Outlier Detection with Autoencoder Ensembles

Mauro Gizdulić,[1] Jan Šubarić[2] and Stjepan Turk[3]

Abstract— **This paper focuses on the implementation of Randomize Neural Network for outlier detection. Outlier in data analysis and machine learning can have a significant impact on statistical measures, data modeling and decision-making processes. Therefore, by identifying and distinguishing these unusual observations from the rest of the data set, we can ensure the validity of the data set. RandNet combines the flexibility of neural networks with the robustness of ensemble techniques to identify outliers within datasets. In this paper we explore the construction and training of ensemble components which are individual autoencoders. Furthermore, we combine multiple autoencoders to get more robust way to identify outliers within datasets. Through extensive experiments on benchmark datasets, the effectiveness of RandNet in outlier detection is evaluated. The results demonstrate the promising performance of RandNet in accurately identifying outliers and highlight its potential for practical outlier detection applications.**

I. INTRODUCTION

The detection of outliers, data points that significantly deviate from the majority of the dataset is a critical task in data analysis and has implications across various domains. Outliers can distort statistical analyses, affect the accuracy of predictive models and provide valuable insights into unique patterns or abnormalities in the data.

Numerous techniques have been proposed for scoring data points for outlierness. These include distance-based methods [2], statistical approaches [3], density-based methods [4], and ensemble-based techniques [5]. Each method has its strengths and limitations, and the choice of technique depends on the characteristics of the data and the specific requirements of the application. In our study, we have adopted a method based on neural networks for outlier detection. The utilization of neural network-based approaches for outlier detection has been explored in previous research papers [6], [7]. These papers provide valuable insights into the application of neural networks in outlier detection tasks and demonstrate the potential of this approach in identifying anomalous patterns in various domains.

While deep neural networks are known for their effectiveness in learning from large datasets, their performance on smaller datasets is questionable due to the risk of overfitting caused by a high number of parameters. Training such networks often leads to convergence to local optima, limiting their generalizability. Increasing the dataset size can help alleviate overfitting, but it may also introduce computational challenges, as larger datasets require more computational resources and time for training. Therefore, striking a balance between dataset size and model complexity is crucial to ensure effective learning without sacrificing computational feasibility.

Our approach utilizes autoencoders as neural networks for outlier detection. However, instead of using fully connected autoencoders, we employ randomly connected autoencoders with varying structures and connection densities. This modification reduces the computational complexity associated with fully connected architectures. Furthermore, to reduce the risk of overfitting inherent in the autoencoder structure, we adopt an ensemble learning method. This involves utilizing multiple autoencoders within the ensemble, allowing for improved generalization and robustness in outlier detection. By combining the advantages of randomization and ensemble learning, we aim to enhance the performance and reliability of our outlier detection model based on autoencoders. This approach is commonly referred to as Randomized Neural Network [1].

II. METHODOLOGY

In order to accomplish our objective of outlier detection, the first step was to carefully select the datasets that we would utilize for our analysis. We took into consideration various factors such as dataset size, diversity, and relevance to the problem domain. Subsequently, we focused on preparing the selected datasets for further analysis. The details of the data preparation process will be elaborated upon in the subsequent section of the paper.

Once the dataset was prepared, our next step involved constructing a suitable autoencoder model. In this regard, we opted to leverage the capabilities of the TensorFlow Keras library, which offers a robust and versatile class and methods for autoencoder development. The RandAE class provides a comprehensive set of tools and functionalities for building and training randomized autoencoder models tailored to our specific requirements. By utilizing this well-established class, we were able to streamline the implementation process and benefit from the extensive features and optimizations provided by the TensorFlow Keras framework. This allowed us to focus more on fine-tuning the model architecture and parameters to achieve optimal performance in outlier detection tasks.The architecture of our autoencoder ensemble is shown in the figure 1 below.

[1], M. Gizdulić is with Faculty of Computer Science, Sveučilište u Rijeci, Hrvatska `m.gizdulic at riteh.hr`

[2], J. Šubarić is with Faculty of Computer Science, Sveučilište u Rijeci, Hrvatska `j.subaric at riteh.hr`

[3], S. Turk is with Faculty of Computer Science, Sveučilište u Rijeci, Hrvatska `s.turk at riteh.hr`

