CS464

Introduction to Machine Learning

Homework 2

Mustafa Mert Türkmen

21902576 – IE

Section 2

Question 1: PCA Analysis

Question 1.1

In order to obtain the principal components, used the numpy.linalg.eig function to return both the eigenvalues and eigenvectors. Then, to compute the PVE for each principal component by dividing the corresponding eigenvalue by the sum of all eigenvalues. The results are below:

Proportion of Variance Explained (PVE) for the first 10 principal components

PC-1: 0.09704664368150995
PC-2: 0.07095924066613629
PC-3: 0.06169088772849623
PC-4: 0.05389419494309184
PC-5: 0.04868797012682377
PC-6: 0.043122313233494455
PC-7: 0.032719299513236835
PC-8: 0.028838954483044483
PC-9: 0.02762029396742113
PC-10: 0.023570005511596343

It can be seen that when we reach the 10. principal components, when we sum them up, it just can capture nearly 49% of the total variance. We are losing information from data, so it is not enough to use it.

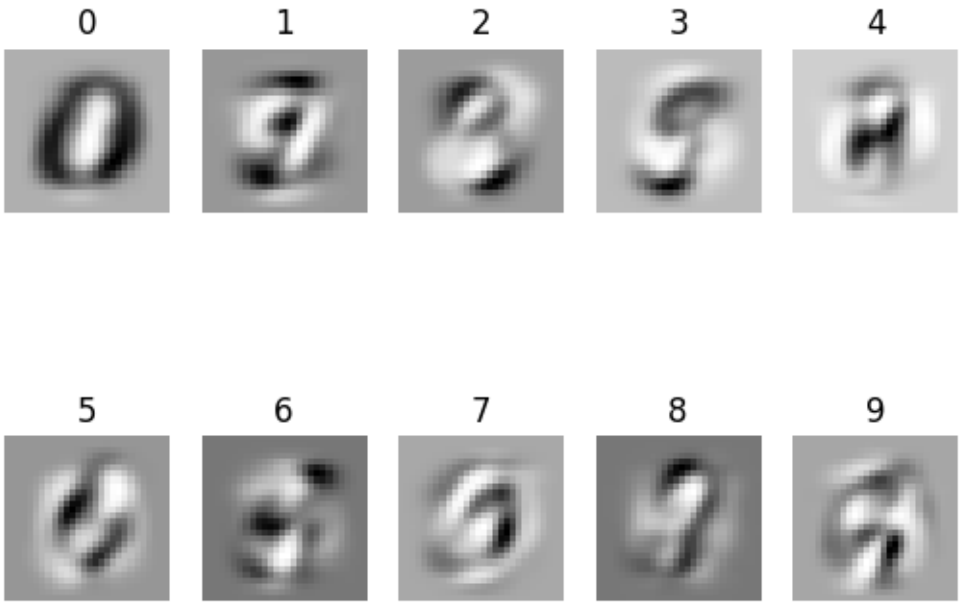
Question 1.2

In order to capture more than %70 of the variance, at least 26 principal components are needed.

Question 1.3

In order to obtain visuals of first 10 eigenvectors, I take i'th principal component on top where i takes values from 1 to 10. It can be said that these images are a not a good representation of all original images since these are the top 10 principal components; however

