CS 464 HW2

Section: 2
Student Name: Utku Kurtulmuş
Student ID: 21903025

Q1)

Question 1.1

Results:
Component 1: PVE = 0.0970
Component 2: PVE = 0.0710
Component 3: PVE = 0.0617
Component 4: PVE = 0.0539
Component 5: PVE = 0.0487
Component 6: PVE = 0.0431
Component 7: PVE = 0.0327
Component 8: PVE = 0.0288
Component 9: PVE = 0.0276
Component 10: PVE = 0.0236

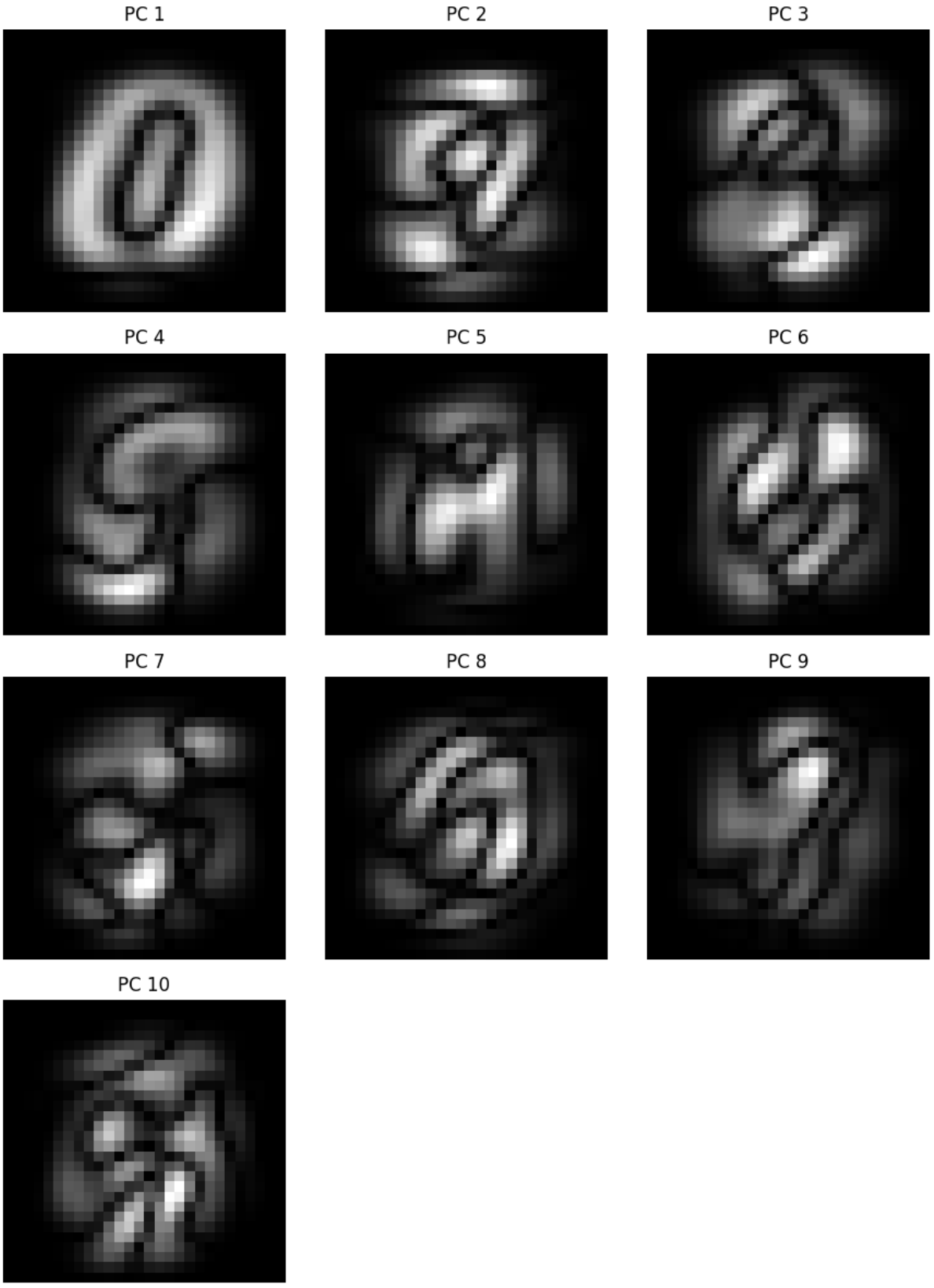
Cumulative PVE for the first 10 components:
[0.09704664 0.16800588 0.22969677 0.28359097 0.33227894 0.37540125
 0.40812055 0.4369595 0.4645798 0.4881498]

These results suggest that with principal component 1, we can capture approximately 10% percent of the variance in the data, and with the first 10 components this number goes up to %48 percent. With 10 principal components we can almost capture half of the variance in the data. But we may want to increase these numbers by increasing the number of components.

