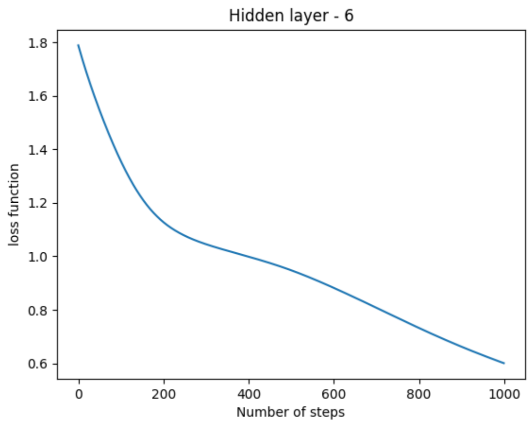
```
Line #    Mem usage    Increment   Occurences   Line Contents
============================================================
    17     95.5 MiB     95.5 MiB           1   @profile
    18                                         def function():
    19     95.5 MiB      0.0 MiB           1       warnings.filterwarnings(action='ignore')
    20
    21    101.0 MiB      5.5 MiB           1       excel_file = pd.read_excel("Book1.xlsx", sheet_name="Sheet1")
    22    101.1 MiB      0.1 MiB           1       excel_content = excel_file.drop('Final Score', axis=1)
    23    101.1 MiB      0.0 MiB           1       X = excel_content.drop('Security Level', axis=1)
    24    101.1 MiB      0.0 MiB           1       Y = excel_content[['Security Level']]
    25
    26    101.2 MiB      0.0 MiB           1       X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.33, random_state=0)
    27    101.2 MiB      0.0 MiB           1       hidden_layer_number = 3
    28    101.2 MiB      0.0 MiB           1       mlp = MLPClassifier(hidden_layer_sizes=hidden_layer_number, max_iter=1000, activation='tanh',
    29
    30    101.2 MiB      0.0 MiB           1       start_training = time.time()
    31    101.6 MiB      0.4 MiB           1       scores = cross_val_score(mlp, X_train, Y_train.values.ravel(), cv=5)
    32    101.6 MiB      0.0 MiB           1       print("-------------- Training Accuracy --------------")
    33    101.6 MiB      0.0 MiB           1       print("%0.2f accuracy with a standard deviation of %0.2f\n" % (scores.mean(), scores.std()))
    34    101.6 MiB      0.0 MiB           1       end_training = time.time()
    35
    36    101.6 MiB      0.0 MiB           1       start_testing = time.time()
    37    101.6 MiB      0.1 MiB           1       mlp.fit(X_train, Y_train.values.ravel())
    38    101.6 MiB      0.0 MiB           1       y_pred = mlp.predict(X_test)
    39    101.6 MiB      0.0 MiB           1       cm = confusion_matrix(Y_test.values, y_pred)
    40    101.6 MiB      0.0 MiB           1       end_testing = time.time()
    41    101.6 MiB      0.0 MiB           1       print('-------------- Confusion Matrix --------------')
    42    101.6 MiB      0.0 MiB           1       print(cm)
    43    101.6 MiB      0.0 MiB           1       print('\n-------------- ANN Model Accuracy --------------')
    44    101.7 MiB      0.0 MiB           1       print(accuracy(cm) * 100)
    45    101.7 MiB      0.0 MiB           1       print('\n-------------- ANN Model Report --------------')
    46    101.7 MiB      0.0 MiB           1       print(classification_report(Y_test, y_pred))
```

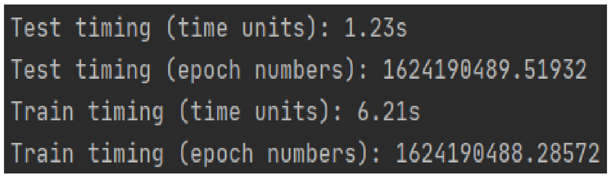
It can be seen that with just 3 neurons in the hidden layer, the accuracy of the model was about 41% despite the training set having an accuracy of 48%. The time taken to complete the testing was 1.35sec and training was 6.73sec. The third image shows the memory consumption of the python script running for the machine learning model. It can be seen that after the predictions have been compared, the memory consumed is 101.6 MiB.

For the next step, the number of neurons were doubled to **6 neurons**:

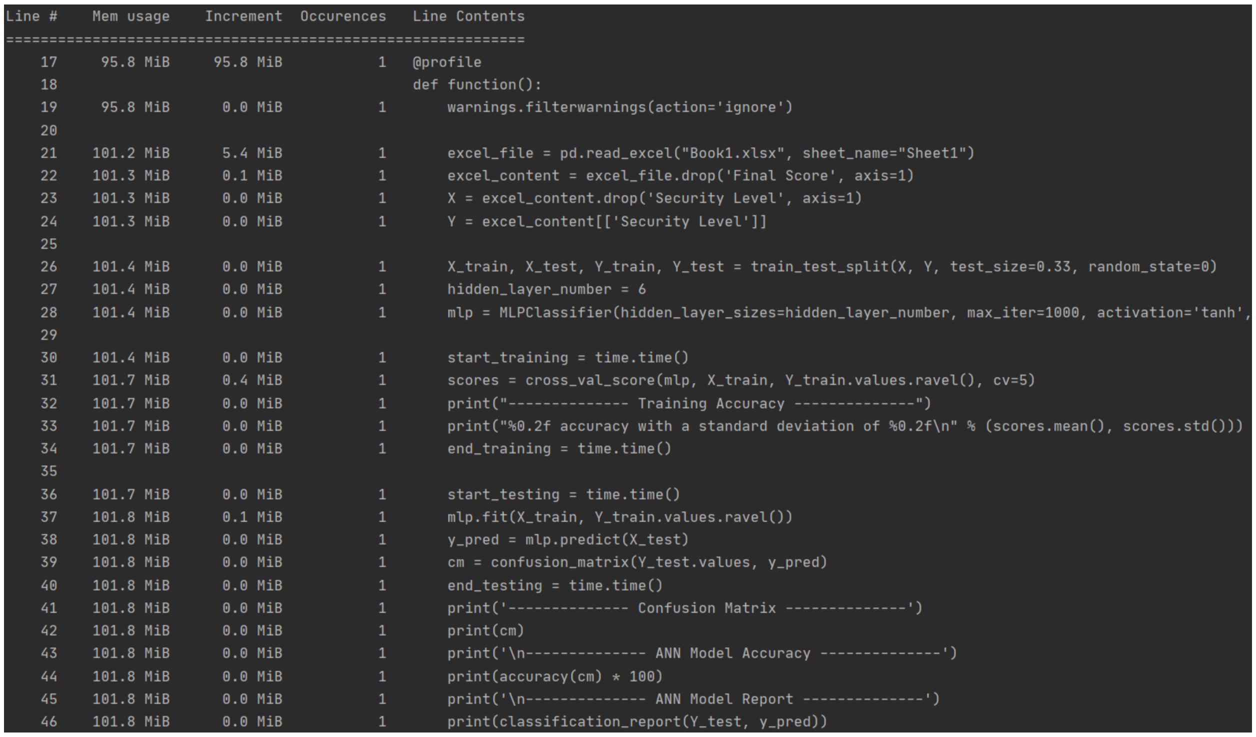


```
-------------- Training Accuracy --------------
0.75 accuracy with a standard deviation of 0.13

-------------- Confusion Matrix --------------
[[ 0  2  0  0]
 [ 0 11  0  0]
 [ 0  0 15  0]
 [ 0  0  4  0]]

-------------- ANN Model Accuracy --------------
81.25

-------------- ANN Model Report --------------
              precision    recall  f1-score   support

           1       0.00      0.00      0.00         2
           2       0.85      1.00      0.92        11
           3       0.79      1.00      0.88        15
           4       0.00      0.00      0.00         4

    accuracy                           0.81        32
   macro avg       0.41      0.50      0.45        32
weighted avg       0.66      0.81      0.73        32
```





```
Test timing (time units): 1.23s
Test timing (epoch numbers): 1624190489.51932
Train timing (time units): 6.21s
Train timing (epoch numbers): 1624190488.28572
```



