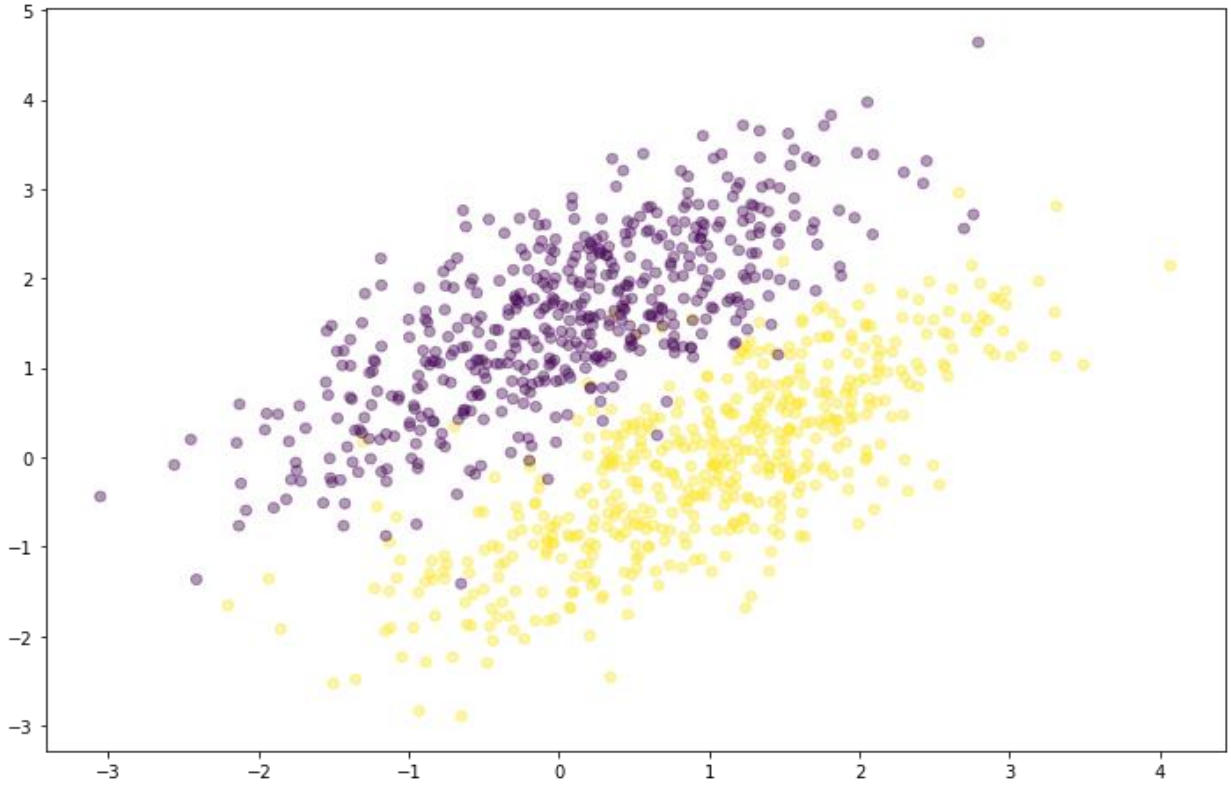
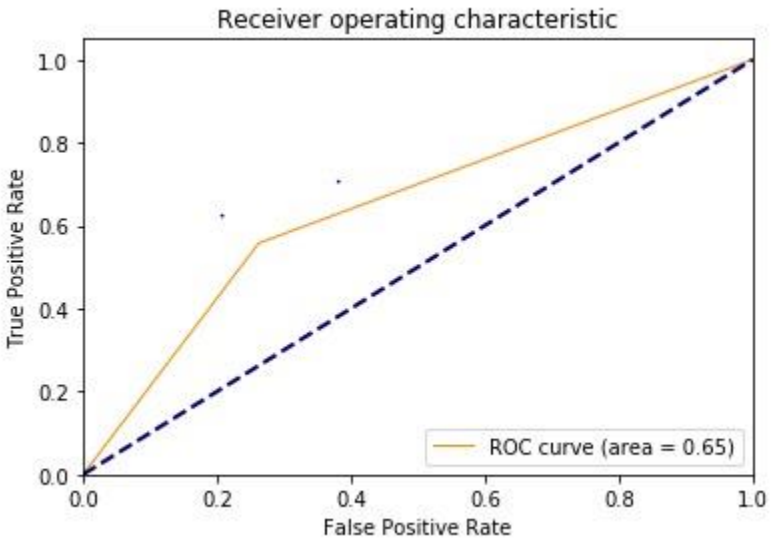
Name: Samarth Manjunath
UTA ID: 1001522809
Assignment-3
Data Mining

Problem-1
1] Batch training

a. The figure shows the scatter
plot of the testing data and the trained decision boundary.

