**Notebook: “Credit Card Fraud Detection”**

**Basic idea of the model:**

Train the model with only non-fraud transactions. The autoencoder model is expected to learn how to reconstruct only non-fraud transactions. So the model is expected not perform well in reconstructing fraud transactions. So for fraud transactions it is expected to have a large error between the actual transaction and the reconstructed transaction. This large error is the critical parameter used to determine if the transaction is a fraud transaction.

**Raw data**: 284807 rows × 31 columns

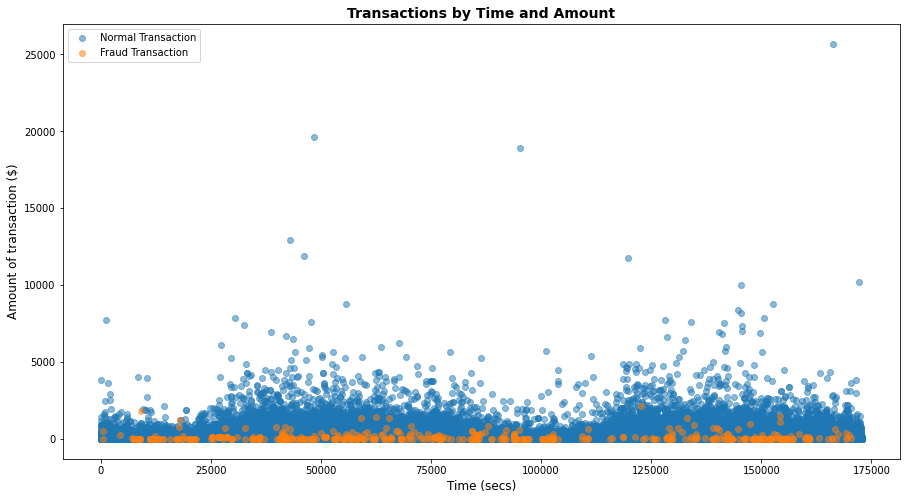
Including: Time, V1 - V28, Amount, Class

We see a total of 31 variables.

* features V1 - V28 are a result of the PCA transformation and are simply numerical representations.
* Time variable is the amount of time that passed from the time when the first transaction took place.
* Amount is the value in dollars of the transaction
* Class represents if the transaction is tagged as being a fraudulent transaction. 0 indicates the transaction is not fraudulent while a 1 indicates a fraudulent transaction. This will be our target variable.

**Pre-processing dataset**:

Visualising the data in term of Time window and Amount of transaction



* The graph above has shown that the fraud or non fraud account does not occur at a specific time window, so this indicates that the time does not hold any important information in detecting fraud transactions. Thus, **time is removed from the data**
* The graph has shown that all the fraud transactions will not have a large transaction amount. (refer to table below)
* The graph showed that the difference between the amount of transactions is large and this will affect the learning of the model. Thus all the **amount of transactions is normalized**
* The raw data also contain 28 columns of **credit card features (V1-V28)**, all these column have also been **normalized**

|  |  |
| --- | --- |

* There are 492 fraud samples and 284315 fraud samples
* Small amount of fraud samples and large amount of non fraud samples

**Split the data into training, validation and test set**

* Training Set and Validation set contain only the non-fraud samples
* Test set contain both the fraud and non-fraud dataset
* Batch size = 1000

| **Dataset** | **Split percentage** | **Number of samples** |
| --- | --- | --- |
| Training Set | 70% | 209255 |
| Validation Set | 10% | 18197 |
| Test Set | 20% | 57355 |

**Explanation for the classifier stage:**

1. **Use the validation set to find the mean error between actual transaction and reconstructed transaction for only non-fraud sample**

**Preds = np.vstack([saved\_model(V(next(valData)).to(device)).cpu().data.numpy() for i in range(len(valData))])**

**# Determine the error between the actual account and the reconstructed account**

**error = np.mean(abs(X\_val - Preds), axis = 1) #axis 1 -> along row**

This code has given the mean error for each non-fraud samples in the validation set

1. **Compute a threshold using the mean error**

**2 Approaches, chosen approach 2**

* **Approach 1**: find the max non-fraud samples’ mean error and use this as the threshold
  + Code: ***threshold = round(max(error), 4)***
  + Limitation: the value is too small and is does not able to detect most of the fraud account
  + Result I obtained only able to detect 3 cases out of 492 fraud cases
* **Approach 2:** find the mean and standard deviation of the non-fraud samples’ mean error and use it as the threshold
  + Code: ***threshold = round((error.mean() + error.std()), 4) #idea from http://www.datadoz.com/blog/detectingfraud.html***
  + This is a better approach as the threshold is a larger number
  + Result: it is able to detect more fraud cases. But also using this threshold it will now detect more non-fraud as fraud.
  + But compare to not detecting fraud account, this threshold has given a better result compared to approach 1