Question 1.2

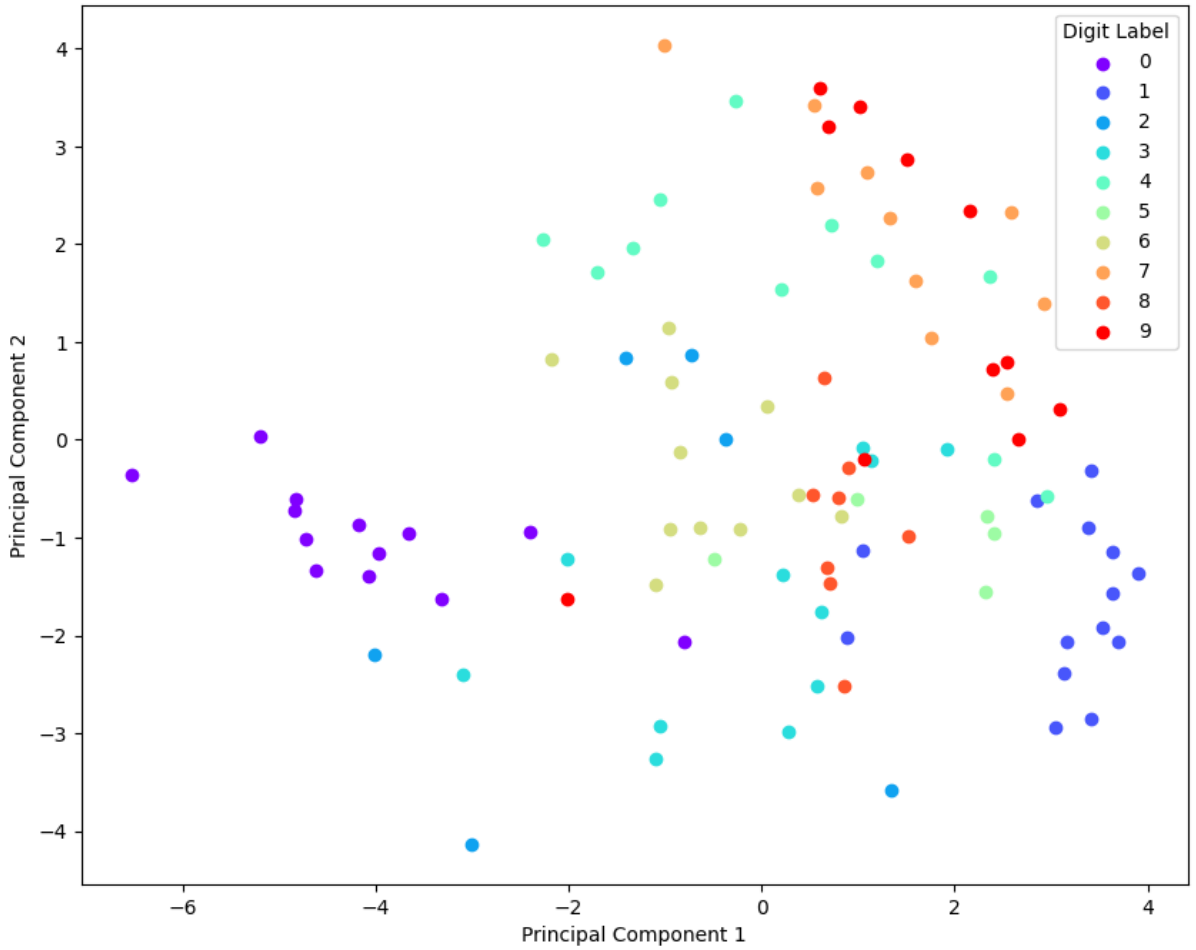
If we increase the number of components from 10 to 26 we can capture %70 percent of the variance in the data. However, if we want to be more greedy and capture %90 percent of the variance in the data, we need to increase this number to 87.

Question 1.3



As we increase the number of components we use we can capture more detail in the image.

Question 1.4



The distribution of the data obtained using the first two principal components looks very similar to the reshaped image of the principal component 2. We can also see some clustering in the data, but they are not very tightly clustered. This suggests that we will need more components to better differentiate the digits.

Question 1.5

To reconstruct an original digit image using principal components we need to project the image data on the principal components.

```
# Specify the values of k
k_values = [1, 50, 100, 250, 500, 784]

# Choose the index of the image to reconstruct (e.g., the first image)
image_index = 0
original_image = images[image_index]
```

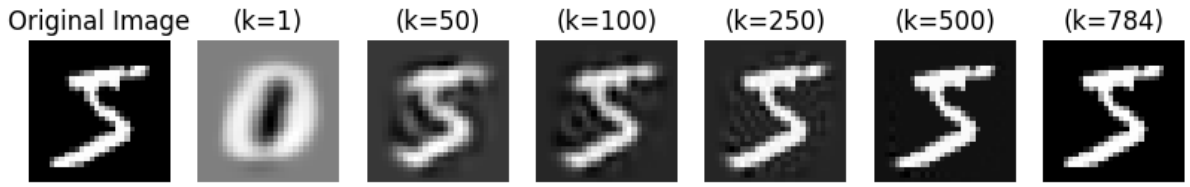
```
# Plot the original image
plt.figure(figsize=(10, 5))
plt.subplot(2, len(k_values) + 1, 1)
plt.imshow(original_image.reshape(28, 28), cmap='gray')
plt.title('Original Image')
plt.axis('off')

# Reconstruct the image using different values of k
for i, k in enumerate(k_values):
    # Project the original image onto the first k principal components
    projected_data_k = np.dot(original_image, all_eigenvectors[:, :k])

    # Reconstruct the image using the projected data and the first k
principal components
    reconstructed_image = np.dot(projected_data_k, all_eigenvectors[:,
:k].T).astype(float)

    # Plot the reconstructed image
    plt.subplot(2, len(k_values) + 1, i + 2)
    plt.imshow(reconstructed_image.reshape(28, 28), cmap='Greys_r')
    plt.title(f'(k={k})')
    plt.axis('off')

plt.show()
```



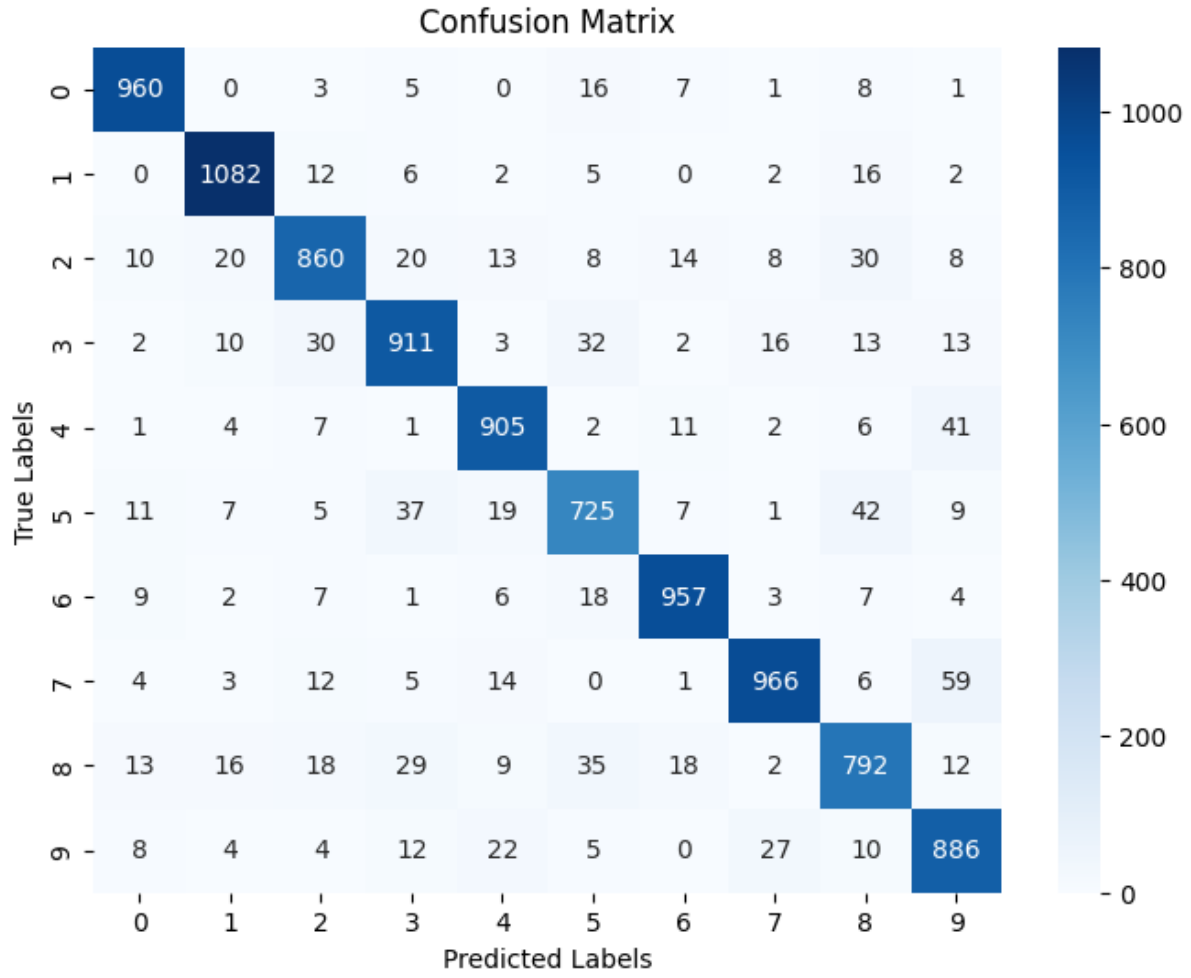
As we increase the number of principal components we can capture more detail in the original image. As we can see also when we use only the first principal component, and project the image on this component the shape is very similar to the shape of the principal component. Moreover, as we calculated previously with the first principal component, we can get 10% of the variance in the data, and this is not enough for us to classify the image correctly. However, as we increase the number to 26 principal components we can get 70% of the variance in the data, and with 86 components this number goes to 90%. Therefore, with 50 components, we are able to identify the reconstructed image correctly.