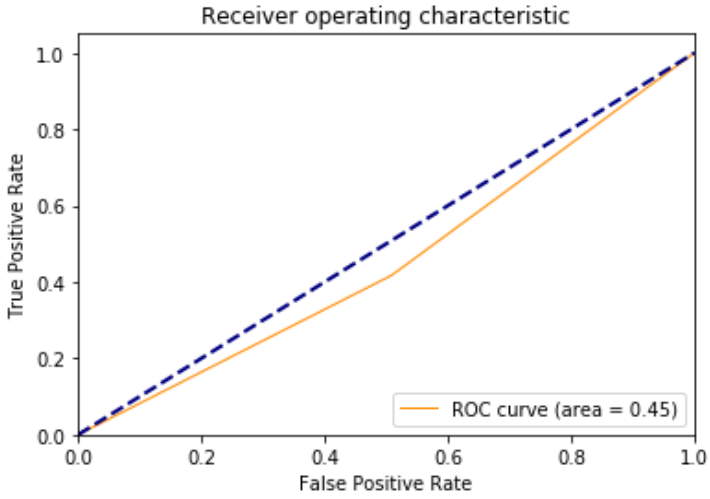


b. Figure of changes of training error (cross entropy) w.r.t. iteration and Figure of changes of norm of gradient w.r.t iteration.
For learning rate=1
Learning Rate=1
Iteration= 1 Cost= 1.5079169673734414
Weights=[-0.76576612 -1.21433115 -2.58130123]
Accuracy from scratch: 0.5925
Result on Test Data
Accuracy from scratch: 0.608
Roc Curve
Accuracy : 0.5925



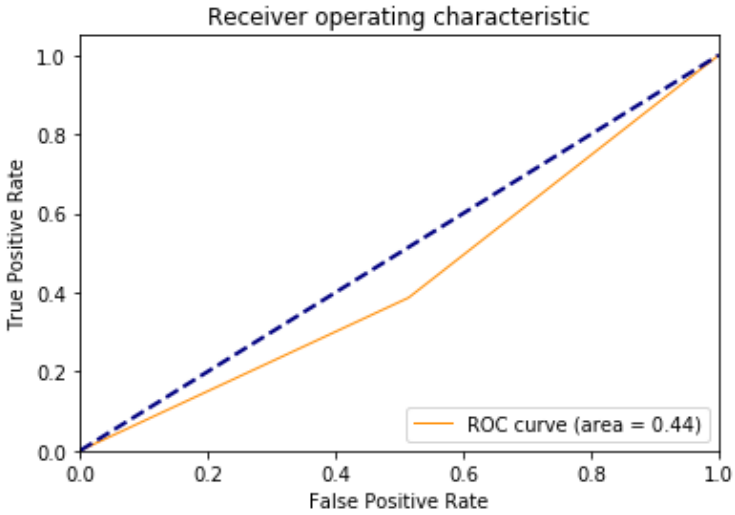
For learning rate=0.1

```
Learning Rate=0.1
Iteration= 1 Cost= 1.50791696737
Weights=[ 0.82342339  0.77856688  0.64186988]
Accuracy from scratch: 0.484
Result on Test Data
Accuracy from scratch: 0.478
Roc Curve
Accuracy  : 0.484
```