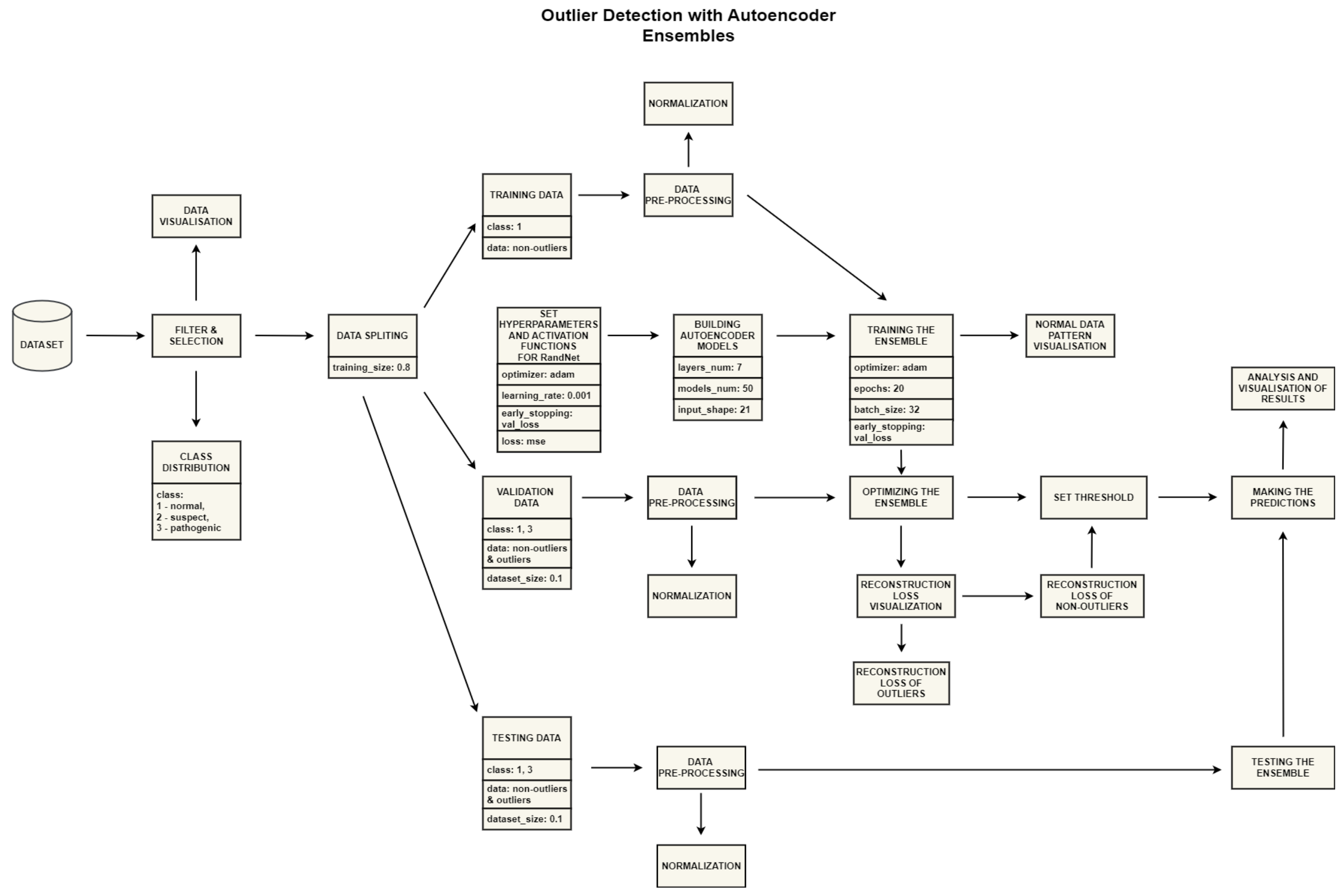


Fig. 1. **The architecture of the autoencoder ensemble**

After conducting numerous experiments and iterations, we determined that a neural network architecture with 7 hidden layers would be most suitable for our outlier detection task. In this architecture, each layer, from the initial layer to the bottleneck layer, is designed to have 4 fewer dimensions than the previous layer. In terms of the activation functions used, we employed the Sigmoid activation function for the input and output layer, while the ReLU (Rectified Linear Unit) activation function was utilized for the remaining hidden layers. In addition, we incorporated the convenience of the built-in function *EarlyStopping* into our model. This function plays a crucial role in preventing overfitting by monitoring the validation error during training. If no significant improvement is observed in the validation error over a specified number of consecutive epochs, the training process is automatically halted. This early stopping mechanism serves as a valuable safeguard, ensuring that the model does not continue to train unnecessarily and potentially overfit the data. The ensemble is made out of 50 randomized autoencoder models.

III. CASE STUDY

In the scientific article [1] which we were given for research and making a project, several different datasets were mentioned. From several mentioned datasets we chose one for our project and that is CTG or in longer terms Cardiotocography. CTG is a continuous recording of the fetal heart rate obtained with an ultrasound transducer placed on the mother's abdomen. It is also important to mention that it is widely used in pregnancy. The mentioned dataset contains 2126 (rows) measurements and classifications of fetal heart rate (FHR) signals and 42 (columns) measurement parameters. The mentioned parameters can be divided into two groups. First group contains basic data like filename and classification and it is shown in table I and second group also called Features contains the data necessary for the execution of this project and it is shown in table II.

The most important data is shown in table II. It contains 21 Feature with an NSP at the end of table. NSP represents the following three classes: Normal, Suspect and Pathologic. Class Normal represents data which is normal or data with normal measurements. Class Suspect represents data that slightly deviates from Normal data and for the purpose of our project we dropped the specified data. Class Pathologic represents data that is significantly deviating from Normal data and the specified data is also called outlier.

A. *Datasets*

Before data splitting, we edited the dataset according to the needs of the project. We checked first if column NSP

Filename	Name of CTG examination
Date	Date of CTG examination
b	Start instant
e	End instant
LBE	Baseline value (medical expert)
A	Calm sleep
B	REM sleep
C	Calm vigilance
D	Active vigilance
SH	Shift pattern (A or Susp with shifts)
AD	Accelerative/Decelerative pattern (stress situation)
DE	Decelerative pattern (vagal stimulation)
LD	Largely decelerative pattern
FS	Flat-Sinusoidal pattern (Pathological state)
SUSP	Suspect pattern
CLASS	Class code (1 to 10) for classes A to SUSP
NSP	Normal = 1 Suspect = 2 Pathologic = 3

