Credit Card Fraud Detection

Anomaly detection of fraudulent transactions

Anthea Silvia Sasdelli
Course of Artificial Intelligence in Industry

Project development

1. Dataset analysis

2. GMM

3. Autoencoder

Dataset analysis

Credit Card Fraud Detection dataset from kaggle

The dataset was taken from Kaggle and is composed by :

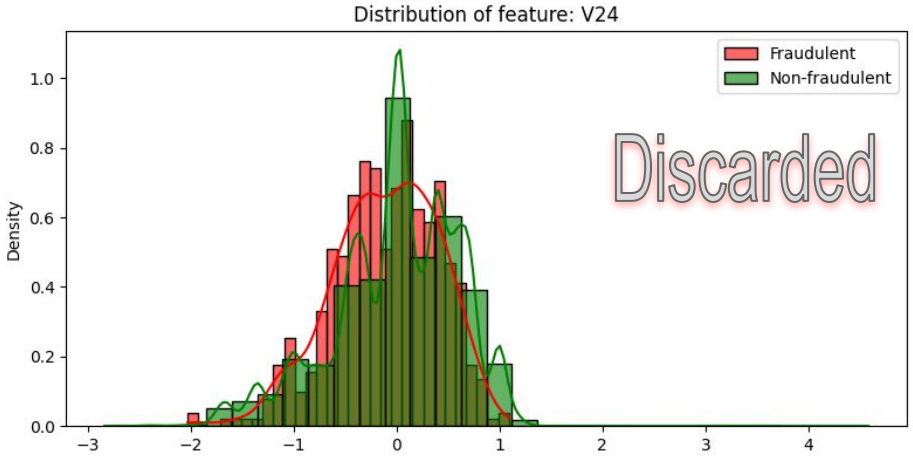
- 284807 transaction made by credit cards in September 2013 by European cardholders.
- Each transaction contains:
  - **The time**: number of seconds elapsed between each transaction
  - **Features V1 to V28**: Principal components derived from PCA
  - **Amount**: The monetary value of each transaction.
  - **Class**: The response variable indicating whether the transaction is fraudulent (1) or legitimate (0).

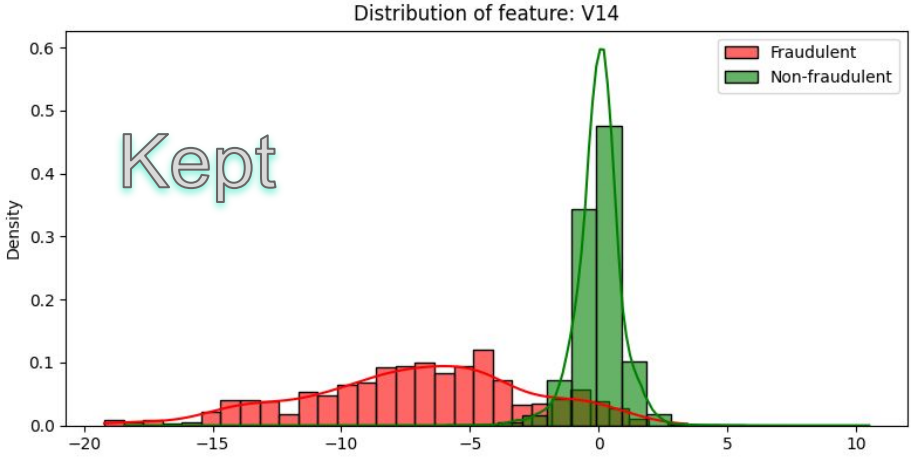
	Time	V1	V2	V3	V4	V5	...	V28	Amount	Class
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	...	-0.021053	149.62	0
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	...	0.014724	2.69	0
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	...	-0.059752	378.66	0
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	...	0.061458	123.50	0
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	...	0.215153	69.99	0

Dataset analysis

Dataset simplification and data processing

- The distributions of the features are then plotted to see how they were distributed. Since some of them had huge overlaps and unclear distinction between the two classes, the dataset has been simplified.





Dataset analysis

Dataset simplification and data processing

- The distributions of the features are then plotted to see how they were distributed. Since some of them had huge overlaps and unclear distinction between the two classes, the dataset has been simplified.

- It was used Random Forest to find the best features.

	Class	V10	V12	V14	V16	V17
0	0	0.090794	-0.617801	-0.311169	-0.470401	0.207971
1	0	-0.166974	1.065235	-0.143772	0.463917	-0.114805
2	0	0.207643	0.066084	-0.165946	-2.890083	1.109969
3	0	-0.054952	0.178228	-0.287924	-1.059647	-0.684093
4	0	0.753074	0.538196	-1.119670	-0.451449	-0.237033

Dataset analysis

Dataset simplification and data processing

- The dataset is then **normalized**, an operation recommended for GMM but essential for Autoencoder.

	V10	V12	V14	V16	V17	Class
0	0.510600	0.680908	0.635591	0.434392	0.737173	0
1	0.505267	0.744342	0.641219	0.464105	0.727794	0
2	0.513018	0.706683	0.640473	0.357443	0.763381	0
3	0.507585	0.710910	0.636372	0.415653	0.711253	0
4	0.524303	0.724477	0.608406	0.434995	0.724243	0

- The dataset is then divided in :
  - *normal*: every normal transaction (Class == 0)
  - *fraud*: every fraudulent transaction (Class == 1)

Dataset analysis

Dataset simplification and data processing

- Upon further analysis of the dataset, a strong imbalance between the elements of the two classes was noted.

```
The initial normal dataset has this shape: (284315, 6)
The fraudulent dataset has this shape: (492, 6)
```

- Because of this it was decided to divide the normal dataset in two:
  - the test dataset, that contains a ratio of 3:1 (three normal transaction each fraudulent one)
  - the train dataset, that contains the rest of the normal transactions

  In this way it was possible to create a balanced normal test dataset.

Dataset analysis

Dataset split

The train dataset thas this shape: (282839, 5)

The test dataset thas this shape: (1968, 5)

GMM

What are the Gaussian Mixture Models

- A Gaussian mixture model is a probabilistic model that assumes all the data points are generated from a mixture of a finite number of Gaussian distributions with unknown parameters. One can think of mixture models as generalizing k-means clustering to incorporate information about the covariance structure of the data as well as the centers of the latent Gaussians.

- For this project it has been used the class of scikit-learn : ***sklearn.mixture.GaussianMixture***

```
# Fit the GMM
gmm = mixture.GaussianMixture()
gmm.fit(X_train)
```

```
GaussianMixture
GaussianMixture()
```

GMM

Pipeline

- The model was trained with the normal X_train, which doesn't contain any fraudulent transaction (**UNSUPERVISED LEARNING**)

- Then the log-probabilities are calculated on the train split.
  They indicate how well each point fits the distribution learned from the model.