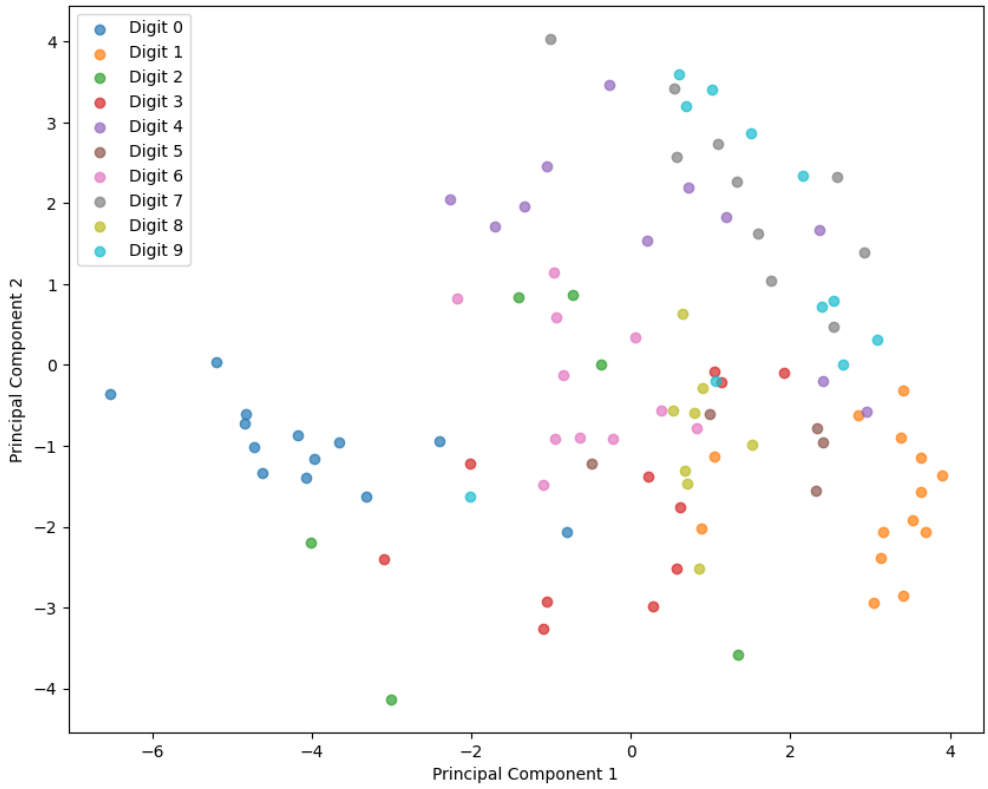
they just capture of the variance almost 50%. Maybe linearly combining these images, we can obtain a close approximation of the original images. The resulting ten images can be seen below.



Picture 1. Grayscale Principal Component Images

Question 1.4

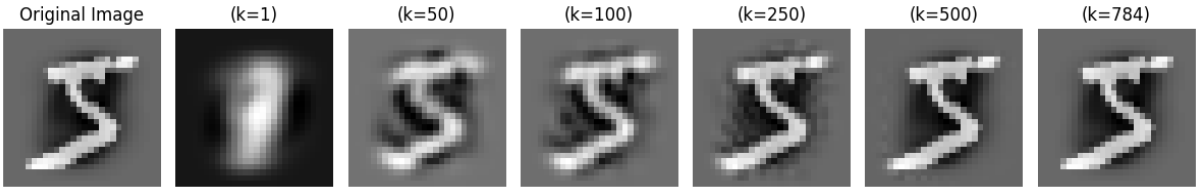
I can say that the first two principal components are catching some of the digits, for example number 0 and 1. Both digits take similar values from these two components relatively, digit 0 takes -1 to 0 from principal component 2 and take densely -4 to -5 from principal components 1. It shows that the first two principal components are somewhat successful in catching these digits. However, it can not say some of the digits like digit 2. It takes values from a wider range and didn't gather in specific values. It can easily be said that the first two principal components are not enough to get information from digits images.



Plot 1. First Two Principal Component Values of First 100 Images

Question 1.5

First, I take the mean of the images and subtract it from the original image to center images. Then I tried to find the projection of them with using eigenvectors with help np.dot function. I found the dot product of centered image array and transposed eigenvector array. I have control of how many of the eigenvectors are going to use so that I can try it with 1 to 784. Then with this projection arrays I again find dot product with eigenvector but now I didn't take its transpose and I added mean of the images. So, it gives me to opportunity to visualize them with imshow function.