TABLE I
BASIC DATA

LB	Baseline value (SisPorto)
AC	Accelerations (SisPorto)
FM	Fetal movement (SisPorto)
UC	Uterine contractions (SisPorto)
DL	Light decelerations
DS	Severe decelerations
DP	Prolongued decelerations
ASTV	Percentage of time with abnormal short term variability (SisPorto)
MSTV	Mean value of short term variability (SisPorto)
ALTV	Percentage of time with abnormal long term variability (SisPorto)
MLTV	Mean value of long term variability (SisPorto)
Width	Histogram width
Min	Low freq. of the histogram
Max	High freq. of the histogram
Nmax	Number of histogram peaks
Nzeros	Number of histogram zeros
Mode	Histogram mode
Mean	Histogram mean
Median	Histogram median
Variance	Histogram variance
Tendency	Left asymmetric = -1 Symmetric = 0 Right asymmetric = 1
NSP	Normal = 1 Suspect = 2 Pathologic = 3

TABLE II
FEATURES

contains null values. In this case there was no null values. So the next step was to show NSP distribution and it is shown in figure 2. We can conclude from figure that from maximum data of 2126, we have 1655 normal data and about 176 outlier data.

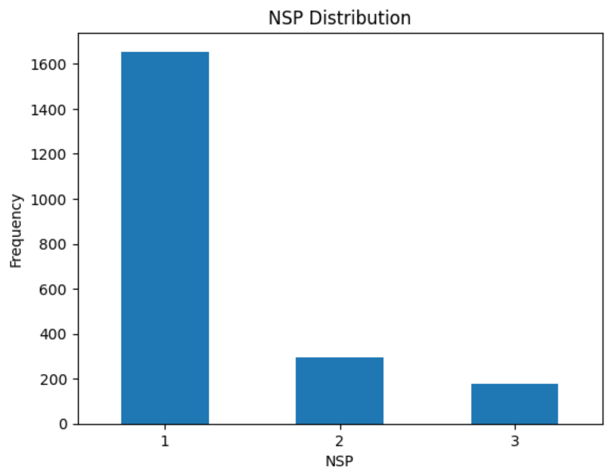


Fig. 2. **NSP Distribution**

Since only two classes (Normal and Pathologic) are needed for our project, we decided to drop the data that was represented with class Suspect. After droping the data, for easier data use, we represent normal data with value 1 and outlier data with value 0.

B. Data splitting

First we import sklearn.model_selection library to use train_test_split() function. The specified function receives the following data as parameters: data, labels, train_size and random_state. Parameter labels contains just NSP values and those are 0 and 1. Parameter data contains all values except NSP values. Train size can accept the value between 0 and 1 and random state represent the random number generator. In our case train size contains value 0.8 which means that 80% of data is using for training and 20% of data is remaining for optimizing and testing the model. That 20% of data we will divide into two equal groups: validation data and testing data. The validation set is used to optimize model hyperparameters and determine the threshold, while the test set is used to check accuracy/correctness of the ensemble. In the code below we can see how it is performed.

```
# Just values, without column name
raw_data = df.values

# Values of NSP columns (0,1)
labels = raw_data[:, -1]
```

```
# All other values
data = raw_data[:, 0:-1]

X_train, X_rem, y_train, y_rem = /
train_test_split(data, labels, /
train_size=0.8, random_state = 21)

# Now since we want the valid and test /
size to be equal /
(10% each of overall data).
# We have to define valid_size=0.5 /
(that is 50% of remaining data)

X_valid, X_test, y_valid, y_test = /
train_test_split(X_rem,y_rem, /
test_size=0.5)
```

C. Normalization

To overcome the issue of model learning, we implemented data normalization. We utilized the MinMaxScaler from the sklearn.preprocessing library for this purpose. Firstly, we fitted the scaler to the training data using the MinMaxScaler().fit() function. The function was given the parameter X_train, which represents our training data. Subsequently, we applied the scaler to normalize all three datasets: X_train, X_test, and X_valid. The code snippet demonstrating the performed operations is displayed below.

```
from sklearn.preprocessing /
import MinMaxScaler

# fit scaler on training data
norm = MinMaxScaler().fit(X_train)

# transform training data
X_train = norm.transform(X_train)
X_test = norm.transform(X_test)
X_valid = norm.transform(X_valid)

train_labels = y_train.astype(bool)
test_labels = y_test.astype(bool)

normal_train_data = X_train[train_labels]
normal_test_data = X_test[test_labels]

anomalous_train_data = /
X_train[~train_labels]

anomalous_test_data = /
X_test[~test_labels]
```

Variables train_labels, test_labels, normal_train_data, normal_test_data, anomalous_train_data, anomalous_test_data are used for plotting normal and outlier data. Specified plots are visible in section IV-A.

D. Model training

Our training model is previously mentioned RandAE(). Randnet autoencoder is a type of symmetrical neural network with the encoder and decoder part. Model receives three parameters: input layer, hidden layers with number of nodes and drop ratio. The input and output layers have the same number of nodes. The middle hidden layer between the encoder and decoder has the smallest number of nodes and because of that it is called the Bottleneck. In general, the bottleneck layer constrains the amount of information that goes through our autoencoder. While training our model, we use the EarlyStopping() function to stop training when the monitored metric stops improving. Also for our training we use Adam's optimizer with a learning rate of 0.001. The reconstruction loss is shown as MSE (Mean Squared Error). We train our model on normal_train_data with 100 epochs and batch size of 32.

E. Ensemble training

After optimizing the single model and achieving the best results, we constructed an ensemble consisting of 50 randomized autoencoder models. Each individual model was trained using training data with an input shape of 21 nodes, 7 hidden layers, 100 epochs, and a dropout ratio of 0.33. The dropout ratio refers to the percentage of randomly selected neurons that are ignored during the training phase. To prevent overfitting, we also implemented an early stopping function to halt training. For optimization, we utilized the Adam optimizer with a learning rate of 0.001 and MSE for calculating the reconstruction loss. The entire training process for the ensemble lasted over 2 hours.