Q2)

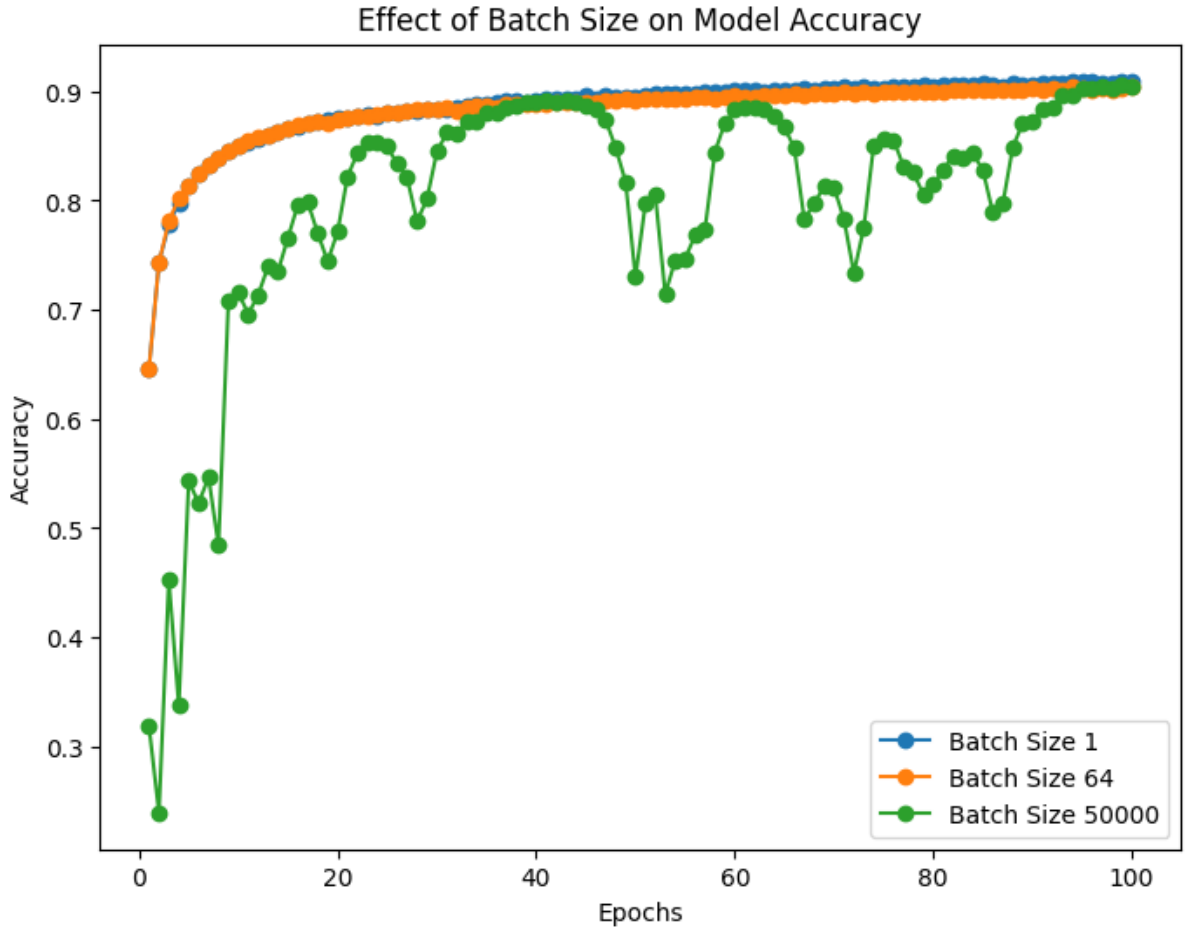
Question 2.1)