- The threshold was calculated by checking which percentile maximized the F1 score on the test set. Specifically, for each percentile, it was computed a threshold based on the GMM score distribution for the training data.

GMM

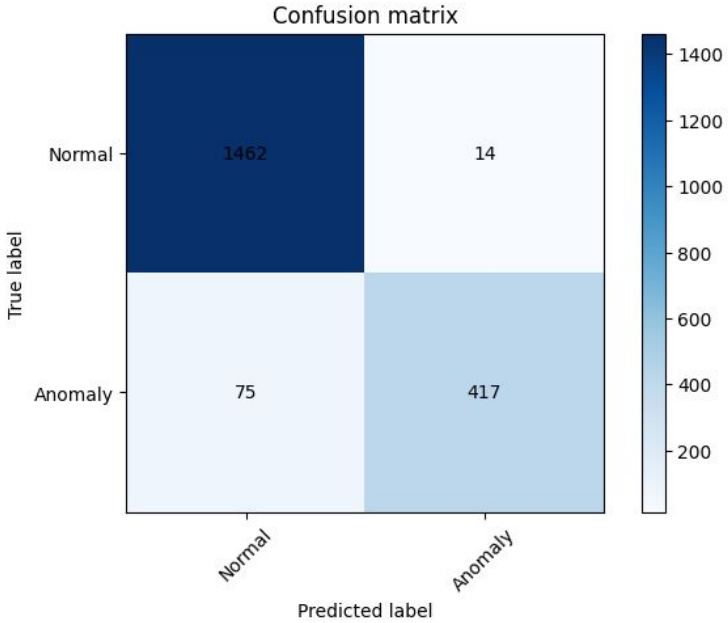
Results

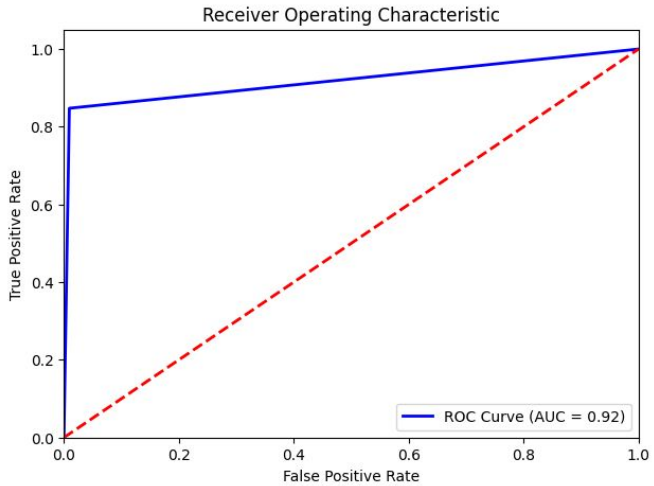
	Precision	Recall	F1-score	Support
Normal	0.95	0.99	0.97	1476
Anomaly	0.97	0.85	0.90	492
Accuracy	-	-	0.95	1968
Macro avg	0.96	0.92	0.94	1968
Weighted avg	0.96	0.95	0.95	1968

Accuracy : 0.9547764

GMM

Results

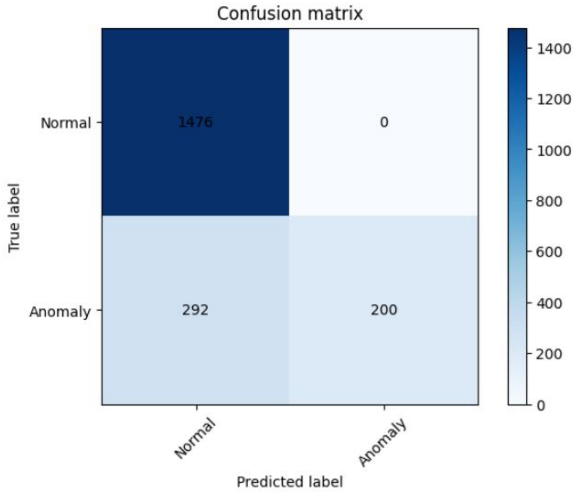

Confusion matrix
Normal
Anomaly
1462
14
75
417
True label
Predicted label
Normal
Anomaly
1400
1200
1000
800
600
400
200


Receiver Operating Characteristic
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0
ROC Curve (AUC = 0.92)

Autoencoder

More data preprocessing

- Another modification had to be made to the train set as it was too high in relation to the number of fraudulent examples in the test set.
  In fact, as can be seen in the diagram below, the model was unable to identify the anomalies.



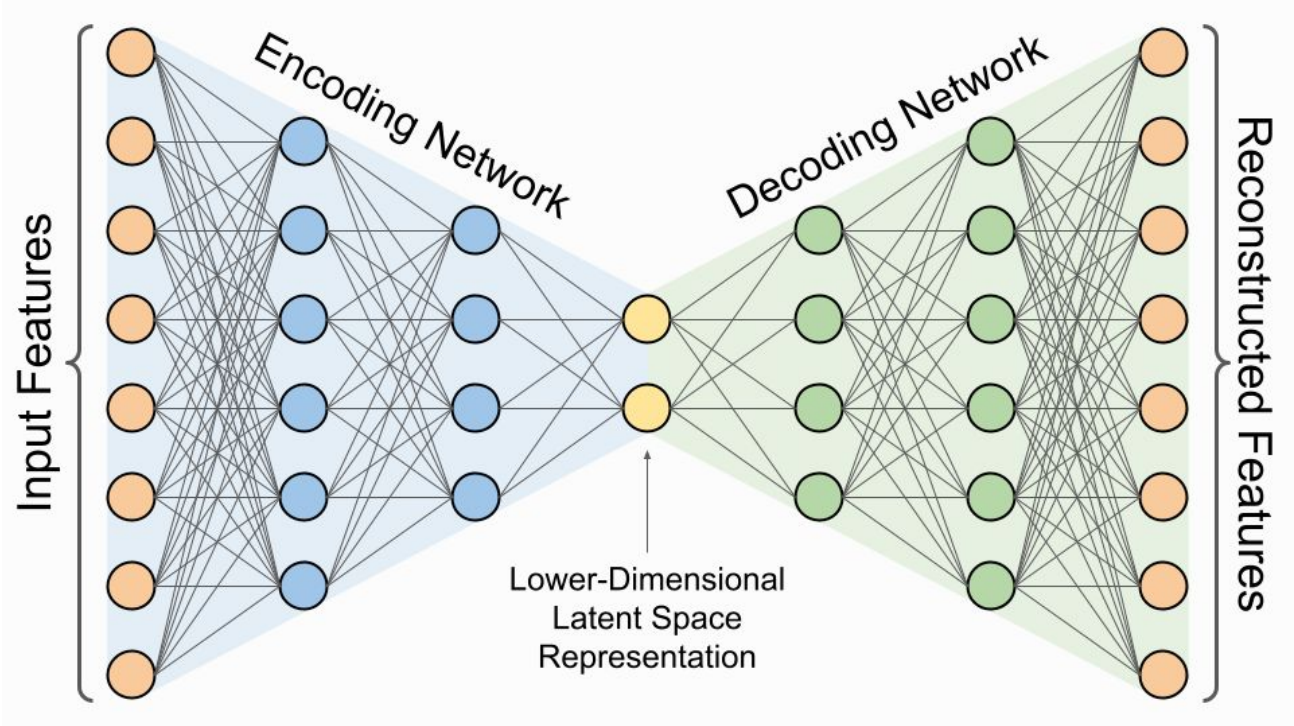
	precision	recall	f1-score	support
Normal	0.83	1.00	0.91	1476
Anomaly	1.00	0.41	0.58	492
accuracy			0.85	1968
macro avg	0.92	0.70	0.74	1968
weighted avg	0.88	0.85	0.83	1968