Picture 2. Reconstructed Digit Images with using k Principal Components

After investigating the pictures, I can easily say that using just 1 principal components is not enough to see the digits, we can start to see it after using 50 principal components. It

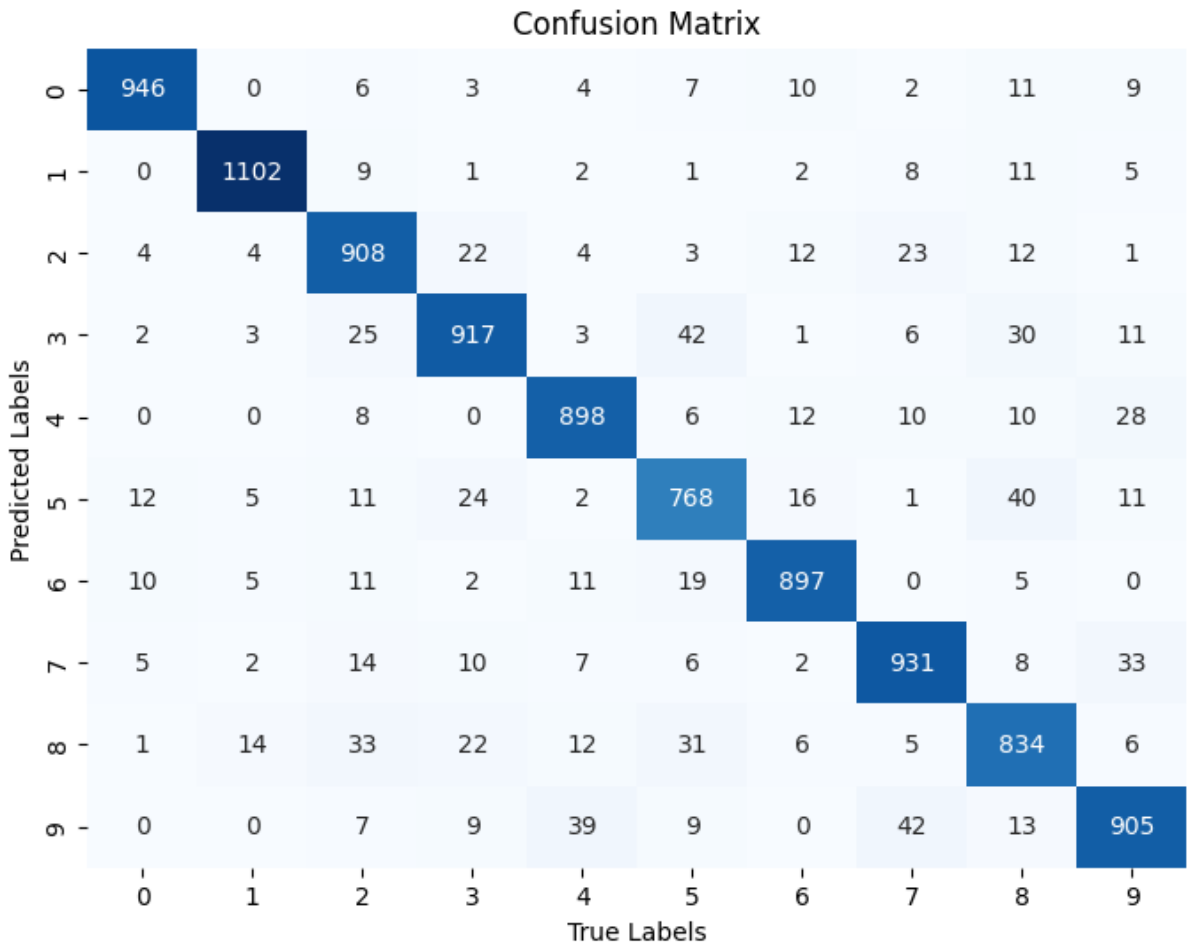
can be said that there is no necessity for using 100 principal components, the digit can be seen at 50 with some blurry.

Using all the principal components gives us nearly original image with more sharply version of it.