Validation Accuracy: 90.44%

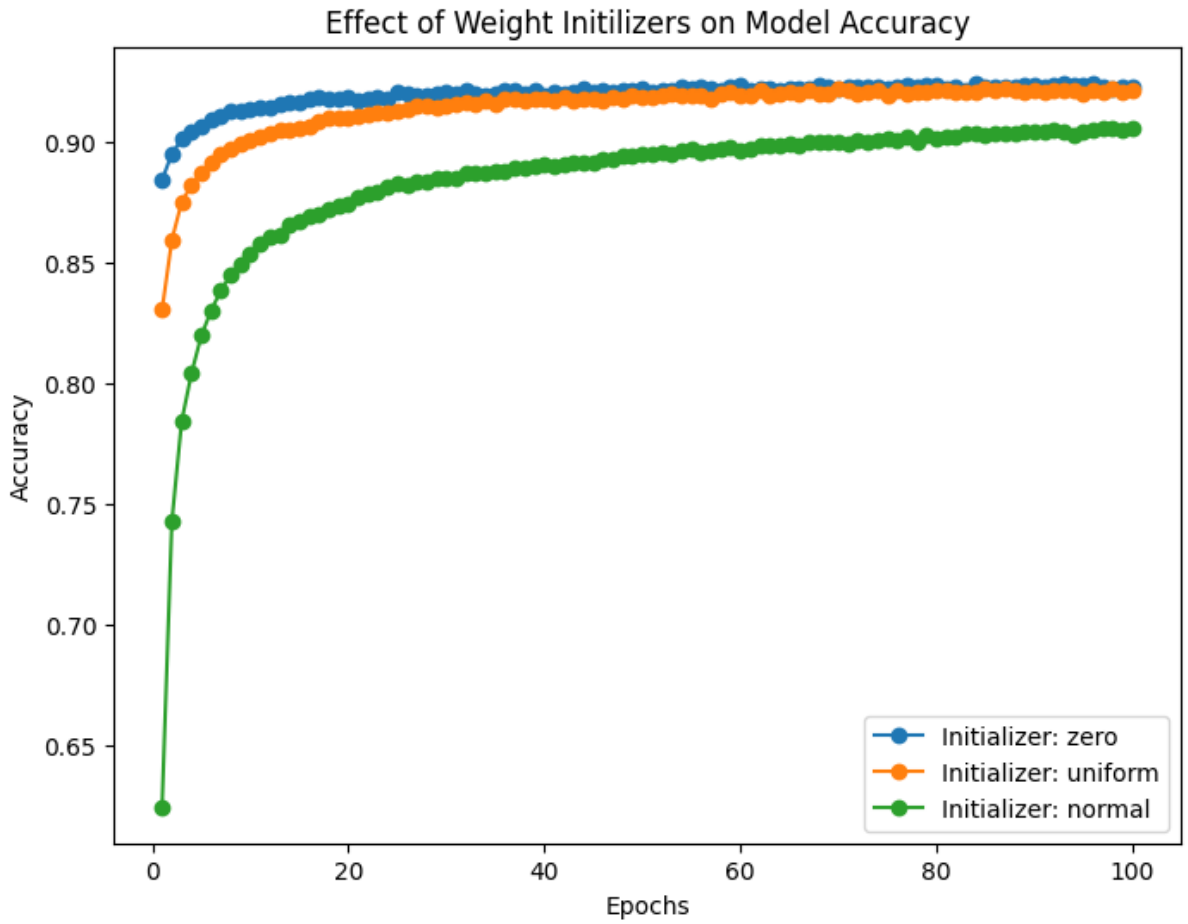
Confusion Matrix:



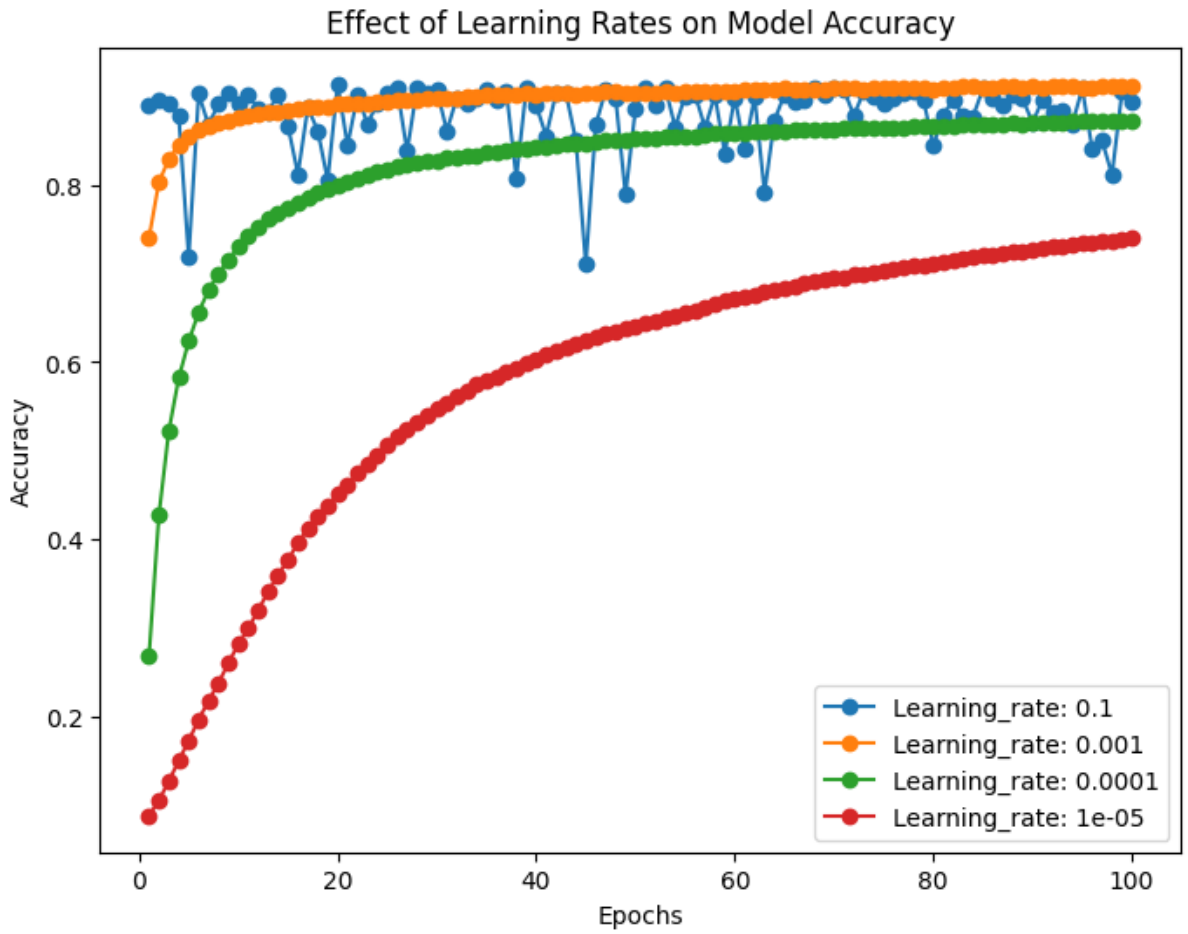
Question 2.2)



