SecureHeart*

Nureen Barakat
Intelligent Systems Department
Faculty of Computers and Data Science
Alexandria, Egypt
nureenbrakat3@gmail.com

Bassant Mohamed
Intelligent Systems Department
Faculty of Computers and Data Science
Alexandria, Egypt
bassantmohamed414@gmail.com

Zeinab Mohamed
Intelligent Systems Department
Faculty of Computers and Data Science
Alexandria, Egypt
cds.zainabMohamed2022@alexu.edu.eg

***Abstract*—Heart disease remains a significant global health concern, demanding effective predictive models for early diagnosis and intervention. Deep learning techniques, particularly neural networks, have shown promise in medical diagnostics. This study presents "SecureHeart," an investigation into the development of a deep learning-based model for heart disease prediction using a dataset sourced from Kaggle. Additionally, it explores the model's resilience against cybersecurity attacks, evaluating its robustness under adversarial conditions. The study involves preprocessing the dataset, splitting it into training and testing sets, training the model, attacking the model using adversarial examples, and implementing defense mechanisms to protect the model from such attacks. Through this research, insights into the effectiveness of deep learning in heart disease prediction and its susceptibility to adversarial manipulation will be gained, contributing to the development of more reliable and secure healthcare applications of deep learning.**

I. Introduction

Heart disease is a leading cause of mortality worldwide, necessitating accurate and timely diagnosis for effective intervention. Deep learning, a subset of artificial intelligence, has emerged as a promising approach for medical diagnostics due to its ability to extract complex patterns from large datasets. However, the deployment of deep learning models in critical applications such as healthcare introduces new challenges, including susceptibility to cybersecurity attacks. Adversarial attacks, where subtle perturbations are introduced to input data to deceive the model, pose a significant threat to the reliability and integrity of deep learning-based diagnostic systems. In this context, this study aims to develop a deep learning model for heart disease prediction while investigating its vulnerability to adversarial manipulation. The research will follow a structured methodology, including data preprocessing, model training, adversarial attack simulation, and the implementation of defense mechanisms to enhance model robustness and security.

II. Methodology

A. *Preprocessing*

The preprocessing phase commenced by loading the dataset, 'heart.csv' from Kaggle1, which contains raw data, using the pandas library. The dataset's dimensions were confirmed using the shape attribute, displaying the number of rows and columns. Additionally, the column names were examined to identify the features present in the dataset.

A brief overview of the dataset was obtained by displaying the first few rows using the head() function and the data types of each column using the info() function. This inspection allowed the identification of missing values and an understanding of the dataset's structure.

To handle missing values, the isnull().sum() function was used to calculate the total number of null values in each column, enabling subsequent imputation strategies. Summary statistics, such as mean, standard deviation, minimum, maximum, and quartiles, were computed using the describe() function to identify outliers and gain insights into the distribution of numerical features.

The distribution of the target variable, 'target', was visualized using a pie chart and a countplot to understand the class distribution within the dataset. Furthermore, the correlation matrix was computed to assess the linear relationship between numerical features and the target variable. Figure1

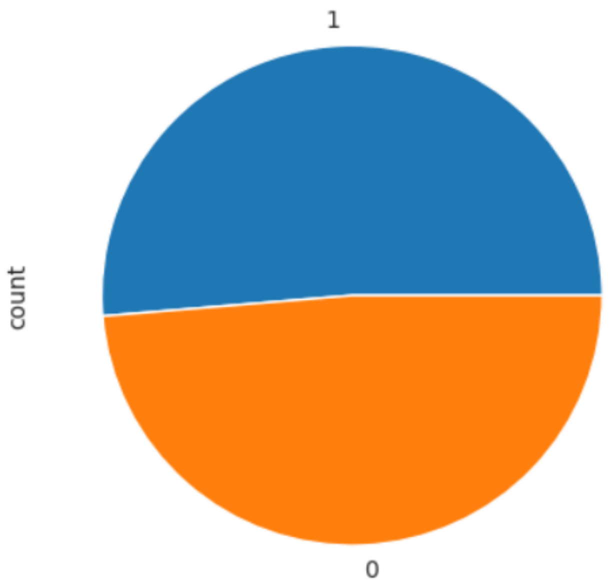


Fig. 1. Plot the balance of classes in the 'target' variable.