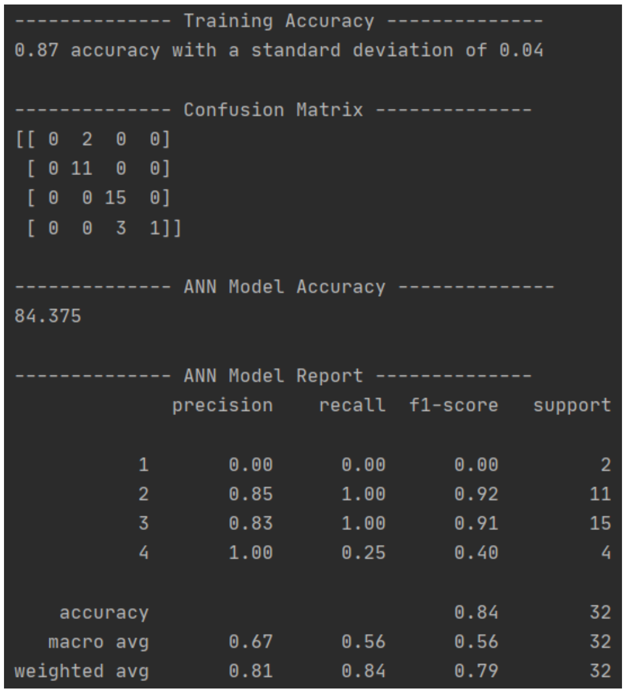
```
Line #    Mem usage    Increment  Occurences   Line Contents
============================================================
    17     95.8 MiB     95.8 MiB           1   @profile
    18                                         def function():
    19     95.8 MiB      0.0 MiB           1       warnings.filterwarnings(action='ignore')
    20
    21    101.2 MiB      5.4 MiB           1       excel_file = pd.read_excel("Book1.xlsx", sheet_name="Sheet1")
    22    101.3 MiB      0.1 MiB           1       excel_content = excel_file.drop('Final Score', axis=1)
    23    101.3 MiB      0.0 MiB           1       X = excel_content.drop('Security Level', axis=1)
    24    101.3 MiB      0.0 MiB           1       Y = excel_content[['Security Level']]
    25
    26    101.4 MiB      0.0 MiB           1       X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.33, random_state=0)
    27    101.4 MiB      0.0 MiB           1       hidden_layer_number = 6
    28    101.4 MiB      0.0 MiB           1       mlp = MLPClassifier(hidden_layer_sizes=hidden_layer_number, max_iter=1000, activation='tanh',
    29
    30    101.4 MiB      0.0 MiB           1       start_training = time.time()
    31    101.7 MiB      0.4 MiB           1       scores = cross_val_score(mlp, X_train, Y_train.values.ravel(), cv=5)
    32    101.7 MiB      0.0 MiB           1       print("-------------- Training Accuracy --------------")
    33    101.7 MiB      0.0 MiB           1       print("%0.2f accuracy with a standard deviation of %0.2f\n" % (scores.mean(), scores.std()))
    34    101.7 MiB      0.0 MiB           1       end_training = time.time()
    35
    36    101.7 MiB      0.0 MiB           1       start_testing = time.time()
    37    101.8 MiB      0.1 MiB           1       mlp.fit(X_train, Y_train.values.ravel())
    38    101.8 MiB      0.0 MiB           1       y_pred = mlp.predict(X_test)
    39    101.8 MiB      0.0 MiB           1       cm = confusion_matrix(Y_test.values, y_pred)
    40    101.8 MiB      0.0 MiB           1       end_testing = time.time()
    41    101.8 MiB      0.0 MiB           1       print('-------------- Confusion Matrix --------------')
    42    101.8 MiB      0.0 MiB           1       print(cm)
    43    101.8 MiB      0.0 MiB           1       print('\n-------------- ANN Model Accuracy --------------')
    44    101.8 MiB      0.0 MiB           1       print(accuracy(cm) * 100)
    45    101.8 MiB      0.0 MiB           1       print('\n-------------- ANN Model Report --------------')
    46    101.8 MiB      0.0 MiB           1       print(classification_report(Y_test, y_pred))
```

As it can be seen that the accuracy on increasing the number of neurons to 6 increased to 81% but the time in epoch numbers is more along with the memory consumption which increased to about 101.7MiB.

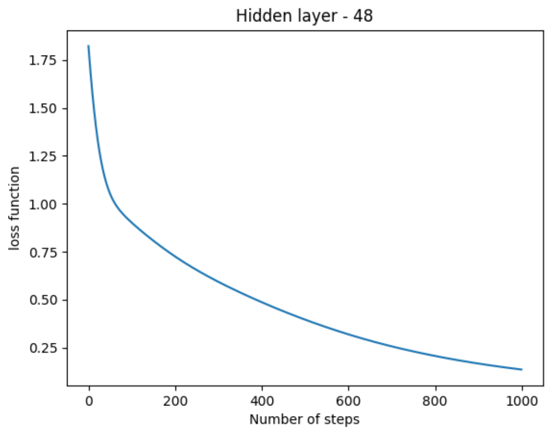
The number of neurons were doubled again to **12 neurons** and the output obtained did not have much of a difference in accuracy. The accuracy for 12 neurons was still 81% and the time consumed in epoch numbers was more again. The memory consumption was almost the same as before.

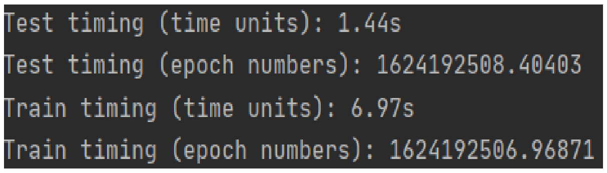
The number of neurons were doubled again to **24 neurons** and the output obtained did not have much of a difference in accuracy. The accuracy for 24 neurons was still 81% and the time consumed in epoch numbers was more again. The memory consumption was almost the same as before.