F. Evaluation

The metrics used for model training were MSE (Mean Squared Error) and validation loss. For evaluation, classification report metrics such as precision, recall, accuracy, and F1 score were used to test the ensemble. MSE is utilized to measure the difference between the original and generated data, known as the reconstruction loss. Validation loss is a metric used to halt model training and prevent overfitting. Precision measures the accuracy of positive predictions, while recall measures the completeness of positive predictions. Precision is often seen as a measure of quality, while recall is a measure of quantity. Accuracy is a metric used at the end of testing to describe how well the ensemble performs on both non-outliers and outliers. The validation dataset is employed to optimize model hyperparameters and determine the optimal threshold. The reconstruction loss for a single data point is determined by the median score of reconstruction losses across all ensemble components. The threshold is determined using the numpy function np.quantile(), which takes an array of median losses and the mean score of validation labels.

IV. RESULTS

A. *Normal and outlier data plot*

Normal and outlier data plots are shown in two graphs. First graph shows example of normal data and the second graph shows example of anomalous data. The X-axis represents the Features and the Y-axis represents the Features values in both graphs. Displayed values on the Y-axis are between 0 and 1 because normal and anomalous data are normalized. Both graphs are shown in figures 3 and 7.

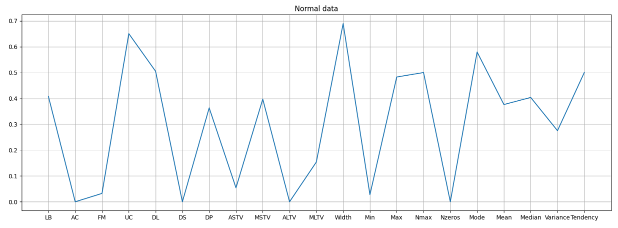


Fig. 3. **Normal data**

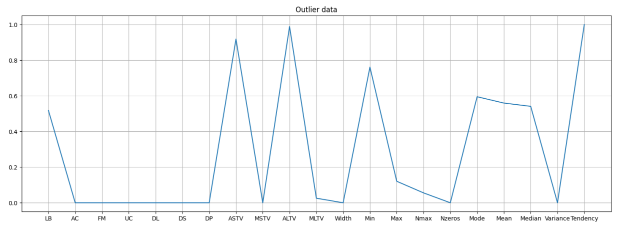


Fig. 4. **Outlier data**

Figures 5 and 6 provide visual representations of the data reconstruction outcomes. In the first graph, represented by the red line, we observe the reconstruction of a normal data point, while in the second graph, we see the reconstruction of an outlier data point. Notably, the reconstruction of normal data demonstrates a higher level of accuracy compared to that of outlier data. This distinction in reconstruction quality emphasizes the success of our approach in capturing and reproducing the patterns present in the normal data while highlighting the challenges faced when dealing with outlier data.

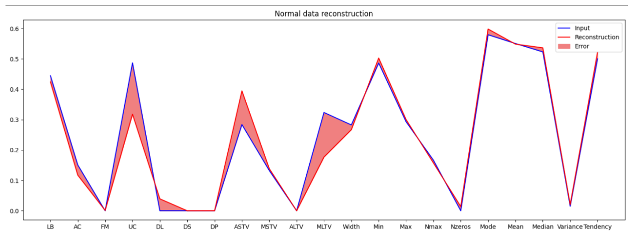


Fig. 5. **Normal data reconstruction**

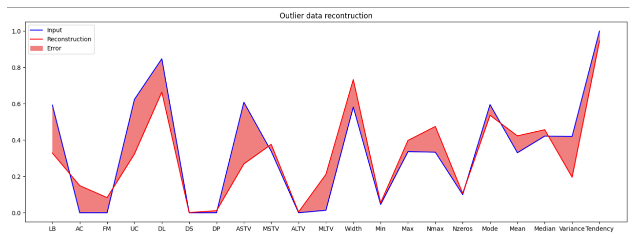


Fig. 6. **Outlier data reconstruction**

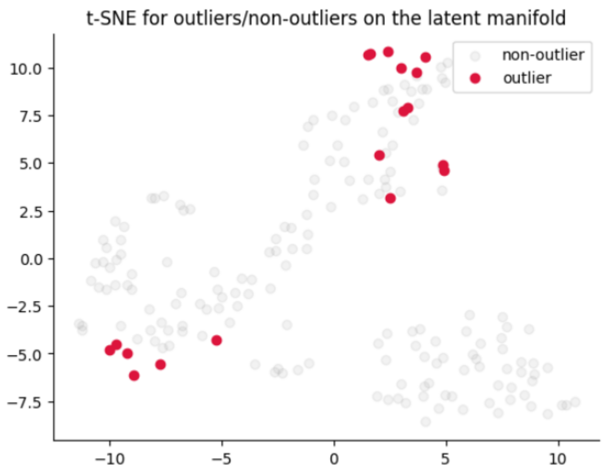
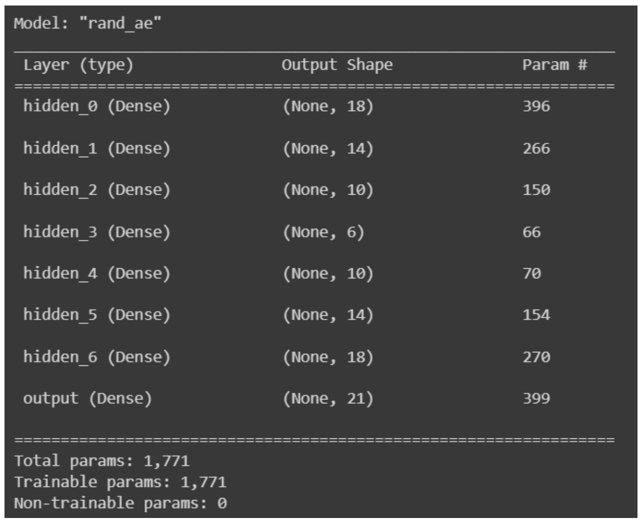