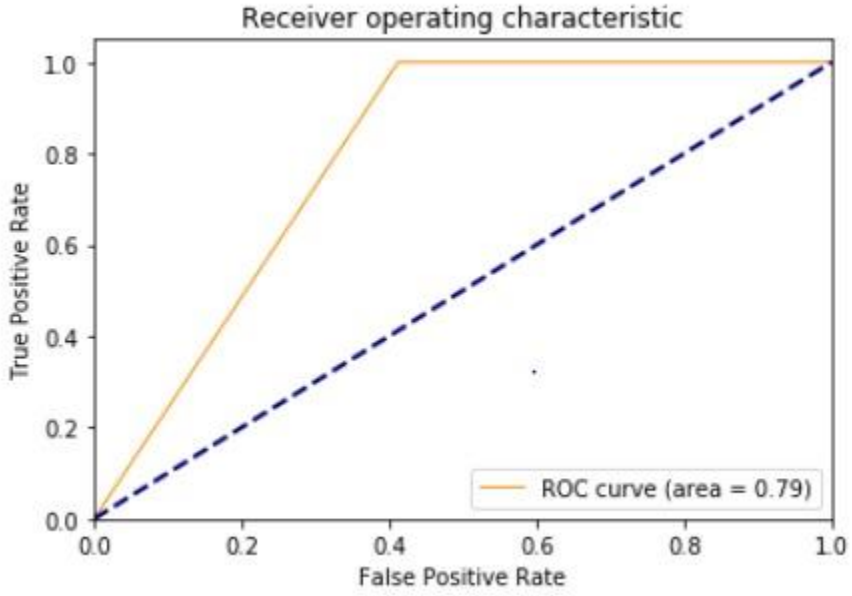


For learning rate=0.01

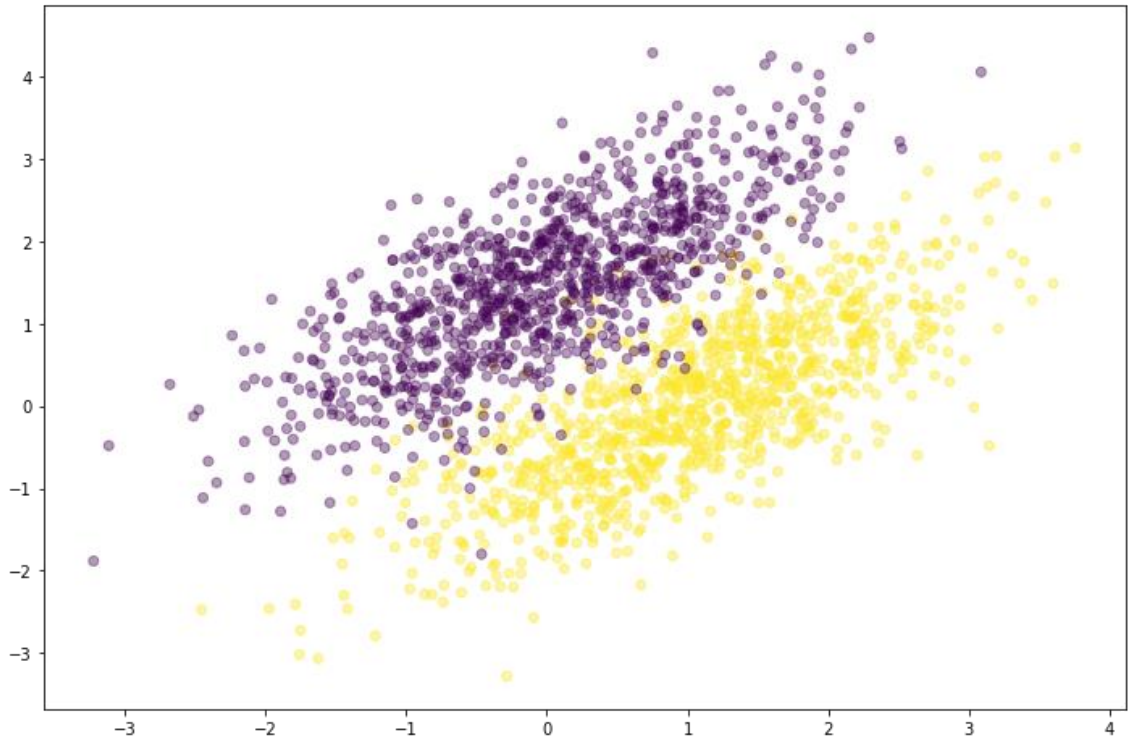
```
Learning Rate=0.01
Iteration= 1 Cost= 1.50791696737
Weights=[ 0.98234234  0.97785669  0.96418699]
Accuracy from scratch: 0.4745
Result on Test Data
Accuracy from scratch: 0.463
Roc Curve
Accuracy  : 0.4745
```



Learning Rate=0.001
Iteration= 1 Cost= 1.5079169673734414
Weights=[0.99823423 0.99778567 0.9964187]
Accuracy from scratch: 0.4735
Result on Test Data
Accuracy from scratch: 0.461
Roc Curve
Accuracy : 0.4735



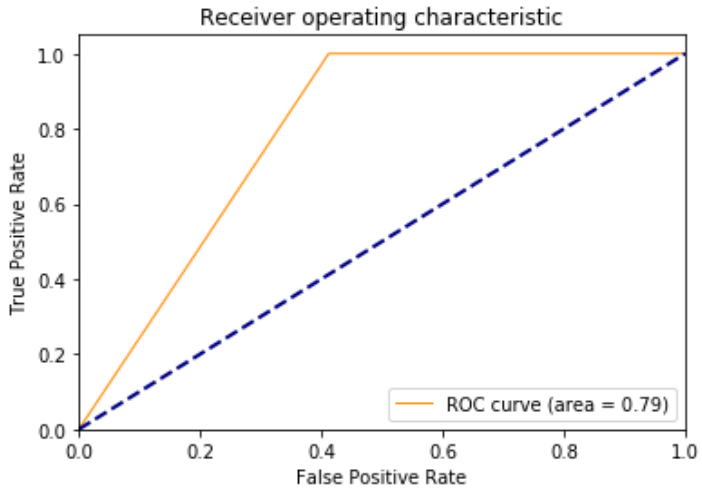
2] **Online training**
a. The figure shows the scatter
plot of the testing data and the trained decision boundary.



b. Figure of changes of training error (cross entropy) w.r.t. iteration and Figure of changes of norm of gradient w.r.t iteration.