The number of neurons were doubled again to **48 neurons**:

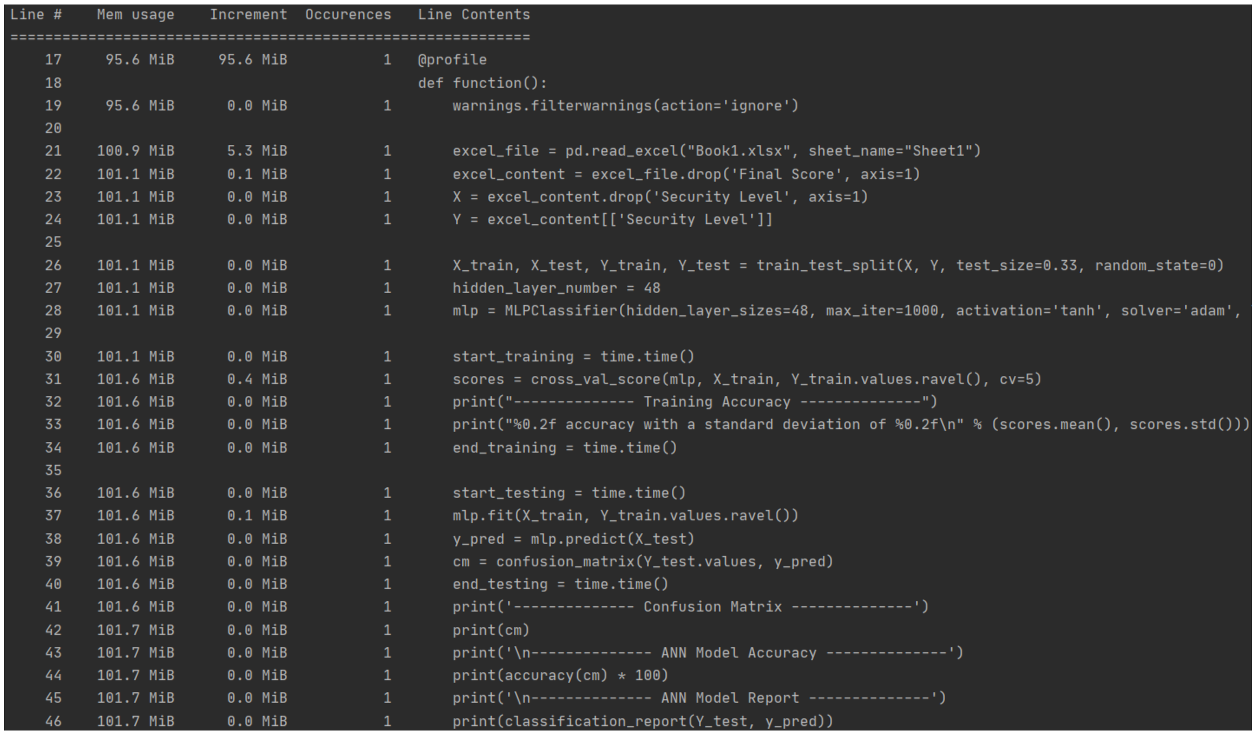


```
-------------- Training Accuracy --------------
0.87 accuracy with a standard deviation of 0.04

-------------- Confusion Matrix --------------
[[ 0  2  0  0]
 [ 0 11  0  0]
 [ 0  0 15  0]
 [ 0  0  3  1]]

-------------- ANN Model Accuracy --------------
84.375

-------------- ANN Model Report --------------
              precision    recall  f1-score   support

           1       0.00      0.00      0.00         2
           2       0.85      1.00      0.92        11
           3       0.83      1.00      0.91        15
           4       1.00      0.25      0.40         4

    accuracy                           0.84        32
   macro avg       0.67      0.56      0.56        32
weighted avg       0.81      0.84      0.79        32
```





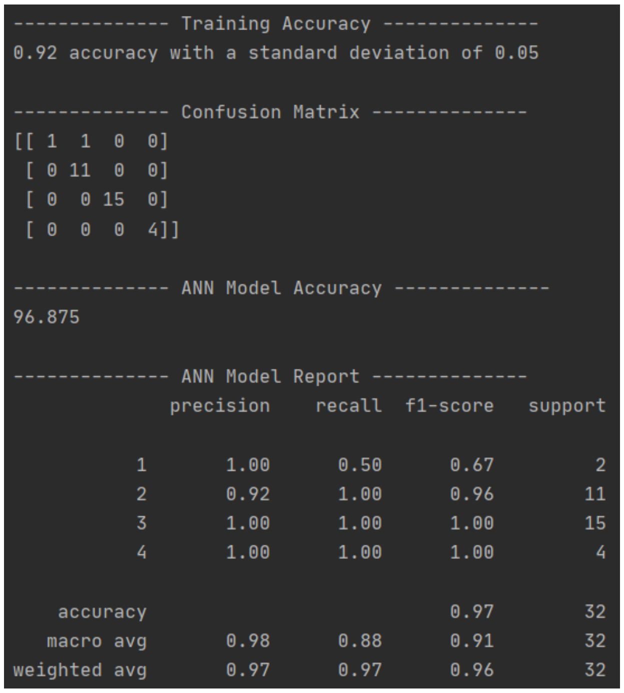
```
Test timing (time units): 1.44s
Test timing (epoch numbers): 1624192508.40403
Train timing (time units): 6.97s
Train timing (epoch numbers): 1624192506.96871
```



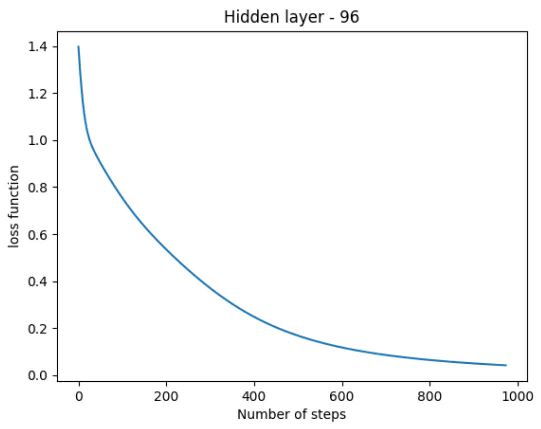
```
Line #    Mem usage    Increment  Occurences   Line Contents
============================================================
    17     95.6 MiB     95.6 MiB           1   @profile
    18                                         def function():
    19     95.6 MiB      0.0 MiB           1       warnings.filterwarnings(action='ignore')
    20
    21    100.9 MiB      5.3 MiB           1       excel_file = pd.read_excel("Book1.xlsx", sheet_name="Sheet1")
    22    101.1 MiB      0.1 MiB           1       excel_content = excel_file.drop('Final Score', axis=1)
    23    101.1 MiB      0.0 MiB           1       X = excel_content.drop('Security Level', axis=1)
    24    101.1 MiB      0.0 MiB           1       Y = excel_content[['Security Level']]
    25
    26    101.1 MiB      0.0 MiB           1       X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.33, random_state=0)
    27    101.1 MiB      0.0 MiB           1       hidden_layer_number = 48
    28    101.1 MiB      0.0 MiB           1       mlp = MLPClassifier(hidden_layer_sizes=48, max_iter=1000, activation='tanh', solver='adam',
    29
    30    101.1 MiB      0.0 MiB           1       start_training = time.time()
    31    101.6 MiB      0.4 MiB           1       scores = cross_val_score(mlp, X_train, Y_train.values.ravel(), cv=5)
    32    101.6 MiB      0.0 MiB           1       print("------------- Training Accuracy -------------")
    33    101.6 MiB      0.0 MiB           1       print("%0.2f accuracy with a standard deviation of %0.2f\n" % (scores.mean(), scores.std()))
    34    101.6 MiB      0.0 MiB           1       end_training = time.time()
    35
    36    101.6 MiB      0.0 MiB           1       start_testing = time.time()
    37    101.6 MiB      0.1 MiB           1       mlp.fit(X_train, Y_train.values.ravel())
    38    101.6 MiB      0.0 MiB           1       y_pred = mlp.predict(X_test)
    39    101.6 MiB      0.0 MiB           1       cm = confusion_matrix(Y_test.values, y_pred)
    40    101.6 MiB      0.0 MiB           1       end_testing = time.time()
    41    101.6 MiB      0.0 MiB           1       print('------------- Confusion Matrix -------------')
    42    101.7 MiB      0.0 MiB           1       print(cm)
    43    101.7 MiB      0.0 MiB           1       print('\n------------- ANN Model Accuracy -------------')
    44    101.7 MiB      0.0 MiB           1       print(accuracy(cm) * 100)
    45    101.7 MiB      0.0 MiB           1       print('\n------------- ANN Model Report -------------')
    46    101.7 MiB      0.0 MiB           1       print(classification_report(Y_test, y_pred))
```