Question 2: Multinomial Logistic Regression Classifier

Question 2.1

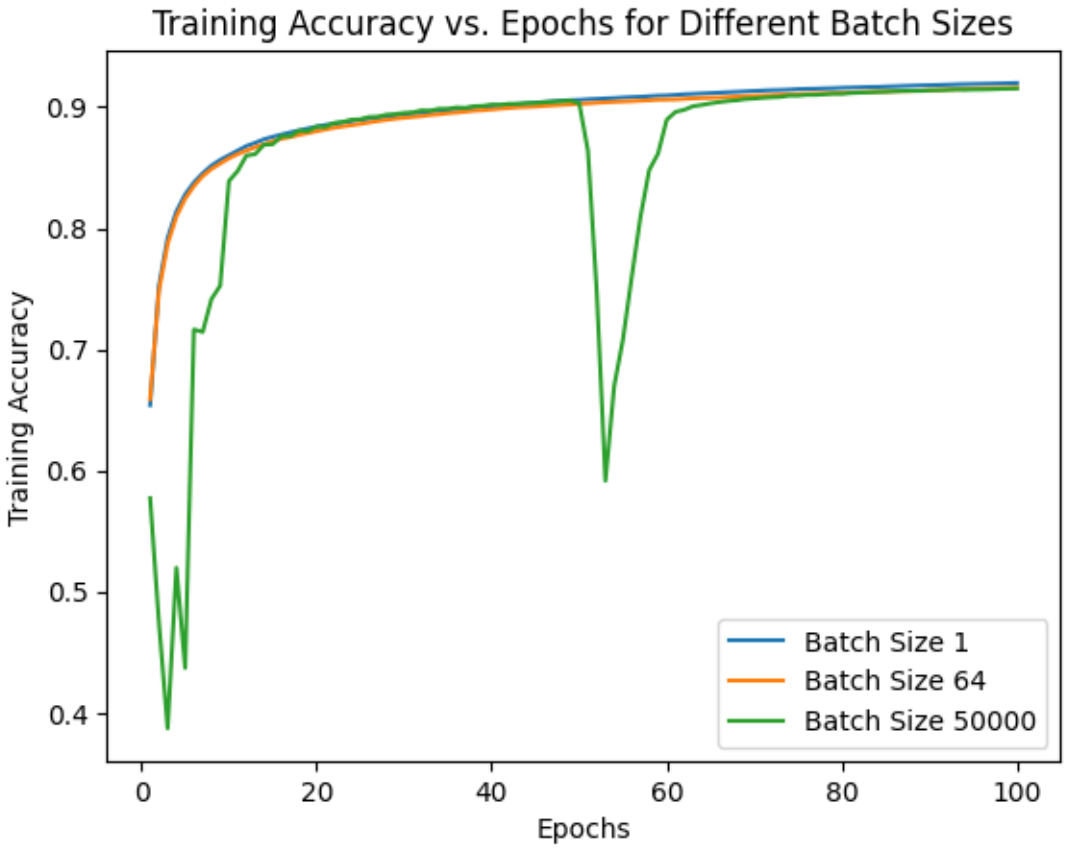
I write a class for Logistic Regression Classification, I write functions to compute the softmax, cross entropy loss and gradients. After computing them, I used them as my tools to train the model in train function. I train the default model and take **91.06**% accuracy on the test data set. And my confusion matrix looks like this:



Plot 2. Confusion Matrix of Default Model Prediction on Test Data

Question 2.2

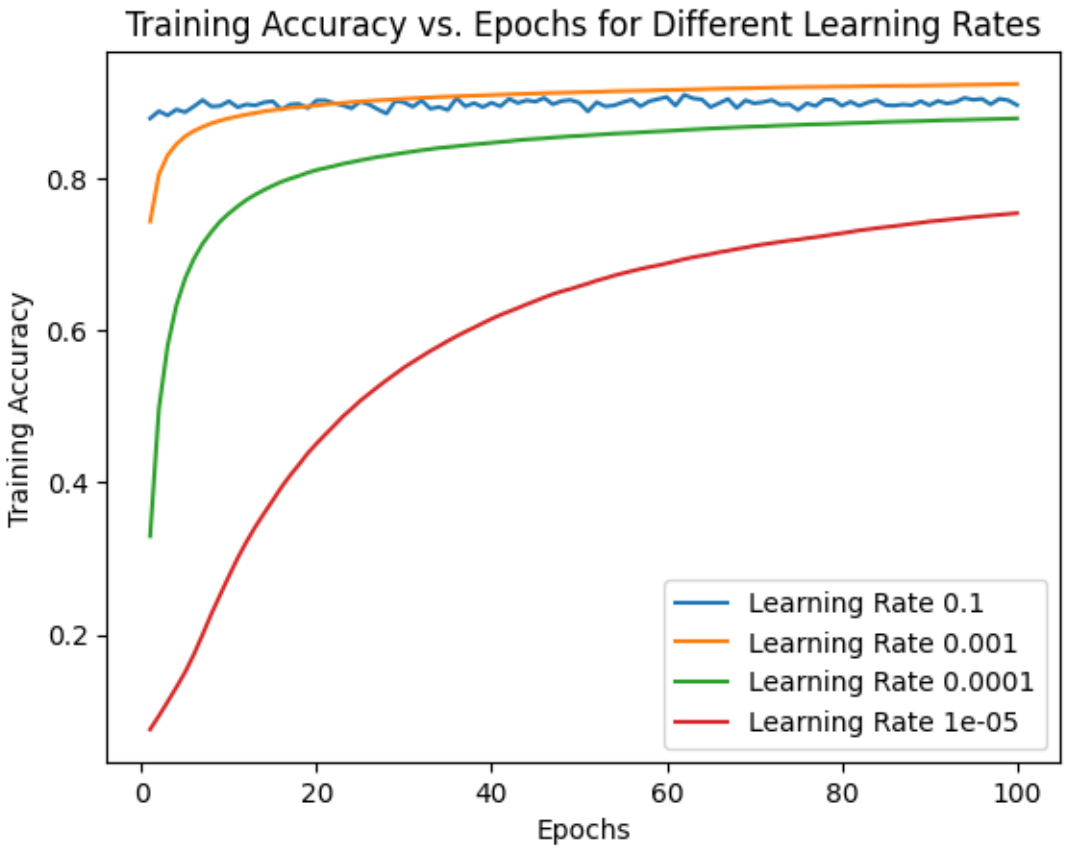
I did separate experiments for each hyperparameter, took different accuracy score on train set. I decided to choose the better performing ones with looking their accuracy vs epoch plots. I tried 13 different model and when I am changing one hyperparameter type, I keep the rest as default model. I started to investigate with changing batch sizes. The result is like this:



Plot 3. Epochs-Accuracy Graph in Different Batch Sizes

With looking this graph, it can easily say that we can eliminate the batch size equals to 50000 among the best model hyperparameters value, because there are fluctuations in the accuracy graph through epochs. The other parameters gives very close results but I will chose the batch size equals to 64 because it has less train time than 1.