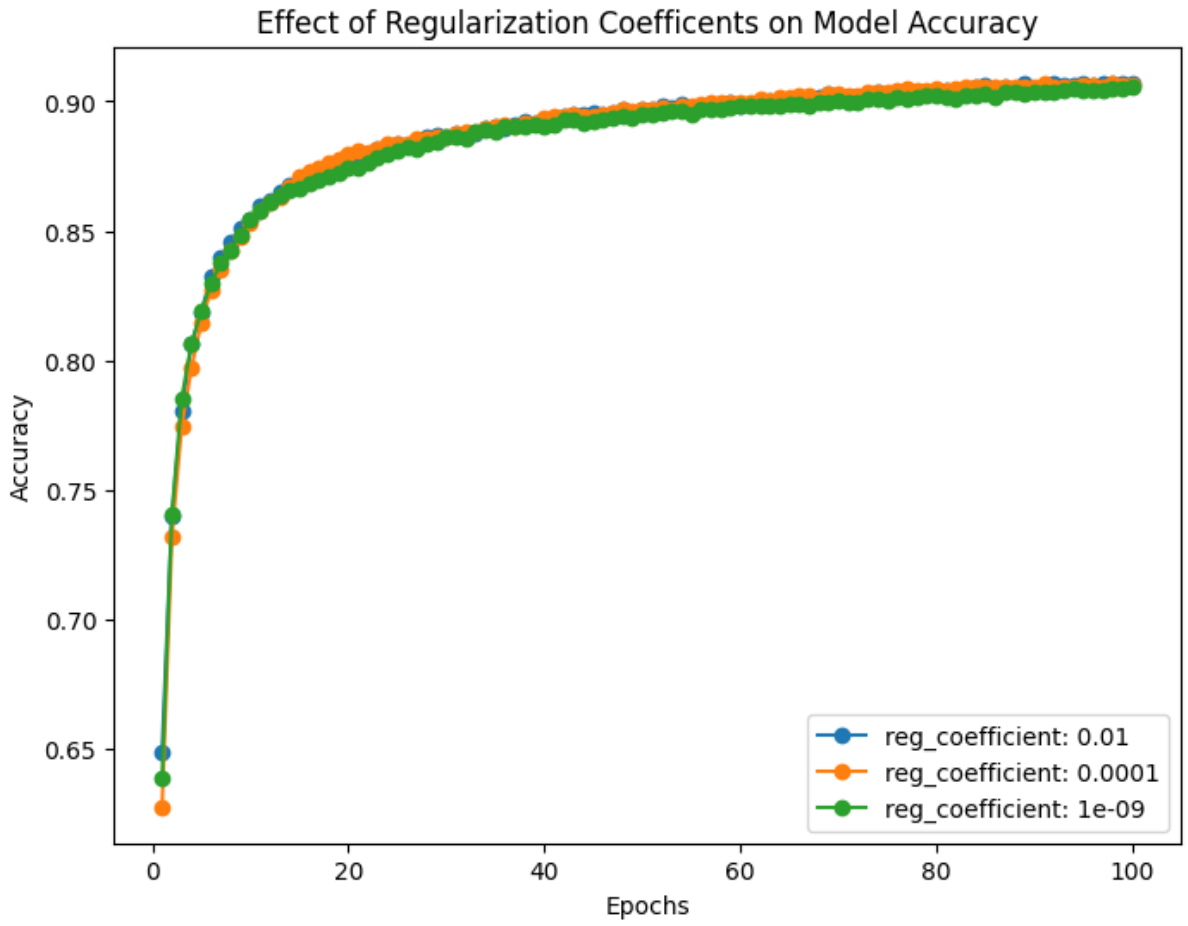
Winner: Batch Size 1 with a slight difference. (However, batch size 1 takes significantly more time to train, choosing batch size 64 is a better approach.)



Winner: Zero initialization with a slight difference.



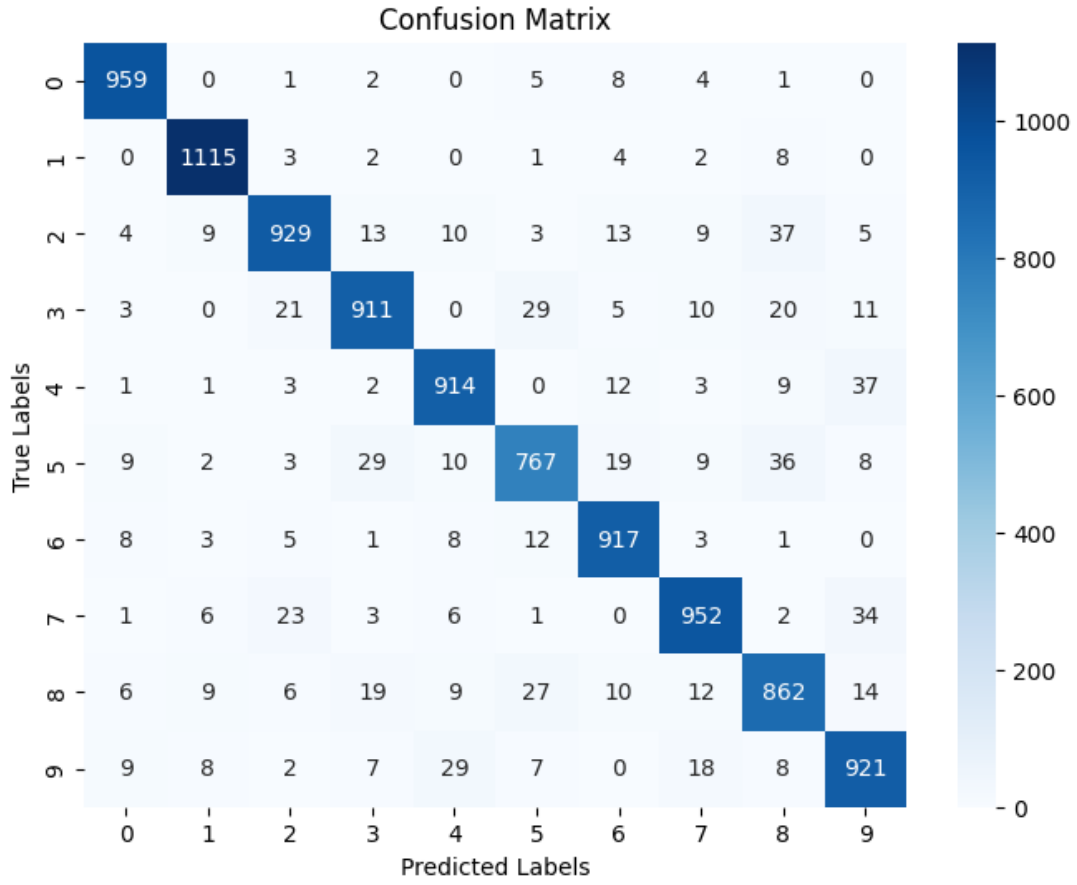
Winner: Learning_rate 0.001. Learning_Rate:0.1 sometimes passes the 0.001, but it oscillates too much and is unstable.