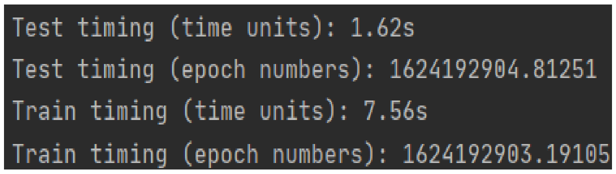
As it can be seen that on increasing the number of neurons to 48 the accuracy increased to about 84%. Since the number of neurons in the hidden layers doubled the time taken for the code also increased in epoch numbers. The memory consumption was almost 101.7 but in bigger terms was close to 101.6MiB.

The number of neurons were doubled to **96 neurons**:

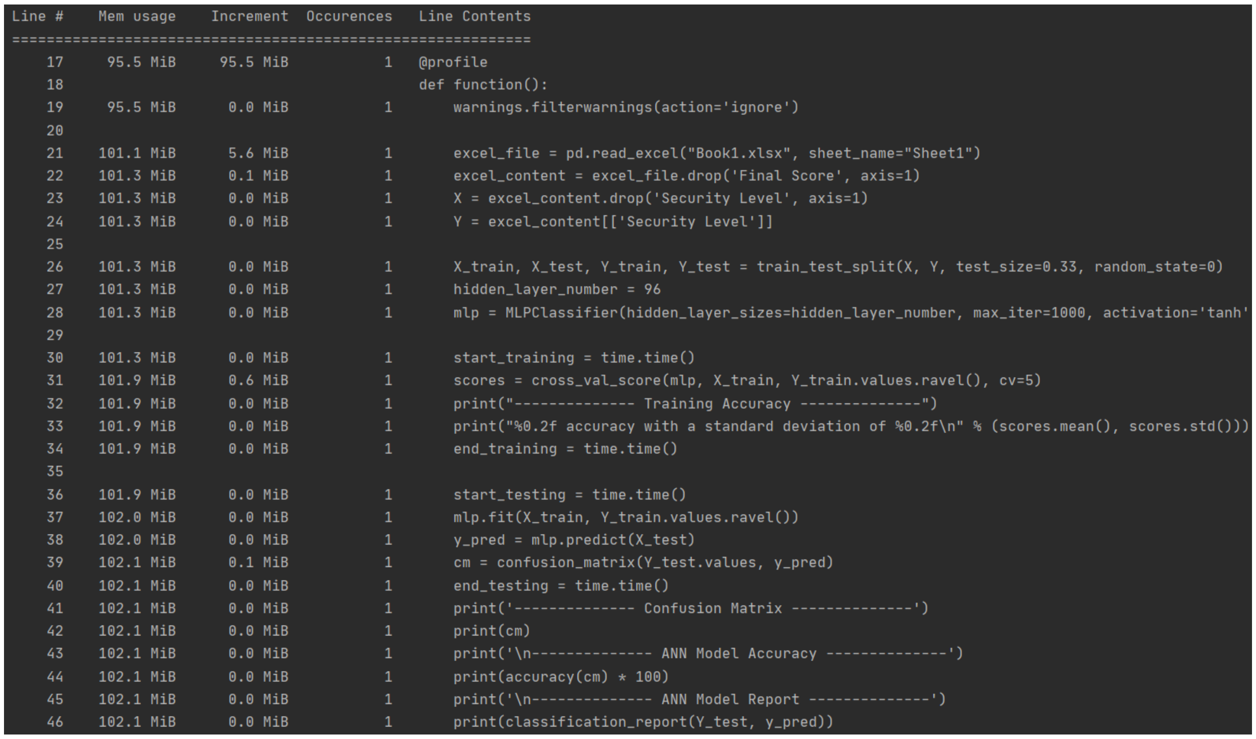


```
-------------- Training Accuracy --------------
0.92 accuracy with a standard deviation of 0.05

-------------- Confusion Matrix --------------
[[ 1  1  0  0]
 [ 0 11  0  0]
 [ 0  0 15  0]
 [ 0  0  0  4]]

-------------- ANN Model Accuracy --------------
96.875

-------------- ANN Model Report --------------
              precision    recall  f1-score   support

           1       1.00      0.50      0.67         2
           2       0.92      1.00      0.96        11
           3       1.00      1.00      1.00        15
           4       1.00      1.00      1.00         4

    accuracy                           0.97        32
   macro avg       0.98      0.88      0.91        32
weighted avg       0.97      0.97      0.96        32
```





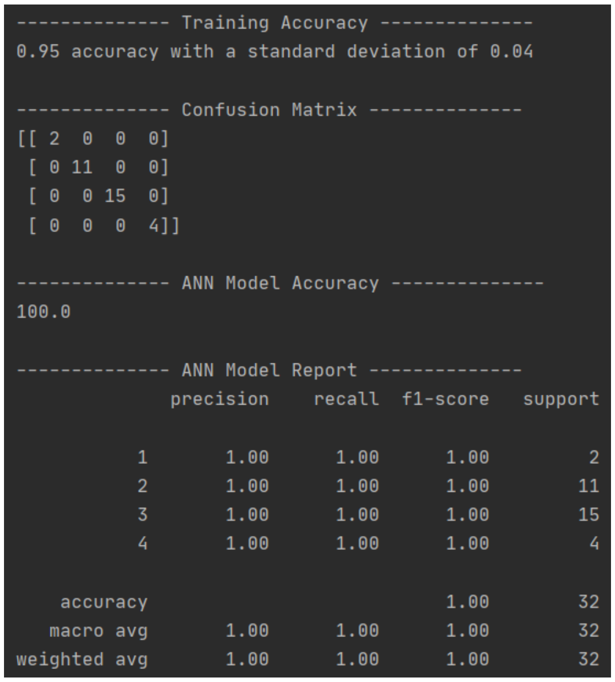
```
Test timing (time units): 1.62s
Test timing (epoch numbers): 1624192904.81251
Train timing (time units): 7.56s
Train timing (epoch numbers): 1624192903.19105
```



