First we start by loading our train data in X_train and y_train, and then we load our test data in X_test and y_test By watching our data we find that our train data has 18000 labeled samples and 784 features (which are pixels of a 28*28 picture) and a label from 0 to 9.

After that we start some checking to see if our data needs any preprocessing. First we check if we have any missing values and thankfully there isn't any. After that we check pixels interval and we see that pixel's intervals vary a lot and we decide to do a min-max scaling to scale them all in interval of [0, 1]. After that we check for imbalancement and again thankfully there isn't so much of imbalancement but for caution we do an oversampling for label 5 and increase its sample numbers to 1750.

After that it's time to implement our autoencoder and set its parameters. We decide to have 4 encoding layers and 4 decoding layers. Because we may lose information during feature reduction, we decide to divide out features by two with each layer so we have that out feature numbers start from 784 and goes like:

784 -> 392 -> 196 -> 98 -> 49

and when we are decoding our feature numbers goes like:

49 -> 98 -> 196 -> 392 -> 784.

And for each nodes activation function we set Relu function except for the last layer of our decoder, we choose sigmoid function because our pixels are in interval of [0, 1].

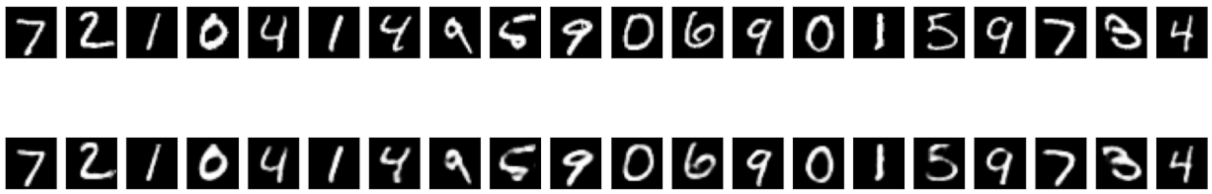
And for our loss function and optimizer we have that:

- **Optimizer = Adam**: It combines the benefits of adaptive learning rates and momentum-based updates. Adam algorithm computes individual adaptive learning rates for each of our parameters and it allows efficient convergence and handles sparse gradients.
- **Loss function = Binary cross-entropy**: It measures the dissimilarity between our predicted outputs and the true binary targets, and it penalizes large discrepancies. And by maximizing the similarity between our

reconstructed outputs and the original inputs, it encourages our autoencoder to learn meaningful binary representations.

Also we use 100 epochs to make sure our encoder learns well and we use batch gradient descend to make it faster (with batch size of 256).

After learning our encoder, its time to encode X_train and X_test and decode some of them to see how our algorithm is doing. We decode all of our test data but we plot only twenty of them.



As you can see our autoencoder is working pretty fine so we can use it for feature extraction. With using this autoencoder we will reduce our features from 784 features to only 49 features. In first sight we may say that we are losing a lot of information, but the truth is autoencoder learns best parameters that specifies our samples pretty well and with using it we are not losing too much information and it will help us a lot to avoid curse of dimensionality and helps our classifiers to learn more accurate and much faster.

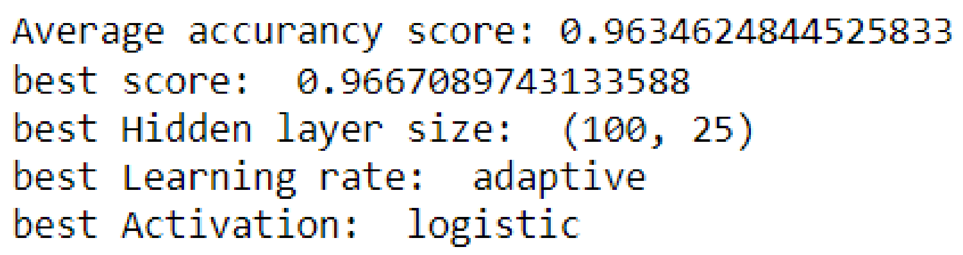
Know we start to learn best parameters for MLP with 5-fold classification and our learning parameters are:

1. **Hidden layers**: here we fix number of layers and size of them, but because of computation limits we choose only one layer and two layer options.
2. **Learning rates**:
    a. Constant: Uses a constant learning rate throughout the training process, which may lead to slower convergence or overshooting in some cases.
    b. Adaptive: Adjusts the learning rate dynamically based on the progress of the training, such as reducing the learning rate when approaching a minimum or increasing it for faster initial learning

    c. Invscaling: Gradually decreases the learning rate over time based on the inverse of a scaling factor, aiding convergence and stability during training.

3. **Activation functions**:
    a. Logistic: Maps the input to a range between 0 and 1, resembling a logistic function, commonly used in binary classification problems.
    b. Identity: Provides a linear mapping where the output is equal to the input, often used in regression problems or as a pass-through activation function.
    c. Relu: Sets all negative inputs to zero and keeps positive inputs unchanged, commonly used in deep learning models to introduce non-linearity.
    d. Tanh: Rescales the input to a range between -1 and 1, similar to the logistic function but symmetric around zero.
4. **Batch size**: Refers to the number of training examples used in each forward and backward pass to update the model's parameters but because of computation limits we have avoided it.
5. **Maximum number of iterations**: Because of computation limits we have avoided it.

After learning we find that best parameters are:

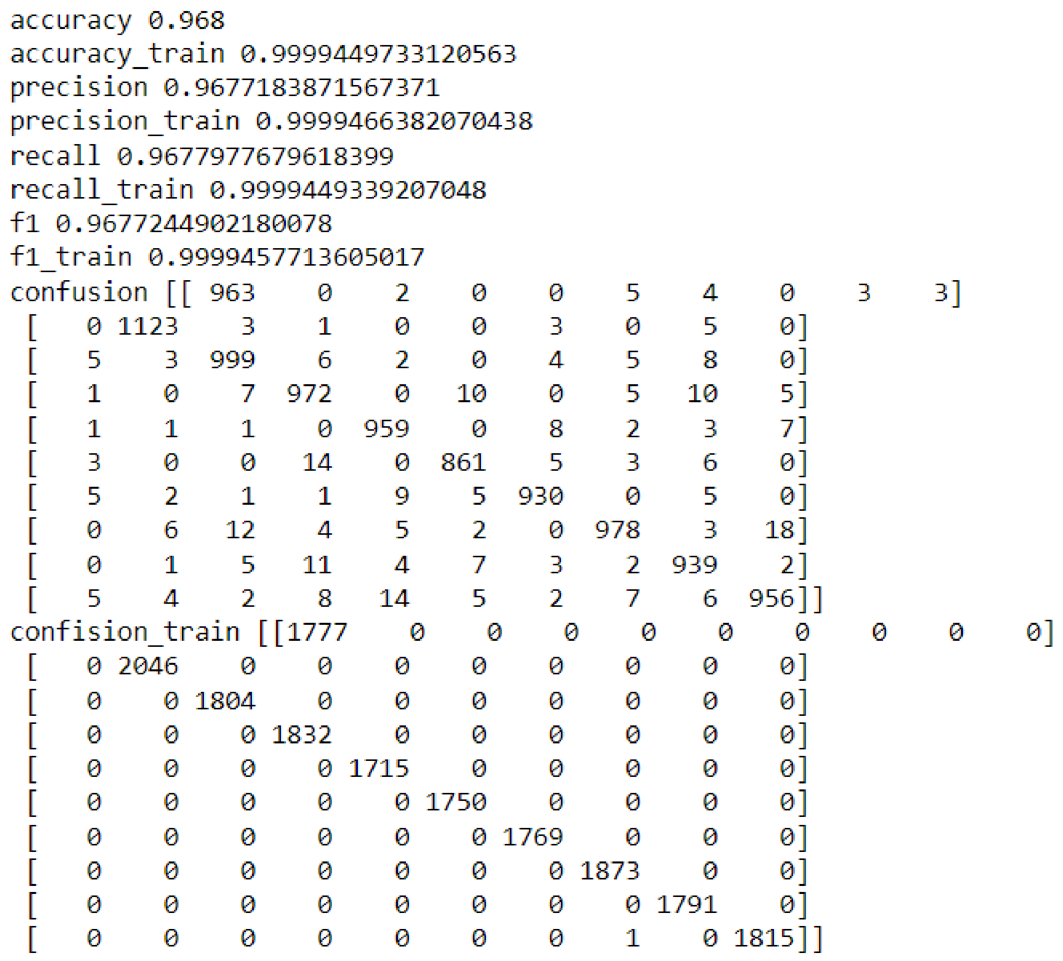


```
Average accurancy score: 0.9634624844525833
best score:  0.9667089743133588
best Hidden layer size:  (100, 25)
best Learning rate:  adaptive
best Activation:  logistic
```