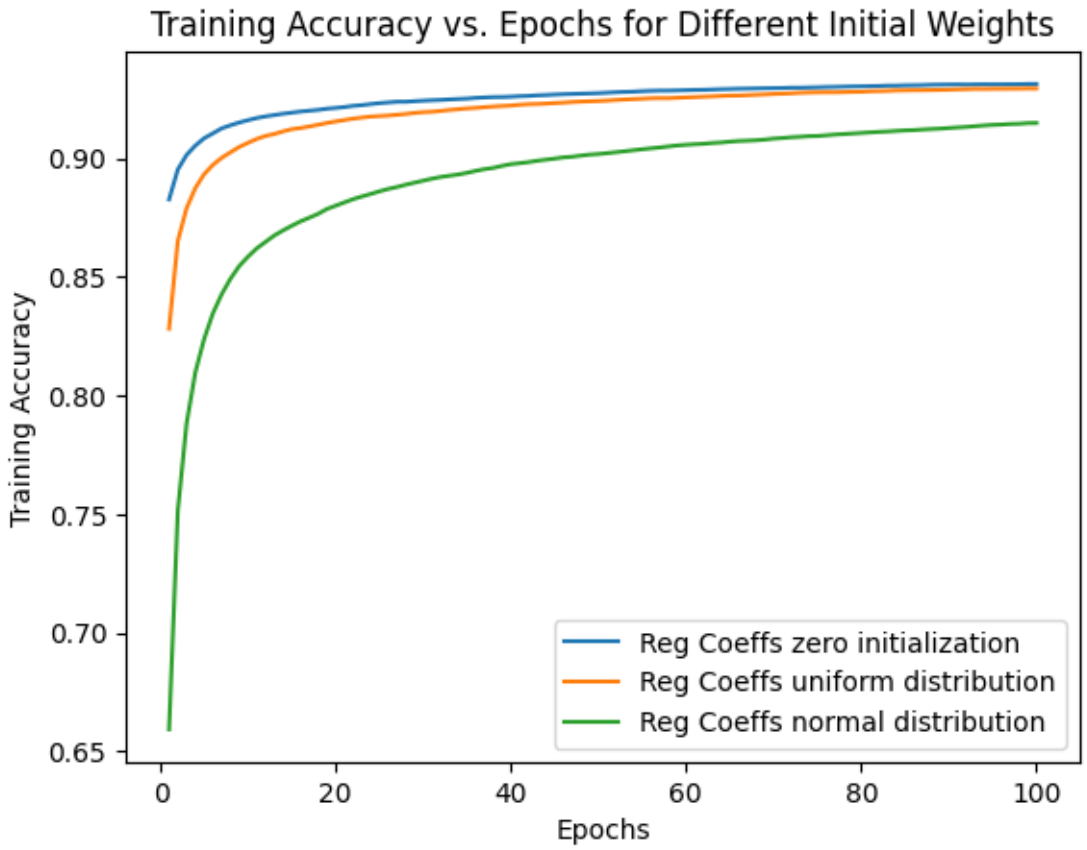
My second investigation was on learning rate. I changed its value 0.1 to 0.00001 and the epoch vs accuracy graph can be seen below.



Plot 4. Epochs-Accuracy Graph in Different Learning Rates

After the observing the graph there are lots of up and downs at learning rate equals to 0.1 even if its value seems like stabile at more than ninety percent. For this hyperparameter I will choose 0.001 as better performing one.