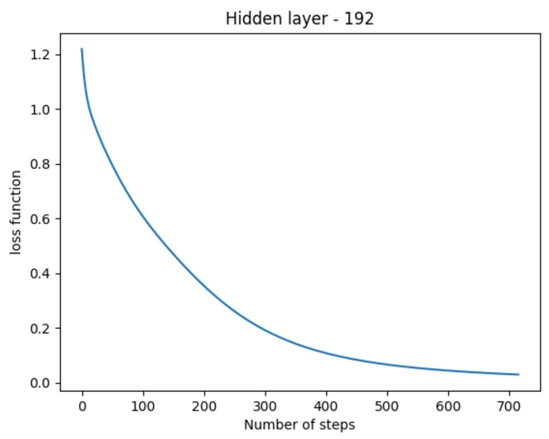
```
Line #    Mem usage    Increment  Occurences   Line Contents
============================================================
    17     95.5 MiB     95.5 MiB           1   @profile
    18                                         def function():
    19     95.5 MiB      0.0 MiB           1       warnings.filterwarnings(action='ignore')
    20
    21    101.1 MiB      5.6 MiB           1       excel_file = pd.read_excel("Book1.xlsx", sheet_name="Sheet1")
    22    101.3 MiB      0.1 MiB           1       excel_content = excel_file.drop('Final Score', axis=1)
    23    101.3 MiB      0.0 MiB           1       X = excel_content.drop('Security Level', axis=1)
    24    101.3 MiB      0.0 MiB           1       Y = excel_content[['Security Level']]
    25
    26    101.3 MiB      0.0 MiB           1       X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.33, random_state=0)
    27    101.3 MiB      0.0 MiB           1       hidden_layer_number = 96
    28    101.3 MiB      0.0 MiB           1       mlp = MLPClassifier(hidden_layer_sizes=hidden_layer_number, max_iter=1000, activation='tanh'
    29
    30    101.3 MiB      0.0 MiB           1       start_training = time.time()
    31    101.9 MiB      0.6 MiB           1       scores = cross_val_score(mlp, X_train, Y_train.values.ravel(), cv=5)
    32    101.9 MiB      0.0 MiB           1       print("------------- Training Accuracy -------------")
    33    101.9 MiB      0.0 MiB           1       print("%0.2f accuracy with a standard deviation of %0.2f\n" % (scores.mean(), scores.std()))
    34    101.9 MiB      0.0 MiB           1       end_training = time.time()
    35
    36    101.9 MiB      0.0 MiB           1       start_testing = time.time()
    37    102.0 MiB      0.0 MiB           1       mlp.fit(X_train, Y_train.values.ravel())
    38    102.0 MiB      0.0 MiB           1       y_pred = mlp.predict(X_test)
    39    102.1 MiB      0.1 MiB           1       cm = confusion_matrix(Y_test.values, y_pred)
    40    102.1 MiB      0.0 MiB           1       end_testing = time.time()
    41    102.1 MiB      0.0 MiB           1       print('------------- Confusion Matrix -------------')
    42    102.1 MiB      0.0 MiB           1       print(cm)
    43    102.1 MiB      0.0 MiB           1       print('\n------------- ANN Model Accuracy -------------')
    44    102.1 MiB      0.0 MiB           1       print(accuracy(cm) * 100)
    45    102.1 MiB      0.0 MiB           1       print('\n------------- ANN Model Report -------------')
    46    102.1 MiB      0.0 MiB           1       print(classification_report(Y_test, y_pred))
```

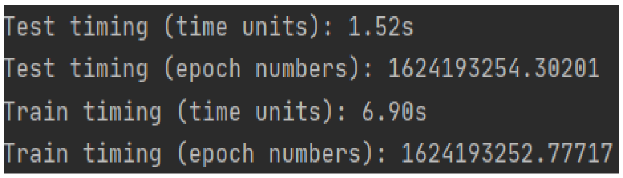
As it can be seen that on increasing the number of neurons to 96, the accuracy has increased again to about 97%. This is really good accuracy for a machine learning model with limited data, but the project will try to obtain 100% as well. The time in epoch numbers has increased again along with the memory consumption that has jumped to about 102.1MiB.

The number of neurons were doubled to **192 neurons**:

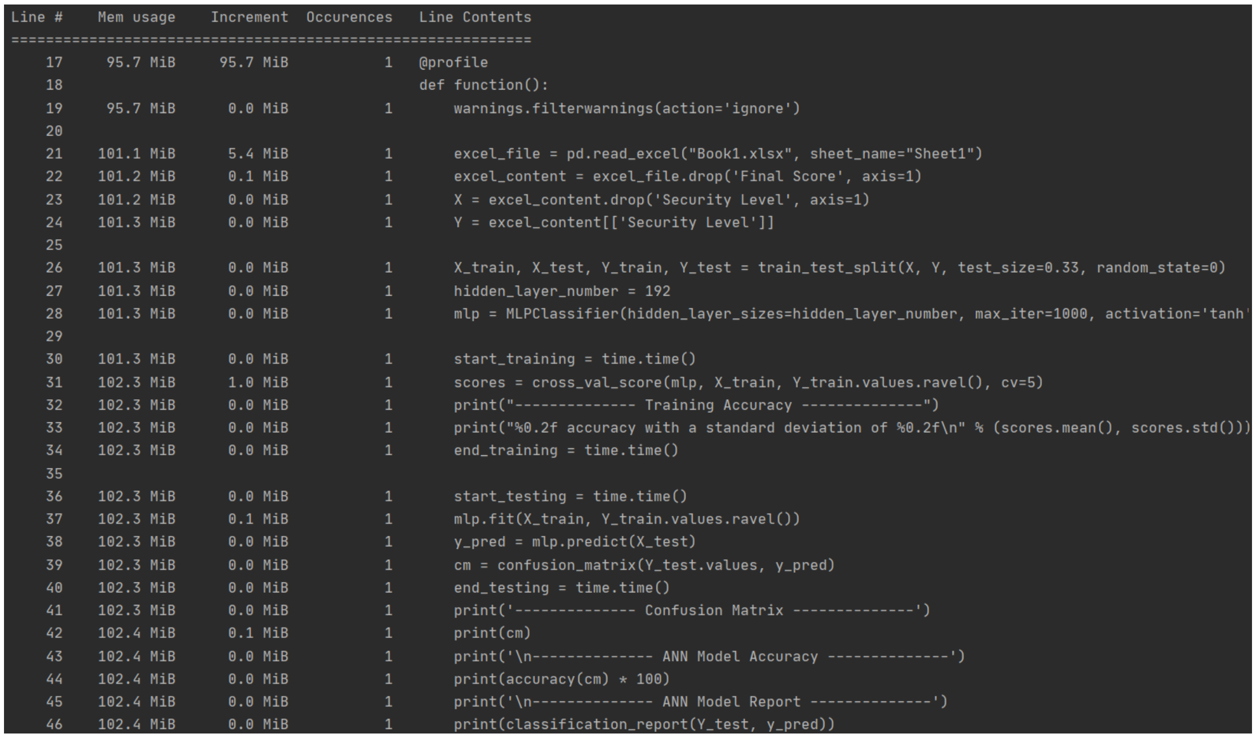


```
------------- Training Accuracy -------------
0.95 accuracy with a standard deviation of 0.04

------------- Confusion Matrix -------------
[[ 2  0  0  0]
 [ 0 11  0  0]
 [ 0  0 15  0]
 [ 0  0  0  4]]

------------- ANN Model Accuracy -------------
100.0

------------- ANN Model Report -------------
              precision    recall  f1-score   support

           1       1.00      1.00      1.00         2
           2       1.00      1.00      1.00        11
           3       1.00      1.00      1.00        15
           4       1.00      1.00      1.00         4

    accuracy                           1.00        32
   macro avg       1.00      1.00      1.00        32
weighted avg       1.00      1.00      1.00        32
```





```
Test timing (time units): 1.52s
Test timing (epoch numbers): 1624193254.30201
Train timing (time units): 6.90s
Train timing (epoch numbers): 1624193252.77717
```



