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.

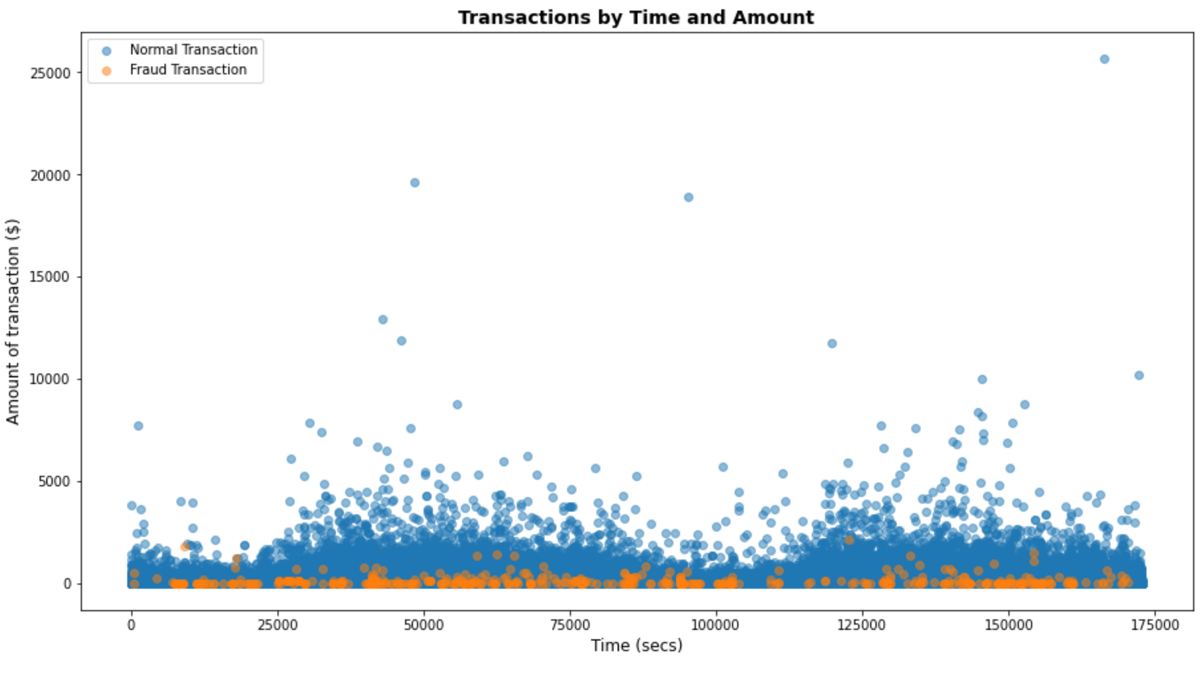
Raw data: 284807 rows × 31 columns
Including: Time, V1 - V28, Amount, Class

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**

```
fraud.Amount.describe()

count      492.000000
mean       122.211321
std        256.683288
min          0.000000
25%          1.000000
50%          9.250000
75%        105.890000
max       2125.870000
Name: Amount, dtype: float64
```

```
nonfraud.Amount.describe()

count    284315.000000
mean         88.291022
std         250.105092
min           0.000000
25%           5.650000
50%          22.000000
75%          77.050000
max       25691.160000
Name: Amount, dtype: float64
```

- 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.nu
mpy() 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

2. **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

3. **Repeat the same process to find the mean error for each sample in the test set which contain both fraud and non-fraud transaction**

4. **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:

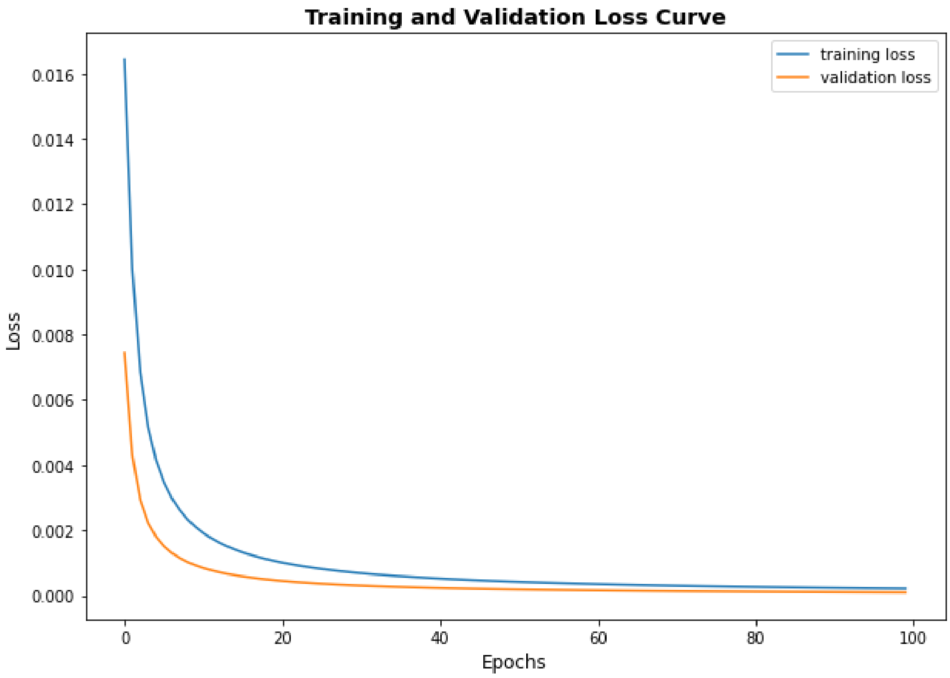
```
y_testpred = []
for idx, err in enumerate (test_error):
    # if difference btw the actual account and the
reconstructed account is > threshold == fraud
    if err > threshold: class_pred = 1
    else: class_pred = 0
    y_testpred.append(class_pred)
```

Model:

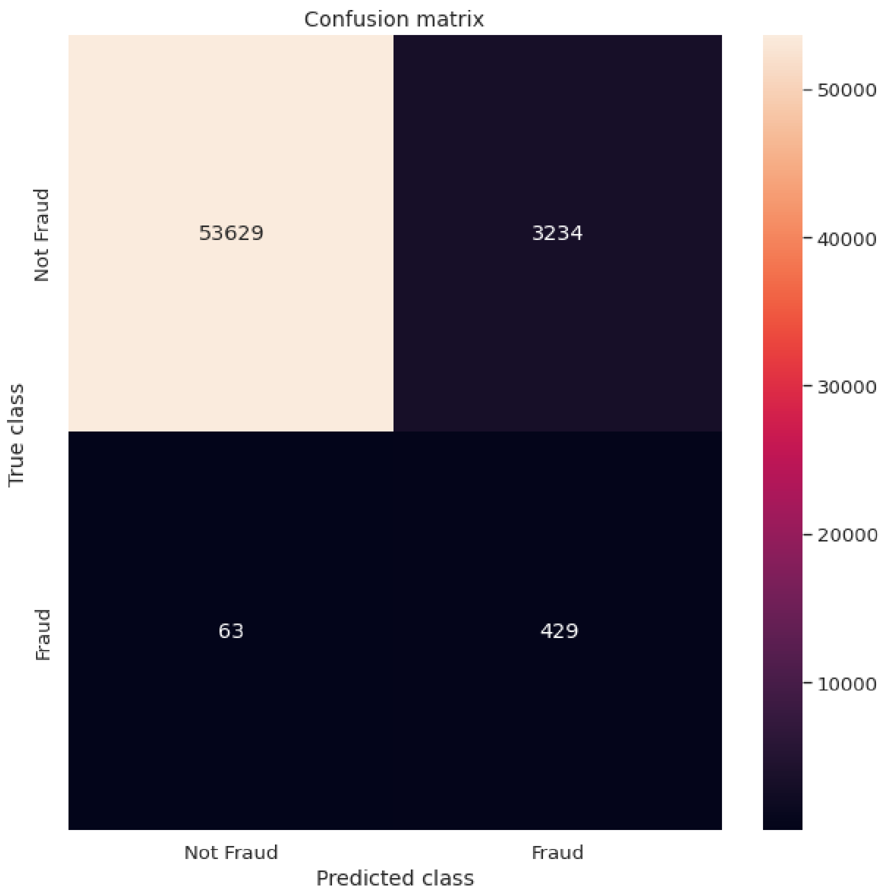
Approach 1:

```
class Autoencoder(nn.Module):
    def __init__(self):
        super(Autoencoder, self).__init__()

        self.fc1 = nn.Linear(29, 19)
        self.fc2 = nn.Linear(19, 9)
        self.fc3 = nn.Linear(9 , 19)
        self.fc4 = nn.Linear(19,29)

        self.tanh = nn.Tanh()

        self.drop = nn.Dropout(0.05)

    def forward(self, data):
        x = self.tanh(self.fc1(data))
        x = self.tanh(self.fc2(x))
        x = self.drop(x)
        x = self.tanh(self.fc3(x))
        out = self.fc4(x)

        return (out)
```

```
model = Autoencoder().to(device)
lr = 1e-4
nepochs = 100
loss = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=lr)
```