Fig. 7. **Outliers and normal data on the latent manifold**

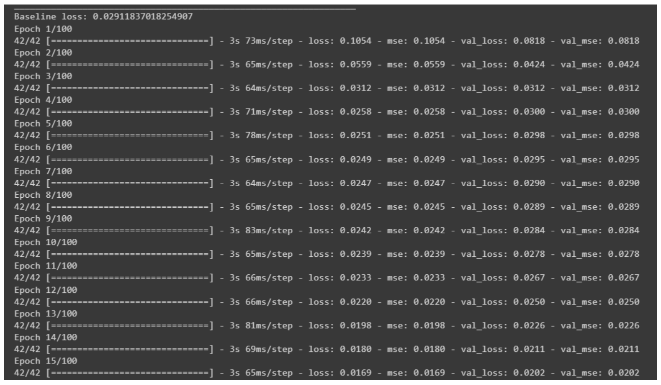
B. *Single model training results*

The RandAE model contains 7 hidden layers with the following node numbers: 18, 14, 10, 6, 10, 14, 18. The 4th layer is our bottleneck because it contains the smallest number of nodes. The first and last layers contain 21 nodes each. The drop ratio is equal to 0. After executing the single model training, the results are as follows.



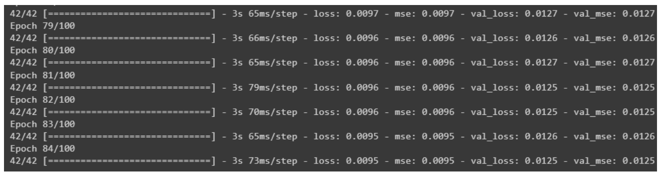
```
Model: "rand_ae"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 hidden_0 (Dense)            (None, 18)                396

 hidden_1 (Dense)            (None, 14)                266

 hidden_2 (Dense)            (None, 10)                150

 hidden_3 (Dense)            (None, 6)                 66

 hidden_4 (Dense)            (None, 10)                70

 hidden_5 (Dense)            (None, 14)                154

 hidden_6 (Dense)            (None, 18)                270

 output (Dense)              (None, 21)                399

=================================================================
Total params: 1,771
Trainable params: 1,771
Non-trainable params: 0
```

Fig. 8. **Model summary**



```
Baseline loss: 0.02911837018254907
Epoch 1/100
42/42 [==============================] - 3s 73ms/step - loss: 0.1054 - mse: 0.1054 - val_loss: 0.0818 - val_mse: 0.0818
Epoch 2/100
42/42 [==============================] - 3s 65ms/step - loss: 0.0559 - mse: 0.0559 - val_loss: 0.0424 - val_mse: 0.0424
Epoch 3/100
42/42 [==============================] - 3s 64ms/step - loss: 0.0312 - mse: 0.0312 - val_loss: 0.0312 - val_mse: 0.0312
Epoch 4/100
42/42 [==============================] - 3s 71ms/step - loss: 0.0258 - mse: 0.0258 - val_loss: 0.0300 - val_mse: 0.0300
Epoch 5/100
42/42 [==============================] - 3s 78ms/step - loss: 0.0251 - mse: 0.0251 - val_loss: 0.0298 - val_mse: 0.0298
Epoch 6/100
42/42 [==============================] - 3s 65ms/step - loss: 0.0249 - mse: 0.0249 - val_loss: 0.0295 - val_mse: 0.0295
Epoch 7/100
42/42 [==============================] - 3s 64ms/step - loss: 0.0247 - mse: 0.0247 - val_loss: 0.0290 - val_mse: 0.0290
Epoch 8/100
42/42 [==============================] - 3s 65ms/step - loss: 0.0245 - mse: 0.0245 - val_loss: 0.0289 - val_mse: 0.0289
Epoch 9/100
42/42 [==============================] - 3s 83ms/step - loss: 0.0242 - mse: 0.0242 - val_loss: 0.0284 - val_mse: 0.0284
Epoch 10/100
42/42 [==============================] - 3s 65ms/step - loss: 0.0239 - mse: 0.0239 - val_loss: 0.0278 - val_mse: 0.0278
Epoch 11/100
42/42 [==============================] - 3s 66ms/step - loss: 0.0233 - mse: 0.0233 - val_loss: 0.0267 - val_mse: 0.0267
Epoch 12/100
42/42 [==============================] - 3s 66ms/step - loss: 0.0220 - mse: 0.0220 - val_loss: 0.0250 - val_mse: 0.0250
Epoch 13/100
42/42 [==============================] - 3s 81ms/step - loss: 0.0198 - mse: 0.0198 - val_loss: 0.0226 - val_mse: 0.0226
Epoch 14/100
42/42 [==============================] - 3s 69ms/step - loss: 0.0180 - mse: 0.0180 - val_loss: 0.0211 - val_mse: 0.0211
Epoch 15/100
42/42 [==============================] - 3s 65ms/step - loss: 0.0169 - mse: 0.0169 - val_loss: 0.0202 - val_mse: 0.0202
```