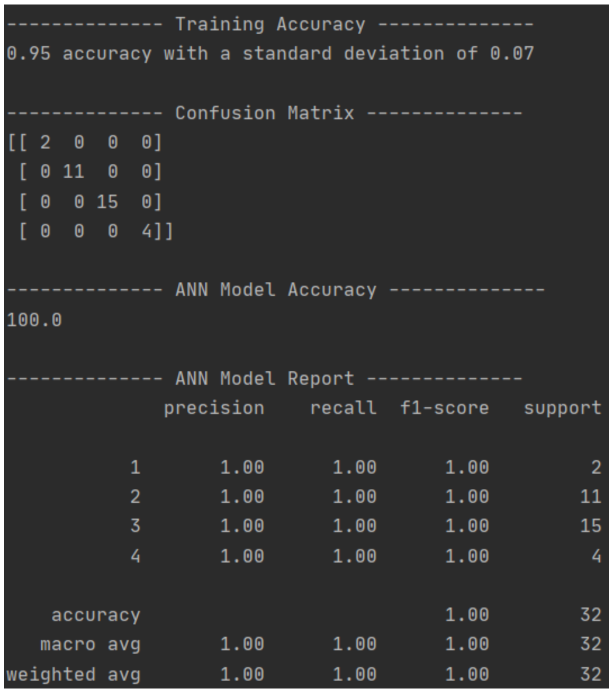
```
Line #    Mem usage    Increment  Occurences   Line Contents
============================================================
    17     95.7 MiB     95.7 MiB           1   @profile
    18                                         def function():
    19     95.7 MiB      0.0 MiB           1       warnings.filterwarnings(action='ignore')
    20
    21    101.1 MiB      5.4 MiB           1       excel_file = pd.read_excel("Book1.xlsx", sheet_name="Sheet1")
    22    101.2 MiB      0.1 MiB           1       excel_content = excel_file.drop('Final Score', axis=1)
    23    101.2 MiB      0.0 MiB           1       X = excel_content.drop('Security Level', axis=1)
    24    101.3 MiB      0.0 MiB           1       Y = excel_content[['Security Level']]
    25
    26    101.3 MiB      0.0 MiB           1       X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.33, random_state=0)
    27    101.3 MiB      0.0 MiB           1       hidden_layer_number = 192
    28    101.3 MiB      0.0 MiB           1       mlp = MLPClassifier(hidden_layer_sizes=hidden_layer_number, max_iter=1000, activation='tanh'
    29
    30    101.3 MiB      0.0 MiB           1       start_training = time.time()
    31    102.3 MiB      1.0 MiB           1       scores = cross_val_score(mlp, X_train, Y_train.values.ravel(), cv=5)
    32    102.3 MiB      0.0 MiB           1       print("-------------- Training Accuracy --------------")
    33    102.3 MiB      0.0 MiB           1       print("%0.2f accuracy with a standard deviation of %0.2f\n" % (scores.mean(), scores.std()))
    34    102.3 MiB      0.0 MiB           1       end_training = time.time()
    35
    36    102.3 MiB      0.0 MiB           1       start_testing = time.time()
    37    102.3 MiB      0.1 MiB           1       mlp.fit(X_train, Y_train.values.ravel())
    38    102.3 MiB      0.0 MiB           1       y_pred = mlp.predict(X_test)
    39    102.3 MiB      0.0 MiB           1       cm = confusion_matrix(Y_test.values, y_pred)
    40    102.3 MiB      0.0 MiB           1       end_testing = time.time()
    41    102.3 MiB      0.0 MiB           1       print('-------------- Confusion Matrix --------------')
    42    102.4 MiB      0.1 MiB           1       print(cm)
    43    102.4 MiB      0.0 MiB           1       print('\n-------------- ANN Model Accuracy --------------')
    44    102.4 MiB      0.0 MiB           1       print(accuracy(cm) * 100)
    45    102.4 MiB      0.0 MiB           1       print('\n-------------- ANN Model Report --------------')
    46    102.4 MiB      0.0 MiB           1       print(classification_report(Y_test, y_pred))
```

At this point we can see that the accuracy at 192 neurons for the ANN model has reached 100%. It can also be verified by looking at the confusion matrix that all the values predicted were correct. The time consumed was again more in epoch numbers and the memory increased as well to about 102.3MiB.

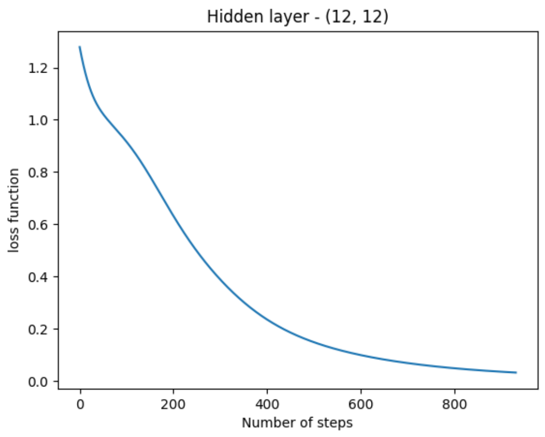
Experiments

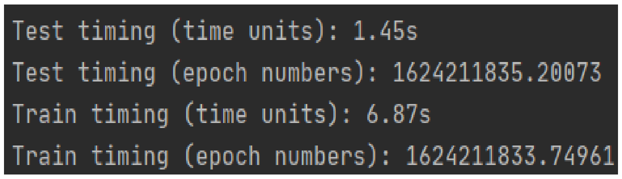
A few experiments were performed to make the model provide better accuracy for lesser memory or time consumption and the following was obtained:

The number of hidden layers was increased from 1 to 2 and then each layer was provided with **12 neurons**:

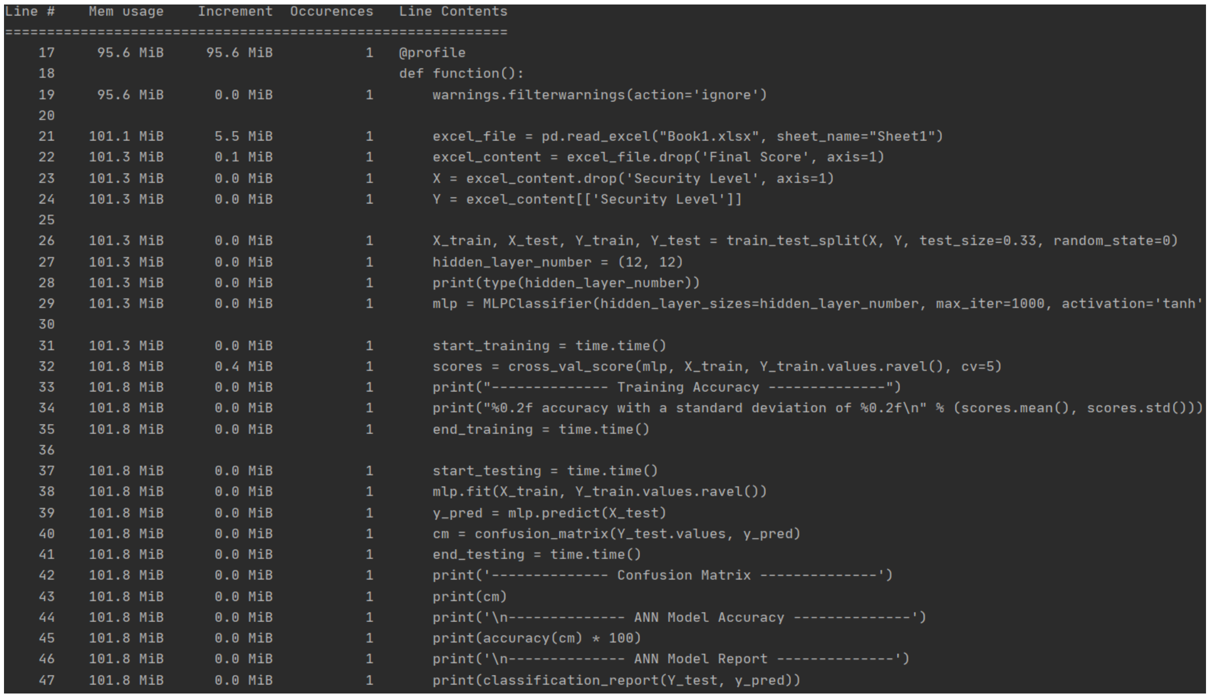


```
-------------- Training Accuracy --------------
0.95 accuracy with a standard deviation of 0.07

-------------- Confusion Matrix --------------
[[ 2  0  0  0]
 [ 0 11  0  0]
 [ 0  0 15  0]
 [ 0  0  0  4]]

-------------- ANN Model Accuracy --------------
100.0

-------------- ANN Model Report --------------
              precision    recall  f1-score   support

           1       1.00      1.00      1.00         2
           2       1.00      1.00      1.00        11
           3       1.00      1.00      1.00        15
           4       1.00      1.00      1.00         4

    accuracy                           1.00        32
   macro avg       1.00      1.00      1.00        32
weighted avg       1.00      1.00      1.00        32
```





