BITS F464 MACHINE LEARNING ASSIGNMENT 2

GROUP MEMBERS :
M. ASHRITHA 2019B3A70472H
SUSHMA REDDY KOLLI 2019B5A70671H

Introduction:
In this report, we will discuss the performance of the following models -Naïve Bayes classifier to predict whether income is more than $50k per year, ANN to classify handwritten digits on a given dataset.
We will compare the performance of these models and identify

the best-performing model based on the average performance metrics of multiple models.

The criteria used to grade this assignment are accuracy, recall, and precision, which are all basic measures of performance.

TP: true positives (number of correctly classified positive instances)
TN: true negatives (number of correctly classified negative instances)
FP: false positives (number of negative instances classified as positive)
FN: false negatives (number of positive instances classified as negative)

Accuracy: (TP + TN) / (TP + TN + FP + FN)
Accuracy measures the proportion of correctly classified instances (both positive and negative) among all instances in the dataset.

Precision: TP / (TP + FP)

Precision measures the proportion of correctly classified positive instances among all instances classified as positive.

Recall: TP / (TP + FN)
Recall measures the proportion of correctly classified positive instances among all actual positive instances.
Confusion Matrix:
A confusion matrix is a table that is used to evaluate the performance of a classification model. It shows the number of true positives (TP), true negatives (TN), false positives (FP), and false negatives (FN) predicted by the model for each class.

To generate a confusion matrix, the model is tested on a dataset where the true class labels are known. For each instance in the test set, the model's predicted class label is compared to the true class label. The counts of true positives, true negatives, false positives, and false negatives are then recorded in the confusion matrix.

The confusion matrix can be used to calculate various performance metrics for the classification model, such as accuracy, precision, recall, and F1-score. It is also useful for identifying which classes the model is performing well on and which classes it is struggling with.

PART-A

Naive Bayes is a type of probabilistic machine learning algorithm that is used for classification tasks. It is based on Bayes' theorem, which is a fundamental theorem in probability theory. Naive Bayes assumes that the features used to predict a class are conditionally independent, meaning that the presence or absence of one feature does not affect the presence or absence of another feature.

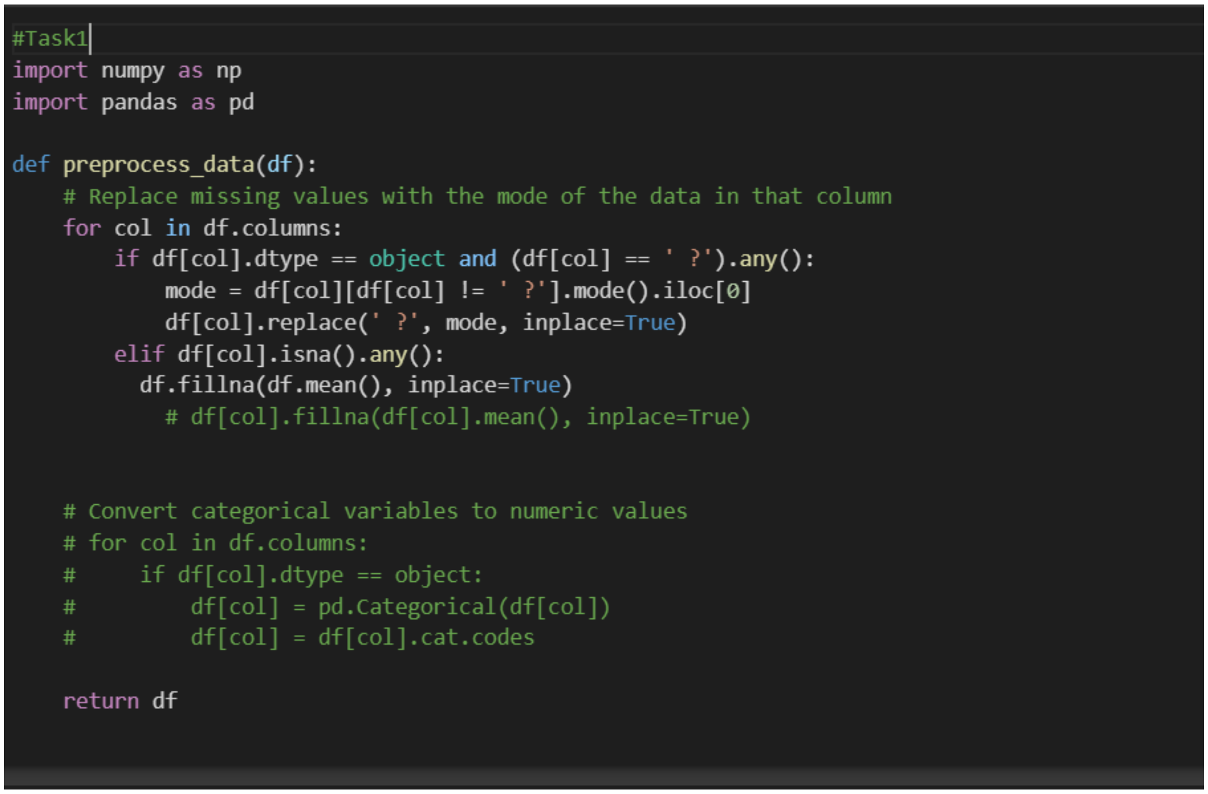
There are different types of Naive Bayes algorithms, including the Gaussian Naive Bayes, Multinomial Naive Bayes, and Bernoulli Naive Bayes. Gaussian Naive Bayes assumes that the continuous input variables are normally distributed, while Multinomial

Naive Bayes assumes that the input variables are discrete counts. Bernoulli Naive Bayes assumes that the input variables are binary.

Naive Bayes is commonly used in text classification tasks, such as spam filtering, sentiment analysis, and topic classification. It is also used in other applications such as recommendation systems, fraud detection, and medical diagnosis.

TASK 1:

Preprocessing Data and Replacing the Missing Values ('?')' with Mode and Mean of the Dataset



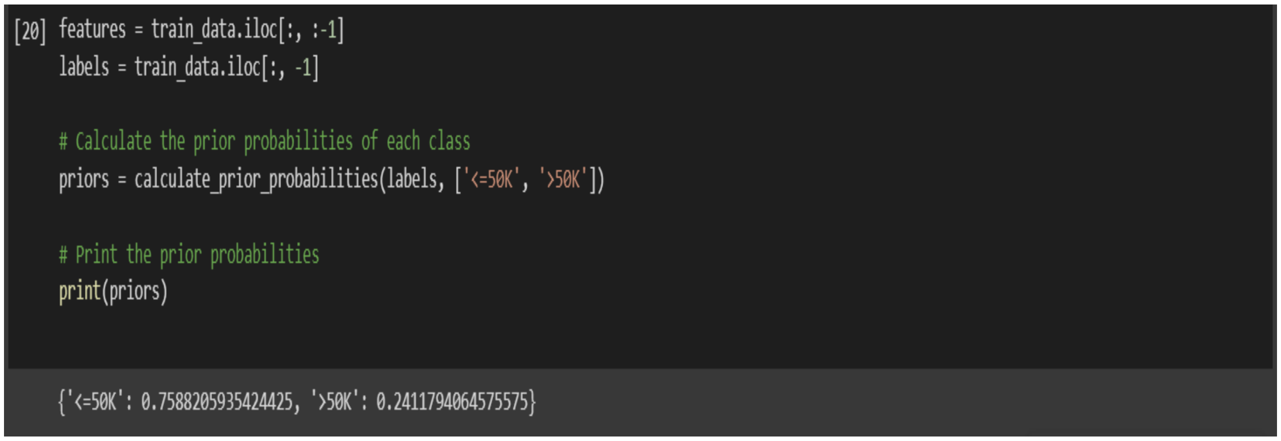
```
#Task1
import numpy as np
import pandas as pd

def preprocess_data(df):
    # Replace missing values with the mode of the data in that column
    for col in df.columns:
        if df[col].dtype == object and (df[col] == ' ?').any():
            mode = df[col][df[col] != ' ?'].mode().iloc[0]
            df[col].replace(' ?', mode, inplace=True)
        elif df[col].isna().any():
          df.fillna(df.mean(), inplace=True)
            # df[col].fillna(df[col].mean(), inplace=True)


    # Convert categorical variables to numeric values
    # for col in df.columns:
    #     if df[col].dtype == object:
    #         df[col] = pd.Categorical(df[col])
    #         df[col] = df[col].cat.codes

    return df
```

Naive-Bayes Implementation

TASK 2:

Prior Probability:

1.

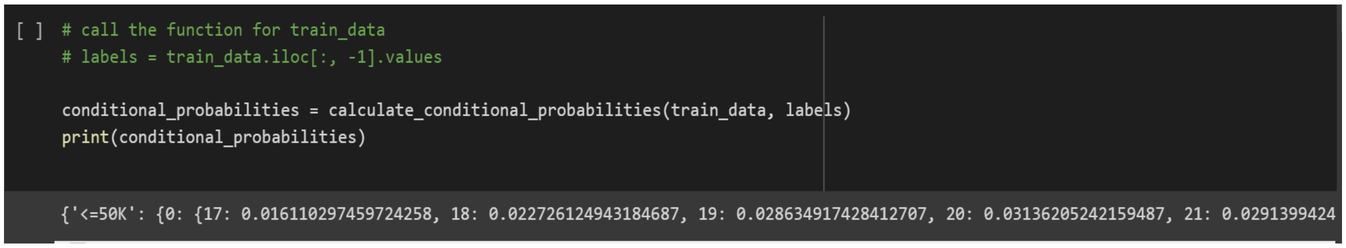


```
[20] features = train_data.iloc[:, :-1]
     labels = train_data.iloc[:, -1]

     # Calculate the prior probabilities of each class
     priors = calculate_prior_probabilities(labels, ['<=50K', '>50K'])

     # Print the prior probabilities
     print(priors)
```

```
{'<=50K': 0.7588205935424425, '>50K': 0.2411794064575575}
```

Conditional Probability:

2.

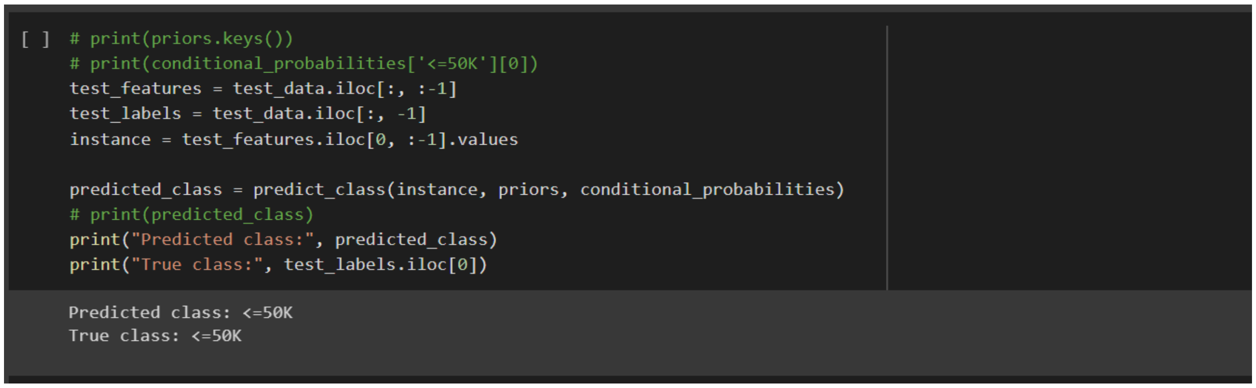


```
[ ] # call the function for train_data
    # labels = train_data.iloc[:, -1].values

    conditional_probabilities = calculate_conditional_probabilities(train_data, labels)
    print(conditional_probabilities)
```

```
{'<=50K': {0: {17: 0.016110297459724258, 18: 0.022726124943184687, 19: 0.028634917428412707, 20: 0.03136205242159487, 21: 0.0291399424
```

Predict Class of given Instance:

3.



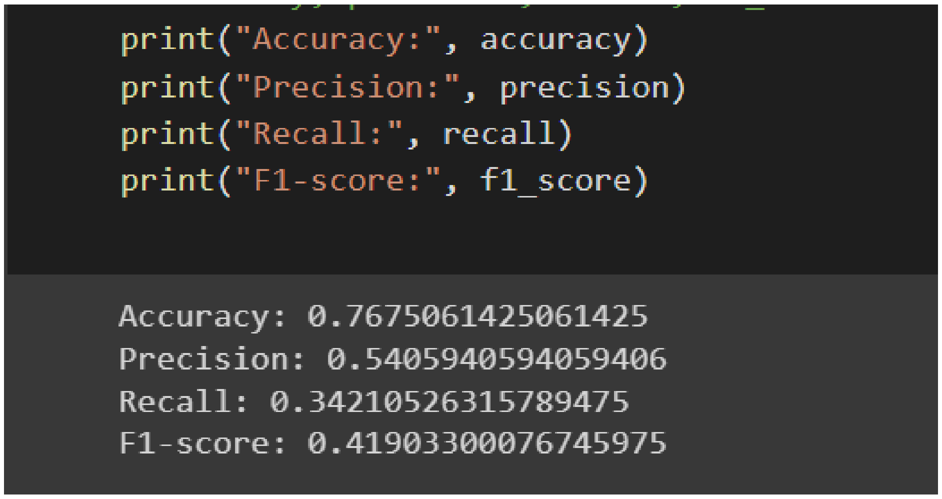
```
[ ] # print(priors.keys())
    # print(conditional_probabilities['<=50K'][0])
    test_features = test_data.iloc[:, :-1]
    test_labels = test_data.iloc[:, -1]
    instance = test_features.iloc[0, :-1].values

    predicted_class = predict_class(instance, priors, conditional_probabilities)
    # print(predicted_class)
    print("Predicted class:", predicted_class)
    print("True class:", test_labels.iloc[0])
```

```
Predicted class: <=50K
True class: <=50K
```

TASK 3:

Accuracy, Precision, Recall and F1-Score:

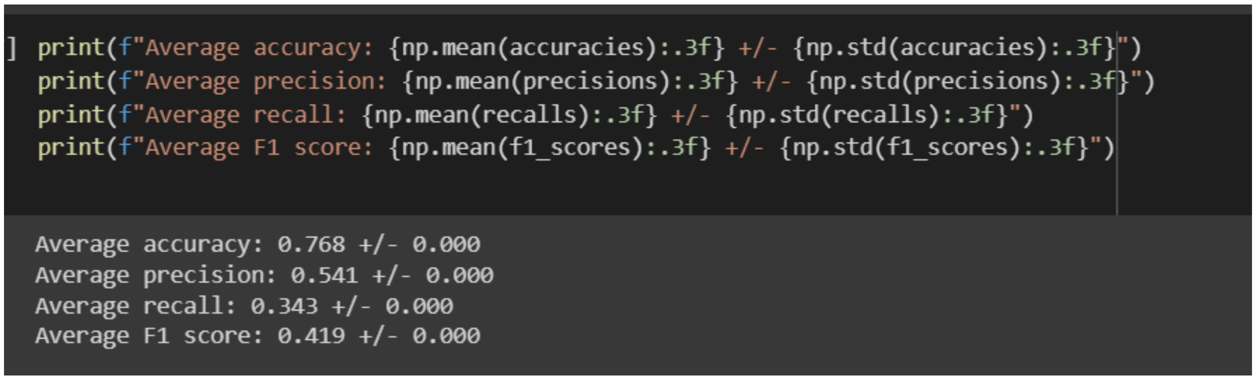
1.


```
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1_score)
```

```
Accuracy: 0.7675061425061425
Precision: 0.5405940594059406
Recall: 0.34210526315789475
F1-score: 0.41903300076745975
```

For 10 different splits the average accuracy, precision and recall:

2.


```
] print(f"Average accuracy: {np.mean(accuracies):.3f} +/- {np.std(accuracies):.3f}")
  print(f"Average precision: {np.mean(precisions):.3f} +/- {np.std(precisions):.3f}")
  print(f"Average recall: {np.mean(recalls):.3f} +/- {np.std(recalls):.3f}")
  print(f"Average F1 score: {np.mean(f1_scores):.3f} +/- {np.std(f1_scores):.3f}")
```

```
Average accuracy: 0.768 +/- 0.000
Average precision: 0.541 +/- 0.000
Average recall: 0.343 +/- 0.000
Average F1 score: 0.419 +/- 0.000
```

Smoothening Techniques:

Smoothing is a technique used to avoid zero probabilities in probability-based models, such as Naive Bayes classifiers, where zero probabilities can cause errors in the model's performance. Smoothing methods help to assign non-zero probabilities to unknown features or classes that have zero frequencies in the training set.

Two common smoothing techniques used in Naive Bayes classifiers are Laplace smoothing and Add-k smoothing.

Laplace Smoothing:
Also known as additive smoothing, Laplace smoothing is a simple and popular technique used to avoid zero probabilities. In Laplace smoothing, we add one to the count of each feature and class in the training set. This is known as the pseudocount or additive factor. By adding one, we ensure that there are no zero probabilities in the model.

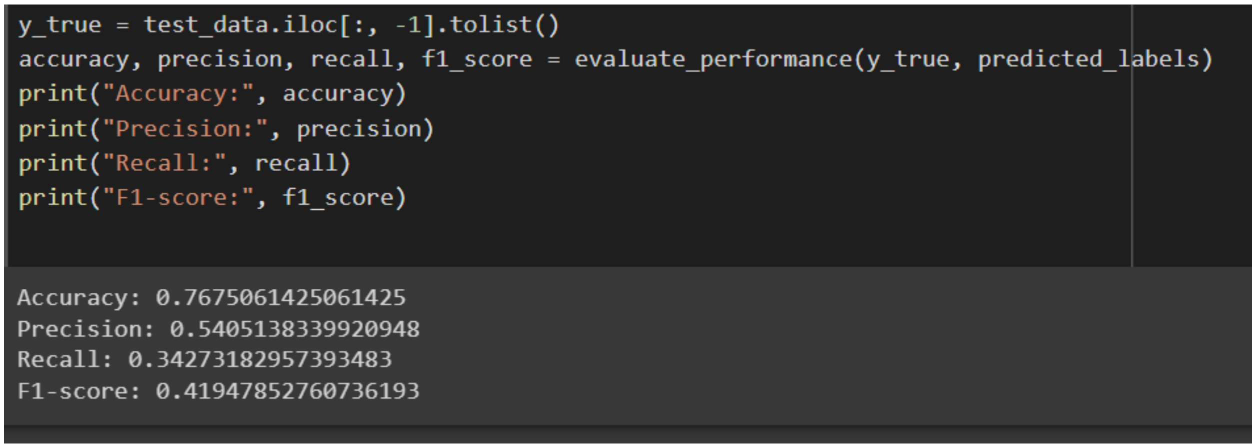
The formula for Laplace smoothing is:

P(x_i | y) = (count(x_i, y) + 1) / (count(y) + |V|)

where count(x_i, y) is the number of occurrences of feature x_i in class y, count(y) is the total number of instances in class y, and |V| is the total number of features in the vocabulary.

Add-k Smoothing:
Add-k smoothing is a generalization of Laplace smoothing, where we add a value k instead of one to the count of each feature and class. This technique is used to adjust the degree of smoothing and is often used when the training set is small.

Laplace smoothning:
4.



```
y_true = test_data.iloc[:, -1].tolist()
accuracy, precision, recall, f1_score = evaluate_performance(y_true, predicted_labels)
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1_score)
```

```
Accuracy: 0.7675061425061425
Precision: 0.5405138339920948
Recall: 0.34273182957393483
F1-score: 0.41947852760736193
```

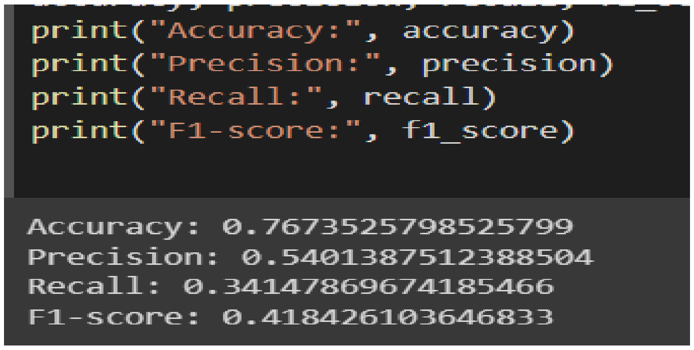
ADD-K smoothing
Add-k smoothing is a generalization of Laplace smoothing, where we add a value k instead of one to the count of each feature and class. This technique is used to adjust the degree of smoothing and is often used when the training set is small.

The formula for Add-k smoothing is:

P(x_i | y) = (count(x_i, y) + k) / (count(y) + k * |V|)

where count(x_i, y) is the number of occurrences of feature x_i in class y, count(y) is the total number of instances in class y, |V| is the total number of features in the vocabulary, and k is the smoothing parameter.

5.



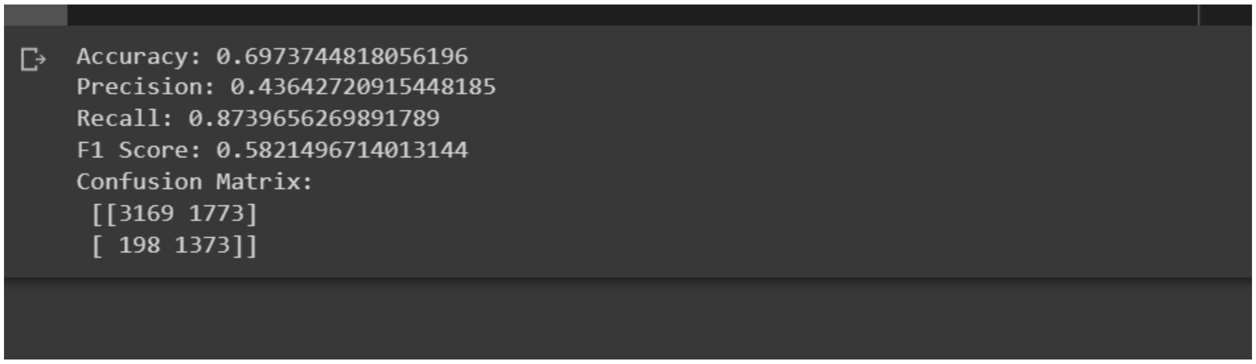
```
print("Accuracy:", accuracy)
print("Precision:", precision)
print("Recall:", recall)
print("F1-score:", f1_score)
```

```
Accuracy: 0.7673525798525799
Precision: 0.5401387512388504
Recall: 0.34147869674185466
F1-score: 0.418426103646833
```

Both Laplace and Add-k smoothing are effective techniques for handling zero probabilities in Naive Bayes classifiers. However, the choice between the two techniques depends on the size of the training set and the degree of smoothing required. If the training set is small, then Add-k smoothing may be preferred over Laplace smoothing.

Logistic Regression:

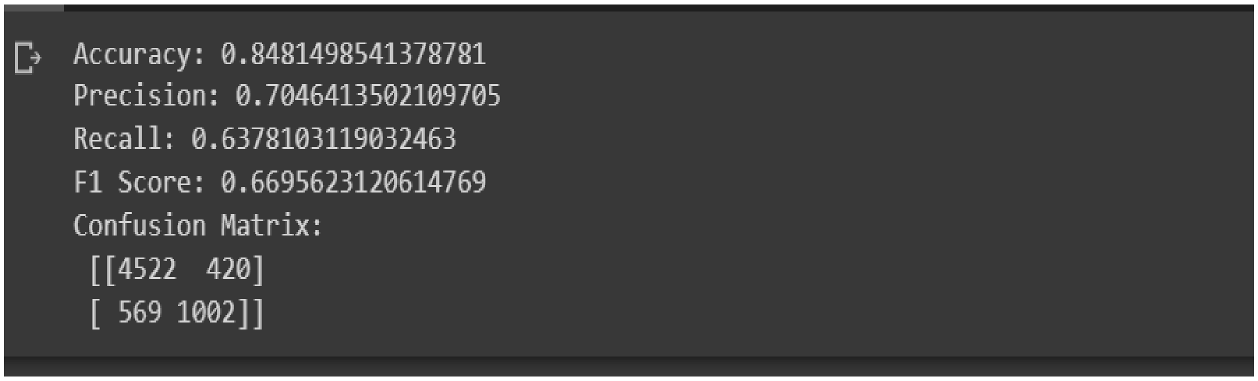
4.



```
Accuracy: 0.6973744818056196
Precision: 0.43642720915448185
Recall: 0.8739656269891789
F1 Score: 0.5821496714013144
Confusion Matrix:
 [[3169 1773]
 [ 198 1373]]
```

KNN :

5.



```
Accuracy: 0.8481498541378781
Precision: 0.7046413502109705
Recall: 0.6378103119032463
F1 Score: 0.6695623120614769
Confusion Matrix:
 [[4522  420]
 [ 569 1002]]
```

KNN is the better model here because higher the F1 Score

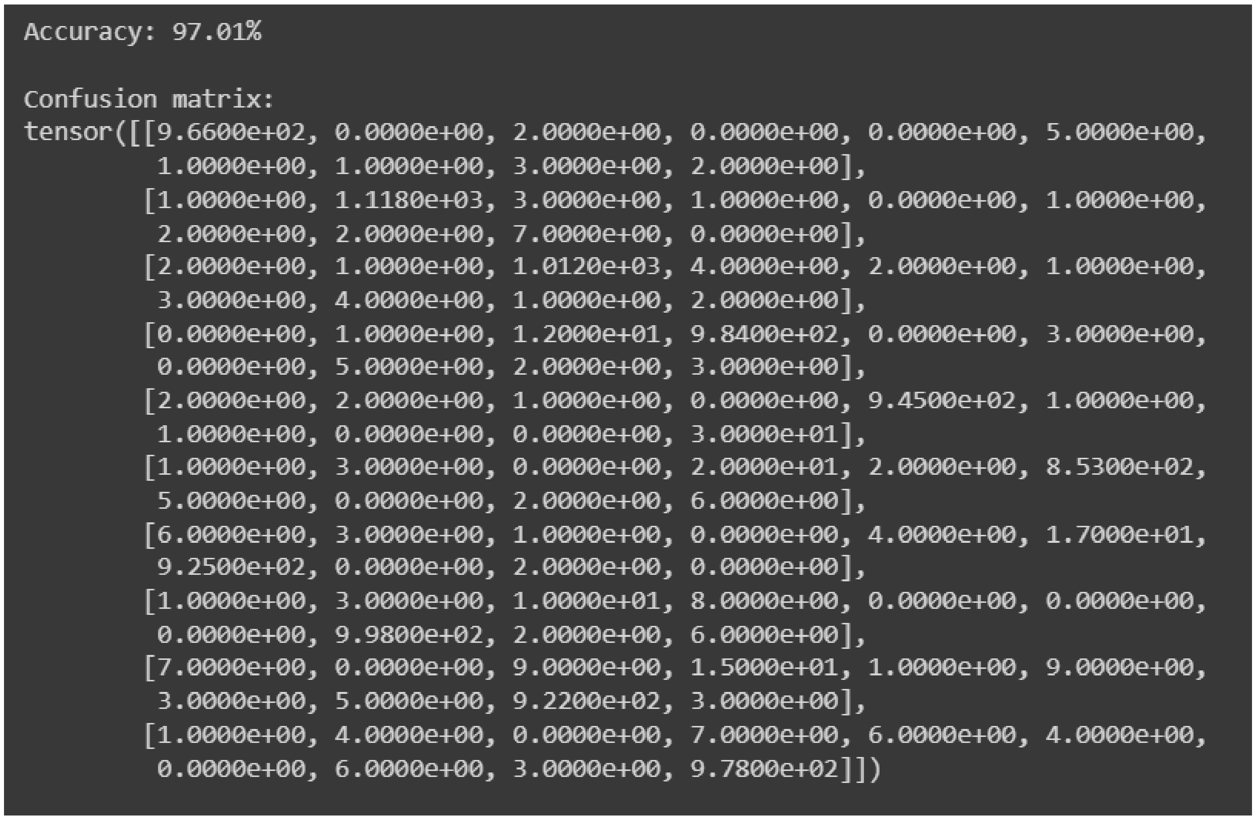
PART -B

Adam optimizer:

- The Adam optimizer is an extension of stochastic gradient descent used for deep learning.
- It adjusts the learning rate for each weight based on its historical gradient values.
- It uses momentum to accelerate the optimization process by smoothing out the noise in the gradient direction.
- The adaptive learning rate and momentum methods make the Adam optimizer faster and more reliable in converging to the global minimum of the loss function.
- The Adam optimizer is robust to hyperparameters and requires less fine-tuning.
- It stores only the first and second moments of the gradient, making it more memory-efficient.
- Adam optimizer is widely used in various deep learning tasks and has a high degree of success.
- It is a reliable and efficient optimization algorithm that is easier to use than other optimization algorithms.
- The Adam optimizer is a preferred choice for many deep learning applications because of its advantages over stochastic gradient descent.
- It helps deep learning models converge faster and more accurately, leading to better results.