(And you can see accuracy, precision, recall and f1_score for each model in Main.ipynb)

It shows that here 2 layers works better than one layer, and adjusting learning rate adaptive and using logistic activation function also helps our model to predict better.

Now we learn our best model and calculate accuracy, precision, recall, f1_score and confusion matrixes for it.



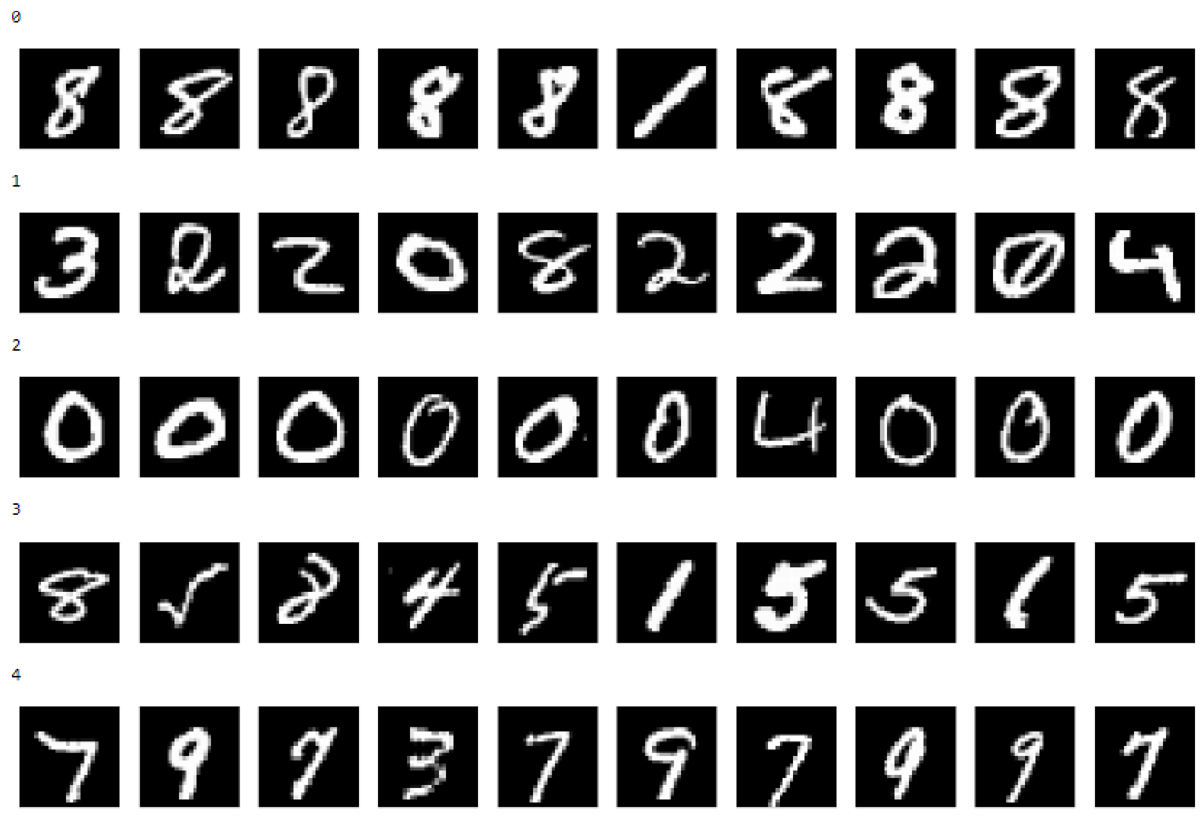
```
accuracy 0.968
accuracy_train 0.9999449733120563
precision 0.9677183871567371
precision_train 0.9999466382070438
recall 0.9677977679618399
recall_train 0.9999449339207048
f1 0.9677244902180078
f1_train 0.9999457713605017
confusion [[ 963    0    2    0    0    5    4    0    3    3]
 [   0 1123    3    1    0    0    3    0    5    0]
 [   5    3  999    6    2    0    4    5    8    0]
 [   1    0    7  972    0   10    0    5   10    5]
 [   1    1    1    0  959    0    8    2    3    7]
 [   3    0    0   14    0  861    5    3    6    0]
 [   5    2    1    1    9    5  930    0    5    0]
 [   0    6   12    4    5    2    0  978    3   18]
 [   0    1    5   11    4    7    3    2  939    2]
 [   5    4    2    8   14    5    2    7    6  956]]
confision_train [[1777    0    0    0    0    0    0    0    0    0]
 [   0 2046    0    0    0    0    0    0    0    0]
 [   0    0 1804    0    0    0    0    0    0    0]
 [   0    0    0 1832    0    0    0    0    0    0]
 [   0    0    0    0 1715    0    0    0    0    0]
 [   0    0    0    0    0 1750    0    0    0    0]
 [   0    0    0    0    0    0 1769    0    0    0]
 [   0    0    0    0    0    0    0 1873    0    0]
 [   0    0    0    0    0    0    0    0 1791    0]
 [   0    0    0    0    0    0    0    1    0 1815]]
```