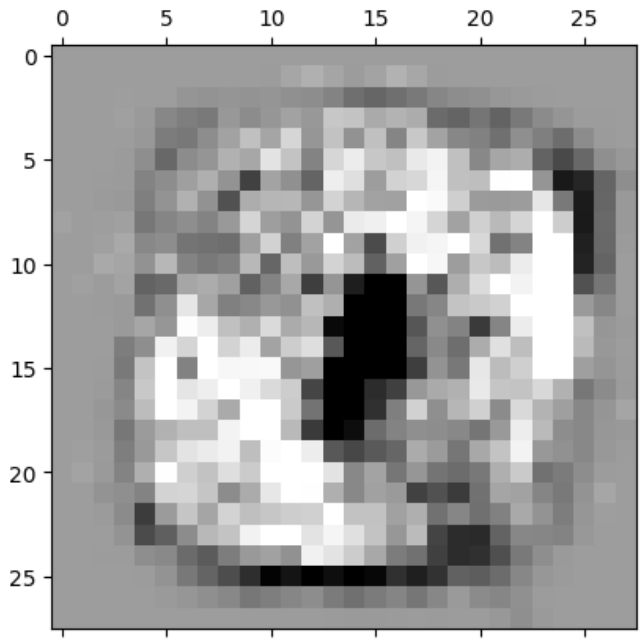
Winner: reg_coefficient 0.01 with a slight difference.

Question 2.3
Parameters for the best model:
num_classes = 10
input_dim = X_train.shape[1]
learning_rate = 1e-3
batch_size = 64
initializer = 'zero'
l2_reg_coefficient = 1e-2
epochs = 100

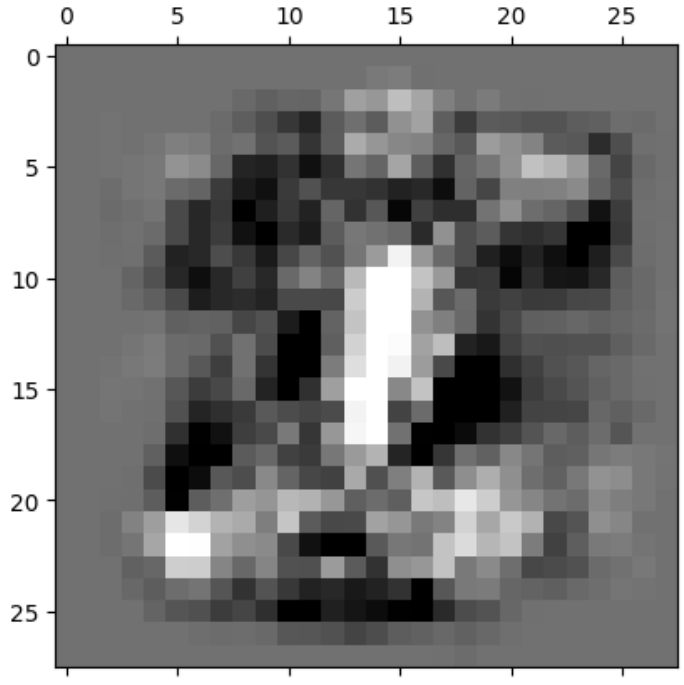
Test Accuracy: 92.47%



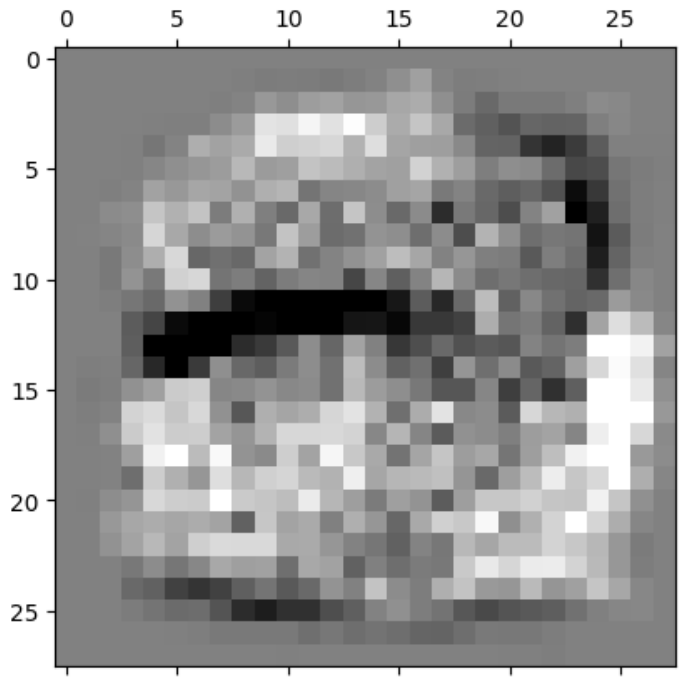
Question 2.4



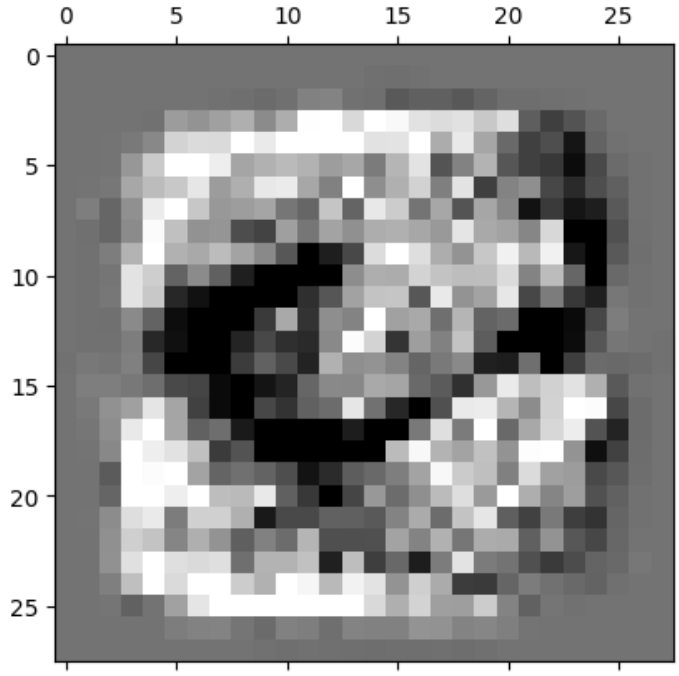
This weight vector looks like 0.