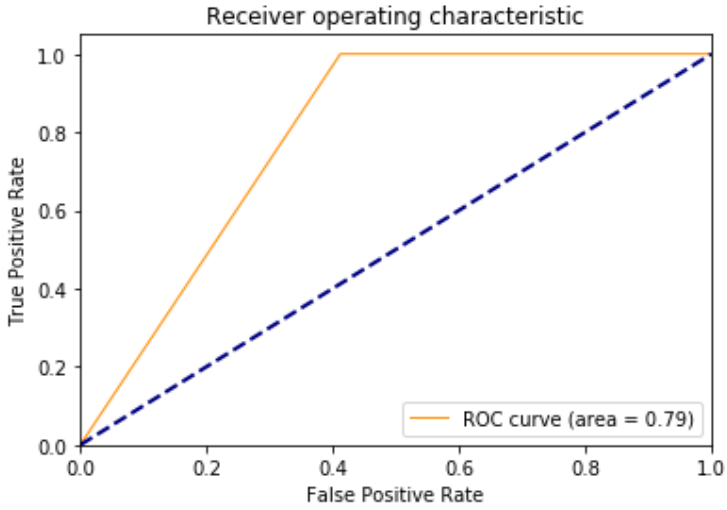
Learning rate=0.1

```
Learning Rate=0.1
Iteration= 470 Cost= 0.555995453752
weights[ 0.56981147  0.30314135 -0.3027044 ]
Accuracy from scratch: 0.6495
Result on Test Data
Accuracy from scratch: 0.653
Roc Curve
Accuracy  : 0.6495
```

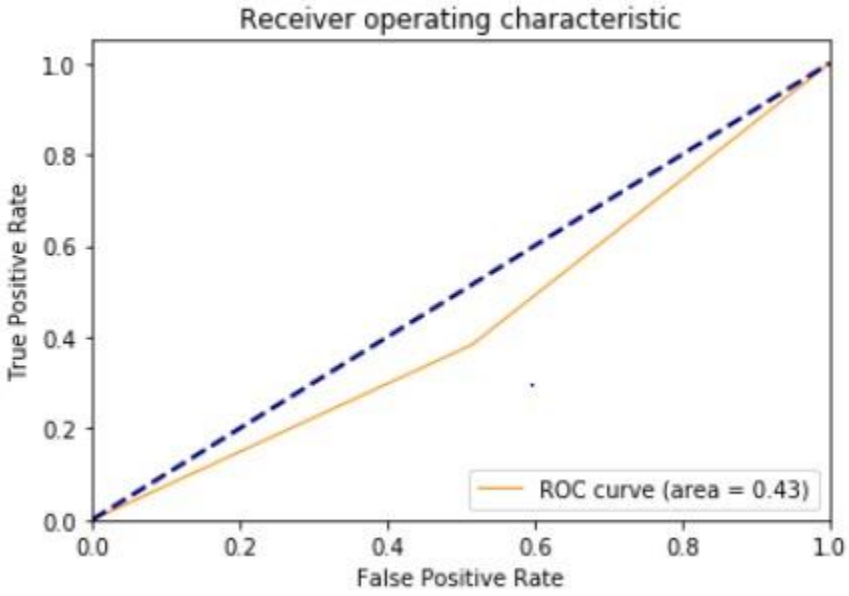


Learning rate=0.01

```
Learning Rate=0.01
Iteration= 4405 Cost= 0.555995453752
weights[ 0.56981147  0.30314135 -0.3027044 ]
Accuracy from scratch: 0.6495
Result on Test Data
Accuracy from scratch: 0.653
Roc Curve
Accuracy  : 0.6495
```



Learning Rate=0.001
Iteration= 39856 Cost= 0.5559954537518565
weights[0.56981147 0.30314135 -0.3027044]
Accuracy from scratch: 0.6495
Result on Test Data
Accuracy from scratch: 0.653
Roc Curve
Accuracy : 0.6495



Problem 2

***1. (10pt) In the report, write comments for each line of code given above and explain what this* framework is doing.**

```
#Importing the tensorflow library as tf
import tensorflow as tf
#Loading the MNIST dataset. MNIST is a dataset of handwritten digits.
mnist = tf.keras.datasets.mnist
#Loading training and testing data.
(x_train, y_train),(x_test, y_test) = mnist.load_data()
#Converting the integers to floating points.
 x_train, x_test = x_train / 255.0, x_test / 255.0
```

```
#Sequential model is constructed.
model = tf.keras.models.Sequential([
#The first layer Flatten transforms the image to  1d array.
tf.keras.layers.Flatten(),
#This layer after flatten has 512 nodes ,gives a probability values as array of 512 values showing which class the input data classifies to.
 tf.keras.layers.Dense(512, activation=tf.nn.relu),
#Dropout rate= 0.2 drops which means 20% is a modeling error
tf.keras.layers.Dropout(0.2),
#Layer has 10 nodes and uses softmax activation function to give an output of 10 probability values.
tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
#Before compiling we'll configure few settings.
#Optimizer is used to update the model based on the loss function.
#Loss function is used to check how accurate the model is.
#Metrics are made use to monitor the training and testing data.
model.compile(optimizer='adam',loss='sparse_categorical_crossentropy', metrics=['accuracy'])
#We input the training samples and their labels as input to the model to learn the data.
# Epochs gives how many times we want to run the training data on the model to update it.
model.fit(x_train, y_train, epochs=5)
#Then we test the model by giving the testing samples and labels, then check the accuracy and loss determining how well our model makes predictions for the data.
model.evaluate(x_test, y_test)
```

2. (10pt) Change the number of hidden nodes to 5, 10, 128 and 512. Report how the testing accuracy changes for the testing data. Report the result and your observation in the report.