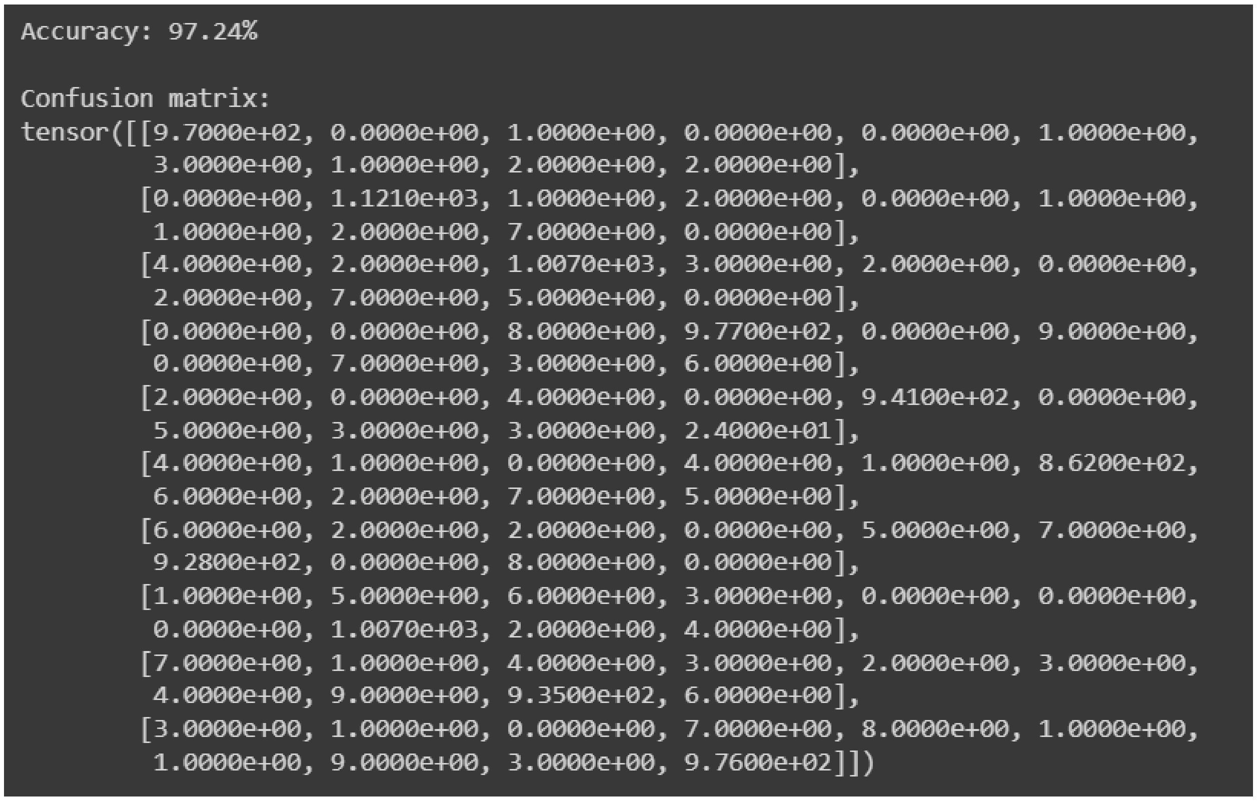
i) 2 hidden layers

100 neurons in hidden layers, relu as activation function



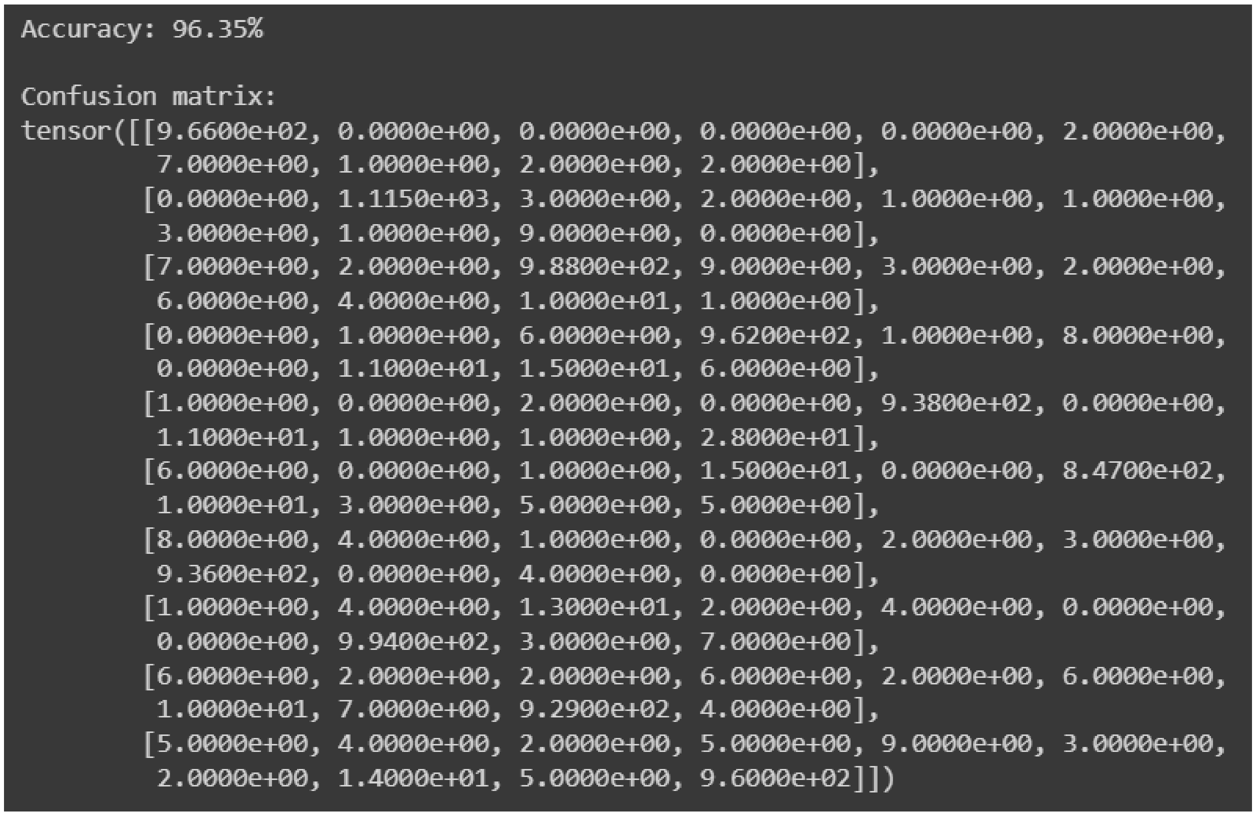
```
Accuracy: 97.01%

Confusion matrix:
tensor([[9.6600e+02, 0.0000e+00, 2.0000e+00, 0.0000e+00, 0.0000e+00, 5.0000e+00,
         1.0000e+00, 1.0000e+00, 3.0000e+00, 2.0000e+00],
        [1.0000e+00, 1.1180e+03, 3.0000e+00, 1.0000e+00, 0.0000e+00, 1.0000e+00,
         2.0000e+00, 2.0000e+00, 7.0000e+00, 0.0000e+00],
        [2.0000e+00, 1.0000e+00, 1.0120e+03, 4.0000e+00, 2.0000e+00, 1.0000e+00,
         3.0000e+00, 4.0000e+00, 1.0000e+00, 2.0000e+00],
        [0.0000e+00, 1.0000e+00, 1.2000e+01, 9.8400e+02, 0.0000e+00, 3.0000e+00,
         0.0000e+00, 5.0000e+00, 2.0000e+00, 3.0000e+00],
        [2.0000e+00, 2.0000e+00, 1.0000e+00, 0.0000e+00, 9.4500e+02, 1.0000e+00,
         1.0000e+00, 0.0000e+00, 0.0000e+00, 3.0000e+01],
        [1.0000e+00, 3.0000e+00, 0.0000e+00, 2.0000e+01, 2.0000e+00, 8.5300e+02,
         5.0000e+00, 0.0000e+00, 2.0000e+00, 6.0000e+00],
        [6.0000e+00, 3.0000e+00, 1.0000e+00, 0.0000e+00, 4.0000e+00, 1.7000e+01,
         9.2500e+02, 0.0000e+00, 2.0000e+00, 0.0000e+00],
        [1.0000e+00, 3.0000e+00, 1.0000e+01, 8.0000e+00, 0.0000e+00, 0.0000e+00,
         0.0000e+00, 9.9800e+02, 2.0000e+00, 6.0000e+00],
        [7.0000e+00, 0.0000e+00, 9.0000e+00, 1.5000e+01, 1.0000e+00, 9.0000e+00,
         3.0000e+00, 5.0000e+00, 9.2200e+02, 3.0000e+00],
        [1.0000e+00, 4.0000e+00, 0.0000e+00, 7.0000e+00, 6.0000e+00, 4.0000e+00,
         0.0000e+00, 6.0000e+00, 3.0000e+00, 9.7800e+02]])
```

100 neurons in hidden layers, sigmoid as activation function



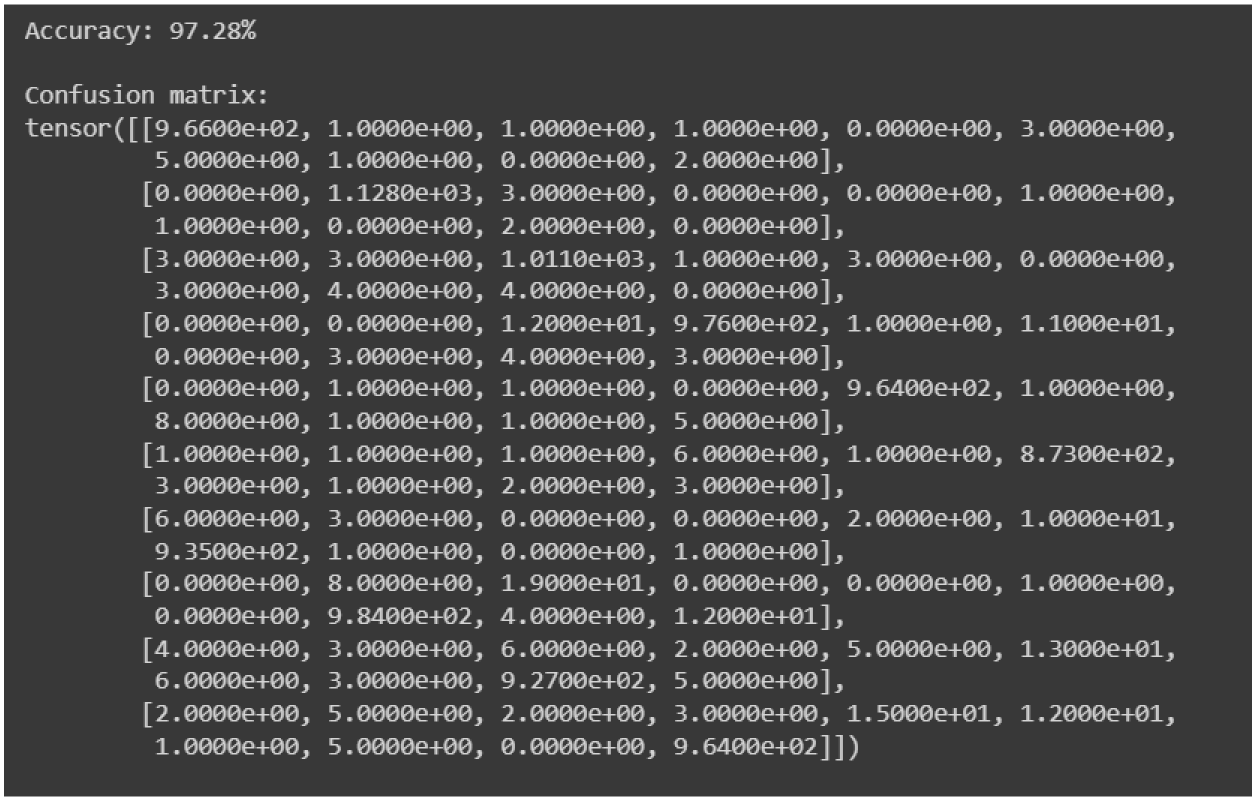
```
Accuracy: 97.24%

Confusion matrix:
tensor([[9.7000e+02, 0.0000e+00, 1.0000e+00, 0.0000e+00, 0.0000e+00, 1.0000e+00,
         3.0000e+00, 1.0000e+00, 2.0000e+00, 2.0000e+00],
        [0.0000e+00, 1.1210e+03, 1.0000e+00, 2.0000e+00, 0.0000e+00, 1.0000e+00,
         1.0000e+00, 2.0000e+00, 7.0000e+00, 0.0000e+00],
        [4.0000e+00, 2.0000e+00, 1.0070e+03, 3.0000e+00, 2.0000e+00, 0.0000e+00,
         2.0000e+00, 7.0000e+00, 5.0000e+00, 0.0000e+00],
        [0.0000e+00, 0.0000e+00, 8.0000e+00, 9.7700e+02, 0.0000e+00, 9.0000e+00,
         0.0000e+00, 7.0000e+00, 3.0000e+00, 6.0000e+00],
        [2.0000e+00, 0.0000e+00, 4.0000e+00, 0.0000e+00, 9.4100e+02, 0.0000e+00,
         5.0000e+00, 3.0000e+00, 3.0000e+00, 2.4000e+01],
        [4.0000e+00, 1.0000e+00, 0.0000e+00, 4.0000e+00, 1.0000e+00, 8.6200e+02,
         6.0000e+00, 2.0000e+00, 7.0000e+00, 5.0000e+00],
        [6.0000e+00, 2.0000e+00, 2.0000e+00, 0.0000e+00, 5.0000e+00, 7.0000e+00,
         9.2800e+02, 0.0000e+00, 8.0000e+00, 0.0000e+00],
        [1.0000e+00, 5.0000e+00, 6.0000e+00, 3.0000e+00, 0.0000e+00, 0.0000e+00,
         0.0000e+00, 1.0070e+03, 2.0000e+00, 4.0000e+00],
        [7.0000e+00, 1.0000e+00, 4.0000e+00, 3.0000e+00, 2.0000e+00, 3.0000e+00,
         4.0000e+00, 9.0000e+00, 9.3500e+02, 6.0000e+00],
        [3.0000e+00, 1.0000e+00, 0.0000e+00, 7.0000e+00, 8.0000e+00, 1.0000e+00,
         1.0000e+00, 9.0000e+00, 3.0000e+00, 9.7600e+02]])
```