Fig. 9. **Training results 1**



```
42/42 [==============================] - 3s 65ms/step - loss: 0.0097 - mse: 0.0097 - val_loss: 0.0127 - val_mse: 0.0127
Epoch 79/100
42/42 [==============================] - 3s 66ms/step - loss: 0.0096 - mse: 0.0096 - val_loss: 0.0126 - val_mse: 0.0126
Epoch 80/100
42/42 [==============================] - 3s 65ms/step - loss: 0.0096 - mse: 0.0096 - val_loss: 0.0127 - val_mse: 0.0127
Epoch 81/100
42/42 [==============================] - 3s 79ms/step - loss: 0.0096 - mse: 0.0096 - val_loss: 0.0125 - val_mse: 0.0125
Epoch 82/100
42/42 [==============================] - 3s 70ms/step - loss: 0.0096 - mse: 0.0096 - val_loss: 0.0125 - val_mse: 0.0125
Epoch 83/100
42/42 [==============================] - 3s 65ms/step - loss: 0.0095 - mse: 0.0095 - val_loss: 0.0126 - val_mse: 0.0126
Epoch 84/100
42/42 [==============================] - 3s 73ms/step - loss: 0.0095 - mse: 0.0095 - val_loss: 0.0125 - val_mse: 0.0125
```

Fig. 10. **Training results 2**

The figure 8 shows the first result. It shows the model name, hidden layers, number of nodes in hidden layers, number of total and trainable parameters. On figures 9 and 10 it is shown baseline loss, executing of every epoch with training and validation loss. Also, both losses are shown on figure 11. The x-axis shows the number of epochs, while the y-axis shows the loss. Based on the displayed figure, it is evident that the loss of validation is slightly higher than the loss of training, and that by increasing the number of epochs, both graphs approach each other, which means that it is an adequate model.

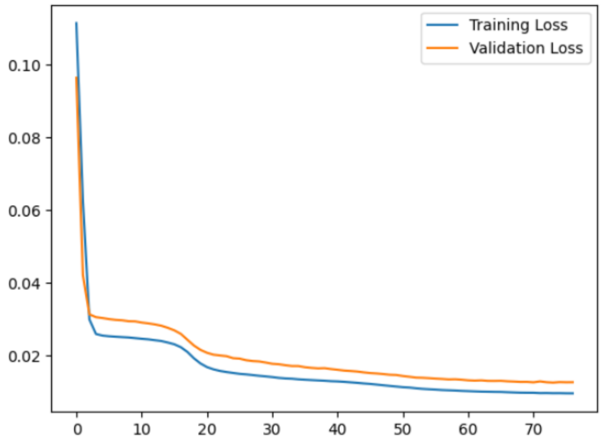


Fig. 11. **Training results 3**

Figure 12 depicts the training reconstruction loss, showcasing the disparity between the original data and the predicted outputs generated by the model during the training phase. On the contrary, Figure 13 showcases the validation reconstruction loss, accompanied by a threshold line to determine the boundary for acceptable performance.

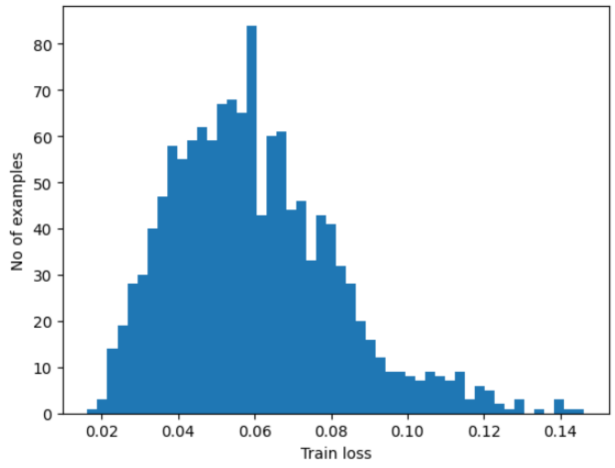


Fig. 12. **Training reconstruction loss**

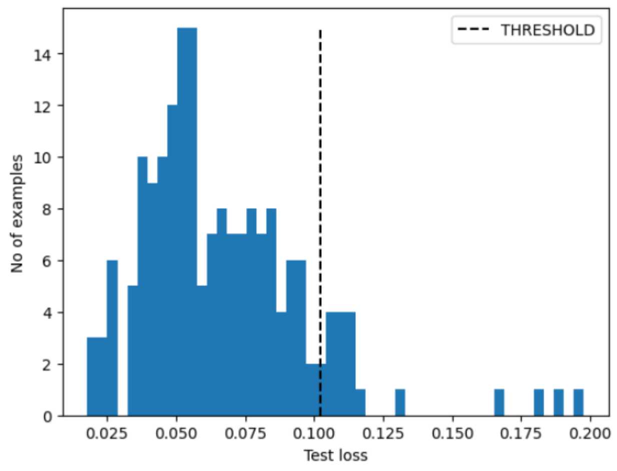
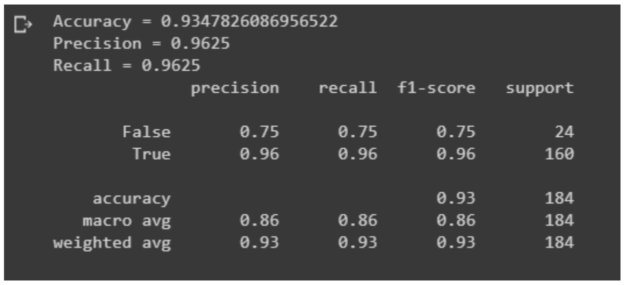