5 hidden nodes are present: Accuracy= 85.55%

```
Epoch 1/5
60000/60000 [==============================] - 3s 56us/sample - loss: 1.2737 - acc: 0.5402
Epoch 2/5
60000/60000 [==============================] - 3s 53us/sample - loss: 1.0187 - acc: 0.6361
Epoch 3/5
60000/60000 [==============================] - 3s 53us/sample - loss: 0.9759 - acc: 0.6516
Epoch 4/5
60000/60000 [==============================] - 3s 53us/sample - loss: 0.9530 - acc: 0.6615
Epoch 5/5
60000/60000 [==============================] - 3s 53us/sample - loss: 0.9413 - acc: 0.6725
10000/10000 [==============================] - 0s 38us/sample - loss: 0.5661 - acc: 0.8555
[0.5660864552497864, 0.8555]
```

10 hidden nodes are present: Accuracy =91.8%

```
Epoch 1/5
60000/60000 [==============================] - 3s 56us/sample - loss: 0.8630 - acc: 0.7200
Epoch 2/5
60000/60000 [==============================] - 3s 55us/sample - loss: 0.6043 - acc: 0.8032
Epoch 3/5
60000/60000 [==============================] - 3s 55us/sample - loss: 0.5624 - acc: 0.8148
Epoch 4/5
60000/60000 [==============================] - 3s 54us/sample - loss: 0.5454 - acc: 0.8194
Epoch 5/5
60000/60000 [==============================] - 3s 55us/sample - loss: 0.5362 - acc: 0.8220
10000/10000 [==============================] - 0s 37us/sample - loss: 0.2966 - acc: 0.9180
[0.2965687126159668, 0.918]
```

128 hidden nodes are present: Accuracy = 97.98%

```
Epoch 1/5
60000/60000 [==============================] - 9s 142us/sample - loss: 0.2924 - acc: 0.9155
Epoch 2/5
60000/60000 [==============================] - 9s 146us/sample - loss: 0.1415 - acc: 0.9579
Epoch 3/5
60000/60000 [==============================] - 7s 115us/sample - loss: 0.1079 - acc: 0.9672
Epoch 4/5
60000/60000 [==============================] - 8s 126us/sample - loss: 0.0870 - acc: 0.9731
Epoch 5/5
60000/60000 [==============================] - 8s 139us/sample - loss: 0.0746 - acc: 0.9764
10000/10000 [==============================] - 1s 57us/sample - loss: 0.0678 - acc: 0.9798
[0.06783388150855899, 0.9798]
```

512 hidden nodes are present: Accuracy = 97.92%

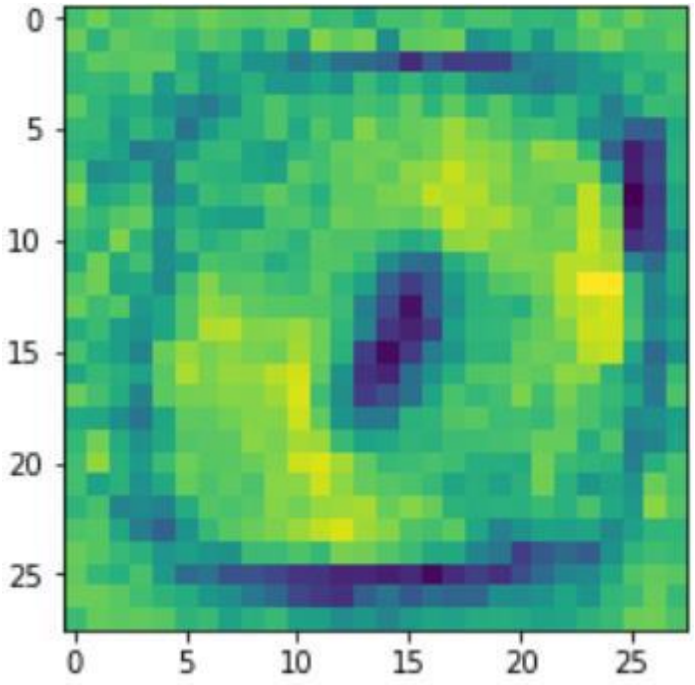
```
Epoch 1/5
60000/60000 [==============================] - 13s 218us/sample - loss: 0.2215 - acc: 0.9347
Epoch 2/5
60000/60000 [==============================] - 13s 210us/sample - loss: 0.0978 - acc: 0.9699
Epoch 3/5
60000/60000 [==============================] - 12s 208us/sample - loss: 0.0684 - acc: 0.9786
Epoch 4/5
60000/60000 [==============================] - 13s 209us/sample - loss: 0.0548 - acc: 0.9829
Epoch 5/5
60000/60000 [==============================] - 13s 209us/sample - loss: 0.0449 - acc: 0.9857
10000/10000 [==============================] - 1s 61us/sample - loss: 0.0697 - acc: 0.9792
[0.06969205481236568, 0.9792]
```

3. (20pt) Now, remove the hidden layer in the code and train the model. The trained model contains the weights that it has learned from training. Plot the "new representation" that it has learned for each number from training and include them in the report. That is, reshape the learned weights (i.e., vector) to the image dimension (in 2D, i.e., 28x28) and show them. You will see some number-like features.