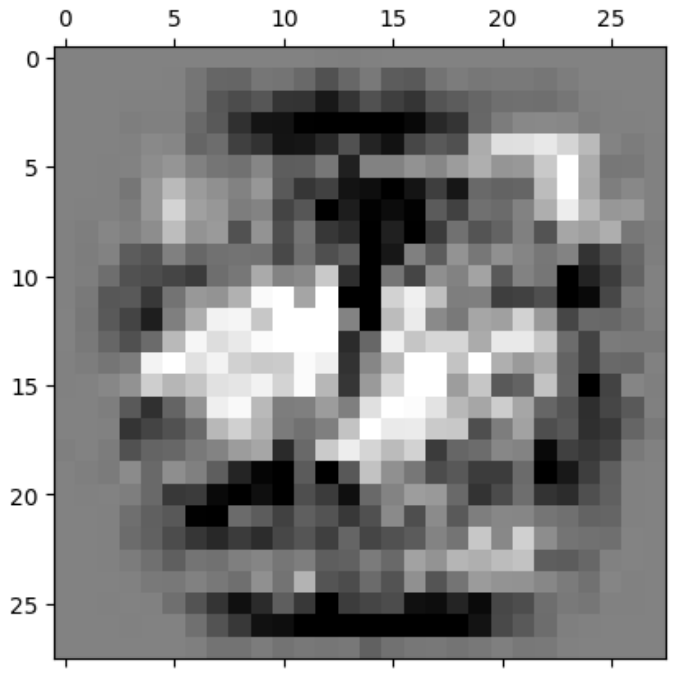
This weight vector looks like 1 and 2. It is hard to classify.



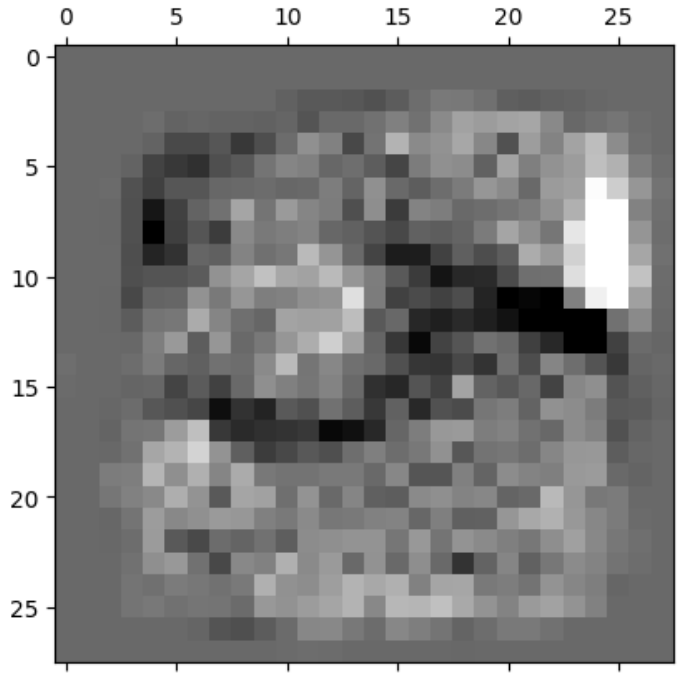
This weight vector looks like 2.



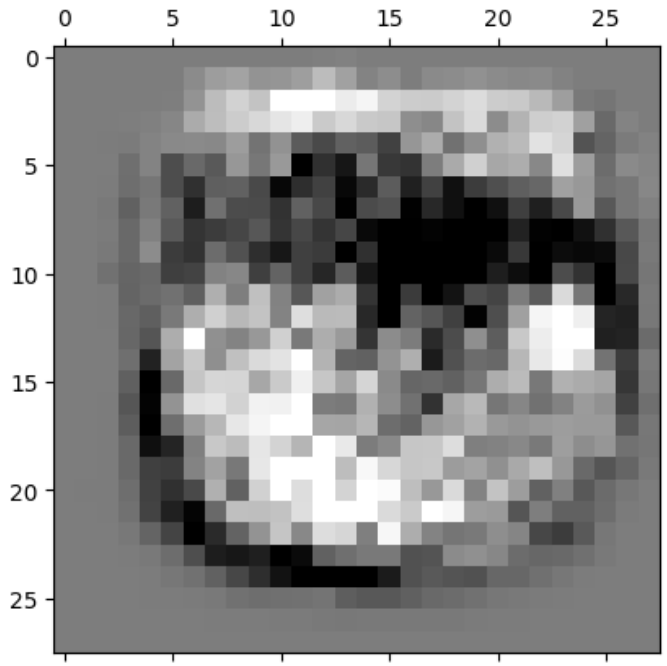
This weight vector looks like 3.



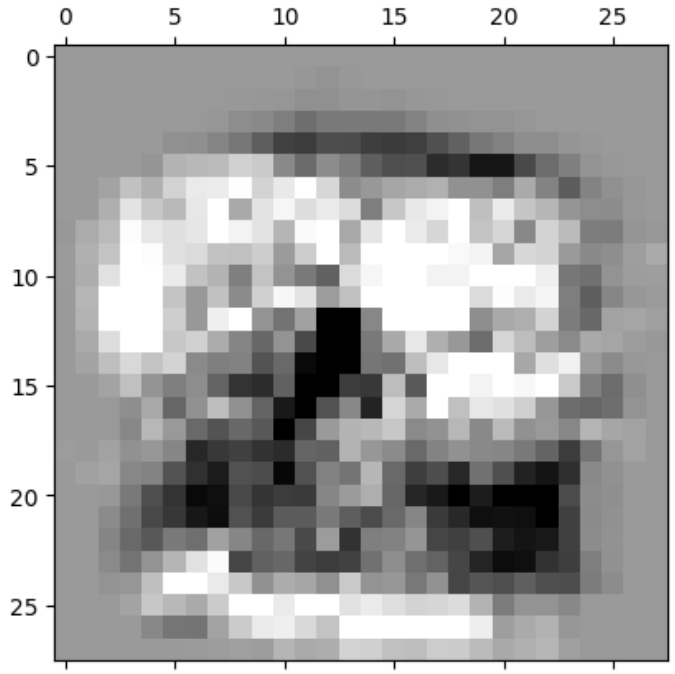
It is hard to classify but looks like 4.