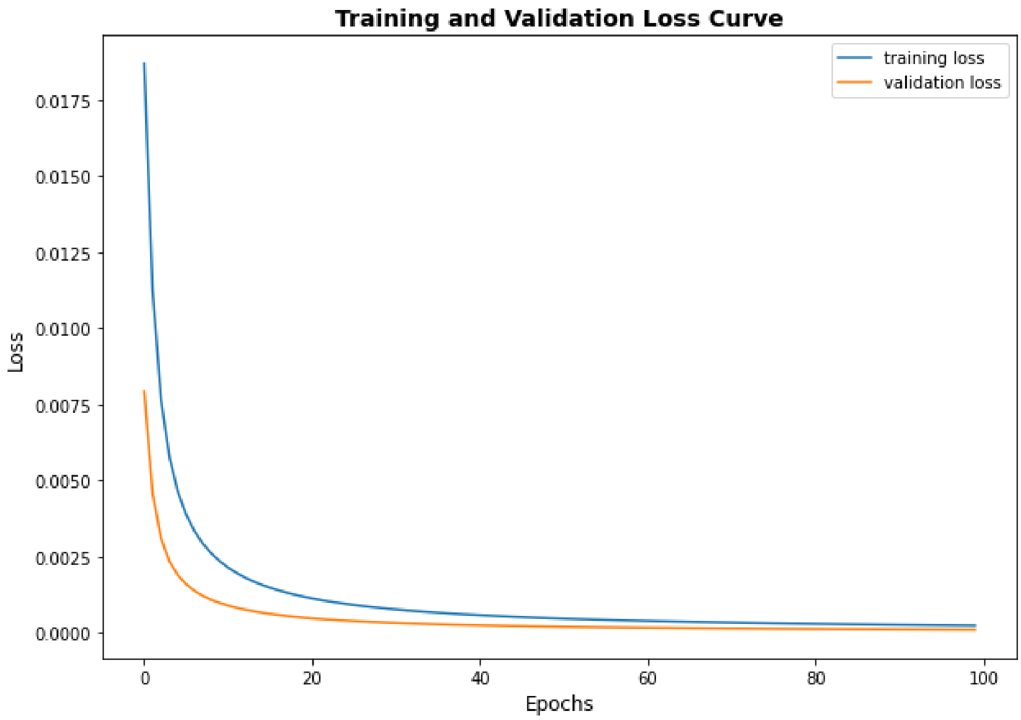
Saved model: “Autoencoder_Approach1.pth”



Approach 2:

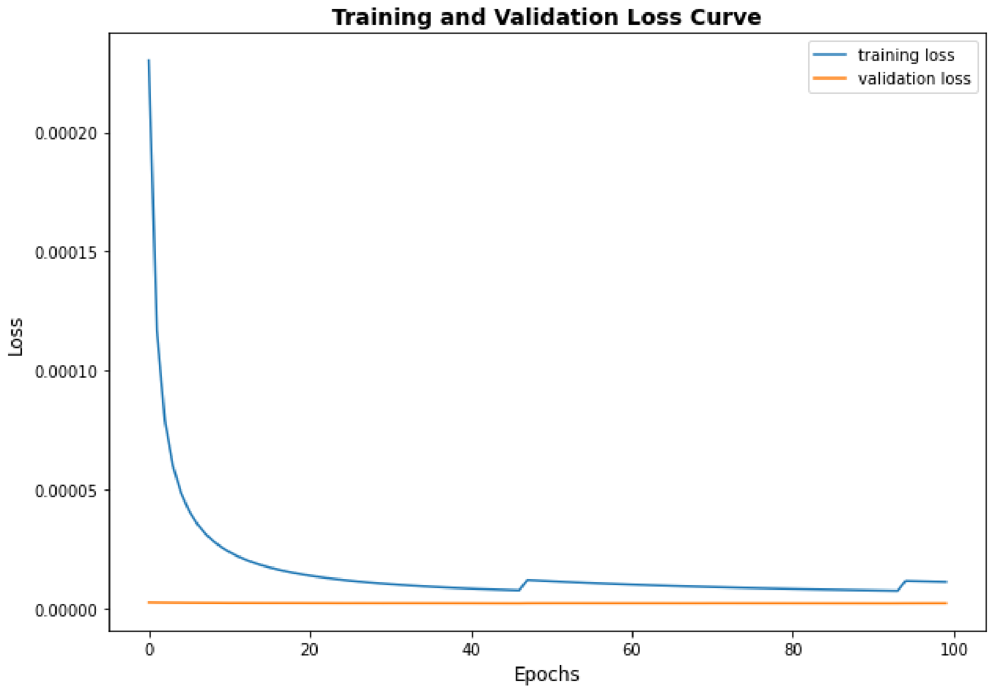
```python
class Autoencoder(nn.Module):
    def __init__(self):
        super(Autoencoder, self).__init__()

        # Layers in the model
        self.fc1 = nn.Linear(29, 22)
        self.fc2 = nn.Linear(22, 15)
        self.fc3 = nn.Linear(15, 10)
        self.fc4 = nn.Linear(10, 15)
        self.fc5 = nn.Linear(15, 22)
        self.fc6 = nn.Linear(22, 29)
        # Activation function
        self.tanh = nn.Tanh()
        # Dropout
        self.drop = nn.Dropout(0.05)

    def forward(self, x):
        out1 = self.tanh(self.fc1(x))
        out2 = self.tanh(self.fc2(out1))
        out3 = self.tanh(self.fc3(out2))
        out4 = self.drop(out3)
        out5 = self.tanh(self.fc4(out4))
        out6 = self.tanh(self.fc5(out5))
        out7 = self.fc6(out6)
        return (out7)
```

```python
model = Autoencoder().to(device)
lr = 1e-4
nepochs = 100
loss = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=lr)
```