Accuracy: 0.8516260162601627

It was decided to put a ratio of 10:1 in the train set.

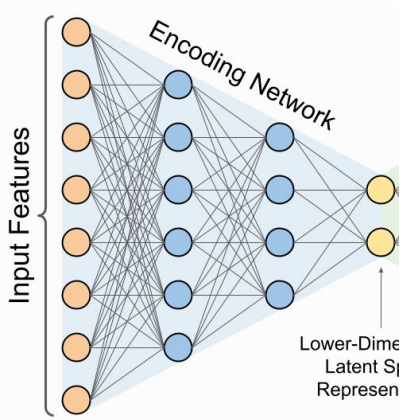
Autoencoder

What is an autoencoder?



Autoencoder

Encoder

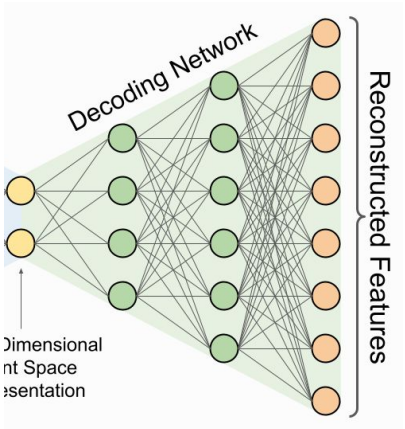



```
input_data = Input(shape=(input_dim,), name='encoder_input')

# Hidden layers
#encoder = Dense(units= 125, activation= 'tanh', name='encoder_1')(input_data)
#encoder = Dropout(.1)(encoder)
encoder = Dense(64,activation='tanh', name='encoder_2')(input_data)
encoder = Dropout(.1)(encoder)
encoder = Dense(48,activation='tanh', name='encoder_3')(encoder)
encoder = Dropout(.1)(encoder)
encoder = Dense(16,activation='tanh', name='encoder_4')(encoder)
encoder = Dropout(.1)(encoder)

# Bottleneck layer
latent_encoding = Dense(latent_dim, activation='linear', name='l
```

Layer (type)
encoder_input (InputLayer)
encoder_2 (Dense)
dropout_3 (Dropout)
encoder_3 (Dense)
dropout_4 (Dropout)
encoder_4 (Dense)
dropout_5 (Dropout)
latent_encoding (Dense)

Autoencoder

Decoder

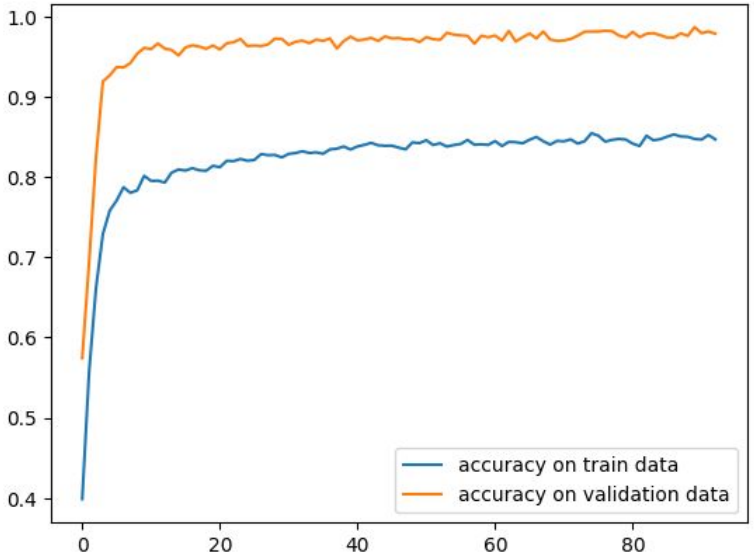


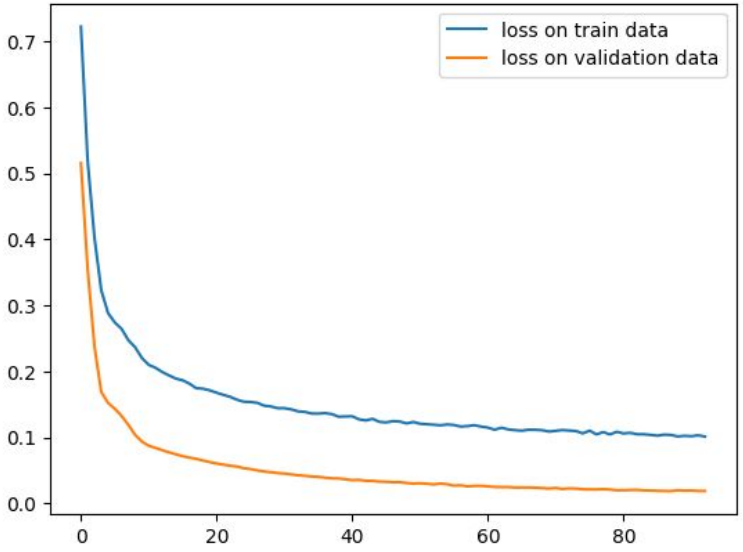

```
# The decoder network is a mirror image of the encoder network.
decoder = Dense(16, activation='tanh', name='decoder_1')(latent_encoding)
decoder = Dropout(.1)(decoder)
decoder = Dense(48, activation='tanh', name='decoder_2')(decoder)
decoder = Dropout(.1)(decoder)
decoder = Dense(64, activation='tanh', name='decoder_3')(decoder)
decoder = Dropout(.1)(decoder)
#decoder = Dense(125, activation='tanh', name='decoder_4')(decoder)
#decoder = Dropout(.1)(decoder)

# The output is the same dimension as the input data we are reconstructing.
reconstructed_data = Dense(units = input_dim, activation='linear', name='rec
```