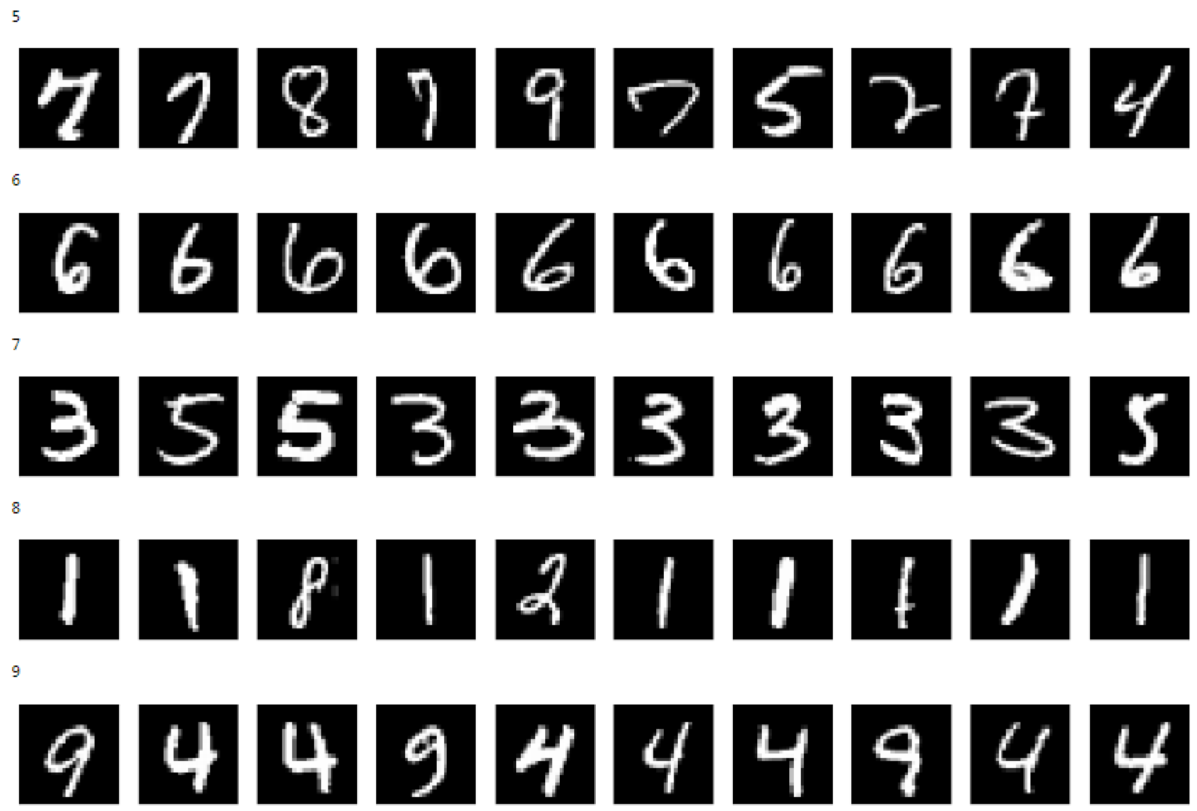
We see that we have 96.8% accuracy for test data and almost 100% accuracy for our train data. Our accuracy on train data alerts us that we have overfitting but when we see that our test accuracy is very high too, we can ignore it and let it be the way it is. By watching our test confusion matrix we see that our most accurate labels are labels 0 and 1 and 6, and our worst labels are 3 and 9 which shows how separable each digit is.