Fig. 13. **Validation reconstruction loss with threshold**

C. Single model testing results

The single model demonstrated impressive performance in outlier detection, achieving an accuracy of 0.935, indicating a substantial proportion of correctly classified instances. Furthermore, the precision value of 0.962 highlights the model's ability to accurately predict outliers among the total predicted outliers. Additionally, with a recall of 0.962, the model successfully captured a large proportion of the actual outliers, underscoring its effectiveness in outlier identification. These results emphasize the model's high accuracy and its capability to reliably detect outliers. ROC curve analysis resulted in an AUC value of 0.856 which is shown in the figure 16.



```
Accuracy = 0.9347826086956522
Precision = 0.9625
Recall = 0.9625
              precision    recall  f1-score   support

       False       0.75      0.75      0.75        24
        True       0.96      0.96      0.96       160

    accuracy                           0.93       184
   macro avg       0.86      0.86      0.86       184
weighted avg       0.93      0.93      0.93       184
```

Fig. 14. **Testing results - single model**

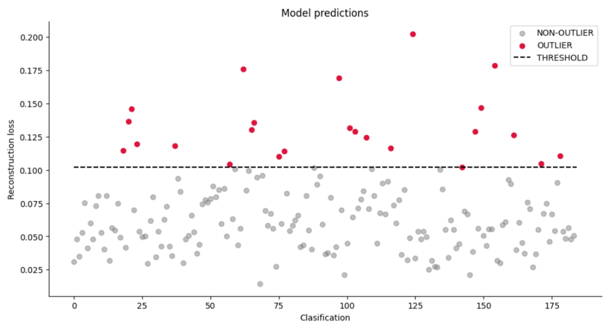


Fig. 15. **Data points with threshold - single model**

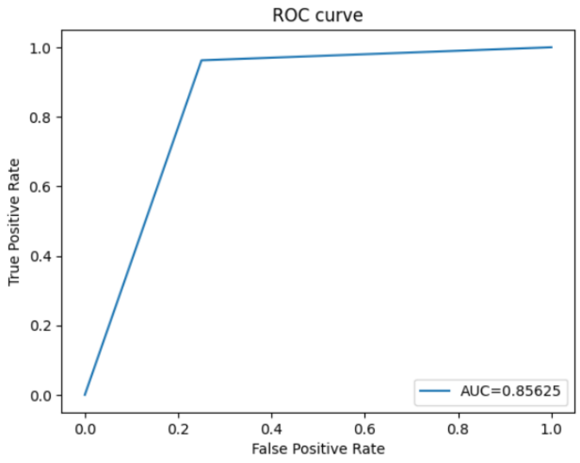
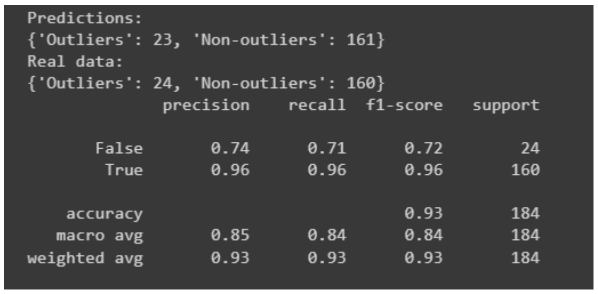


Fig. 16. **ROC curve - single model**

D. *Ensemble results*

The ensemble approach yielded even more impressive results in outlier detection. It achieved an accuracy of 0.930, indicating a high proportion of correctly classified instances. The precision score of 0.961 further demonstrates the model's ability to accurately predict outliers among the total predicted outliers. Additionally, with a recall of 0.964, the ensemble approach successfully captured a significant proportion of the actual outliers. These results indicate that the ensemble approach outperformed the single model in terms of accuracy, precision, and recall, showcasing its superior performance in outlier detection.



```
Predictions:
{'Outliers': 23, 'Non-outliers': 161}
Real data:
{'Outliers': 24, 'Non-outliers': 160}
              precision    recall  f1-score   support

       False       0.74      0.71      0.72        24
        True       0.96      0.96      0.96       160

    accuracy                           0.93       184
   macro avg       0.85      0.84      0.84       184
weighted avg       0.93      0.93      0.93       184
```

Fig. 17. **Testing results - ensemble**

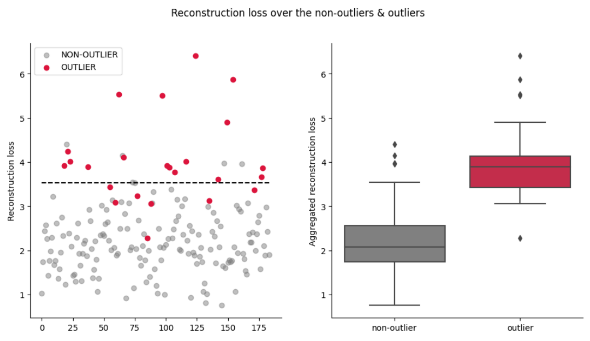


Fig. 18. **Data points with threshold - ensemble**

To evaluate the performance of our outlier detection models, we utilized the Receiver Operating Characteristic (ROC) curve and calculated the Area Under the Curve metric. ROC curve analysis resulted in an AUC value of 0.835, indicating the model's good performance in distinguishing between normal and anomalous instances. Figure 20 displays the ROC curve.