Approach 2 - different learning rate

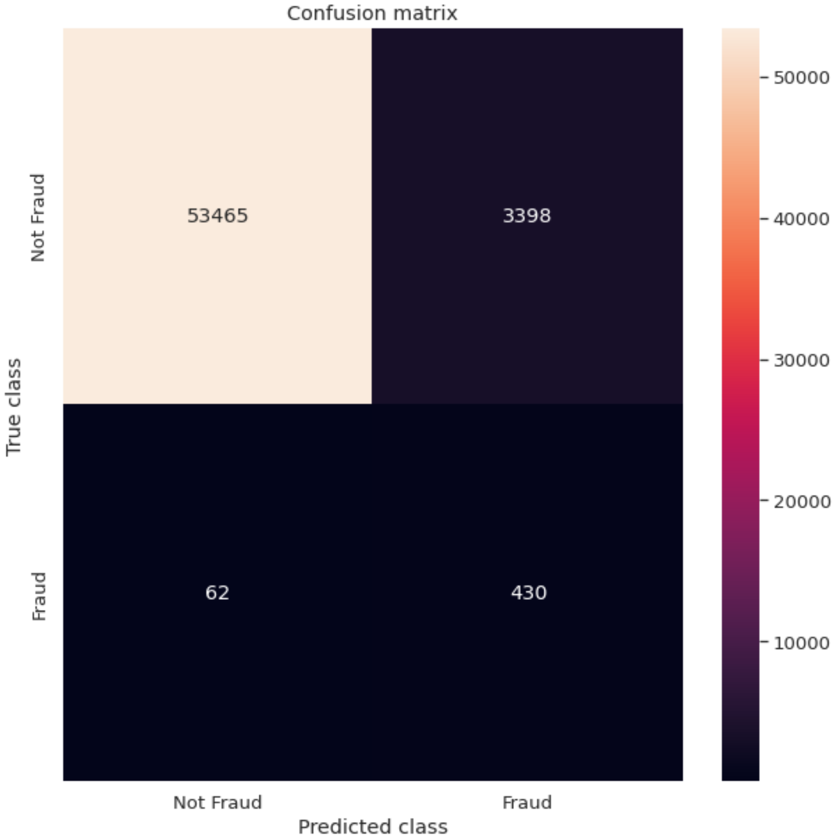
```
model = Autoencoder().to(device)
lr = 1e-2
nepochs = 100
loss = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=lr)
```



The validation set is not learning

Chosen learning rate 1e-4, 100 epochs

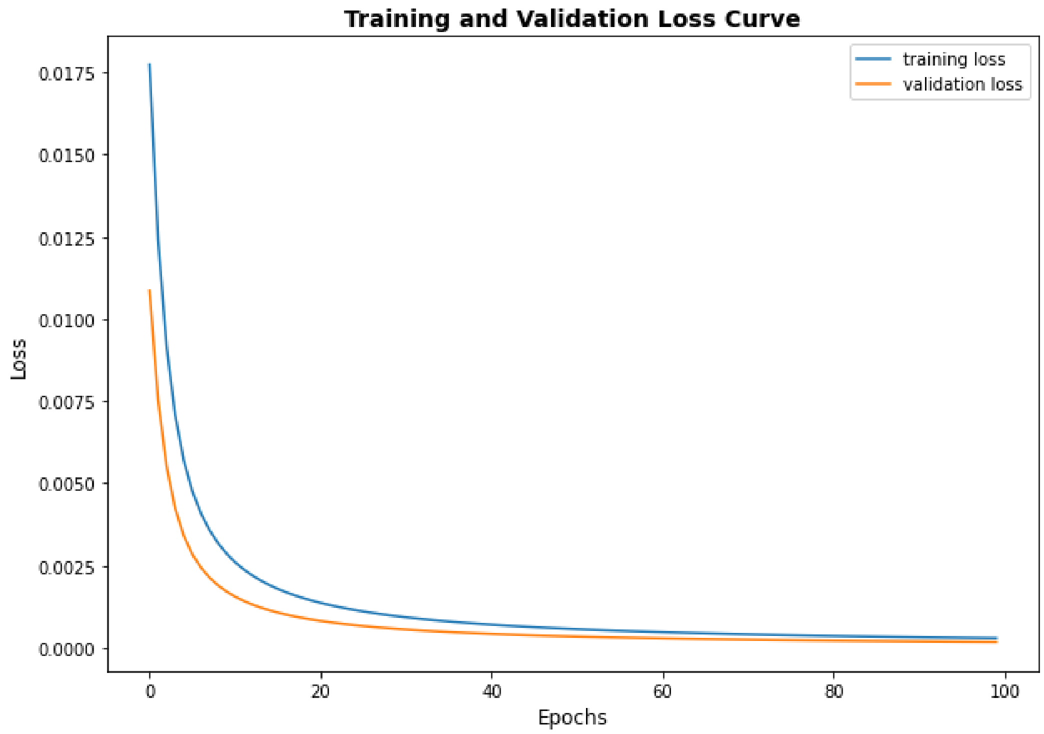
Saved model: “Autoencoder_Approach2.pth”