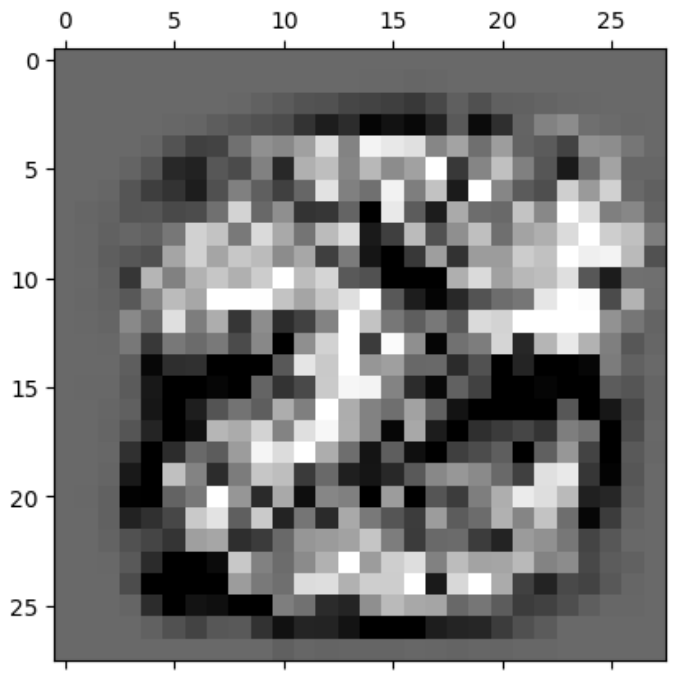
This weight vector looks like 5.



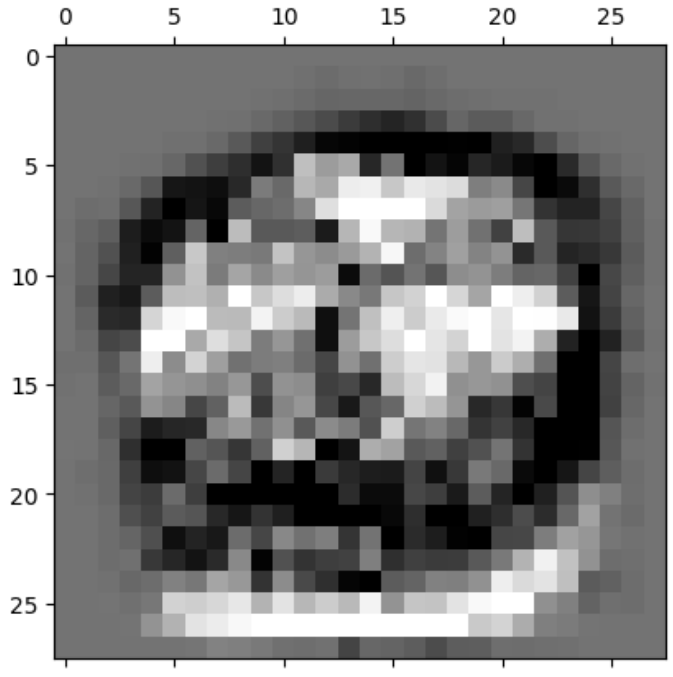
This weight vector looks like 6.



This weight vector looks like 7 and 2.



This weight vector looks like 8.



This weight vector looks like 9

Question 2.5

Class 0: Precision = 0.9610, Recall = 0.9806, F1 Score = 0.9707, F2 Score = 0.9766
The model is able to correctly identify the 0 values. From the weight images we can also see that 0 is easy to identify. Moreover, from the confusion matrix, we can see that the model is mostly able to identify the 0 value they are consistent as expected.

Class 1: Precision = 0.9695, Recall = 0.9806, F1 Score = 0.9750, F2 Score = 0.9784
The model is able to correctly identify the 1 value. However, from the weight image, I find it a little bit difficult to identify the digit as 1, but it doesn't look like any other digit too except the number 2. Additionally, from the confusion matrix, we can see that the model is mostly able to identify the 1 value they are consistent as expected.

Class 2: Precision = 0.9394, Recall = 0.8857, F1 Score = 0.9117, F2 Score = 0.8959
The model has some problems with recalling the value 2 compared to 0 and 1, but it is still not a bad result. From the weight image, I can identify it as 2 but maybe the model mixed it 5. From the confusion matrix, the model mostly mismatched the 2 with 8, not 5, which is surprising to me.

Class 3: Precision = 0.9003, Recall = 0.9119, F1 Score = 0.9061, F2 Score = 0.9095
The model was mostly able to identify the number 3, but it was worse than the performance on 0 and 1. However, precision and recall values are close to each other, not like in the case of number 2. From the weight image, I can easily identify the number as 3, but it looks like it wasn't as clear as 0 and 1 to the model. From the confusion matrix, the model mostly mismatched the 3 with 5.

Class 4: Precision = 0.9281, Recall = 0.9338, F1 Score = 0.9310, F2 Score = 0.9327
The model did a better job on 4 compared to 3 but worse than 1 and 0. The weight image is also not so clear, but for the model, it looks like it was. The model mismatched 4 mostly with 9 according to the confusion matrix.

Class 5: Precision = 0.9018, Recall = 0.8643, F1 Score = 0.8827, F2 Score = 0.8716
The model did a poor job on 5 compared to other numbers. The weight image is not super clear, but I can identify it as 5. The model mismatched 5 mostly with 3 and 8 according to the confusion matrix.

Class 6: Precision = 0.9374, Recall = 0.9530, F1 Score = 0.9451, F2 Score = 0.9499
The model did a good job on 6 compared to other numbers except the 0 and 1. The weight image is not super clear, but I can identify it as 6. The model mismatched 6 mostly with 5 according to the confusion matrix.

Class 7: Precision = 0.9439, Recall = 0.9163, F1 Score = 0.9299, F2 Score = 0.9217
The model did a good job on 7 compared to other numbers except the 0 and 1. However, recall of the model is low compared to the precision. This indicates that the model has some

problems with recalling the instances of the class. The weight image is hard to identify it looks both like 7 and 2. The model mismatched 7 mostly with 2 and 9 according to the confusion matrix.

Class 8: Precision = 0.8571, Recall = 0.8994, F1 Score = 0.8778, F2 Score = 0.8906
The model did a poor job on 8 compared to other numbers. The weight image is not super clear, but I can identify it as 8. The model mismatched 8 mostly with 3 and 5 according to the confusion matrix.

Class 9: Precision = 0.9048, Recall = 0.9138, F1 Score = 0.9093, F2 Score = 0.9120
The model did an average job on 9 compared to other numbers. The weight image is not super clear, but I can identify it as 9. The model mismatched 9 mostly with 4 and 7 according to the confusion matrix.