100 neurons in hidden layers, tanh as activation function



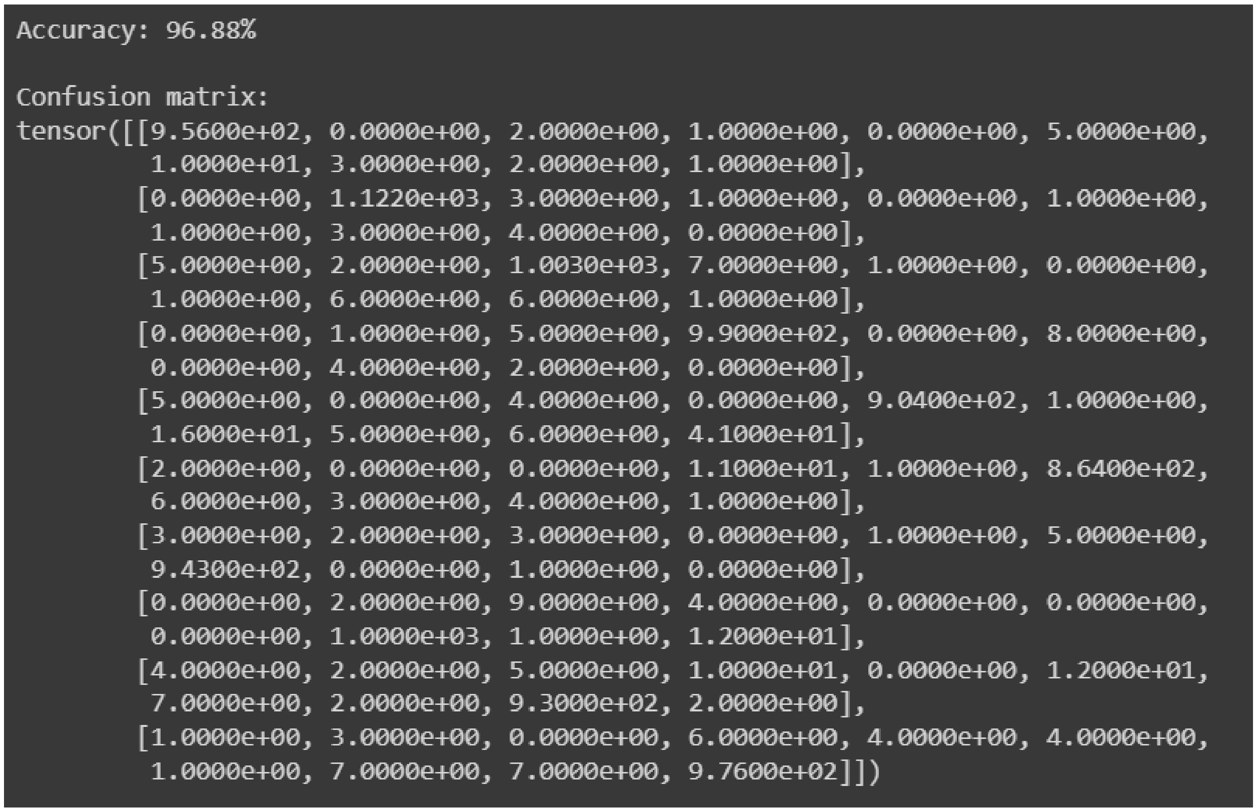
```
Accuracy: 96.35%

Confusion matrix:
tensor([[9.6600e+02, 0.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 2.0000e+00,
         7.0000e+00, 1.0000e+00, 2.0000e+00, 2.0000e+00],
        [0.0000e+00, 1.1150e+03, 3.0000e+00, 2.0000e+00, 1.0000e+00, 1.0000e+00,
         3.0000e+00, 1.0000e+00, 9.0000e+00, 0.0000e+00],
        [7.0000e+00, 2.0000e+00, 9.8800e+02, 9.0000e+00, 3.0000e+00, 2.0000e+00,
         6.0000e+00, 4.0000e+00, 1.0000e+01, 1.0000e+00],
        [0.0000e+00, 1.0000e+00, 6.0000e+00, 9.6200e+02, 1.0000e+00, 8.0000e+00,
         0.0000e+00, 1.1000e+01, 1.5000e+01, 6.0000e+00],
        [1.0000e+00, 0.0000e+00, 2.0000e+00, 0.0000e+00, 9.3800e+02, 0.0000e+00,
         1.1000e+01, 1.0000e+00, 1.0000e+00, 2.8000e+01],
        [6.0000e+00, 0.0000e+00, 1.0000e+00, 1.5000e+01, 0.0000e+00, 8.4700e+02,
         1.0000e+01, 3.0000e+00, 5.0000e+00, 5.0000e+00],
        [8.0000e+00, 4.0000e+00, 1.0000e+00, 0.0000e+00, 2.0000e+00, 3.0000e+00,
         9.3600e+02, 0.0000e+00, 4.0000e+00, 0.0000e+00],
        [1.0000e+00, 4.0000e+00, 1.3000e+01, 2.0000e+00, 4.0000e+00, 0.0000e+00,
         0.0000e+00, 9.9400e+02, 3.0000e+00, 7.0000e+00],
        [6.0000e+00, 2.0000e+00, 2.0000e+00, 6.0000e+00, 2.0000e+00, 6.0000e+00,
         1.0000e+01, 7.0000e+00, 9.2900e+02, 4.0000e+00],
        [5.0000e+00, 4.0000e+00, 2.0000e+00, 5.0000e+00, 9.0000e+00, 3.0000e+00,
         2.0000e+00, 1.4000e+01, 5.0000e+00, 9.6000e+02]])
```

150 neurons in hidden layers, relu as activation function



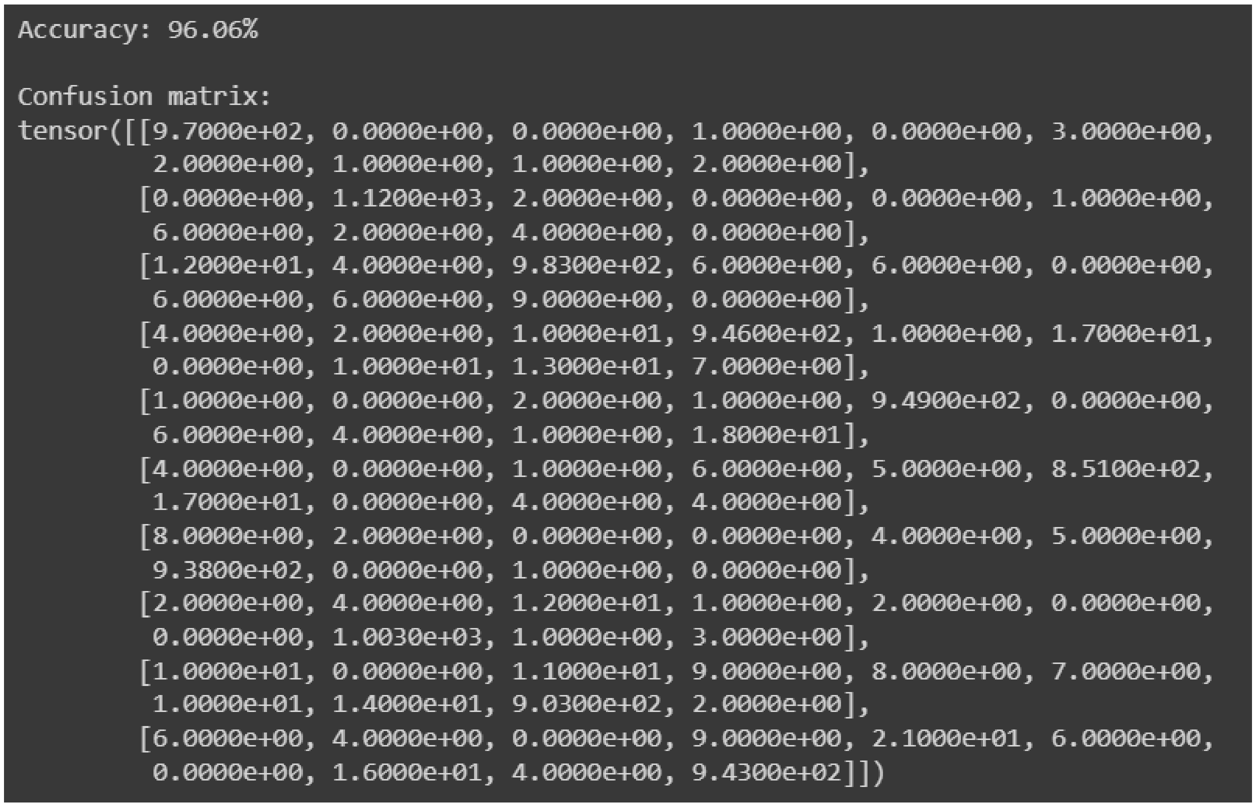
```
Accuracy: 97.28%

Confusion matrix:
tensor([[9.6600e+02, 1.0000e+00, 1.0000e+00, 1.0000e+00, 0.0000e+00, 3.0000e+00,
         5.0000e+00, 1.0000e+00, 0.0000e+00, 2.0000e+00],
        [0.0000e+00, 1.1280e+03, 3.0000e+00, 0.0000e+00, 0.0000e+00, 1.0000e+00,
         1.0000e+00, 0.0000e+00, 2.0000e+00, 0.0000e+00],
        [3.0000e+00, 3.0000e+00, 1.0110e+03, 1.0000e+00, 3.0000e+00, 0.0000e+00,
         3.0000e+00, 4.0000e+00, 4.0000e+00, 0.0000e+00],
        [0.0000e+00, 0.0000e+00, 1.2000e+01, 9.7600e+02, 1.0000e+00, 1.1000e+01,
         0.0000e+00, 3.0000e+00, 4.0000e+00, 3.0000e+00],
        [0.0000e+00, 1.0000e+00, 1.0000e+00, 0.0000e+00, 9.6400e+02, 1.0000e+00,
         8.0000e+00, 1.0000e+00, 1.0000e+00, 5.0000e+00],
        [1.0000e+00, 1.0000e+00, 1.0000e+00, 6.0000e+00, 1.0000e+00, 8.7300e+02,
         3.0000e+00, 1.0000e+00, 2.0000e+00, 3.0000e+00],
        [6.0000e+00, 3.0000e+00, 0.0000e+00, 0.0000e+00, 2.0000e+00, 1.0000e+01,
         9.3500e+02, 1.0000e+00, 0.0000e+00, 1.0000e+00],
        [0.0000e+00, 8.0000e+00, 1.9000e+01, 0.0000e+00, 0.0000e+00, 1.0000e+00,
         0.0000e+00, 9.8400e+02, 4.0000e+00, 1.2000e+01],
        [4.0000e+00, 3.0000e+00, 6.0000e+00, 2.0000e+00, 5.0000e+00, 1.3000e+01,
         6.0000e+00, 3.0000e+00, 9.2700e+02, 5.0000e+00],
        [2.0000e+00, 5.0000e+00, 2.0000e+00, 3.0000e+00, 1.5000e+01, 1.2000e+01,
         1.0000e+00, 5.0000e+00, 0.0000e+00, 9.6400e+02]])
```

150 neurons in hidden layers, sigmoid as activation function



```
Accuracy: 96.88%

Confusion matrix:
tensor([[9.5600e+02, 0.0000e+00, 2.0000e+00, 1.0000e+00, 0.0000e+00, 5.0000e+00,
         1.0000e+01, 3.0000e+00, 2.0000e+00, 1.0000e+00],
        [0.0000e+00, 1.1220e+03, 3.0000e+00, 1.0000e+00, 0.0000e+00, 1.0000e+00,
         1.0000e+00, 3.0000e+00, 4.0000e+00, 0.0000e+00],
        [5.0000e+00, 2.0000e+00, 1.0030e+03, 7.0000e+00, 1.0000e+00, 0.0000e+00,
         1.0000e+00, 6.0000e+00, 6.0000e+00, 1.0000e+00],
        [0.0000e+00, 1.0000e+00, 5.0000e+00, 9.9000e+02, 0.0000e+00, 8.0000e+00,
         0.0000e+00, 4.0000e+00, 2.0000e+00, 0.0000e+00],
        [5.0000e+00, 0.0000e+00, 4.0000e+00, 0.0000e+00, 9.0400e+02, 1.0000e+00,
         1.6000e+01, 5.0000e+00, 6.0000e+00, 4.1000e+01],
        [2.0000e+00, 0.0000e+00, 0.0000e+00, 1.1000e+01, 1.0000e+00, 8.6400e+02,
         6.0000e+00, 3.0000e+00, 4.0000e+00, 1.0000e+00],
        [3.0000e+00, 2.0000e+00, 3.0000e+00, 0.0000e+00, 1.0000e+00, 5.0000e+00,
         9.4300e+02, 0.0000e+00, 1.0000e+00, 0.0000e+00],
        [0.0000e+00, 2.0000e+00, 9.0000e+00, 4.0000e+00, 0.0000e+00, 0.0000e+00,
         0.0000e+00, 1.0000e+03, 1.0000e+00, 1.2000e+01],
        [4.0000e+00, 2.0000e+00, 5.0000e+00, 1.0000e+01, 0.0000e+00, 1.2000e+01,
         7.0000e+00, 2.0000e+00, 9.3000e+02, 2.0000e+00],
        [1.0000e+00, 3.0000e+00, 0.0000e+00, 6.0000e+00, 4.0000e+00, 4.0000e+00,
         1.0000e+00, 7.0000e+00, 7.0000e+00, 9.7600e+02]])
```