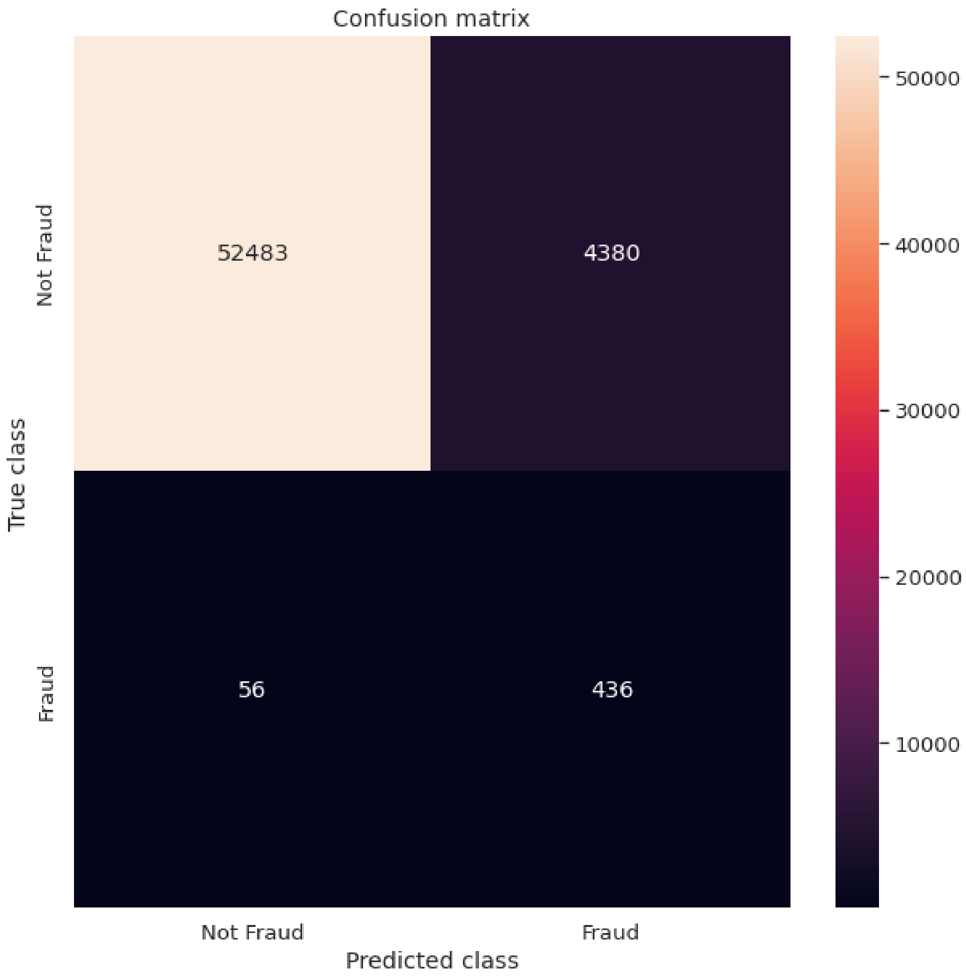


Approach 3:
Saved Model: “Autoencoder_Approach3.pth”

```
# Approach 3
class Autoencoder(nn.Module):
    def __init__(self):
        super(Autoencoder, self).__init__()

        # Layers in the model
        self.fc1 = nn.Linear(29, 14)
        self.fc2 = nn.Linear(14, 7)
        self.fc3 = nn.Linear(7, 14)
        self.fc4 = nn.Linear(14, 29)
        # Activation function
        self.tanh = nn.Tanh()
        self.relu = nn.ReLU()

    def forward(self, x):
        # Encoder
        x = self.fc1(x)
        x = self.tanh(x)

        x = self.fc2(x)
        x = self.relu(x)

        #Decoder
        x = self.fc3(x)
        x = self.tanh(x)

        x = self.fc4(x)
        x = self.relu(x)

        return x
```

```
model = Autoencoder().to(device)
lr = 1e-4
nepochs = 100
loss = nn.MSELoss()
optimizer = torch.optim.Adam(model.parameters(), lr=lr)
```