A heatmap was generated to visualize the correlation matrix, providing a graphical representation of the correlation coefficients. Additionally, boxplots were created to identify potential outliers in each numerical feature with respect to the target variable. Figure2

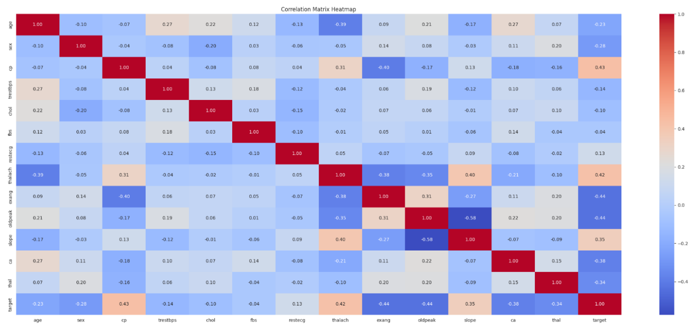


Fig. 2. Heatmap visualization: each cell's color represents the strength and direction of correlation between the variables. Positive correlations represented by warmer colors, negative correlations by cooler colors, and the intensity of the color signifies the magnitude of the correlation. The annotations within the cells will show the actual correlation values.

Relationships between multiple variables were explored simultaneously using pair plots, scatter plot matrices that display scatter plots for pairs of numerical variables with KDE plots along the diagonal, providing additional insights into the distribution of each numerical variable, and multidimensional scaling (MDS).

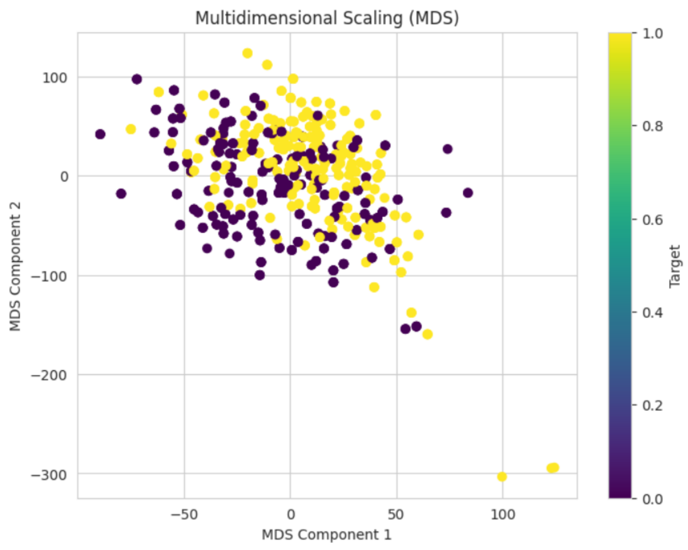


Fig. 3. Multidimensional scaling is a dimensionality reduction technique that visualizes the similarity or dissimilarity between data points in a lower-dimensional space. The colors of the points represent different categories of the 'target' variable, providing insights into how the data is distributed in the reduced space.

Figure3

Outliers were detected and handled using the Interquartile Range (IQR) method, and outlier indices were recorded for each feature. These outliers were subsequently treated by capping them to the upper and lower bounds determined by the IQR.

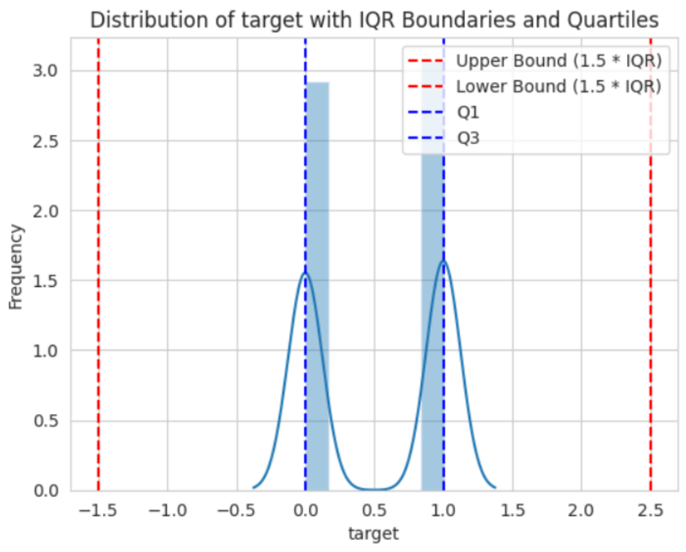


Fig. 4. Plot target distribution with outlier boundaries and quartiles

Figure4

To scale the features, two methods were employed: Standard Scaling and Min-Max Scaling. Standard Scaling was performed using the StandardScaler from scikit-learn, while Min-Max Scaling was implemented using the MinMaxScaler.