Layer (type)
encoder_input (InputLayer)
encoder_2 (Dense)
dropout_3 (Dropout)
encoder_3 (Dense)
dropout_4 (Dropout)
encoder_4 (Dense)
dropout_5 (Dropout)
latent_encoding (Dense)
decoder_1 (Dense)
dropout_6 (Dropout)
decoder_2 (Dense)
dropout_7 (Dropout)
decoder_3 (Dense)
dropout_8 (Dropout)
reconstructed_data (Dense)

Autoencoder

Some graph of the training





Autoencoder

Pipeline

- The autoencoder model was trained on the normal data from X_train, containing no anomalies (**UNSUPERVISED LEARNING**).
- Next, the model reconstructs X_test_combined, and reconstruction scores (mean squared errors) are calculated. These scores indicate how well each sample is represented by the model, with higher scores suggesting potential anomalies.
- The threshold was chosen to maximize the F1 score on the test set. Specifically, it was set based on the point where the difference between the True Positive Rate (TPR) and False Positive Rate (FPR) was greatest.

Autoencoder

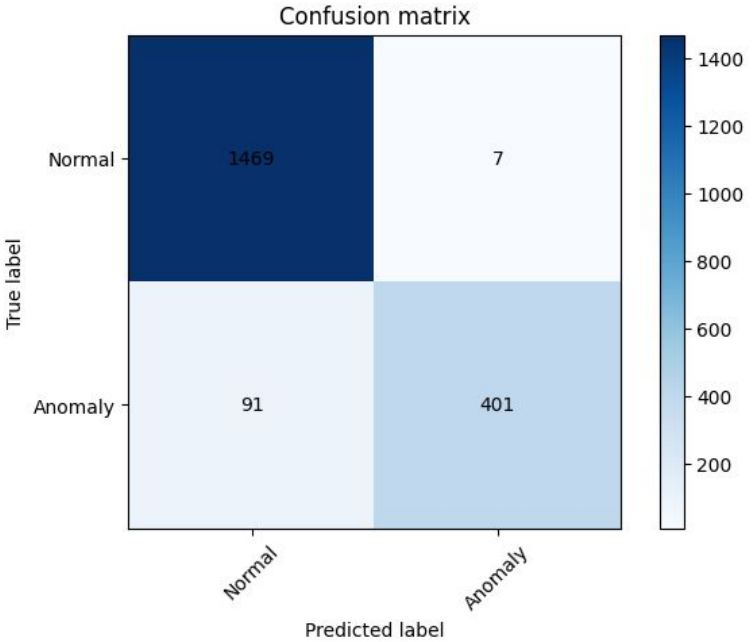
Results

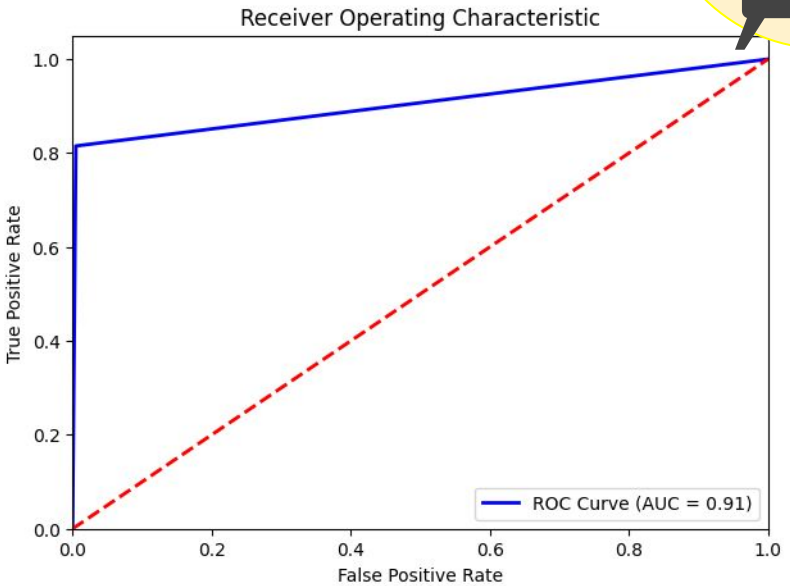
	Precision	Recall	F1-score	Support
Normal	0.94	1.00	0.97	1476
Anomaly	0.98	0.82	0.89	492
Accuracy	-	-	0.95	1968
Macro avg	0.96	0.91	0.93	1968
Weighted avg	0.95	0.95	0.95	1968

Accuracy : 0.950203252

Autoencoder

Results





Conclusions

- The **GMM** demonstrated a strong capability for anomaly detection, achieving an average accuracy of 95.5% on the test data. This model effectively captured the data distribution, allowing it to detect anomalies reliably. Notably, GMM also showed a superior trade-off between false positives and false negatives compared to the Autoencoder, which is essential for minimizing costly false alarms in fraud detection.

- On the other hand, the **Autoencoder** achieved a slightly lower accuracy of 95%. While its architecture offers flexibility, the model's complexity did not translate to higher accuracy. Additionally, the Autoencoder's false positive-to-false negative ratio was less balanced than GMM's, impacting its overall reliability in this context.

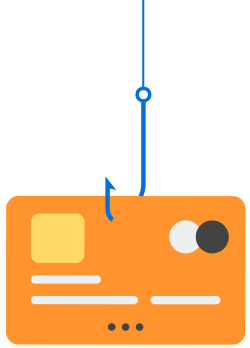
Overall, the results suggest that the **GMM** s a slightly more effective choice for fraud detection in this context, providing a better balance between accuracy and the cost of misclassifications. This makes it a robust option when prioritizing both detection precision and manageable false alarm rates.

Further improvements

- **Dataset managing** : as has been pointed out previously the dataset has been simplified, both in the choice of components to be used and in the rationalization (caused by the imbalance of the two classes). So a possible development might be:
  - to increase the number of components of the simplified dataset, if not use it in its entirety;
  - to use synthetic data (*data augmentation*) to create more fraudulent data .