Training and Validation Loss Curve
training loss
validation loss
Loss
Epochs
0.0175
0.0150
0.0125
0.0100
0.0075
0.0050
0.0025
0.0000
0
20
40
60
80
100


Confusion matrix
True class
Not Fraud
Fraud
52483
4380
56
436
Not Fraud
Fraud
Predicted class
50000
40000
30000
20000
10000

Compare all the 3 approaches:

Approach 1

```python
class Autoencoder(nn.Module):
    def __init__(self):
        super(Autoencoder, self).__init__()

        self.fc1 = nn.Linear(29, 19)
        self.fc2 = nn.Linear(19, 9)
        self.fc3 = nn.Linear(9 , 19)
        self.fc4 = nn.Linear(19,29)

        self.tanh = nn.Tanh()

        self.drop = nn.Dropout(0.05)

    def forward(self, data):
        x = self.tanh(self.fc1(data))
        x = self.tanh(self.fc2(x))
        x = self.drop(x)
        x = self.tanh(self.fc3(x))
        out = self.fc4(x)

        return (out)
```

- 4 fc layers
- Use tanh and dropout

`Total Trainable Params: 1520`

Approach 2

```python
class Autoencoder(nn.Module):
    def __init__(self):
        super(Autoencoder, self).__init__()

        # Layers in the model
        self.fc1 = nn.Linear(29, 22)
        self.fc2 = nn.Linear(22, 15)
        self.fc3 = nn.Linear(15, 10)
        self.fc4 = nn.Linear(10, 15)
        self.fc5 = nn.Linear(15, 22)
        self.fc6 = nn.Linear(22, 29)
        # Activation function
        self.tanh = nn.Tanh()
        # Dropout
        self.drop = nn.Dropout(0.05)

    def forward(self, x):
        out1 = self.tanh(self.fc1(x))
        out2 = self.tanh(self.fc2(out1))
        out3 = self.tanh(self.fc3(out2))
        out4 = self.drop(out3)
        out5 = self.tanh(self.fc4(out4))
        out6 = self.tanh(self.fc5(out5))
        out7 = self.fc6(out6)
        return (out7)
```