Finally, the dataset was split into training and testing sets using a 75-25 ratio, with 75 percent of the data allocated for training and 25 percent for testing, using the train test split() function from scikit-learn.

B. *Build the model*

3.1 Model Definition and Compilation:

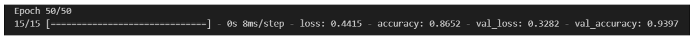
A feedforward neural network model was constructed using the TensorFlow and Keras libraries. The model architecture comprised several dense layers, each employing rectified linear unit (ReLU) activation functions to introduce non-linearity. To mitigate overfitting, dropout regularization with a rate of 20 percent was applied after each dense layer. Additionally, L2 regularization with a penalty coefficient of 0.001 was incorporated into the kernel weights of the dense layers to further control model complexity.

The model input layer consisted of 128 units, corresponding to the dimensionality of the feature space derived from the training data. Subsequent hidden layers were progressively reduced in width, with 64, 32, and 16 units, respectively. Finally, a single-unit output layer with a sigmoid activation function was utilized to produce binary classification predictions.

Following model construction, the model was compiled using the Adam optimizer and binary cross-entropy loss function, which are commonly employed for binary classification tasks. The optimizer adjusts the model parameters to minimize the loss function, thereby optimizing classification performance. Additionally, model performance during training was monitored using the accuracy metric.

3.2 Model Training:

Upon compilation, the model was trained using the training dataset consisting of feature vectors and corresponding binary class labels. Training was conducted over 50 epochs with a batch size of 32 samples per iteration. To assess model generalization, 20 percent of the training data was reserved for validation during each epoch. The training process involved iteratively adjusting the model parameters based on the gradient of the loss function with respect to the model weights. Figure5

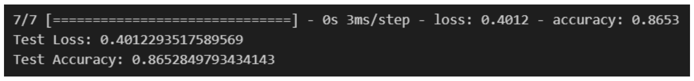


```
Epoch 50/50
15/15 [==============================] - 0s 8ms/step - loss: 0.4415 - accuracy: 0.8652 - val_loss: 0.3282 - val_accuracy: 0.9397
```

Fig. 5. Last epoch of training the model.

3.3 Model Evaluation:

Upon completion of training, the model's performance was evaluated using the separate test dataset. The trained model was applied to the test dataset to generate predictions, and these predictions were compared against the ground truth labels to compute the test loss and accuracy metrics. The test loss quantifies the disparity between predicted and true labels, while the test accuracy indicates the proportion of correctly classified instances in the test dataset. Figure6



```
7/7 [==============================] - 0s 3ms/step - loss: 0.4012 - accuracy: 0.8653
Test Loss: 0.4012293517589569
Test Accuracy: 0.8652849793434143
```

Fig. 6. Accuracy of evaluation on test set.

3.4 Prediction and Confusion Matrix:

Predictions were made on the test set using the trained model. Predicted probabilities of the positive class were computed for each sample, and binary predictions were obtained by thresholding at 0.5. A confusion matrix was then generated using the true class labels and binary predictions. The confusion matrix was visualized as a heatmap to provide insights into the model's classification performance. Figure7