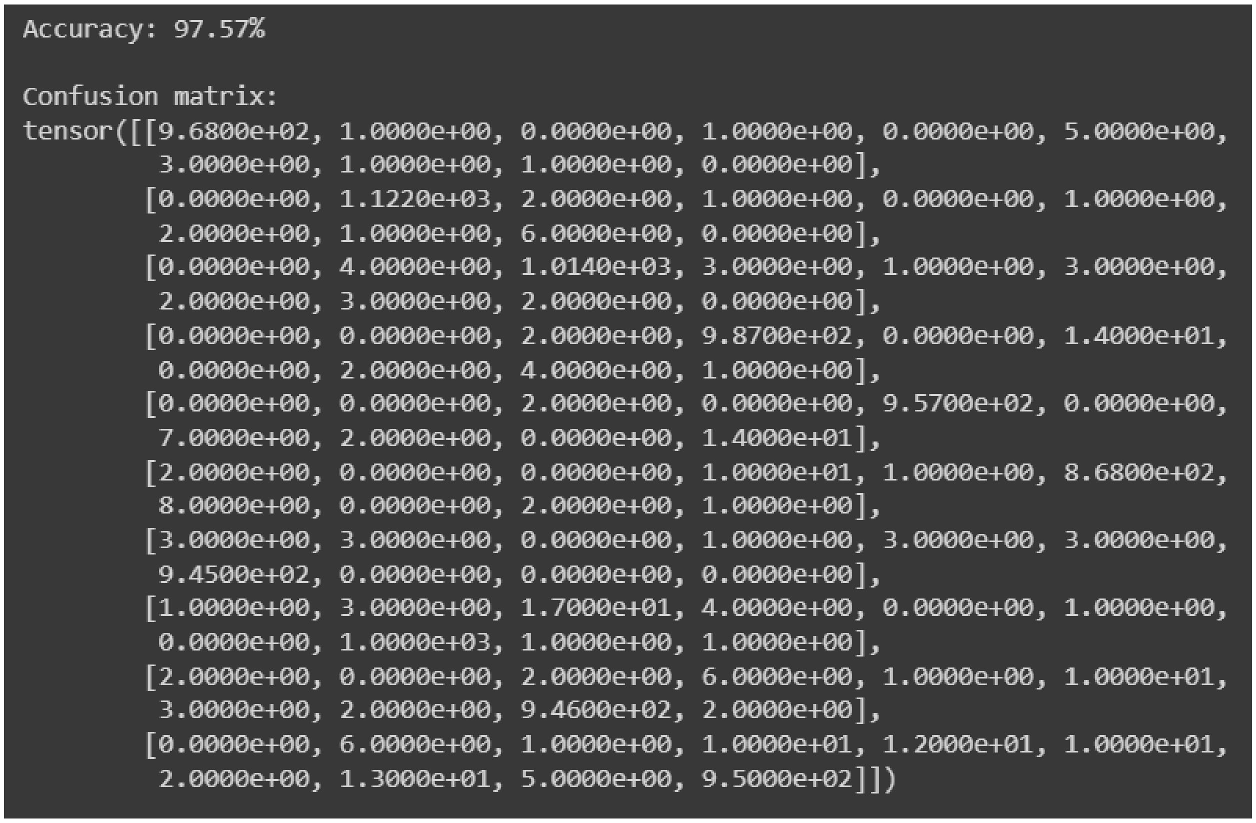
150 neurons in hidden layers, tanh as activation function



```
Accuracy: 96.06%

Confusion matrix:
tensor([[9.7000e+02, 0.0000e+00, 0.0000e+00, 1.0000e+00, 0.0000e+00, 3.0000e+00,
         2.0000e+00, 1.0000e+00, 1.0000e+00, 2.0000e+00],
        [0.0000e+00, 1.1200e+03, 2.0000e+00, 0.0000e+00, 0.0000e+00, 1.0000e+00,
         6.0000e+00, 2.0000e+00, 4.0000e+00, 0.0000e+00],
        [1.2000e+01, 4.0000e+00, 9.8300e+02, 6.0000e+00, 6.0000e+00, 0.0000e+00,
         6.0000e+00, 6.0000e+00, 9.0000e+00, 0.0000e+00],
        [4.0000e+00, 2.0000e+00, 1.0000e+01, 9.4600e+02, 1.0000e+00, 1.7000e+01,
         0.0000e+00, 1.0000e+01, 1.3000e+01, 7.0000e+00],
        [1.0000e+00, 0.0000e+00, 2.0000e+00, 1.0000e+00, 9.4900e+02, 0.0000e+00,
         6.0000e+00, 4.0000e+00, 1.0000e+00, 1.8000e+01],
        [4.0000e+00, 0.0000e+00, 1.0000e+00, 6.0000e+00, 5.0000e+00, 8.5100e+02,
         1.7000e+01, 0.0000e+00, 4.0000e+00, 4.0000e+00],
        [8.0000e+00, 2.0000e+00, 0.0000e+00, 0.0000e+00, 4.0000e+00, 5.0000e+00,
         9.3800e+02, 0.0000e+00, 1.0000e+00, 0.0000e+00],
        [2.0000e+00, 4.0000e+00, 1.2000e+01, 1.0000e+00, 2.0000e+00, 0.0000e+00,
         0.0000e+00, 1.0030e+03, 1.0000e+00, 3.0000e+00],
        [1.0000e+01, 0.0000e+00, 1.1000e+01, 9.0000e+00, 8.0000e+00, 7.0000e+00,
         1.0000e+01, 1.4000e+01, 9.0300e+02, 2.0000e+00],
        [6.0000e+00, 4.0000e+00, 0.0000e+00, 9.0000e+00, 2.1000e+01, 6.0000e+00,
         0.0000e+00, 1.6000e+01, 4.0000e+00, 9.4300e+02]])
```

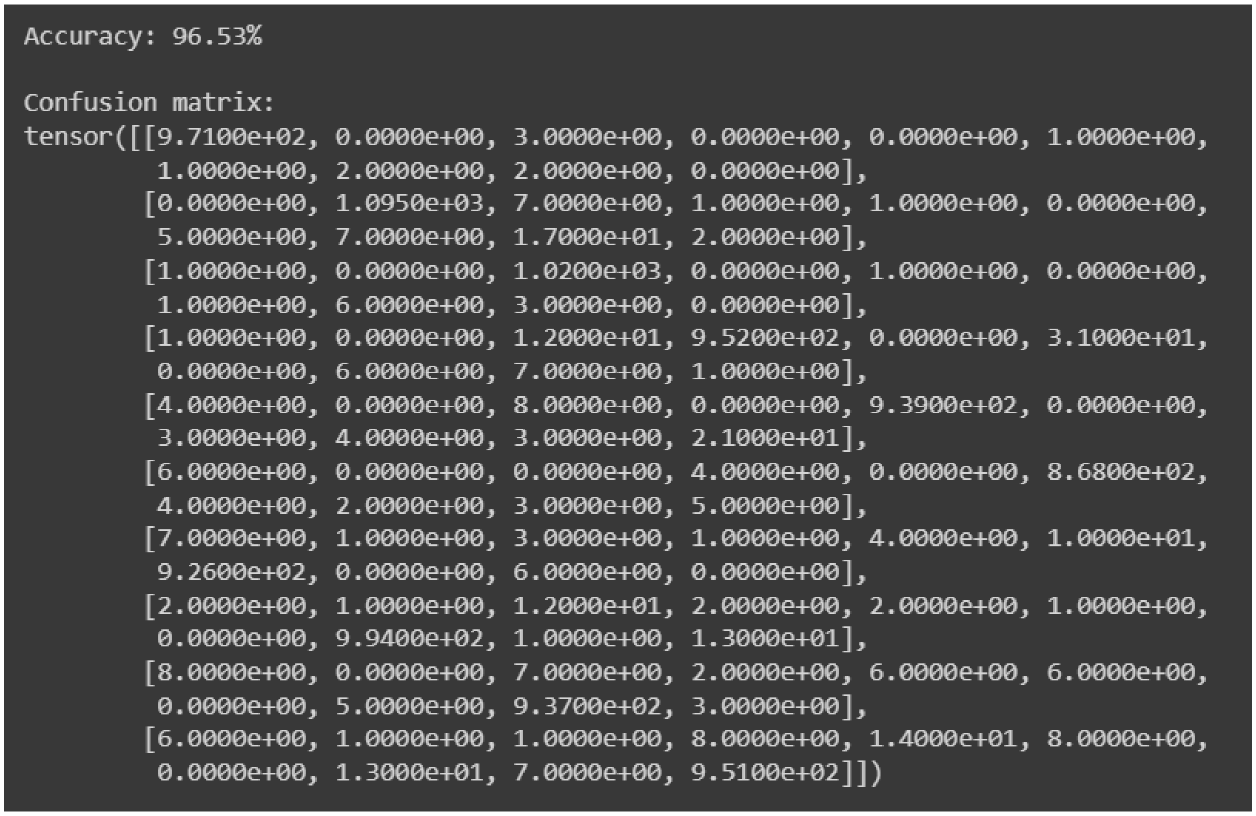
ii) 3 hidden layers

100 neurons in hidden layers, relu as activation function



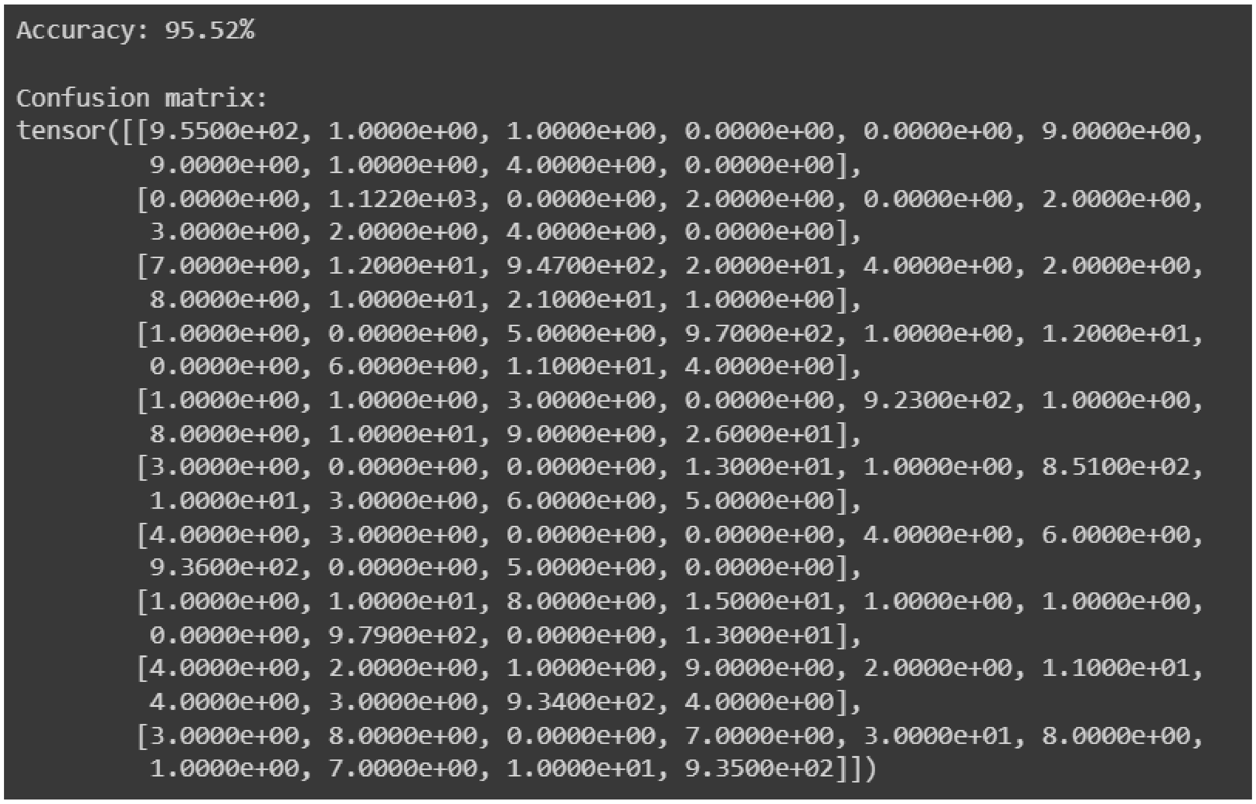
```
Accuracy: 97.57%

Confusion matrix:
tensor([[9.6800e+02, 1.0000e+00, 0.0000e+00, 1.0000e+00, 0.0000e+00, 5.0000e+00,
         3.0000e+00, 1.0000e+00, 1.0000e+00, 0.0000e+00],
        [0.0000e+00, 1.1220e+03, 2.0000e+00, 1.0000e+00, 0.0000e+00, 1.0000e+00,
         2.0000e+00, 1.0000e+00, 6.0000e+00, 0.0000e+00],
        [0.0000e+00, 4.0000e+00, 1.0140e+03, 3.0000e+00, 1.0000e+00, 3.0000e+00,
         2.0000e+00, 3.0000e+00, 2.0000e+00, 0.0000e+00],
        [0.0000e+00, 0.0000e+00, 2.0000e+00, 9.8700e+02, 0.0000e+00, 1.4000e+01,
         0.0000e+00, 2.0000e+00, 4.0000e+00, 1.0000e+00],
        [0.0000e+00, 0.0000e+00, 2.0000e+00, 0.0000e+00, 9.5700e+02, 0.0000e+00,
         7.0000e+00, 2.0000e+00, 0.0000e+00, 1.4000e+01],
        [2.0000e+00, 0.0000e+00, 0.0000e+00, 1.0000e+01, 1.0000e+00, 8.6800e+02,
         8.0000e+00, 0.0000e+00, 2.0000e+00, 1.0000e+00],
        [3.0000e+00, 3.0000e+00, 0.0000e+00, 1.0000e+00, 3.0000e+00, 3.0000e+00,
         9.4500e+02, 0.0000e+00, 0.0000e+00, 0.0000e+00],
        [1.0000e+00, 3.0000e+00, 1.7000e+01, 4.0000e+00, 0.0000e+00, 1.0000e+00,
         0.0000e+00, 1.0000e+03, 1.0000e+00, 1.0000e+00],
        [2.0000e+00, 0.0000e+00, 2.0000e+00, 6.0000e+00, 1.0000e+00, 1.0000e+01,
         3.0000e+00, 2.0000e+00, 9.4600e+02, 2.0000e+00],
        [0.0000e+00, 6.0000e+00, 1.0000e+00, 1.0000e+01, 1.2000e+01, 1.0000e+01,
         2.0000e+00, 1.3000e+01, 5.0000e+00, 9.5000e+02]])
```