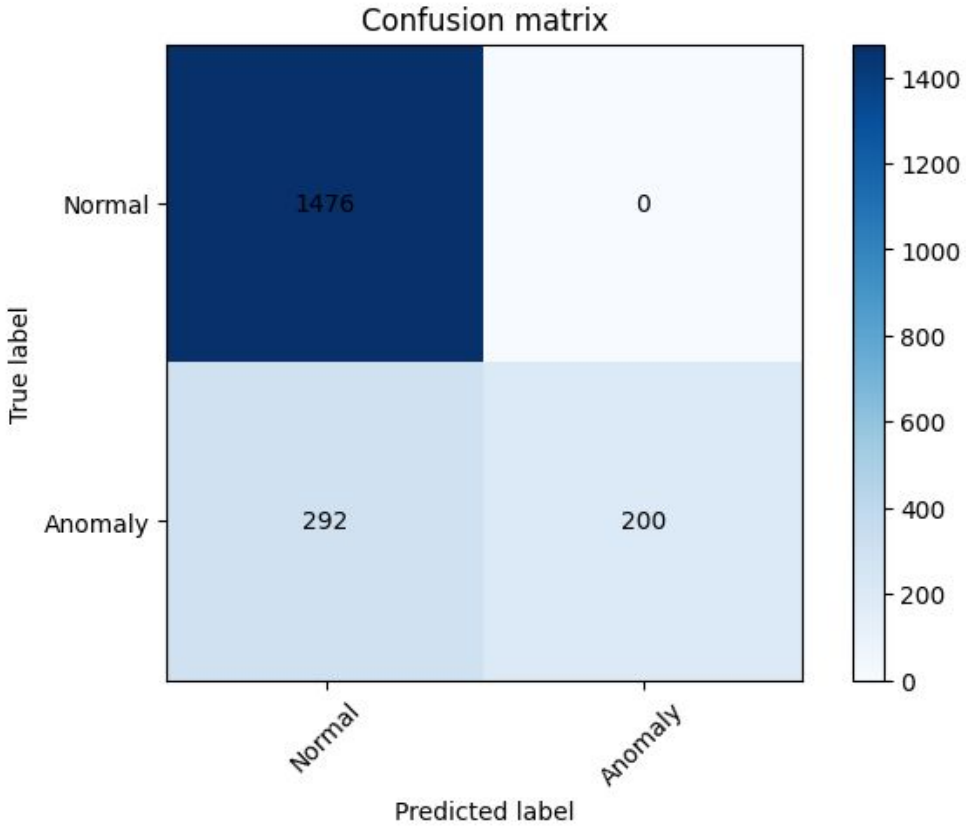
- **Different Autoencoder Structure or hyperparameters** : the autoencoder can be setted with different hyperparameters or built with a different structure (more layer, different activation, more/less neurons, ...)

- **Choose different models for the Anomaly Detection**



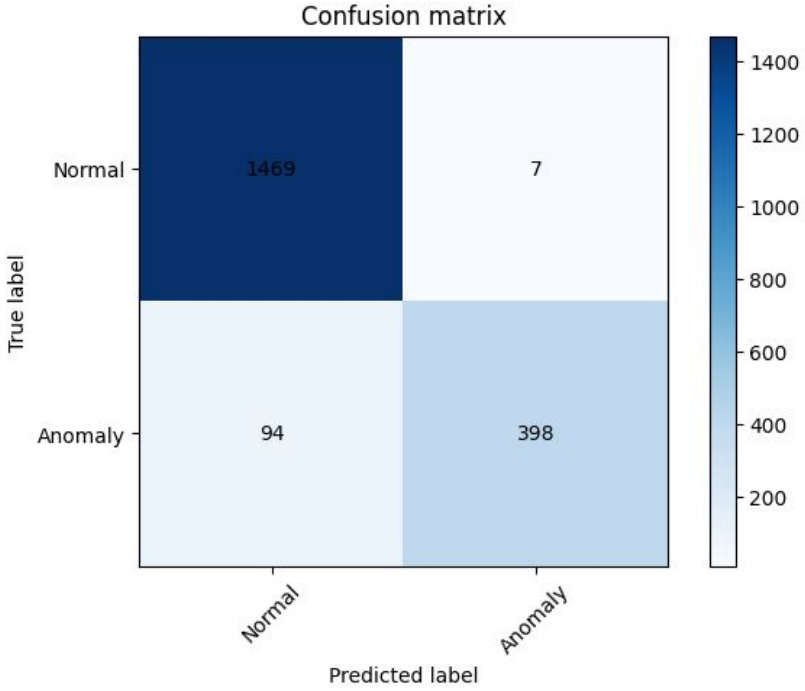
Thanks for your attention

If all the dataset is used for the train



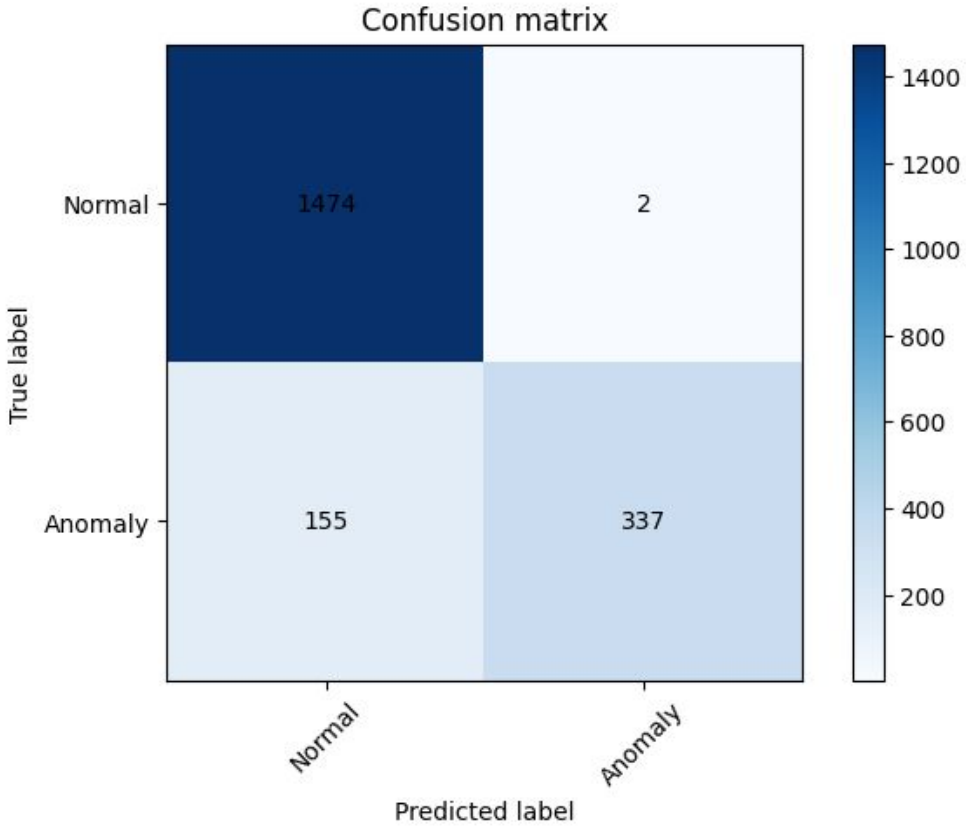

```
Accuracy: 0.8516260162601627
              precision    recall  f1-score   support

      Normal       0.83      1.00      0.91      1476
     Anomaly       1.00      0.41      0.58       492

    accuracy                           0.85      1968
   macro avg       0.92      0.70      0.74      1968
weighted avg       0.88      0.85      0.83      1968
```