1. **Repeat the same process to find the mean error for each sample in the test set which contain both fraud and non-fraud transaction**
2. **Use the threshold to compute the predicted label for each sample**

* If mean error > threshold => fraud transaction. (class label = 1)
* Mean error <= threshold => non-fraud (class label = 0)
* Code:

**y\_testpred = []**

**for idx, err in enumerate (test\_error):**

**# if difference btw the actual account and the reconstructed account is > threshold == fraud**

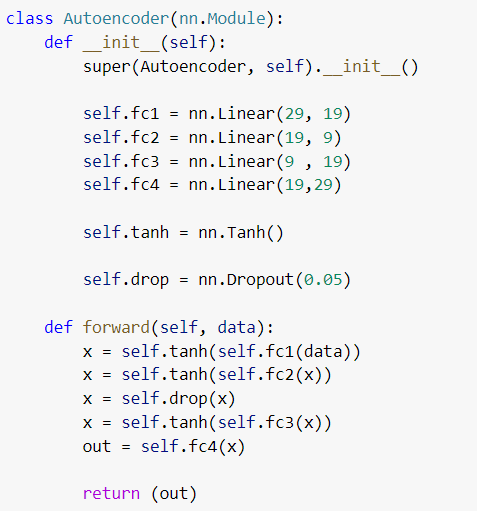
**if err > threshold: class\_pred = 1**

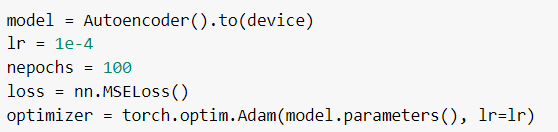
**else: class\_pred = 0**

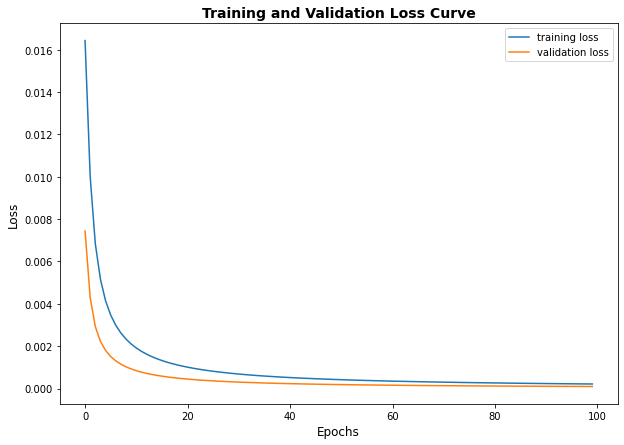
**y\_testpred.append(class\_pred)**

**Model:**

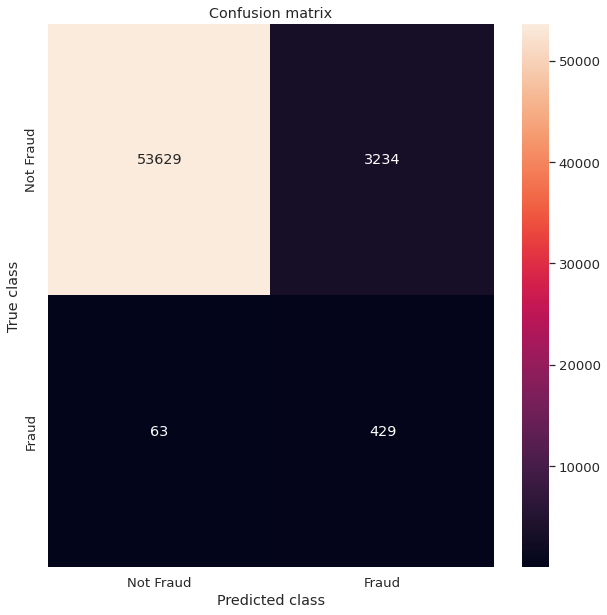
**Approach 1:**

****

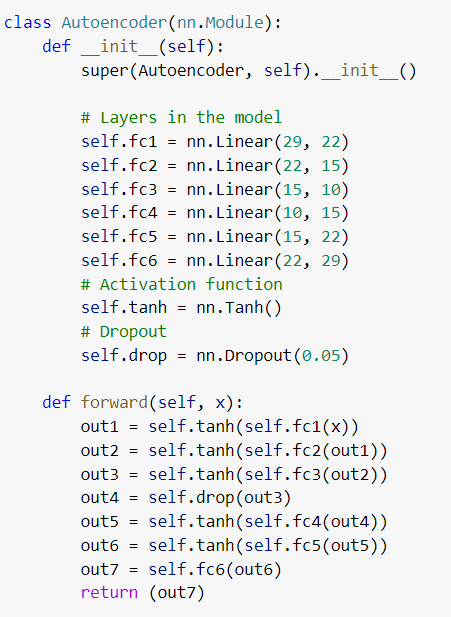
****

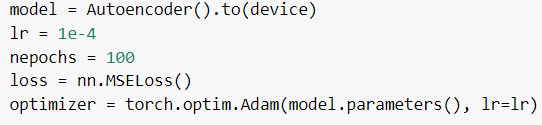
****

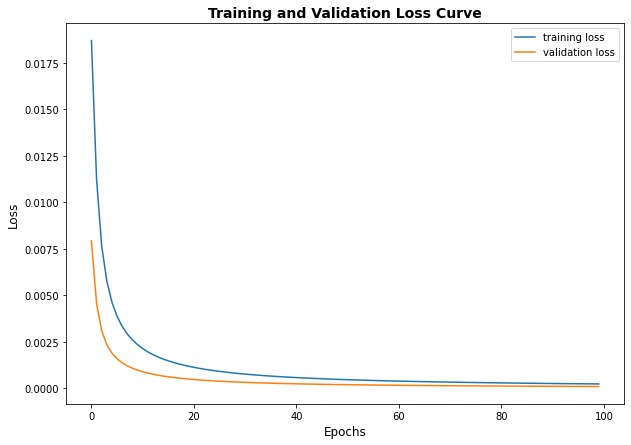
Saved model: “Autoencoder\_Approach1.pth”