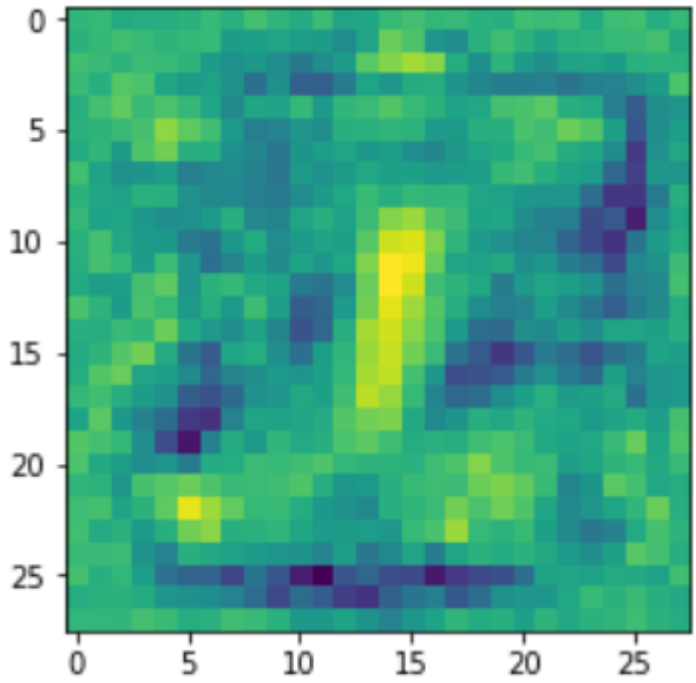
The learned weights are as follows from (0-9):