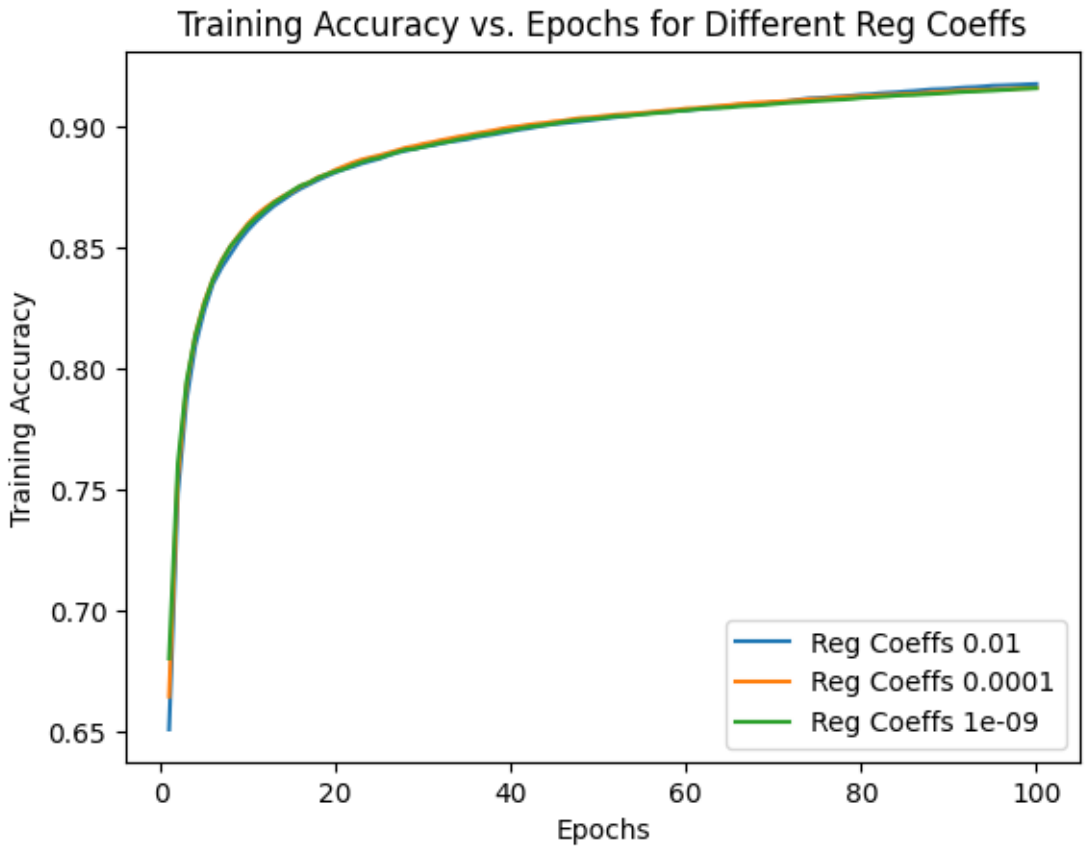
The third investigation on weight initialization technique. I changed it to zero initialization, uniform distribution, and normal distribution. The graph can be seen below.



Plot 5. Epochs-Accuracy Graph in Different Initial Weights

Even if the uniform distribution and make weights as zero at the first gives pretty close results, choosing zero initialization will be better choice.

The last hyperparameter experiment was on regularization coefficients. We used L2 Regularization method to prevent our model from overfitting, so I think it has also had an important effect on our models.



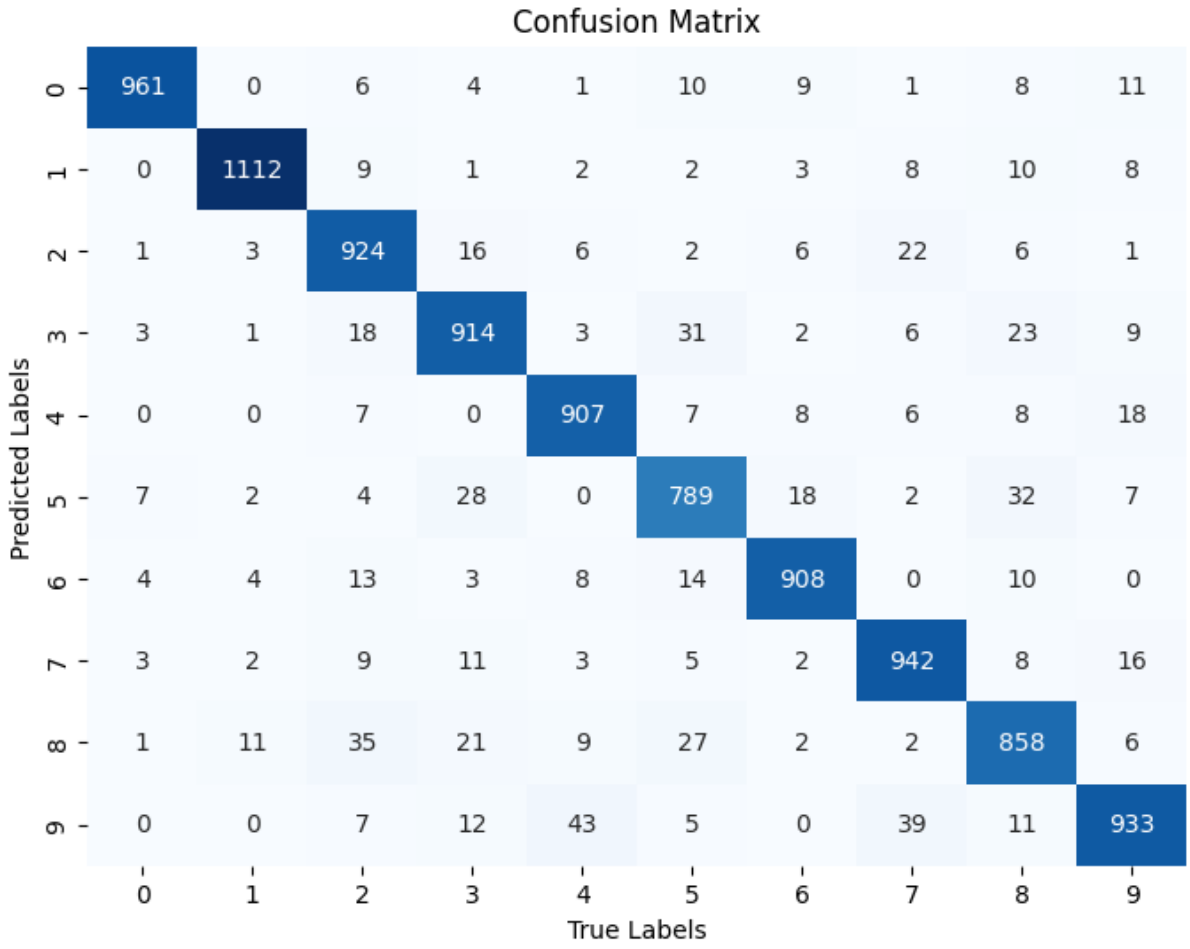
Plot 6. Epochs-Accuracy Graph in Different Regularization Coefficients