If we use the ratio 10:1



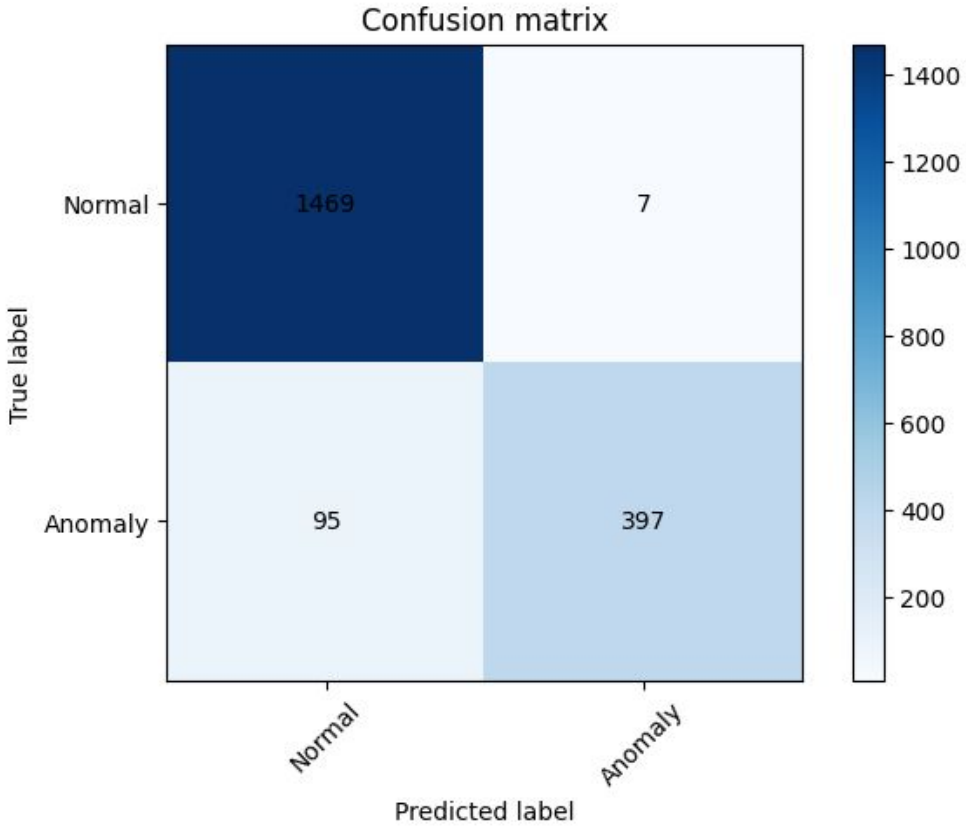

```
...  Accuracy: 0.9486788617886179
                precision    recall  f1-score   support

        Normal       0.94      1.00      0.97      1476
       Anomaly       0.98      0.81      0.89       492

      accuracy                           0.95      1968
     macro avg       0.96      0.90      0.93      1968
  weighted avg       0.95      0.95      0.95      1968
```

If we use the ratio 10:1 with simplif. autoencoder in the bottom



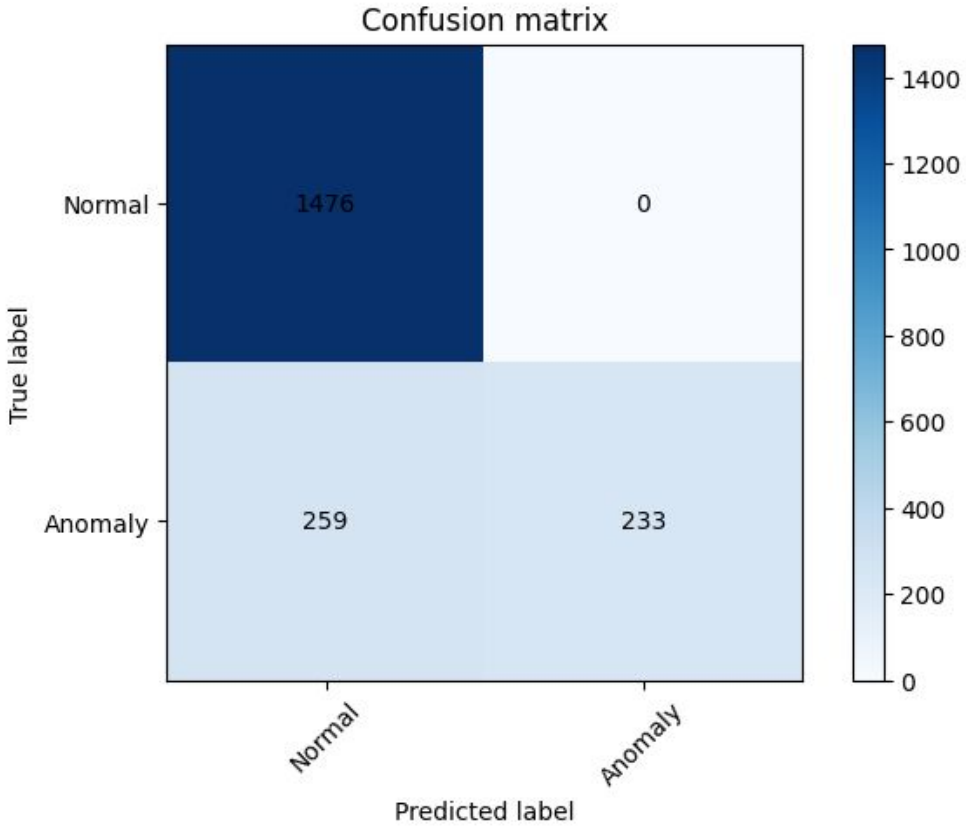

```
Accuracy: 0.9202235772357723
              precision    recall  f1-score   support

      Normal       0.90      1.00      0.95      1476
     Anomaly       0.99      0.68      0.81       492

    accuracy                           0.92      1968
   macro avg       0.95      0.84      0.88      1968
weighted avg       0.93      0.92      0.91      1968
```