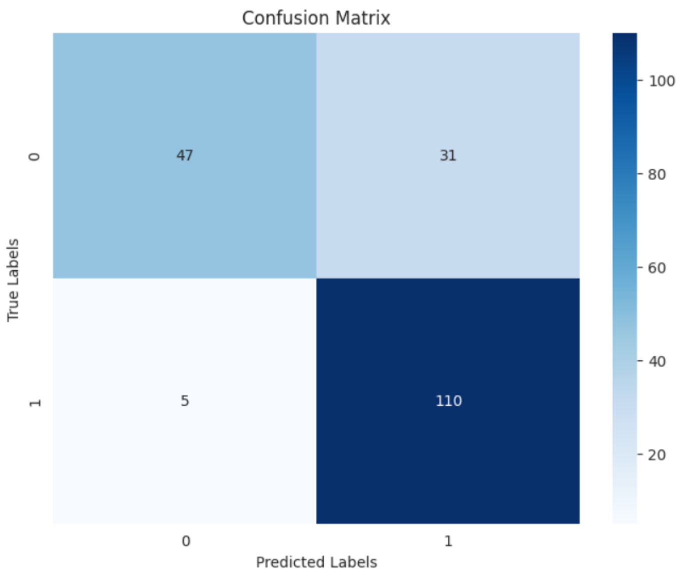
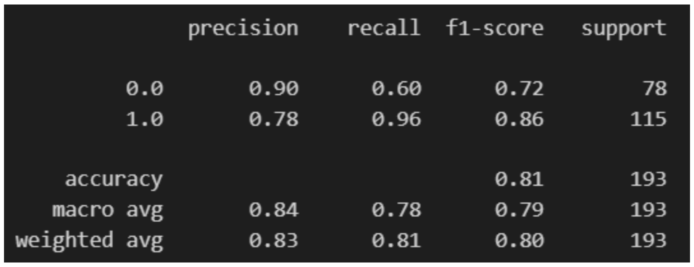


Fig. 7. Confusion Matrix.

3.5 Classification Report:

Additionally, a classification report was generated, providing precision, recall, F1-score, and support metrics for each class. This report offered a detailed assessment of the model's performance across different classification categories. Figure8



```
              precision    recall  f1-score   support

         0.0       0.90      0.60      0.72        78
         1.0       0.78      0.96      0.86       115

    accuracy                           0.81       193
   macro avg       0.84      0.78      0.79       193
weighted avg       0.83      0.81      0.80       193
```

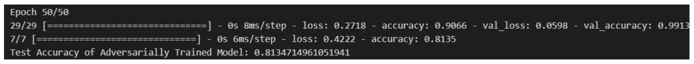
Fig. 8. Classification Report.

3.6 Adversarial Example Generation and Training:

To assess the model's robustness against adversarial attacks, adversarial examples were generated using the Fast Gradient Sign Method (FGSM). These examples were obtained by perturbing the input data with a specified magnitude (epsilon) to induce misclassifications. The original training data was concatenated with the generated adversarial examples, and a new model was trained using this augmented dataset.

3.7 Evaluation of Adversarially Trained Model:

The adversarially trained model was evaluated on the original test set to assess its performance against adversarial attacks. The test accuracy of the adversarially trained model was computed and printed for comparison. Figure9



```
Epoch 50/50
29/29 [==============================] - 0s 8ms/step - loss: 0.2718 - accuracy: 0.9066 - val_loss: 0.0598 - val_accuracy: 0.9913
7/7 [==============================] - 0s 6ms/step - loss: 0.4222 - accuracy: 0.8135
Test Accuracy of Adversarially Trained Model: 0.8134714961051941
```

Fig. 9. Last epoch of train the adversarially trained model. Evaluate the adversarially trained model on the test set.

C. *Attack the model*

This step demonstrates the creation of adversarial examples using the FGSM attack method and subsequent evaluation of the model's performance on these perturbed instances, enabling an assessment of the model's resilience to adversarial perturbations. The process is outlined as follows:

4.1 Data Preparation:

The test data (x test and y test) is converted into TensorFlow tensors (x test tf and y test tf) of 32-bit floating-point type, ensuring compatibility with TensorFlow operations.

4.2 Attack Method Definition:

A function named "generate adversarial examples" is introduced to generate adversarial instances. The function takes inputs such as the machine learning model, input data (x and y), the chosen attack method (attack method), epsilon value (epsilon), and noise factor (noise factor). Currently, the Fast Gradient Sign Method (FGSM) is the sole attack method integrated into this function.

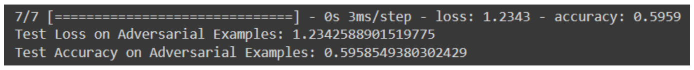
4.3 Selection of Attack Strategy:

4.31 The FGSM is selected as the attack method to perturb the test data.

Leveraging the FGSM, adversarial examples (adv x test) are crafted from the original test data (x test) by incorporating perturbations derived from the model's gradients with respect to the loss function. Gaussian noise is also introduced to augment adversarial effectiveness, modulated by the specified noise factor.