After that, it’s time to cluster our unlabeled samples. For clustering we choose K-means algorithm for some reasons, first of all we can learn K-means for different initial cores and help our algorithm with multiple times of clustering, second reason is that in DBSCAN algorithm we can’t set number of our clusters and if we want to do it in a post-processing job, because we have too many dimensions it will be very difficult.

For clustering we first encode our unlabeled data with our autoencoder and then we perform our clustering to reduce our dimensionality and help our cluster to cluster them better. And we choose random initial points so that we can benefit multiple clustering and choose best one (400 times).

After clustering we should find that what digit is each cluster, and for this we do it manually and we plot 10 samples of each cluster and we map clusters to digits manually and we have that our clusters are like:

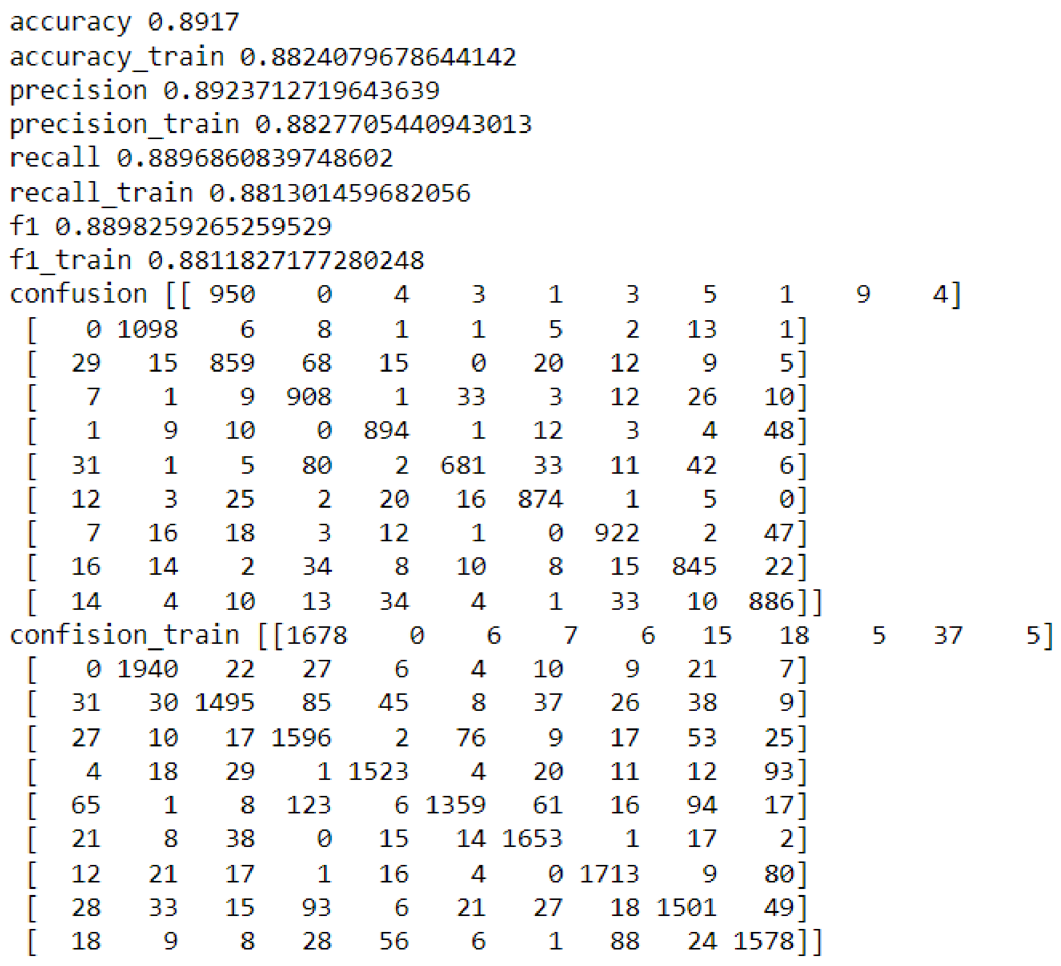




And we decide that our mapping should be:

Cluster	Label
0	8
1	2
2	0
3	5
4	9
5	7
6	6
7	3
8	1
9	4

After labeling our clusters, it’s time to merge them with our train data and learn our previous parameter MLP model on it again. After learning our model it’s time to calculate accuracy, precision, recall, f1_score and confusion matrixes for it.



```
accuracy 0.8917
accuracy_train 0.8824079678644142
precision 0.8923712719643639
precision_train 0.8827705440943013
recall 0.8896860839748602
recall_train 0.881301459682056
f1 0.8898259265259529
f1_train 0.8811827177280248
confusion [[ 950    0    4    3    1    3    5    1    9    4]