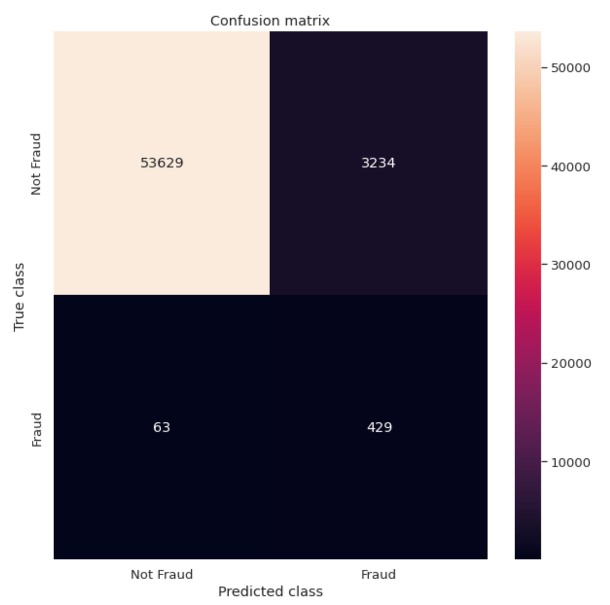
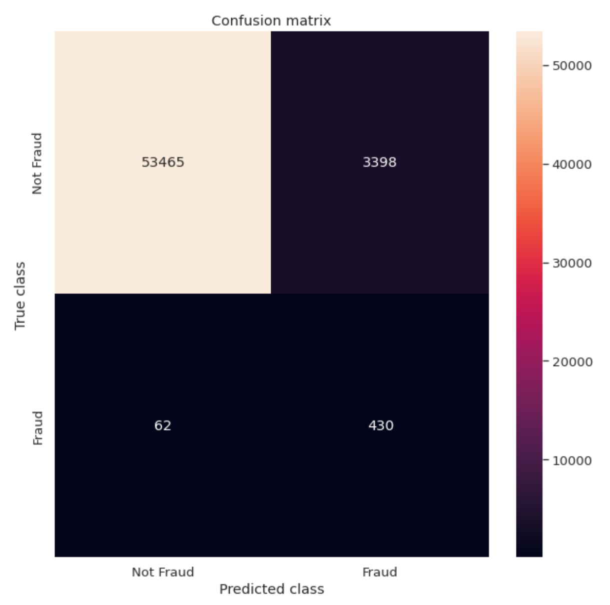
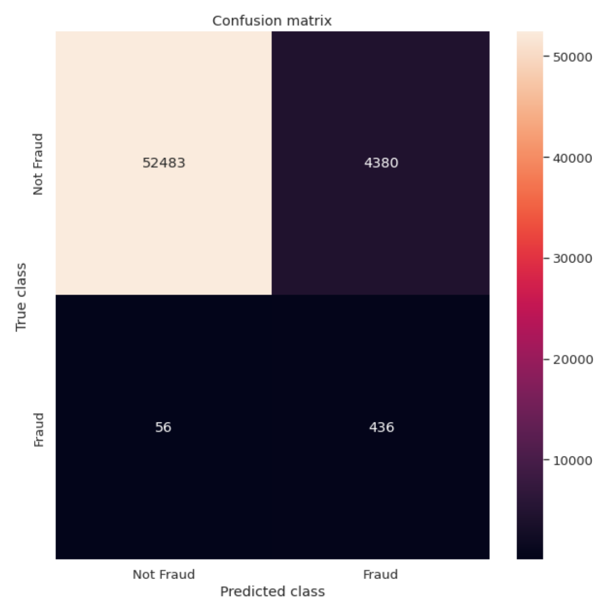
- 6 fc layers
- Use tanh and dropout

Approach 3

```python
# Approach 3
class Autoencoder(nn.Module):
    def __init__(self):
        super(Autoencoder, self).__init__()

        # Layers in the model
        self.fc1 = nn.Linear(29, 14)
        self.fc2 = nn.Linear(14, 7)
        self.fc3 = nn.Linear(7, 14)
        self.fc4 = nn.Linear(14, 29)
        # Activation function
        self.tanh = nn.Tanh()
        self.relu = nn.ReLU()

    def forward(self, x):
        # Encoder
        x = self.fc1(x)
        x = self.tanh(x)

        x = self.fc2(x)
        x = self.relu(x)

        #Decoder
        x = self.fc3(x)
        x = self.tanh(x)

        x = self.fc4(x)
        x = self.relu(x)

        return x
```

- 4 fc layers
- Use tanh and relu

`Total Trainable Params: 1072`

<table>
<tr>
<td>
</td>
<td>
</td>
<td>
</td>
</tr>
<tr>
<td>- Correctly detected non-fraud sample: 53629 out of 56863 => **94.3%**
- Correctly detected fraud sample: 429 out of 492 => **87.2%**</td>
<td>- Correctly detected non-fraud sample: 53465 out of 56863 => **94.02%**
- Correctly detected fraud sample: 430 out of 492 => **87.4%**</td>
<td>- Correctly detected non-fraud sample: 52483 out of 56863 => **92.3%**
- Correctly detected fraud sample: 436 out of 492 => **88.6%**</td>
</tr>
</table>