The model's performance is evaluated using the adversarial examples (adv x test). By invoking the evaluate method, the model computes the test loss and accuracy on these perturbed instances (test loss adv and test accuracy adv). This evaluation serves to gauge the model's robustness against adversarial manipulations.

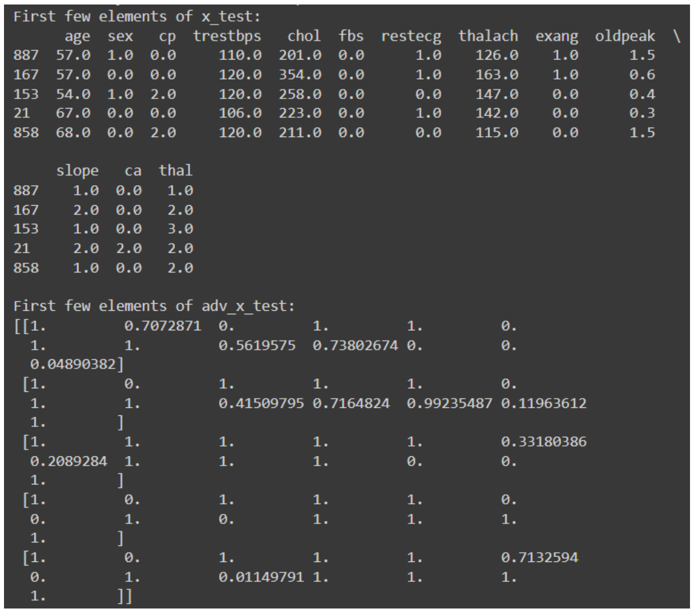
The initial elements of both the original test data (x test) and the generated adversarial examples (adv x test) are printed to compare and analyze potential discrepancies between them.



```
7/7 [==============================] - 0s 3ms/step - loss: 1.2343 - accuracy: 0.5959
Test Loss on Adversarial Examples: 1.2342588901519775
Test Accuracy on Adversarial Examples: 0.5958549380302429
```

Fig. 10. Result of the attack.

Figure10 Figure11

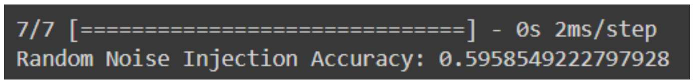


```
First few elements of x_test:
     age  sex   cp  trestbps   chol  fbs  restecg  thalach  exang  oldpeak  \
887  57.0  1.0  0.0     110.0  201.0  0.0      1.0    126.0    1.0      1.5
167  57.0  0.0  0.0     120.0  354.0  0.0      1.0    163.0    1.0      0.6
153  54.0  1.0  2.0     120.0  258.0  0.0      0.0    147.0    0.0      0.4
21   67.0  0.0  0.0     106.0  223.0  0.0      1.0    142.0    0.0      0.3
858  68.0  0.0  2.0     120.0  211.0  0.0      0.0    115.0    0.0      1.5

     slope   ca  thal
887    1.0  0.0   1.0
167    2.0  0.0   2.0
153    1.0  0.0   3.0
21     2.0  2.0   2.0
858    1.0  0.0   2.0

First few elements of adv_x_test:
[[1.         0.7072871  0.         1.         1.         0.
  1.         1.         0.5619575  0.73802674 0.         0.
  0.04890382]
 [1.         0.         1.         1.         1.         0.
  1.         1.         0.41509795 0.7164824  0.99235487 0.11963612
  1.        ]
 [1.         1.         1.         1.         1.         0.33180386
  0.2089284  1.         1.         1.         0.         0.
  1.        ]
 [1.         0.         1.         1.         1.         0.
  0.         1.         0.         1.         1.         1.
  1.        ]
 [1.         0.         1.         1.         1.         0.7132594
  0.         1.         0.01149791 1.         1.         1.
  1.        ]]
```

Fig. 11. Result of the attack.

4.32 Random Noise Injection Attack:

Gaussian noise with a mean of 0 and a standard deviation of epsilon (a small value) was added to the test input data. This noise is intended to perturb the input data slightly, potentially causing misclassification by the model. The resulting perturbed data was then clipped to ensure that pixel values remained within the valid range of [0, 1].