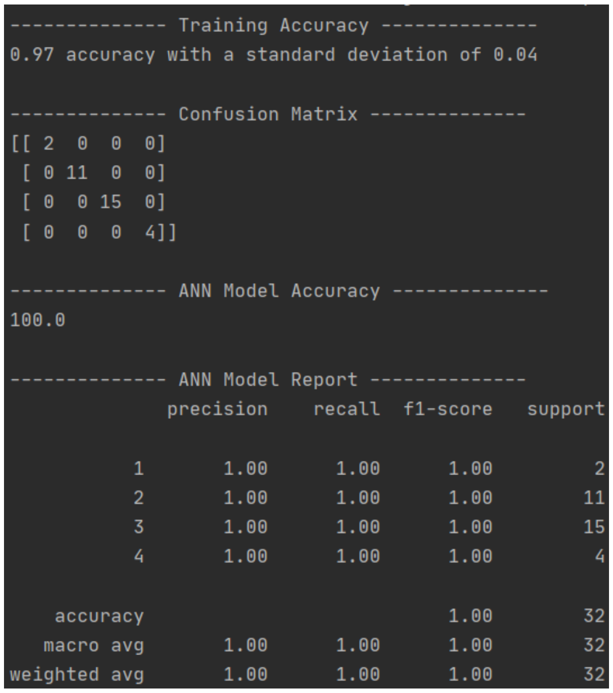
```
Test timing (time units): 1.45s
Test timing (epoch numbers): 1624211835.20073
Train timing (time units): 6.87s
Train timing (epoch numbers): 1624211833.74961
```



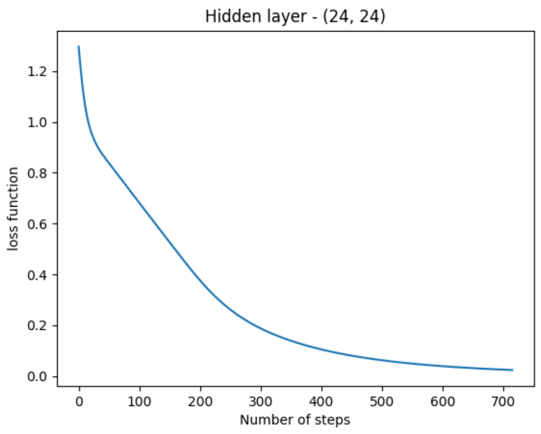
```
Line #    Mem usage    Increment  Occurences   Line Contents
============================================================
    17     95.6 MiB     95.6 MiB           1   @profile
    18                                         def function():
    19     95.6 MiB      0.0 MiB           1       warnings.filterwarnings(action='ignore')
    20
    21    101.1 MiB      5.5 MiB           1       excel_file = pd.read_excel("Book1.xlsx", sheet_name="Sheet1")
    22    101.3 MiB      0.1 MiB           1       excel_content = excel_file.drop('Final Score', axis=1)
    23    101.3 MiB      0.0 MiB           1       X = excel_content.drop('Security Level', axis=1)
    24    101.3 MiB      0.0 MiB           1       Y = excel_content[['Security Level']]
    25
    26    101.3 MiB      0.0 MiB           1       X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.33, random_state=0)
    27    101.3 MiB      0.0 MiB           1       hidden_layer_number = (12, 12)
    28    101.3 MiB      0.0 MiB           1       print(type(hidden_layer_number))
    29    101.3 MiB      0.0 MiB           1       mlp = MLPClassifier(hidden_layer_sizes=hidden_layer_number, max_iter=1000, activation='tanh'
    30
    31    101.3 MiB      0.0 MiB           1       start_training = time.time()
    32    101.8 MiB      0.4 MiB           1       scores = cross_val_score(mlp, X_train, Y_train.values.ravel(), cv=5)
    33    101.8 MiB      0.0 MiB           1       print("-------------- Training Accuracy --------------")
    34    101.8 MiB      0.0 MiB           1       print("%0.2f accuracy with a standard deviation of %0.2f\n" % (scores.mean(), scores.std()))
    35    101.8 MiB      0.0 MiB           1       end_training = time.time()
    36
    37    101.8 MiB      0.0 MiB           1       start_testing = time.time()
    38    101.8 MiB      0.0 MiB           1       mlp.fit(X_train, Y_train.values.ravel())
    39    101.8 MiB      0.0 MiB           1       y_pred = mlp.predict(X_test)
    40    101.8 MiB      0.0 MiB           1       cm = confusion_matrix(Y_test.values, y_pred)
    41    101.8 MiB      0.0 MiB           1       end_testing = time.time()
    42    101.8 MiB      0.0 MiB           1       print('-------------- Confusion Matrix --------------')
    43    101.8 MiB      0.0 MiB           1       print(cm)
    44    101.8 MiB      0.0 MiB           1       print('\n-------------- ANN Model Accuracy --------------')
    45    101.8 MiB      0.0 MiB           1       print(accuracy(cm) * 100)
    46    101.8 MiB      0.0 MiB           1       print('\n-------------- ANN Model Report --------------')
    47    101.8 MiB      0.0 MiB           1       print(classification_report(Y_test, y_pred))
```

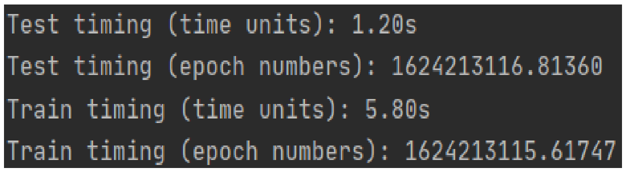
As it can be seen, the 2 hidden layers with 12 neurons each provide a similar output as 192 neurons in a single hidden layer, but this new method also provides advantages in terms of memory consumption. The 192 neurons single hidden layer consumed about 102.3MiB and the 2 hidden layers with 12 neurons each consumed about 101.8MiB. It can be seen that using 2 hidden layers with 12 neurons each is much better than using a single hidden layer with 192 neurons.