100 neurons in hidden layers, sigmoid as activation function



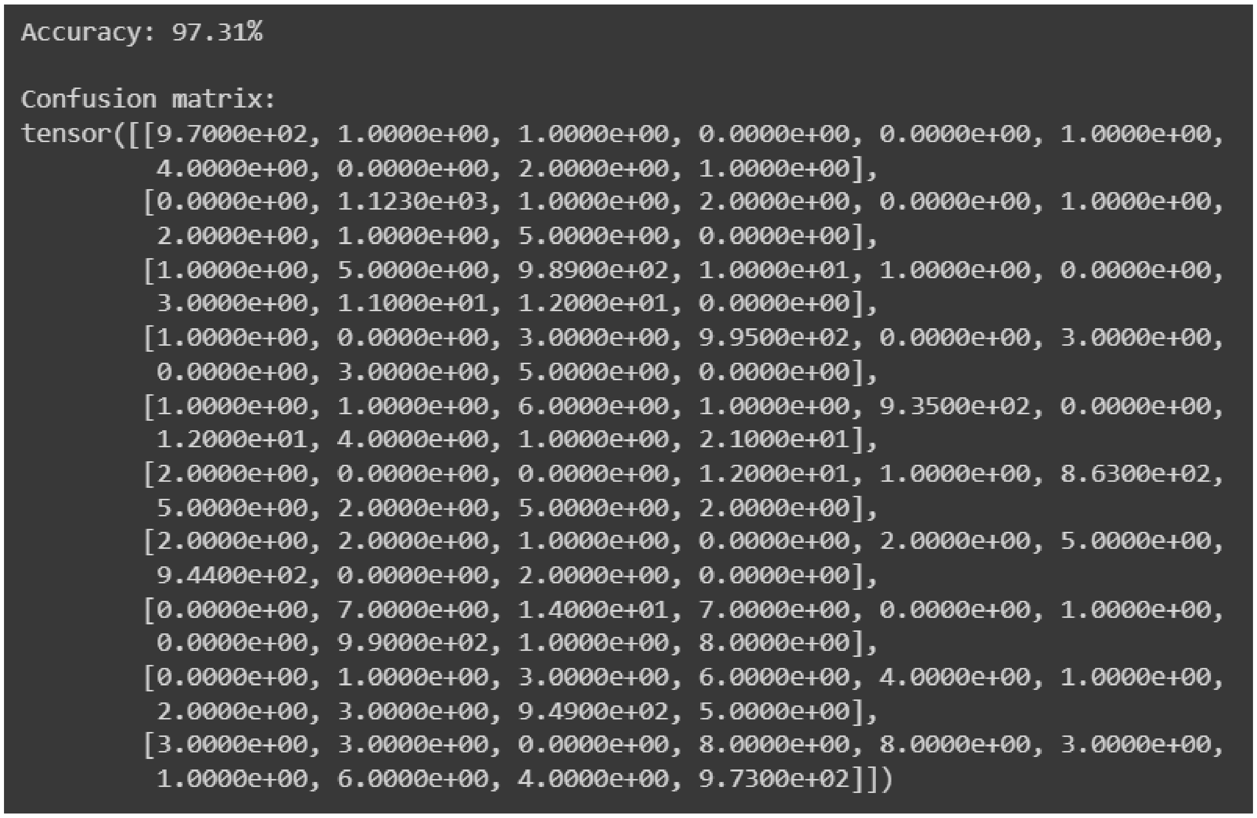
```
Accuracy: 96.53%

Confusion matrix:
tensor([[9.7100e+02, 0.0000e+00, 3.0000e+00, 0.0000e+00, 0.0000e+00, 1.0000e+00,
         1.0000e+00, 2.0000e+00, 2.0000e+00, 0.0000e+00],
        [0.0000e+00, 1.0950e+03, 7.0000e+00, 1.0000e+00, 1.0000e+00, 0.0000e+00,
         5.0000e+00, 7.0000e+00, 1.7000e+01, 2.0000e+00],
        [1.0000e+00, 0.0000e+00, 1.0200e+03, 0.0000e+00, 1.0000e+00, 0.0000e+00,
         1.0000e+00, 6.0000e+00, 3.0000e+00, 0.0000e+00],
        [1.0000e+00, 0.0000e+00, 1.2000e+01, 9.5200e+02, 0.0000e+00, 3.1000e+01,
         0.0000e+00, 6.0000e+00, 7.0000e+00, 1.0000e+00],
        [4.0000e+00, 0.0000e+00, 8.0000e+00, 0.0000e+00, 9.3900e+02, 0.0000e+00,
         3.0000e+00, 4.0000e+00, 3.0000e+00, 2.1000e+01],
        [6.0000e+00, 0.0000e+00, 0.0000e+00, 4.0000e+00, 0.0000e+00, 8.6800e+02,
         4.0000e+00, 2.0000e+00, 3.0000e+00, 5.0000e+00],
        [7.0000e+00, 1.0000e+00, 3.0000e+00, 1.0000e+00, 4.0000e+00, 1.0000e+01,
         9.2600e+02, 0.0000e+00, 6.0000e+00, 0.0000e+00],
        [2.0000e+00, 1.0000e+00, 1.2000e+01, 2.0000e+00, 2.0000e+00, 1.0000e+00,
         0.0000e+00, 9.9400e+02, 1.0000e+00, 1.3000e+01],
        [8.0000e+00, 0.0000e+00, 7.0000e+00, 2.0000e+00, 6.0000e+00, 6.0000e+00,
         0.0000e+00, 5.0000e+00, 9.3700e+02, 3.0000e+00],
        [6.0000e+00, 1.0000e+00, 1.0000e+00, 8.0000e+00, 1.4000e+01, 8.0000e+00,
         0.0000e+00, 1.3000e+01, 7.0000e+00, 9.5100e+02]])
```

100 neurons in hidden layers, tanh as activation function



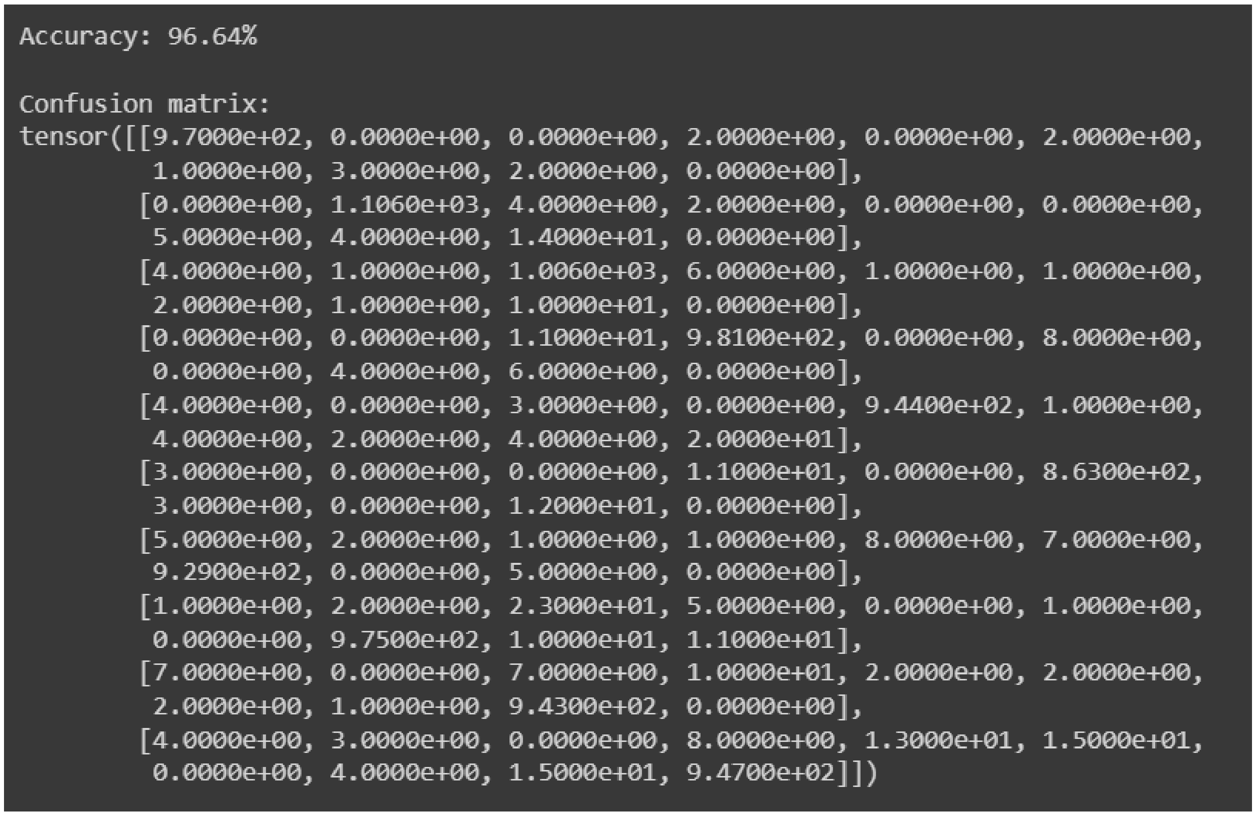
```
Accuracy: 95.52%

Confusion matrix:
tensor([[9.5500e+02, 1.0000e+00, 1.0000e+00, 0.0000e+00, 0.0000e+00, 9.0000e+00,
         9.0000e+00, 1.0000e+00, 4.0000e+00, 0.0000e+00],
        [0.0000e+00, 1.1220e+03, 0.0000e+00, 2.0000e+00, 0.0000e+00, 2.0000e+00,
         3.0000e+00, 2.0000e+00, 4.0000e+00, 0.0000e+00],
        [7.0000e+00, 1.2000e+01, 9.4700e+02, 2.0000e+01, 4.0000e+00, 2.0000e+00,
         8.0000e+00, 1.0000e+01, 2.1000e+01, 1.0000e+00],
        [1.0000e+00, 0.0000e+00, 5.0000e+00, 9.7000e+02, 1.0000e+00, 1.2000e+01,
         0.0000e+00, 6.0000e+00, 1.1000e+01, 4.0000e+00],
        [1.0000e+00, 1.0000e+00, 3.0000e+00, 0.0000e+00, 9.2300e+02, 1.0000e+00,
         8.0000e+00, 1.0000e+01, 9.0000e+00, 2.6000e+01],
        [3.0000e+00, 0.0000e+00, 0.0000e+00, 1.3000e+01, 1.0000e+00, 8.5100e+02,
         1.0000e+01, 3.0000e+00, 6.0000e+00, 5.0000e+00],
        [4.0000e+00, 3.0000e+00, 0.0000e+00, 0.0000e+00, 4.0000e+00, 6.0000e+00,
         9.3600e+02, 0.0000e+00, 5.0000e+00, 0.0000e+00],
        [1.0000e+00, 1.0000e+01, 8.0000e+00, 1.5000e+01, 1.0000e+00, 1.0000e+00,
         0.0000e+00, 9.7900e+02, 0.0000e+00, 1.3000e+01],
        [4.0000e+00, 2.0000e+00, 1.0000e+00, 9.0000e+00, 2.0000e+00, 1.1000e+01,
         4.0000e+00, 3.0000e+00, 9.3400e+02, 4.0000e+00],
        [3.0000e+00, 8.0000e+00, 0.0000e+00, 7.0000e+00, 3.0000e+01, 8.0000e+00,
         1.0000e+00, 7.0000e+00, 1.0000e+01, 9.3500e+02]])
```