==========

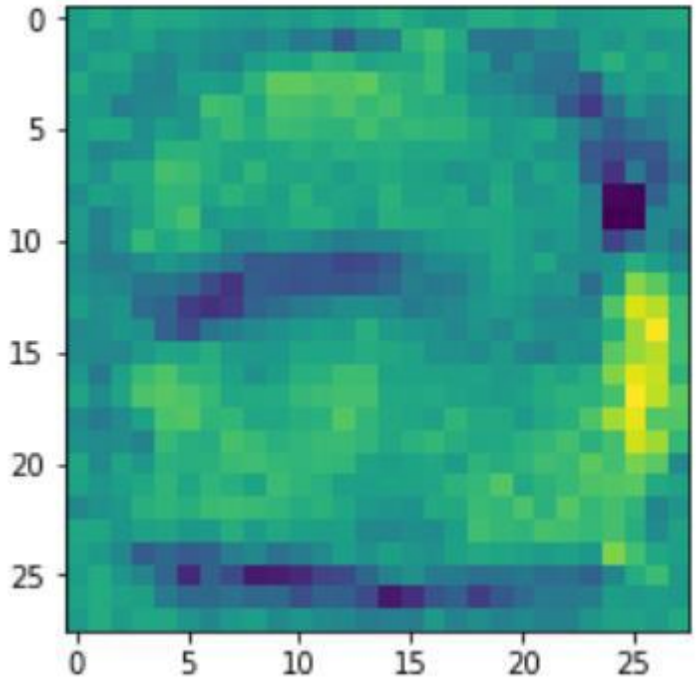
For 0,



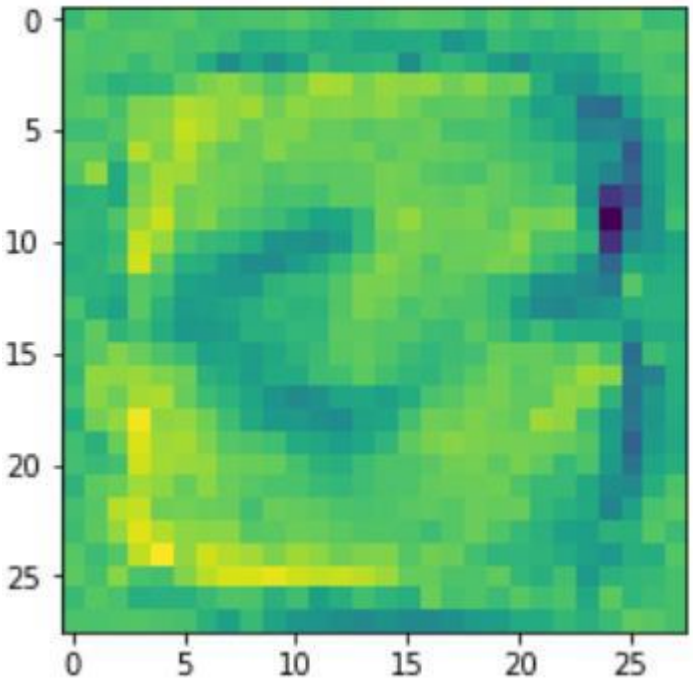
For 1,



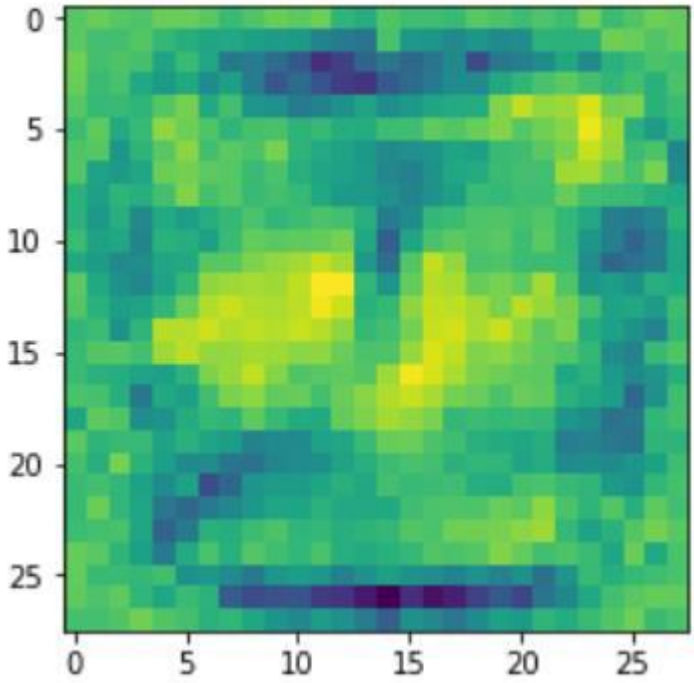
For 2,



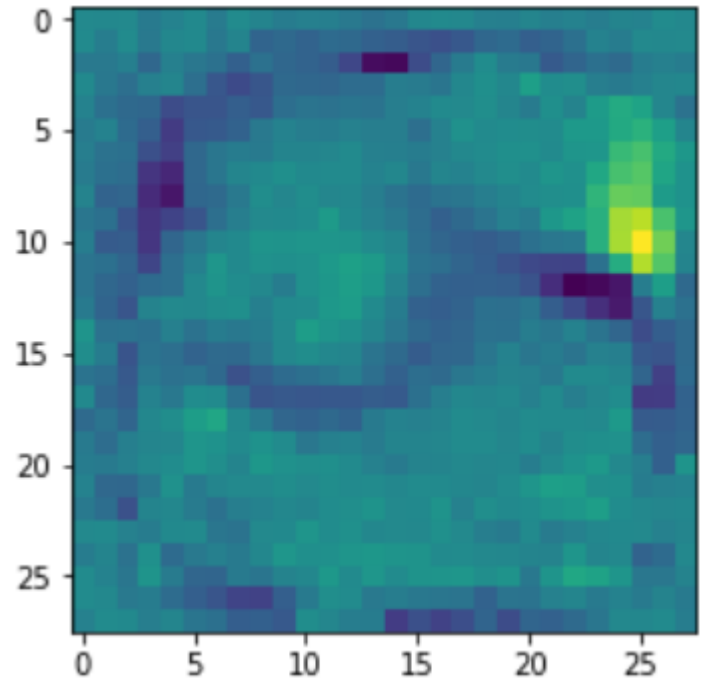
For 3,