****

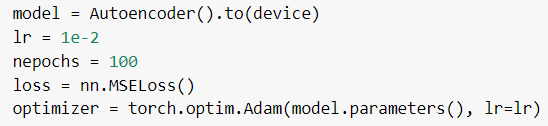
**Approach 2:**

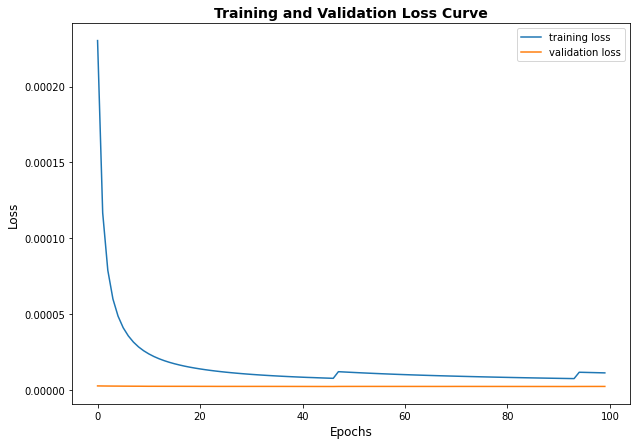
****

****

****

**Approach 2 - different learning rate**

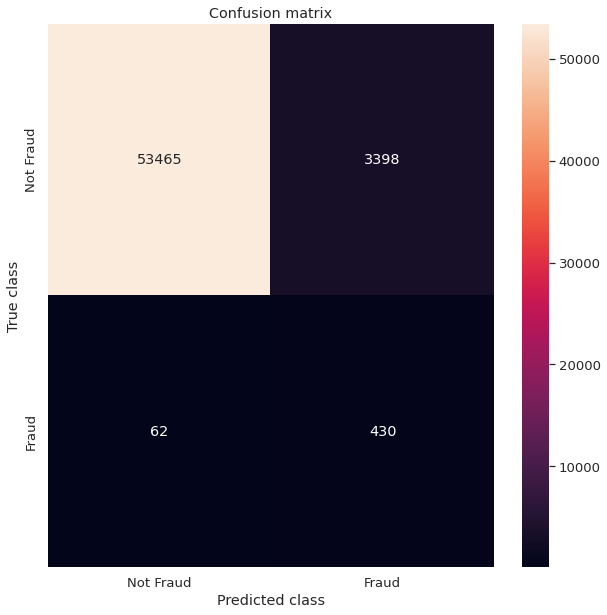
****

****

The validation set is not learning

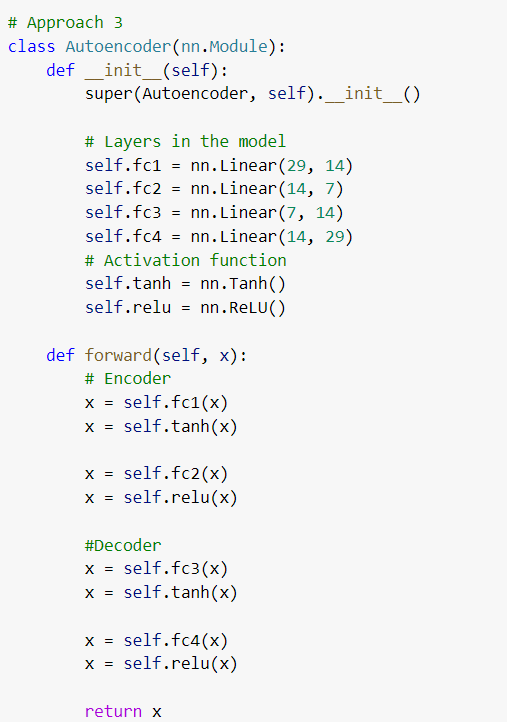
**Chosen learning rate 1e-4, 100 epochs**

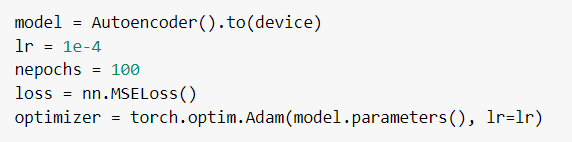
Saved model: “Autoencoder\_Approach2.pth”

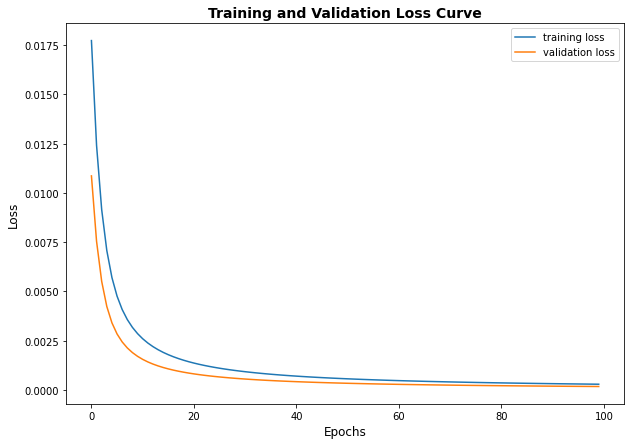


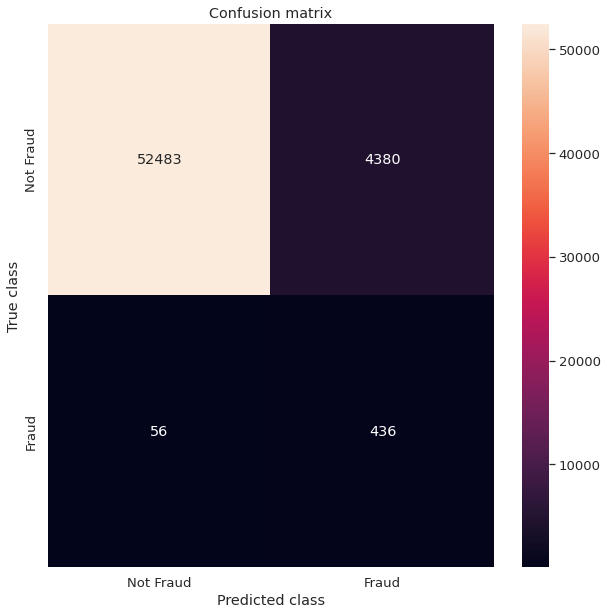
**Approach 3:**

Saved Model: “Autoencoder\_Approach3.pth”

****

****

****



**Compare all the 3 approaches:**

| **Approach 1** | **Approach 2** | **Approach 3** |
| --- | --- | --- |
|  |  |  |
| * 4 fc layers * Use tanh and dropout | * 6 fc layers * Use tanh and dropout | * 4 fc layers * Use tanh and relu |
| Total Trainable Params: 1520 | --- | Total Trainable Params: 1072 |
|  |  |  |
| * Correctly detected non-fraud sample:   53629 out of 56863 => **94.3%**   * Correctly detected fraud sample:   429 out of 492 => **87.2%** | * Correctly detected non-fraud sample:   53465 out of 56863 => **94.02%**   * Correctly detected fraud sample:   430 out of 492 => **87.4%** | * Correctly detected non-fraud sample:   52483 out of 56863 => **92.3%**   * Correctly detected fraud sample:   436 out of 492 => **88.6%** |