150 neurons in hidden layers, relu as activation function



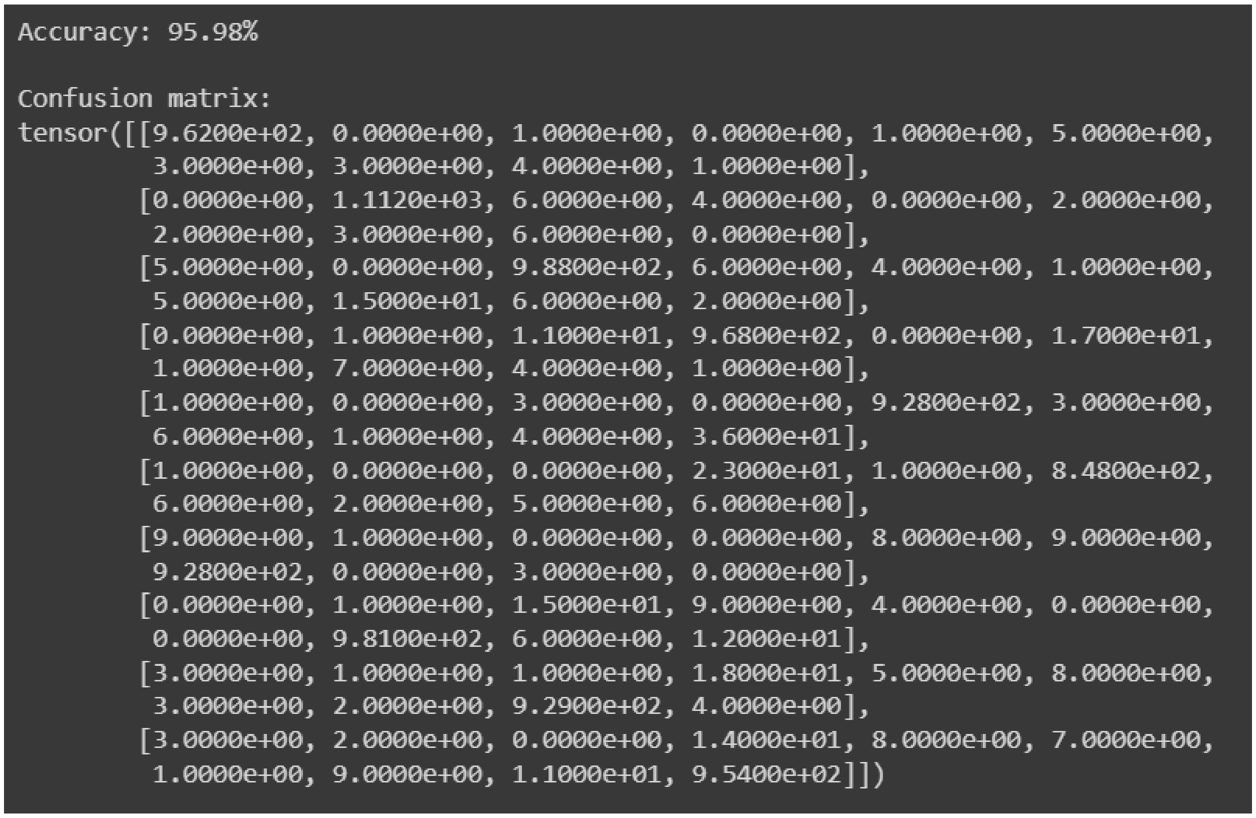
```
Accuracy: 97.31%

Confusion matrix:
tensor([[9.7000e+02, 1.0000e+00, 1.0000e+00, 0.0000e+00, 0.0000e+00, 1.0000e+00,
         4.0000e+00, 0.0000e+00, 2.0000e+00, 1.0000e+00],
        [0.0000e+00, 1.1230e+03, 1.0000e+00, 2.0000e+00, 0.0000e+00, 1.0000e+00,
         2.0000e+00, 1.0000e+00, 5.0000e+00, 0.0000e+00],
        [1.0000e+00, 5.0000e+00, 9.8900e+02, 1.0000e+01, 1.0000e+00, 0.0000e+00,
         3.0000e+00, 1.1000e+01, 1.2000e+01, 0.0000e+00],
        [1.0000e+00, 0.0000e+00, 3.0000e+00, 9.9500e+02, 0.0000e+00, 3.0000e+00,
         0.0000e+00, 3.0000e+00, 5.0000e+00, 0.0000e+00],
        [1.0000e+00, 1.0000e+00, 6.0000e+00, 1.0000e+00, 9.3500e+02, 0.0000e+00,
         1.2000e+01, 4.0000e+00, 1.0000e+00, 2.1000e+01],
        [2.0000e+00, 0.0000e+00, 0.0000e+00, 1.2000e+01, 1.0000e+00, 8.6300e+02,
         5.0000e+00, 2.0000e+00, 5.0000e+00, 2.0000e+00],
        [2.0000e+00, 2.0000e+00, 1.0000e+00, 0.0000e+00, 2.0000e+00, 5.0000e+00,
         9.4400e+02, 0.0000e+00, 2.0000e+00, 0.0000e+00],
        [0.0000e+00, 7.0000e+00, 1.4000e+01, 7.0000e+00, 0.0000e+00, 1.0000e+00,
         0.0000e+00, 9.9000e+02, 1.0000e+00, 8.0000e+00],
        [0.0000e+00, 1.0000e+00, 3.0000e+00, 6.0000e+00, 4.0000e+00, 1.0000e+00,
         2.0000e+00, 3.0000e+00, 9.4900e+02, 5.0000e+00],
        [3.0000e+00, 3.0000e+00, 0.0000e+00, 8.0000e+00, 8.0000e+00, 3.0000e+00,
         1.0000e+00, 6.0000e+00, 4.0000e+00, 9.7300e+02]])
```

150 neurons in hidden layers, sigmoid as activation function



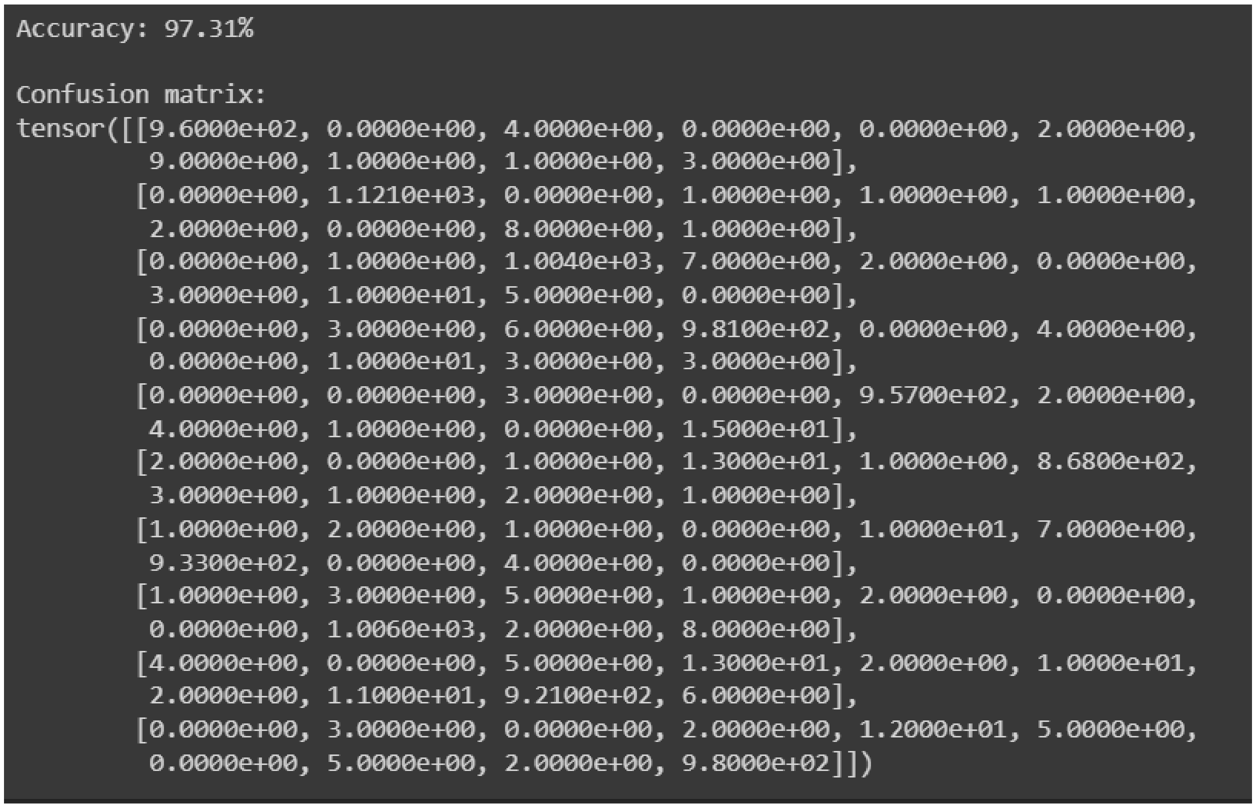
```
Accuracy: 96.64%

Confusion matrix:
tensor([[9.7000e+02, 0.0000e+00, 0.0000e+00, 2.0000e+00, 0.0000e+00, 2.0000e+00,
         1.0000e+00, 3.0000e+00, 2.0000e+00, 0.0000e+00],
        [0.0000e+00, 1.1060e+03, 4.0000e+00, 2.0000e+00, 0.0000e+00, 0.0000e+00,
         5.0000e+00, 4.0000e+00, 1.4000e+01, 0.0000e+00],
        [4.0000e+00, 1.0000e+00, 1.0060e+03, 6.0000e+00, 1.0000e+00, 1.0000e+00,
         2.0000e+00, 1.0000e+00, 1.0000e+01, 0.0000e+00],
        [0.0000e+00, 0.0000e+00, 1.1000e+01, 9.8100e+02, 0.0000e+00, 8.0000e+00,
         0.0000e+00, 4.0000e+00, 6.0000e+00, 0.0000e+00],
        [4.0000e+00, 0.0000e+00, 3.0000e+00, 0.0000e+00, 9.4400e+02, 1.0000e+00,
         4.0000e+00, 2.0000e+00, 4.0000e+00, 2.0000e+01],
        [3.0000e+00, 0.0000e+00, 0.0000e+00, 1.1000e+01, 0.0000e+00, 8.6300e+02,
         3.0000e+00, 0.0000e+00, 1.2000e+01, 0.0000e+00],
        [5.0000e+00, 2.0000e+00, 1.0000e+00, 1.0000e+00, 8.0000e+00, 7.0000e+00,
         9.2900e+02, 0.0000e+00, 5.0000e+00, 0.0000e+00],
        [1.0000e+00, 2.0000e+00, 2.3000e+01, 5.0000e+00, 0.0000e+00, 1.0000e+00,
         0.0000e+00, 9.7500e+02, 1.0000e+01, 1.1000e+01],
        [7.0000e+00, 0.0000e+00, 7.0000e+00, 1.0000e+01, 2.0000e+00, 2.0000e+00,
         2.0000e+00, 1.0000e+00, 9.4300e+02, 0.0000e+00],
        [4.0000e+00, 3.0000e+00, 0.0000e+00, 8.0000e+00, 1.3000e+01, 1.5000e+01,
         0.0000e+00, 4.0000e+00, 1.5000e+01, 9.4700e+02]])
```

150 neurons in hidden layers, tanh as activation function



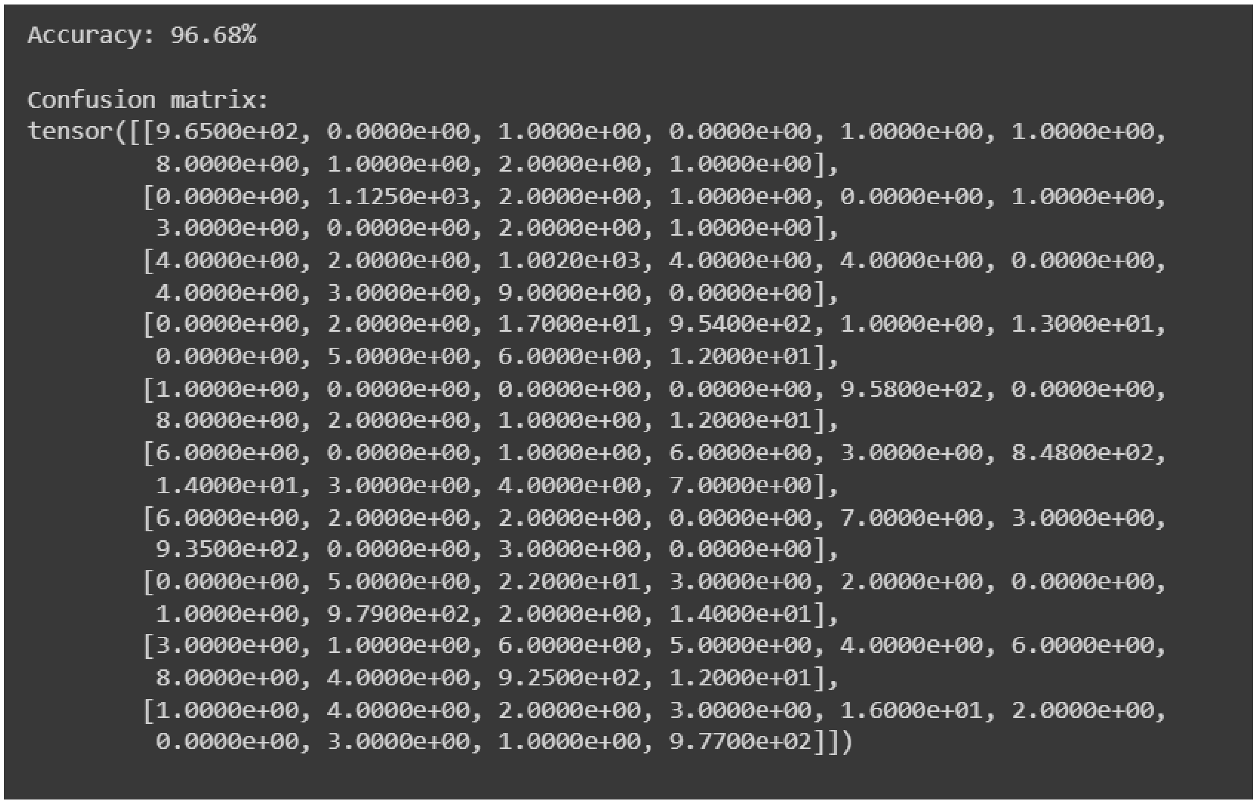
```
Accuracy: 95.98%

Confusion matrix:
tensor([[9.6200e+02, 0.0000e+00, 1.0000e+00, 0.0000e+00, 1.0000e+00, 5.0000e+00,
         3.0000e+00, 3.0000e+00, 4.0000e+00, 1.0000e+00],
        [0.0000e+00, 1.1120e+03, 6.0000e+00, 4.0000e+00, 0.0000e+00, 2.0000e+00,
         2.0000e+00, 3.0000e+00, 6.0000e+00, 0.0000e+00],
        [5.0000e+00, 0.0000e+00, 9.8800e+02, 6.0000e+00, 4.0000e+00, 1.0000e+00,
         5.0000e+00, 1.5000e+01, 6.0000e+00, 2.0000e+00],
        [0.0000e+00, 1.0000e+00, 1.1000e+01, 9.6800e+02, 0.0000e+00, 1.7000e+01,
         1.0000e+00, 7.0000e+00, 4.0000e+00, 1.0000e+00],
        [1.0000e+00, 0.0000e+00, 3.0000e+00, 0.0000e+00, 9.2800e+02, 3.0000e+00,
         6.0000e+00, 1.0000e+00, 4.0000e+00, 3.6000e+01],
        [1.0000e+00, 0.0000e+00, 0.0000e+00, 2.3000e+01, 1.0000e+00, 8.4800e+02,
         6.0000e+00, 2.0000e+00, 5.0000e+00, 6.0000e+00],
        [9.0000e+00, 1.0000e+00, 0.0000e+00, 0.0000e+00, 8.0000e+00, 9.0000e+00,
         9.2800e+02, 0.0000e+00, 3.0000e+00, 0.0000e+00],
        [0.0000e+00, 1.0000e+00, 1.5000e+01, 9.0000e+00, 4.0000e+00, 0.0000e+00,
         0.0000e+00, 9.8100e+02, 6.0000e+00, 1.2000e+01],
        [3.0000e+00, 1.0000e+00, 1.0000e+00, 1.8000e+01, 5.0000e+00, 8.0000e+00,
         3.0000e+00, 2.0000e+00, 9.2900e+02, 4.0000e+00],
        [3.0000e+00, 2.0000e+00, 0.0000e+00, 1.4000e+01, 8.0000e+00, 7.0000e+00,
         1.0000e+00, 9.0000e+00, 1.1000e+01, 9.5400e+02]])
```