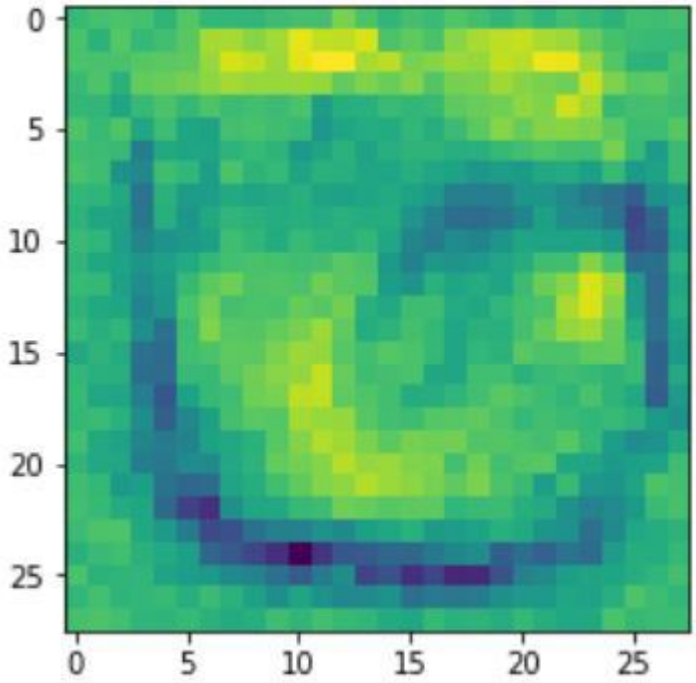
For 4,



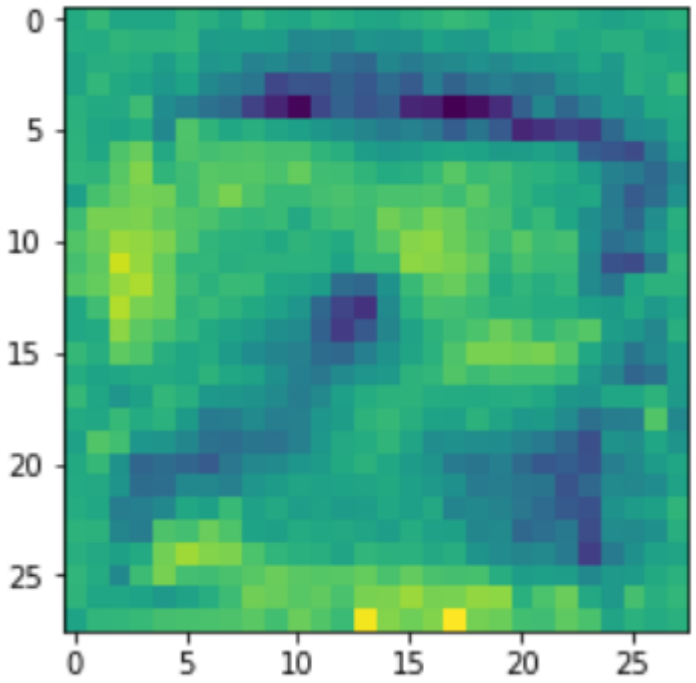
For 5,



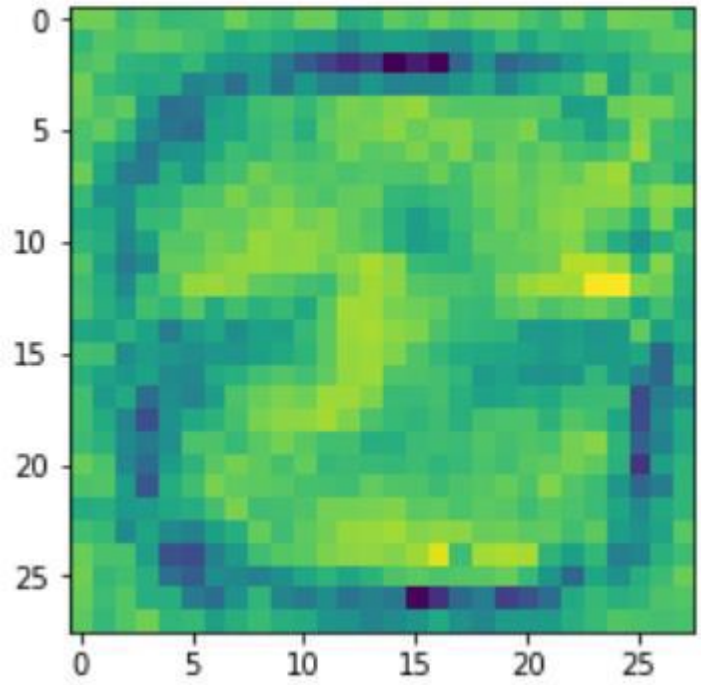
For 6,



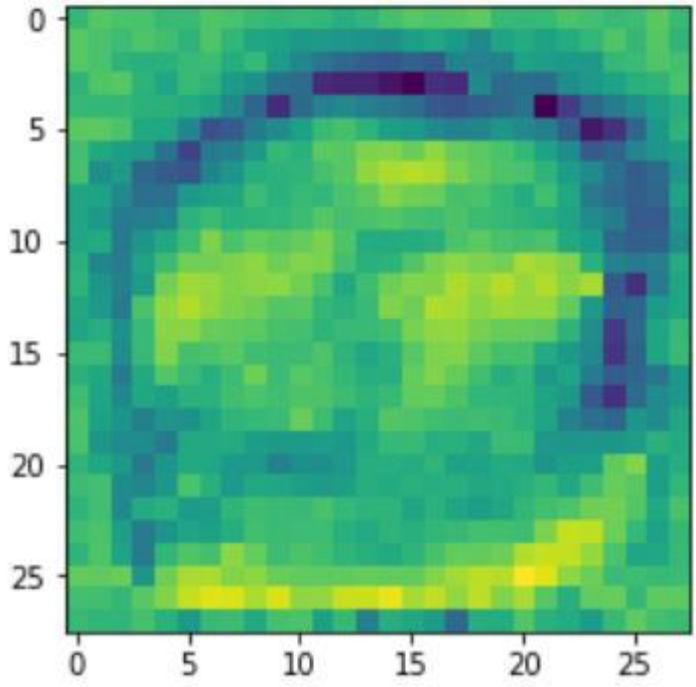
For 7,



For 8,



For 9,


0
5
10
15
20
25
0
5
10
15
20
25