If we use the ratio 10:1 with simplif. autoencoder on top



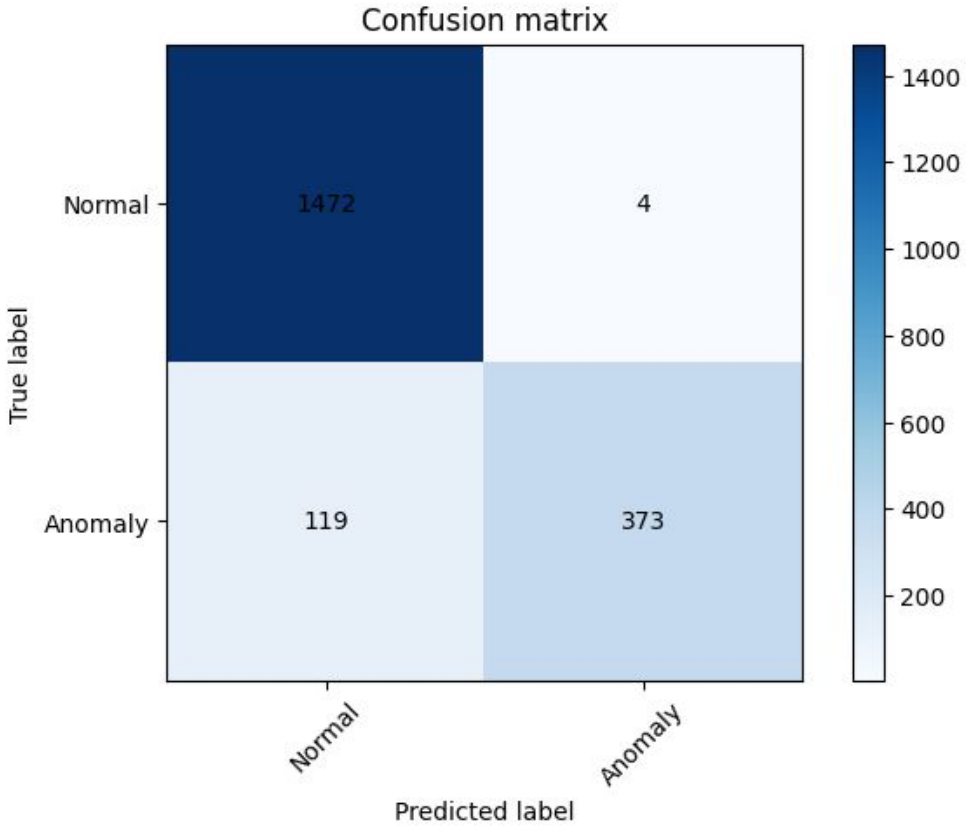

```
Accuracy: 0.948170731707317
              precision    recall  f1-score   support

      Normal       0.94      1.00      0.97      1476
     Anomaly       0.98      0.81      0.89       492

    accuracy                           0.95      1968
   macro avg       0.96      0.90      0.93      1968
weighted avg       0.95      0.95      0.95      1968
```

If we use the ratio 10:1 with simplif. on top without drop



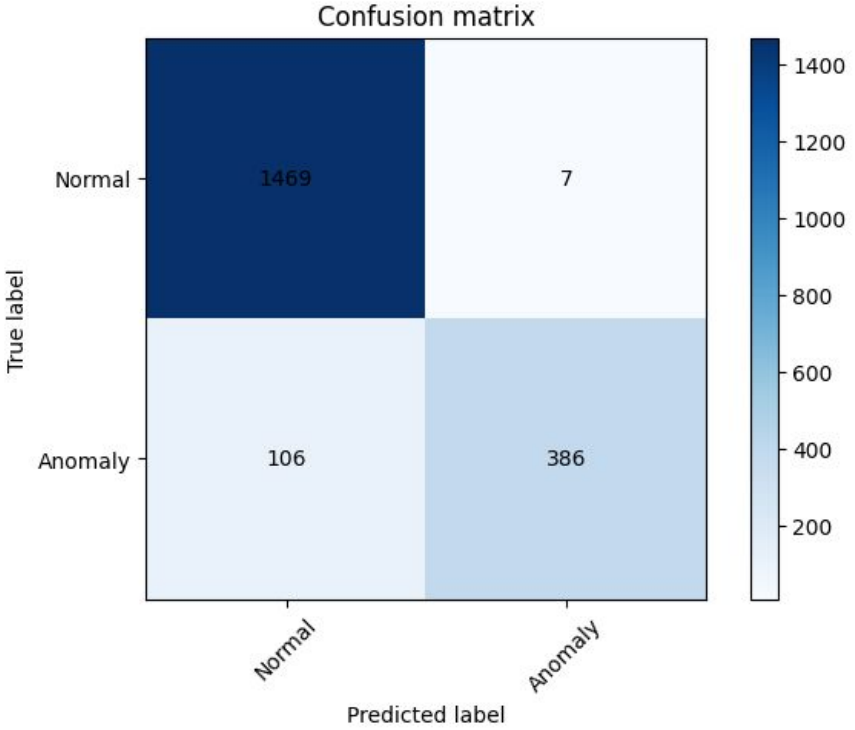

```
Accuracy: 0.8683943089430894
              precision    recall  f1-score   support

      Normal       0.85      1.00      0.92      1476
     Anomaly       1.00      0.47      0.64       492

    accuracy                           0.87      1968
   macro avg       0.93      0.74      0.78      1968
weighted avg       0.89      0.87      0.85      1968
```

If we use ratio 15:1 with simplified autoencoder on top



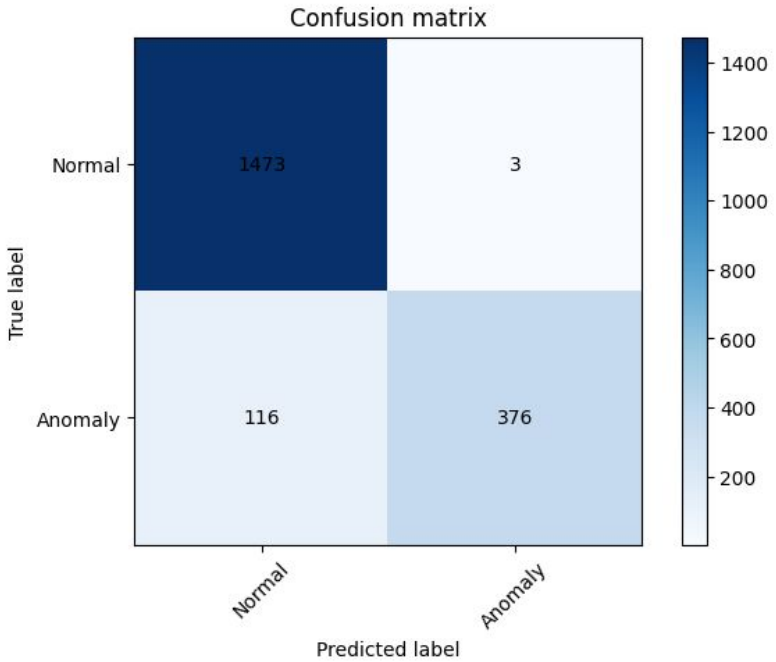

```
...   Accuracy: 0.9375
                precision    recall  f1-score   support

        Normal       0.93      1.00      0.96      1476
       Anomaly       0.99      0.76      0.86       492

      accuracy                           0.94      1968
     macro avg       0.96      0.88      0.91      1968
  weighted avg       0.94      0.94      0.93      1968
```