125 neurons in hidden layers, relu as activation function



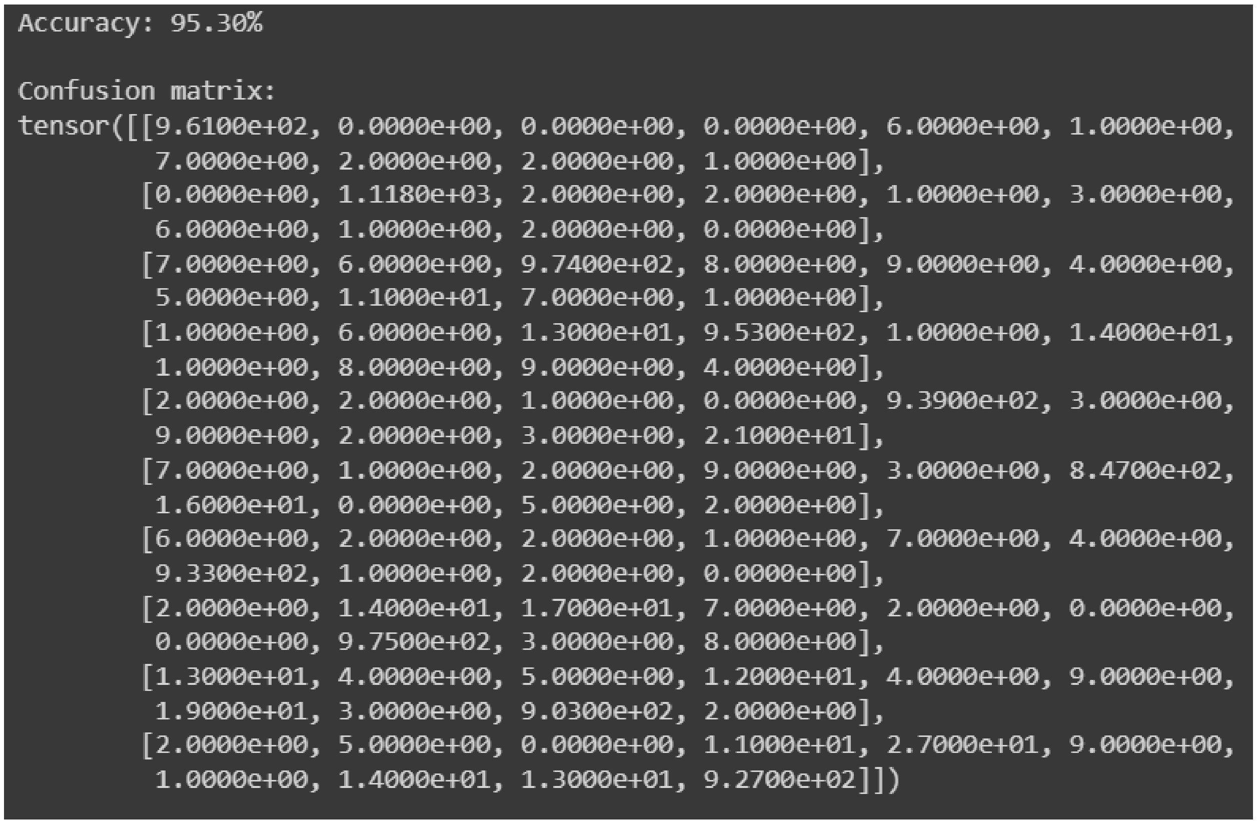
```
Accuracy: 97.31%

Confusion matrix:
tensor([[9.6000e+02, 0.0000e+00, 4.0000e+00, 0.0000e+00, 0.0000e+00, 2.0000e+00,
         9.0000e+00, 1.0000e+00, 1.0000e+00, 3.0000e+00],
        [0.0000e+00, 1.1210e+03, 0.0000e+00, 1.0000e+00, 1.0000e+00, 1.0000e+00,
         2.0000e+00, 0.0000e+00, 8.0000e+00, 1.0000e+00],
        [0.0000e+00, 1.0000e+00, 1.0040e+03, 7.0000e+00, 2.0000e+00, 0.0000e+00,
         3.0000e+00, 1.0000e+01, 5.0000e+00, 0.0000e+00],
        [0.0000e+00, 3.0000e+00, 6.0000e+00, 9.8100e+02, 0.0000e+00, 4.0000e+00,
         0.0000e+00, 1.0000e+01, 3.0000e+00, 3.0000e+00],
        [0.0000e+00, 0.0000e+00, 3.0000e+00, 0.0000e+00, 9.5700e+02, 2.0000e+00,
         4.0000e+00, 1.0000e+00, 0.0000e+00, 1.5000e+01],
        [2.0000e+00, 0.0000e+00, 1.0000e+00, 1.3000e+01, 1.0000e+00, 8.6800e+02,
         3.0000e+00, 1.0000e+00, 2.0000e+00, 1.0000e+00],
        [1.0000e+00, 2.0000e+00, 1.0000e+00, 0.0000e+00, 1.0000e+01, 7.0000e+00,
         9.3300e+02, 0.0000e+00, 4.0000e+00, 0.0000e+00],
        [1.0000e+00, 3.0000e+00, 5.0000e+00, 1.0000e+00, 2.0000e+00, 0.0000e+00,
         0.0000e+00, 1.0060e+03, 2.0000e+00, 8.0000e+00],
        [4.0000e+00, 0.0000e+00, 5.0000e+00, 1.3000e+01, 2.0000e+00, 1.0000e+01,
         2.0000e+00, 1.1000e+01, 9.2100e+02, 6.0000e+00],
        [0.0000e+00, 3.0000e+00, 0.0000e+00, 2.0000e+00, 1.2000e+01, 5.0000e+00,
         0.0000e+00, 5.0000e+00, 2.0000e+00, 9.8000e+02]])
```

125 neurons in hidden layers, sigmoid as activation function



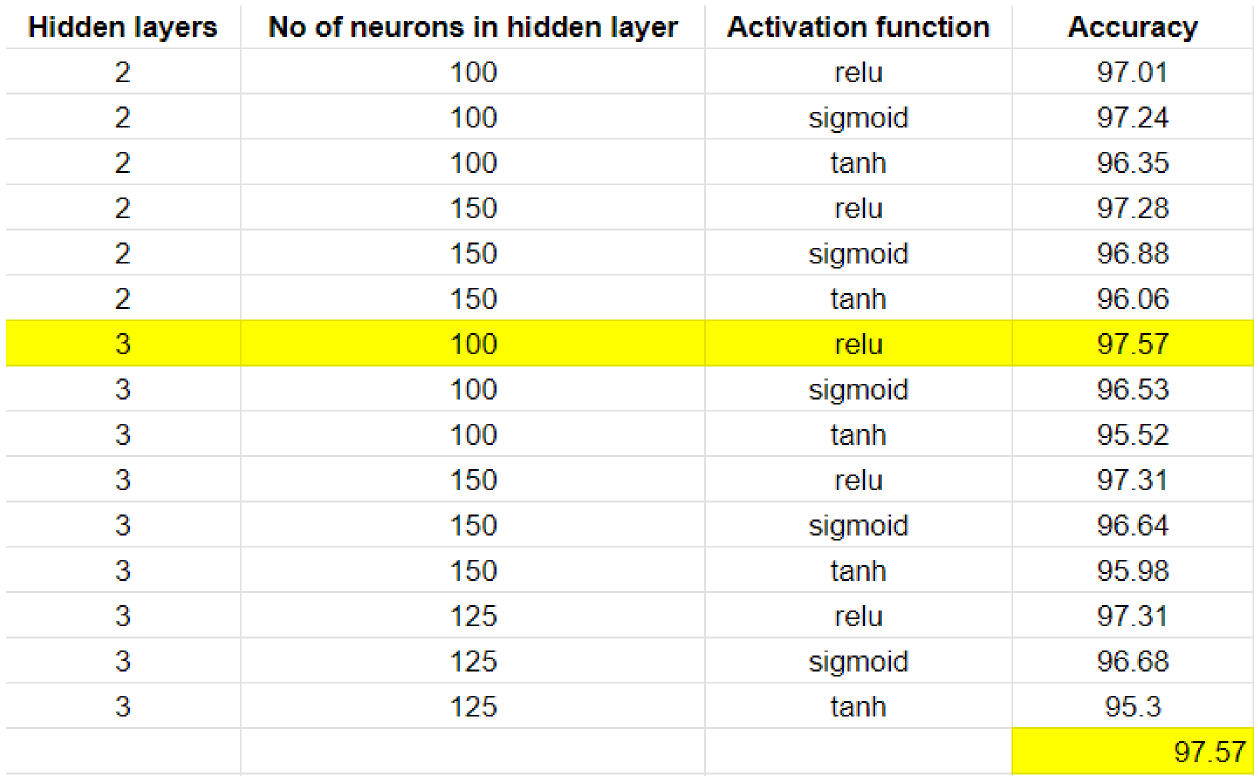
```
Accuracy: 96.68%

Confusion matrix:
tensor([[9.6500e+02, 0.0000e+00, 1.0000e+00, 0.0000e+00, 1.0000e+00, 1.0000e+00,
         8.0000e+00, 1.0000e+00, 2.0000e+00, 1.0000e+00],
        [0.0000e+00, 1.1250e+03, 2.0000e+00, 1.0000e+00, 0.0000e+00, 1.0000e+00,
         3.0000e+00, 0.0000e+00, 2.0000e+00, 1.0000e+00],
        [4.0000e+00, 2.0000e+00, 1.0020e+03, 4.0000e+00, 4.0000e+00, 0.0000e+00,
         4.0000e+00, 3.0000e+00, 9.0000e+00, 0.0000e+00],
        [0.0000e+00, 2.0000e+00, 1.7000e+01, 9.5400e+02, 1.0000e+00, 1.3000e+01,
         0.0000e+00, 5.0000e+00, 6.0000e+00, 1.2000e+01],
        [1.0000e+00, 0.0000e+00, 0.0000e+00, 0.0000e+00, 9.5800e+02, 0.0000e+00,
         8.0000e+00, 2.0000e+00, 1.0000e+00, 1.2000e+01],
        [6.0000e+00, 0.0000e+00, 1.0000e+00, 6.0000e+00, 3.0000e+00, 8.4800e+02,
         1.4000e+01, 3.0000e+00, 4.0000e+00, 7.0000e+00],
        [6.0000e+00, 2.0000e+00, 2.0000e+00, 0.0000e+00, 7.0000e+00, 3.0000e+00,
         9.3500e+02, 0.0000e+00, 3.0000e+00, 0.0000e+00],
        [0.0000e+00, 5.0000e+00, 2.2000e+01, 3.0000e+00, 2.0000e+00, 0.0000e+00,
         1.0000e+00, 9.7900e+02, 2.0000e+00, 1.4000e+01],
        [3.0000e+00, 1.0000e+00, 6.0000e+00, 5.0000e+00, 4.0000e+00, 6.0000e+00,
         8.0000e+00, 4.0000e+00, 9.2500e+02, 1.2000e+01],
        [1.0000e+00, 4.0000e+00, 2.0000e+00, 3.0000e+00, 1.6000e+01, 2.0000e+00,
         0.0000e+00, 3.0000e+00, 1.0000e+00, 9.7700e+02]])
```

125 neurons in hidden layers, tanh as activation function



```
Accuracy: 95.30%

Confusion matrix:
tensor([[9.6100e+02, 0.0000e+00, 0.0000e+00, 0.0000e+00, 6.0000e+00, 1.0000e+00,
         7.0000e+00, 2.0000e+00, 2.0000e+00, 1.0000e+00],
        [0.0000e+00, 1.1180e+03, 2.0000e+00, 2.0000e+00, 1.0000e+00, 3.0000e+00,
         6.0000e+00, 1.0000e+00, 2.0000e+00, 0.0000e+00],
        [7.0000e+00, 6.0000e+00, 9.7400e+02, 8.0000e+00, 9.0000e+00, 4.0000e+00,
         5.0000e+00, 1.1000e+01, 7.0000e+00, 1.0000e+00],
        [1.0000e+00, 6.0000e+00, 1.3000e+01, 9.5300e+02, 1.0000e+00, 1.4000e+01,
         1.0000e+00, 8.0000e+00, 9.0000e+00, 4.0000e+00],
        [2.0000e+00, 2.0000e+00, 1.0000e+00, 0.0000e+00, 9.3900e+02, 3.0000e+00,
         9.0000e+00, 2.0000e+00, 3.0000e+00, 2.1000e+01],
        [7.0000e+00, 1.0000e+00, 2.0000e+00, 9.0000e+00, 3.0000e+00, 8.4700e+02,
         1.6000e+01, 0.0000e+00, 5.0000e+00, 2.0000e+00],
        [6.0000e+00, 2.0000e+00, 2.0000e+00, 1.0000e+00, 7.0000e+00, 4.0000e+00,
         9.3300e+02, 1.0000e+00, 2.0000e+00, 0.0000e+00],
        [2.0000e+00, 1.4000e+01, 1.7000e+01, 7.0000e+00, 2.0000e+00, 0.0000e+00,
         0.0000e+00, 9.7500e+02, 3.0000e+00, 8.0000e+00],
        [1.3000e+01, 4.0000e+00, 5.0000e+00, 1.2000e+01, 4.0000e+00, 9.0000e+00,
         1.9000e+01, 3.0000e+00, 9.0300e+02, 2.0000e+00],
        [2.0000e+00, 5.0000e+00, 0.0000e+00, 1.1000e+01, 2.7000e+01, 9.0000e+00,
         1.0000e+00, 1.4000e+01, 1.3000e+01, 9.2700e+02]])
```

Competitive study:



Hidden layers	No of neurons in hidden layer	Activation function	Accuracy
2	100	relu	97.01
2	100	sigmoid	97.24
2	100	tanh	96.35
2	150	relu	97.28
2	150	sigmoid	96.88
2	150	tanh	96.06
3	100	relu	97.57
3	100	sigmoid	96.53
3	100	tanh	95.52
3	150	relu	97.31
3	150	sigmoid	96.64
3	150	tanh	95.98
3	125	relu	97.31
3	125	sigmoid	96.68
3	125	tanh	95.3
			97.57

Above is the table of all 15 models with their accuracy. Based on accuracy the model with 3 hidden layers with 100 neurons and activation function as relu is the best-performing model

You can find the code at :
https://colab.research.google.com/drive/1DJEm3OJP4pVYxrNguUYZYEatKQEUHDCx#scrollTo=kWyVX1UwfESi

https://colab.research.google.com/drive/1s59RUlVfin_EI5iUS5uRa3n0PdL7UDOg#scrollTo=yREP7aWWtjta