The best result found while experimenting was using **24 neurons for 2 hidden layers** each:

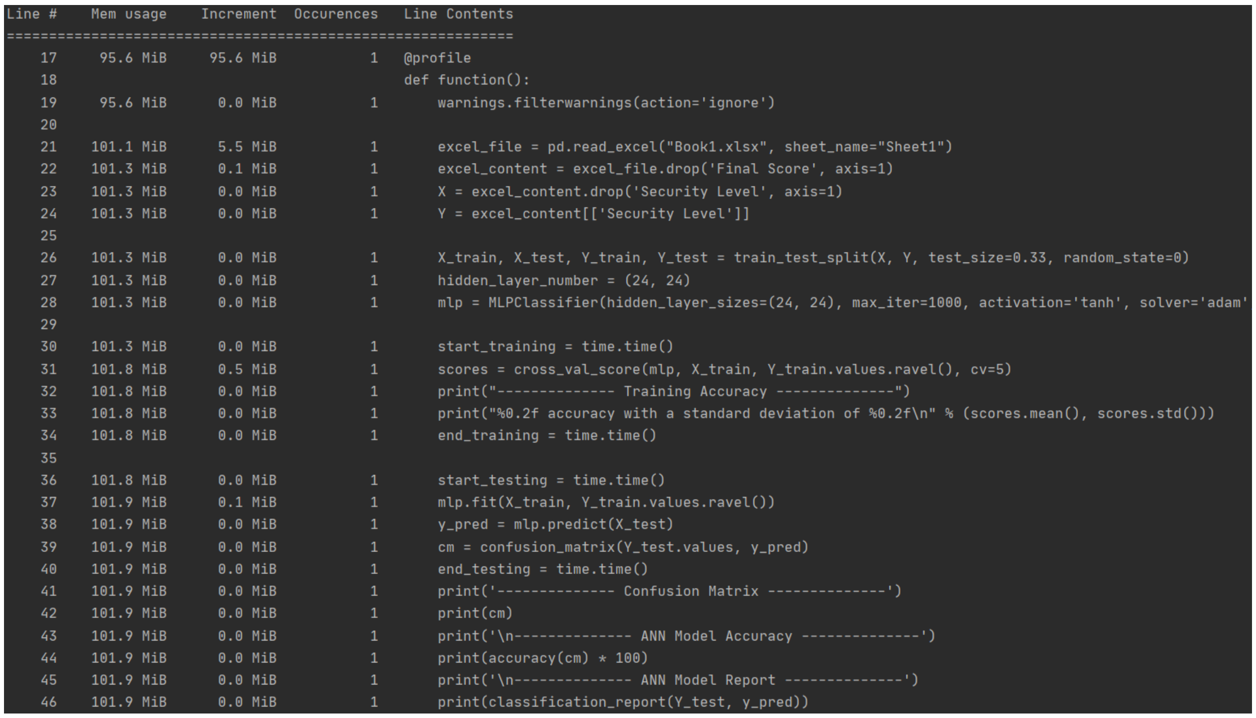


```
-------------- Training Accuracy --------------
0.97 accuracy with a standard deviation of 0.04

-------------- Confusion Matrix --------------
[[ 2  0  0  0]
 [ 0 11  0  0]
 [ 0  0 15  0]
 [ 0  0  0  4]]

-------------- ANN Model Accuracy --------------
100.0

-------------- ANN Model Report --------------
              precision    recall  f1-score   support


           1       1.00      1.00      1.00         2
           2       1.00      1.00      1.00        11
           3       1.00      1.00      1.00        15
           4       1.00      1.00      1.00         4

    accuracy                           1.00        32
   macro avg       1.00      1.00      1.00        32
weighted avg       1.00      1.00      1.00        32
```





```
Test timing (time units): 1.20s
Test timing (epoch numbers): 1624213116.81360
Train timing (time units): 5.80s
Train timing (epoch numbers): 1624213115.61747
```



```
Line #    Mem usage    Increment  Occurences   Line Contents
============================================================
    17     95.6 MiB     95.6 MiB           1   @profile
    18                                         def function():
    19     95.6 MiB      0.0 MiB           1       warnings.filterwarnings(action='ignore')
    20
    21    101.1 MiB      5.5 MiB           1       excel_file = pd.read_excel("Book1.xlsx", sheet_name="Sheet1")
    22    101.3 MiB      0.1 MiB           1       excel_content = excel_file.drop('Final Score', axis=1)
    23    101.3 MiB      0.0 MiB           1       X = excel_content.drop('Security Level', axis=1)
    24    101.3 MiB      0.0 MiB           1       Y = excel_content[['Security Level']]
    25
    26    101.3 MiB      0.0 MiB           1       X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.33, random_state=0)
    27    101.3 MiB      0.0 MiB           1       hidden_layer_number = (24, 24)
    28    101.3 MiB      0.0 MiB           1       mlp = MLPClassifier(hidden_layer_sizes=(24, 24), max_iter=1000, activation='tanh', solver='adam'
    29
    30    101.3 MiB      0.0 MiB           1       start_training = time.time()
    31    101.8 MiB      0.5 MiB           1       scores = cross_val_score(mlp, X_train, Y_train.values.ravel(), cv=5)
    32    101.8 MiB      0.0 MiB           1       print("-------------- Training Accuracy --------------")
    33    101.8 MiB      0.0 MiB           1       print("%0.2f accuracy with a standard deviation of %0.2f\n" % (scores.mean(), scores.std()))
    34    101.8 MiB      0.0 MiB           1       end_training = time.time()
    35
    36    101.8 MiB      0.0 MiB           1       start_testing = time.time()
    37    101.9 MiB      0.1 MiB           1       mlp.fit(X_train, Y_train.values.ravel())
    38    101.9 MiB      0.0 MiB           1       y_pred = mlp.predict(X_test)
    39    101.9 MiB      0.0 MiB           1       cm = confusion_matrix(Y_test.values, y_pred)
    40    101.9 MiB      0.0 MiB           1       end_testing = time.time()
    41    101.9 MiB      0.0 MiB           1       print('-------------- Confusion Matrix --------------')
    42    101.9 MiB      0.0 MiB           1       print(cm)
    43    101.9 MiB      0.0 MiB           1       print('\n-------------- ANN Model Accuracy --------------')
    44    101.9 MiB      0.0 MiB           1       print(accuracy(cm) * 100)
    45    101.9 MiB      0.0 MiB           1       print('\n-------------- ANN Model Report --------------')
    46    101.9 MiB      0.0 MiB           1       print(classification_report(Y_test, y_pred))
```

As it can be seen, this is the best scenario for the model. The accuracy obtained for the ANN model is 100%, this scenario provides us with the best training accuracy (cross_val_score) i.e., 97% and it consumes less memory than 192 neurons in a single hidden layer. The memory consumed for this scenario is 101.9MiB. The time consumed is still more, but it provides us with best output with less memory consumed.

Observations on Graphs

If all the graphs provided are observed together, it can be seen that as the accuracy increases the loss reduces over lesser steps. For the best-case scenarios like a single hidden layer with 192 neutrons model or double hidden layer with 24 neurons each, they both have a good loss curve i.e., their loss reduces within lesser steps.