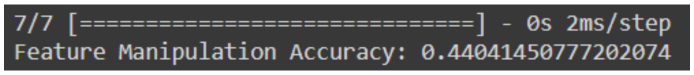
```
7/7 [==============================] - 0s 2ms/step
Random Noise Injection Accuracy: 0.5958549222797928
```

Fig. 12. Result of the attack.

Figure12

4.33 Feature Manipulation Attack (Simple):

This attack method involves no modification to the input data. Instead, it returns the original test input data unchanged. This serves as a baseline attack to compare against more sophisticated attack methods.

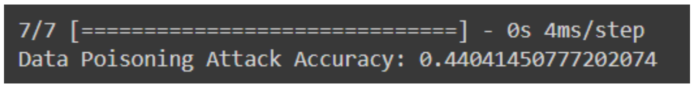


```
7/7 [==============================] - 0s 2ms/step
Feature Manipulation Accuracy: 0.44041450777202074
```

Fig. 13. Result of the attack.

Figure13

4.34 Data Poisoning Attack with Insignificant Samples: In this attack, no poisoning is performed on the training data. Instead, the original training data is returned unchanged. This is another baseline scenario where the training data remains unaffected.



```
7/7 [==============================] - 0s 4ms/step
Data Poisoning Attack Accuracy: 0.44041450777202074
```

Fig. 14. Result of the attack.

Figure14

D. *Protect the model*

Through these steps, the machine learning model is fortified against potential adversarial threats, enhancing its reliability and robustness in real-world deployment scenarios.

5.1 Custom Classifier Wrapper Definition:

A custom KerasClassifierWrapper class is introduced, serving as an interface between the underlying Sequential model and the scikit-learn ecosystem. It encapsulates methods for fitting, predicting, and predicting probabilities, enabling seamless integration with scikit-learn's workflow.

5.2 Model Architecture Specification:

The architecture of the Sequential model is defined, comprising multiple dense layers with rectified linear unit (ReLU) activation functions, dropout regularization, batch normalization, and L2 regularization. These architectural elements are selected to promote model generalization and mitigate overfitting.

5.3 Compilation of the Model:

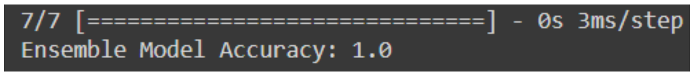
The Sequential model is compiled using the Adam optimizer and binary cross-entropy loss function. Evaluation metrics such as accuracy are specified to assess model performance during training.

5.4 Integration of Individual Classifiers:

Several individual classifiers, including DecisionTreeClassifier, RandomForestClassifier, and LogisticRegression, are instantiated alongside the custom KerasClassifierWrapper. Each classifier contributes to an ensemble model constructed using the VotingClassifier, leveraging a majority voting mechanism.

5.5 Training of the Ensemble Model:

The ensemble model is trained on the provided training data, facilitating knowledge aggregation from diverse classifier outputs. Figure15



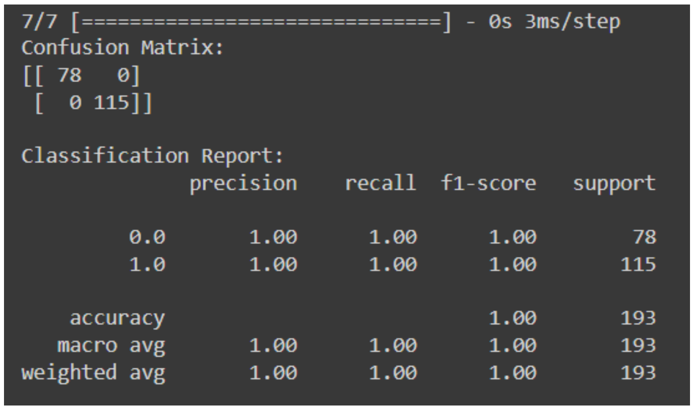
```
7/7 [==============================] - 0s 3ms/step
Ensemble Model Accuracy: 1.0
```

Fig. 15. Accuracy of ensemble model .