If we use ratio 15:1



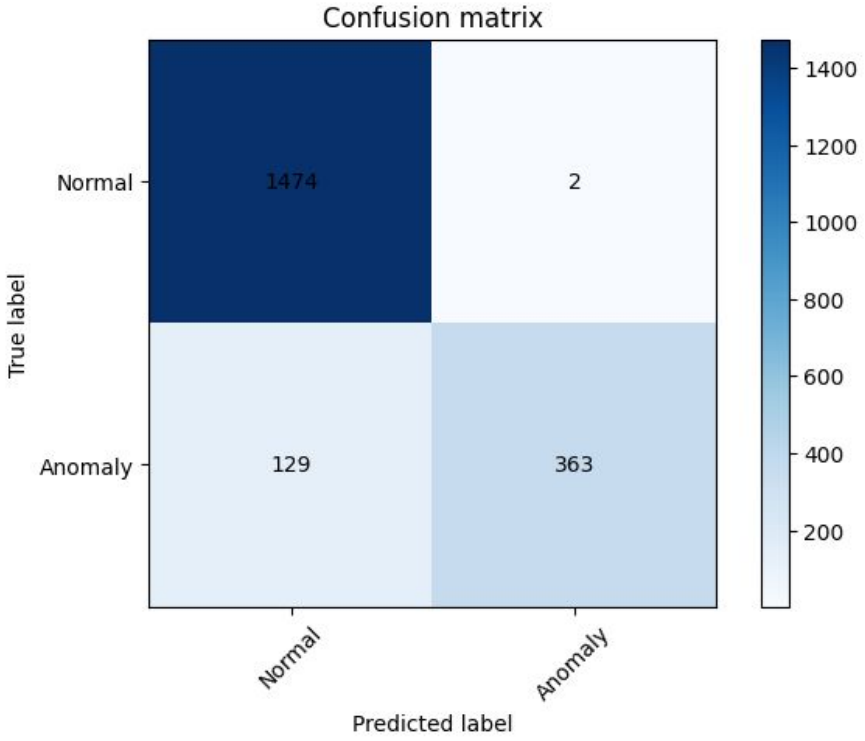

```
Accuracy: 0.9425813008130082
              precision    recall  f1-score   support

      Normal       0.93      1.00      0.96      1476
     Anomaly       0.98      0.78      0.87       492

    accuracy                           0.94      1968
   macro avg       0.96      0.89      0.92      1968
weighted avg       0.95      0.94      0.94      1968
```

If we use ratio 20:1



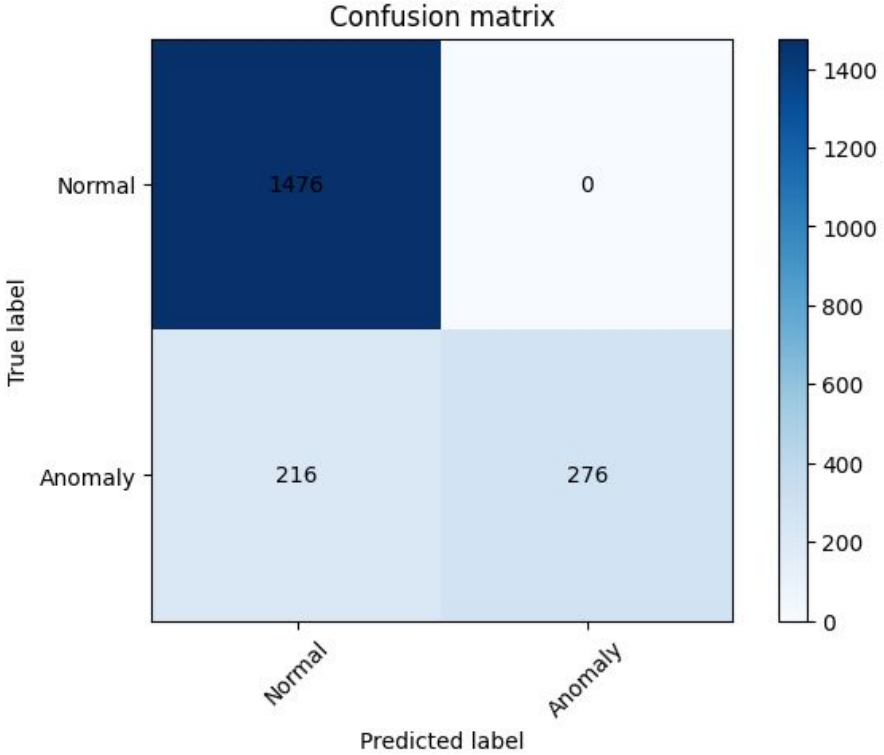

```
Accuracy: 0.9395325203252033
              precision    recall  f1-score   support

      Normal       0.93      1.00      0.96      1476
     Anomaly       0.99      0.76      0.86       492

    accuracy                           0.94      1968
   macro avg       0.96      0.88      0.91      1968
weighted avg       0.94      0.94      0.94      1968
```

If we use ratio 30:1




```
···   Accuracy: 0.9334349593495935
                  precision    recall  f1-score   support

          Normal       0.92      1.00      0.96      1476
         Anomaly       0.99      0.74      0.85       492

        accuracy                           0.93      1968
       macro avg       0.96      0.87      0.90      1968
    weighted avg       0.94      0.93      0.93      1968
```

If we use ratio 30:1 with simplified autoencoder




```
Accuracy: 0.8902439024390244
              precision    recall  f1-score   support

      Normal       0.87      1.00      0.93      1476
     Anomaly       1.00      0.56      0.72       492

    accuracy                           0.89      1968
   macro avg       0.94      0.78      0.83      1968
weighted avg       0.90      0.89      0.88      1968
```