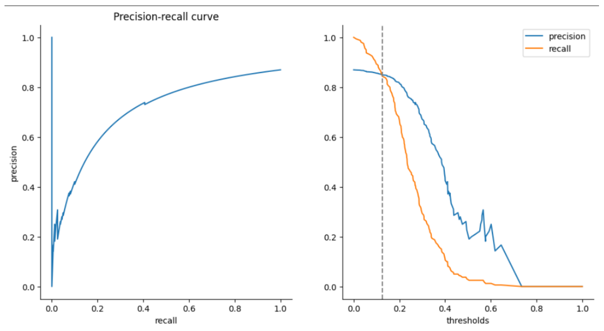


Fig. 19. **Precision-recall curve**

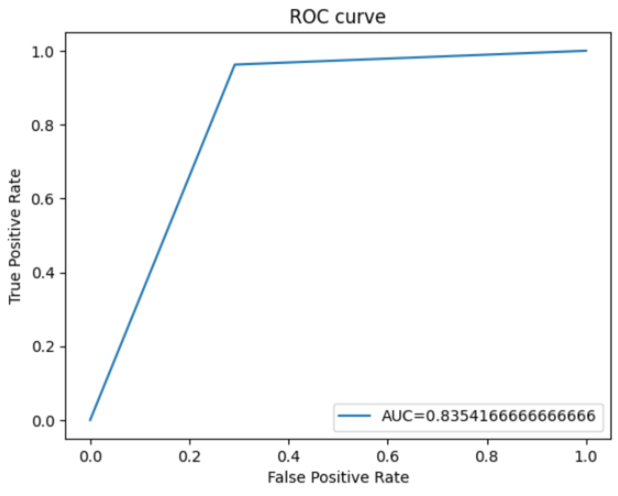


Fig. 20. **ROC curve - ensemble**

V. DISCUSSION

The model exhibited strong performance in capturing the patterns and structures present in the training data, as evident from the convergence of the training and validation losses. This convergence signifies the model's effective learning and reproduction of the data's characteristics.

In our analysis, we successfully compared a single autoencoder model with an ensemble approach for outlier detection. Both methods demonstrated proficiency in identifying outliers, but the ensemble approach outperformed the single autoencoder by leveraging the collective wisdom of multiple models. The ensemble method's superior performance was evident in its ability to achieve more accurate threshold selection.

To establish the threshold for evaluating the outlierness of data, we utilized the validation data and computed the mean of the validation loss augmented by 1.2 times the standard deviation. By doing so, we established a margin for identifying substantial deviations from the norm, enabling us to effectively distinguish outliers from normal data points. On the other hand, in the ensemble, the threshold is determined using np.quantile with median reconstruction loss. On average, it took approximately 2 minutes to train a single autoencoder model, while the entire training process for the ensemble lasted over 2 hours.

However, we must acknowledge potential exceptions and limitations to our conclusions. Factors such as data quality, model architecture, and the representativeness of the validation data can influence the outlier detection methods' performance and generalizability. Additionally, selecting the optimal threshold may vary based on the specific application and the trade-offs between false positives and false negatives.

VI. CONCLUSION

In conclusion, the primary objective of this study was to develop a Randomized Neural Network approach for outlier detection using autoencoders. We successfully achieved this goal by implementing a customized autoencoder model architecture with 7 hidden layers, gradually reducing the dimensions in each layer. We utilized the Sigmoid activation function for the input and output layer, and the ReLU activation function for the intermediate hidden layers. Through rigorous experimentation and optimization, we demonstrated the effectiveness of our approach in detecting outliers in various datasets.

Our results indicate that the Randomized Neural Network approach, coupled with ensemble learning techniques, provides robust outlier detection capabilities. By leveraging the power of autoencoders and the randomization of ensemble components, we achieved improved generalization and reduced overfitting, resulting in more reliable outlier detection.

As for future steps, further research could explore the impact of different activation functions, network architectures, and ensemble configurations on outlier detection performance. Additionally, investigating the applicability of our approach to real-world datasets from different domains would be valuable. Lastly, exploring methods to enhance the interpretability and explainability of the outlier detection results could provide additional insights for practical applications.

REFERENCES

[1] J. Chen, S. Sathe, C. Aggarwal, D. Turaga. Outlier Detection with Autoencoder Ensembles. 2017.
[2] F. Angiulli, C. Pizzuti. Fast outlier detection in high dimensional spaces, PKDD, 2002.
[3] E. Knorr, and R. Ng. Algorithms for Mining Distance-based Outliers in Large Datasets. VLDB, 1998.
[4] M. Breunig, H.-P. Kriegel, R. Ng, J. Sander. LOF: Identifying Density-based Local Outliers, SIGMOD, 2000.
[5] M. Shyu, S. Chen, K. Sarinnapakorn, L. Chang. A novel anomaly detection scheme based on principal component classifier. ICDMW, 2003.
[6] S. Hawkins, H. He, G. Williams, R. Baxter. Outlier detection using replicator neural networks. In DaWaK, pages 170–180. Springer, 2002.
[7] A. Abhaya, B. K. Patra, An efficient method for autoencoder based outlier detection
[8] M. Markou, S. Singh. Novelty detection: a review: part 2: neural network based approaches. Signal processing, 83(12), 2003.
[9] University of California, Cardiotocography dataset: archive.ics.uci.edu/dataset/193/cardiotocography
[10] Keras Github, github.com/keras-team/keras
[11] TesorFlow, https://www.tensorflow.org/tutorials
[12] https://github.com/danieltsoukup/autoencoders/blob/master/outlier_detection_with_autoe
[13] https://medium.com/analytics-vidhya/anomaly-detection-in-cardio-dataset-using-deep-learning-technique-autoencoder-fd24ca9e5c69