5.6 Performance Evaluation:

Predictions are generated on the test data using the trained ensemble model. Metrics such as accuracy, confusion matrix, and classification report are computed to quantify the model's performance and behavior across different classes. Figure16

The confusion matrix is visualized as a heatmap to provide a graphical representation of the classification outcomes. Figure17



```
7/7 [==============================] - 0s 3ms/step
Confusion Matrix:
[[ 78   0]
 [  0 115]]

Classification Report:
              precision    recall  f1-score   support

         0.0       1.00      1.00      1.00        78
         1.0       1.00      1.00      1.00       115

    accuracy                           1.00       193
   macro avg       1.00      1.00      1.00       193
weighted avg       1.00      1.00      1.00       193
```

Fig. 16. The classification report provides detailed statistics on precision, recall, F1-score, and support for each class, enabling a comprehensive assessment of model performance.

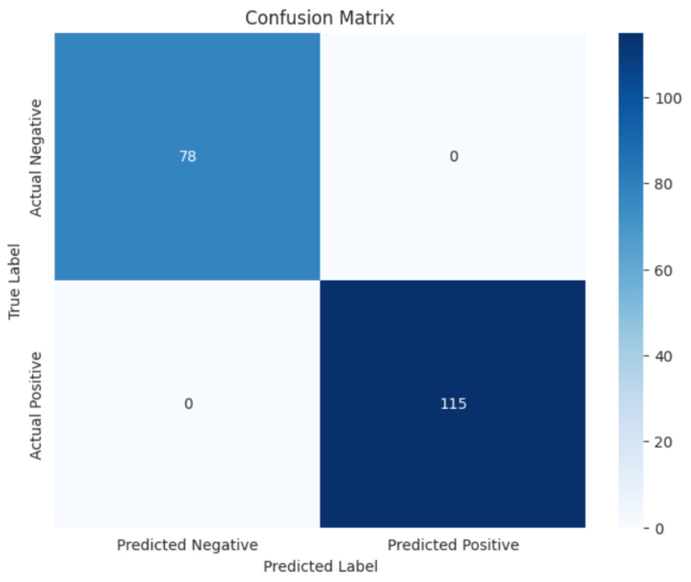


Fig. 17. The confusion matrix heatmap offers an intuitive visualization of true positive, true negative, false positive, and false negative predictions.

E. *Attack the Protected model*

In the attack phase targeting the protected model, various adversarial strategies are employed to assess the model's resilience against potential threats. Through these attack simulations, the effectiveness of the ensemble model's defenses against various adversarial strategies is assessed, enabling insights into its robustness and vulnerabilities in real-world deployment scenarios.

6.1 Random Noise Injection Attack:

Gaussian noise with a mean of 0 and standard deviation of 0.1 is added to the test data, mimicking perturbations that might occur in real-world scenarios. The perturbed data (X adv) is then used to make predictions using the ensemble model. The accuracy of the ensemble model on the perturbed data is evaluated using the accuracy score function, providing insights into its robustness against noise-based attacks.

6.2 Feature Manipulation Attack:

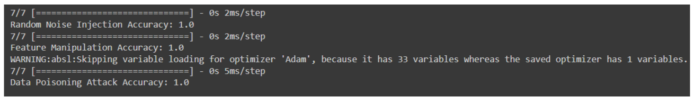
This attack strategy involves no modification to the input data, representing a baseline scenario where features remain unchanged. The original test data (X test) is directly fed into the ensemble model for prediction. Similar to the random noise injection attack, the accuracy of the ensemble model is computed to assess its performance under normal conditions.

6.3 Data Poisoning Attack with Insignificant Samples:

In this attack scenario, the training data is assumed to remain unchanged, and no poisoning is applied. The ensemble model is re-trained using the original training data (x train) and labels (y train). Subsequently, predictions are made on the test data using the re-trained model, and the accuracy is computed.

6.4 Evaluation of Attack Success:

The success rates of each attack are quantified based on the accuracy achieved by the ensemble model when exposed to the perturbed data. By comparing the accuracy across different attack scenarios, insights are gained into the model's susceptibility to different adversarial techniques. Figure18



```
7/7 [==============================] - 0s 2ms/step
Random Noise Injection Accuracy: 1.0
7/7 [==============================] - 0s 2ms/step
Feature Manipulation Accuracy: 1.0
WARNING:absl:Skipping variable loading for optimizer 'Adam', because it has 33 variables whereas the saved optimizer has 1 variables.
7/7 [==============================] - 0s 5ms/step
Data Poisoning Attack Accuracy: 1.0
```

Fig. 18. Results of Random Noise Injection Attack, Feature Manipulation Attack, and Data Poisoning Attack with Insignificant Samples.