 [   0 1098    6    8    1    1    5    2   13    1]
 [  29   15  859   68   15    0   20   12    9    5]
 [   7    1    9  908    1   33    3   12   26   10]
 [   1    9   10    0  894    1   12    3    4   48]
 [  31    1    5   80    2  681   33   11   42    6]
 [  12    3   25    2   20   16  874    1    5    0]
 [   7   16   18    3   12    1    0  922    2   47]
 [  16   14    2   34    8   10    8   15  845   22]
 [  14    4   10   13   34    4    1   33   10  886]]
confision_train [[1678    0    6    7    6   15   18    5   37    5]
 [   0 1940   22   27    6    4   10    9   21    7]
 [  31   30 1495   85   45    8   37   26   38    9]
 [  27   10   17 1596    2   76    9   17   53   25]
 [   4   18   29    1 1523    4   20   11   12   93]
 [  65    1    8  123    6 1359   61   16   94   17]
 [  21    8   38    0   15   14 1653    1   17    2]
 [  12   21   17    1   16    4    0 1713    9   80]
 [  28   33   15   93    6   21   27   18 1501   49]
 [  18    9    8   28   56    6    1   88   24 1578]]
```

When labeling our clusters we see that clusters are not divided that well but it is still pretty convincing so we move forward with it and when we do our clustering on it, it is natural that it’s accuracy decreases from our manually labeled train data but still we see that we have 88% accuracy on our train data and 89% accuracy on our test data and I think it is a very good accuracy. After that with watching train confusion matrix we see that exactly labels 7 and 9 that were our most inseparable digits are predicted worst and labels 1 and 6 that were our most separable digits are predicted best.

Ashkan Zarkhah

610399196