They are just like same but I want to choose 1e-9 as my value because it is starting with more accuracy at epoch equals one.

In conclusion, we come to an end with batch size equals 64, weight initialization technique as assigning zero, learning rate equals 0.001 and regularization coefficient equals 1e-9 as our best performance model hyperparameters. Let's look at its accuracy score on test set to confirm its reality.

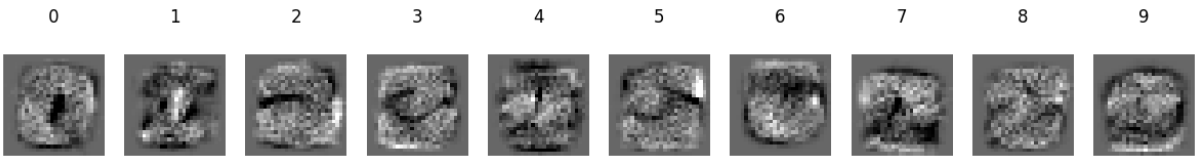
Question 2.3

The best model parameters were assigned in the last question and take an **92.48**% accuracy score at test data. It is more than default models result, so it proves that there is an enhance at model. So, confusion matrix can be seen in the next page.



Plot 7. Confusion Matrix of Best Model Prediction on Test Data

Question 2.4



Picture 3. Weights Images

Weights for 0, 1, 2, 3, 5 can be seen with human eyes but the rest of them I'm having trouble seeing the pattern.

Question 2.5

Best model performance measurements can be seen below table.

Class	0	1	2	3	4	5	6	7	8	9
Precision	0.9505	0.9628	0.9362	0.9050	0.9438	0.8875	0.9419	0.9411	0.8827	0.8886
Recall	0.9806	0.9797	0.8953	0.9050	0.9236	0.8845	0.9478	0.9163	0.8809	0.9247
F1 Score	0.9653	0.9712	0.9153	0.9050	0.9336	0.8860	0.9448	0.9285	0.8818	0.9063
F2 Score	0.9744	0.9763	0.9032	0.9050	0.9276	0.8851	0.9466	0.9212	0.8813	0.9172

Table 1. Best Model Performance Table