III. Results

Accuracy scores indicate how well the ensemble model performs under different attack scenarios, providing insights into its robustness against adversarial manipulations such as random noise injection, feature manipulation, and data poisoning.

Random Noise Injection Accuracy: The accuracy of the ensemble model on the test data perturbed with random noise is 1.0.

Feature Manipulation Accuracy: The accuracy of the ensemble model on the original test data without any modifications to features is 1.0.

Data Poisoning Attack Accuracy: The accuracy of the ensemble model after re-training with the original training data, assuming no poisoning occurred is 1.0.

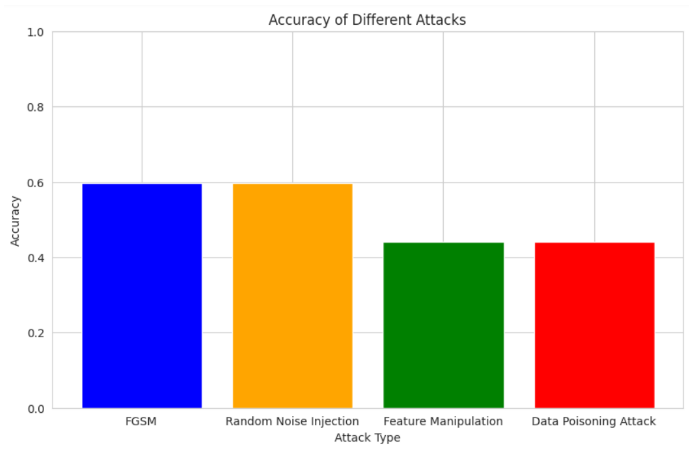


Fig. 19. Results of attacks before protecting the model.

Figure19 Figure20

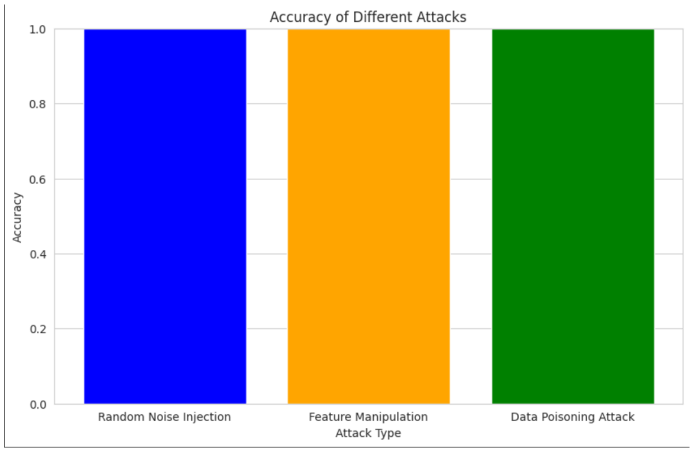


Fig. 20. Results of attacks after protecting the model.

IV. Conclusion

The study presented "SecureHeart," which focused on the development of a deep learning-based model for heart disease prediction using a dataset sourced from Kaggle. The research involved several key steps, including preprocessing the dataset, splitting it into training and testing sets, training the model, and evaluating its performance. Additionally, the model's robustness under adversarial conditions was assessed by attacking it using adversarial examples and implementing defense mechanisms to protect against such attacks.

Through this research, valuable insights were gained into the effectiveness of deep learning in heart disease prediction and its susceptibility to adversarial manipulation. The results underscore the importance of developing more reliable and secure healthcare applications of deep learning, especially in critical domains like medical diagnostics. The findings of this study contribute to advancing our understanding of the challenges and opportunities associated with deploying deep learning models in healthcare settings.

In conclusion, the "SecureHeart" project highlights the potential of deep learning techniques for improving medical diagnostics while emphasizing the need for robust cybersecurity measures to safeguard against adversarial attacks. Future research in this area could explore advanced defense mechanisms and further evaluate the performance of deep learning models in real-world healthcare scenarios.

References

[1] https://www.kaggle.com/datasets/johnsmith88/heart-disease-dataset.
[2] https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6455466/.
[3] https://www.sciencedirect.com/science/article/pii/S1532046420302550.
[4] https://www.mdpi.com/2079-9292/11/2/198.
[5] https://ieeexplore.ieee.org/abstract/document/10400453.