CS 440 Assignment 3 Report

Edbert Linardi, YoungBin Jo, Xiao Tang (3 credit)

Part 1.1

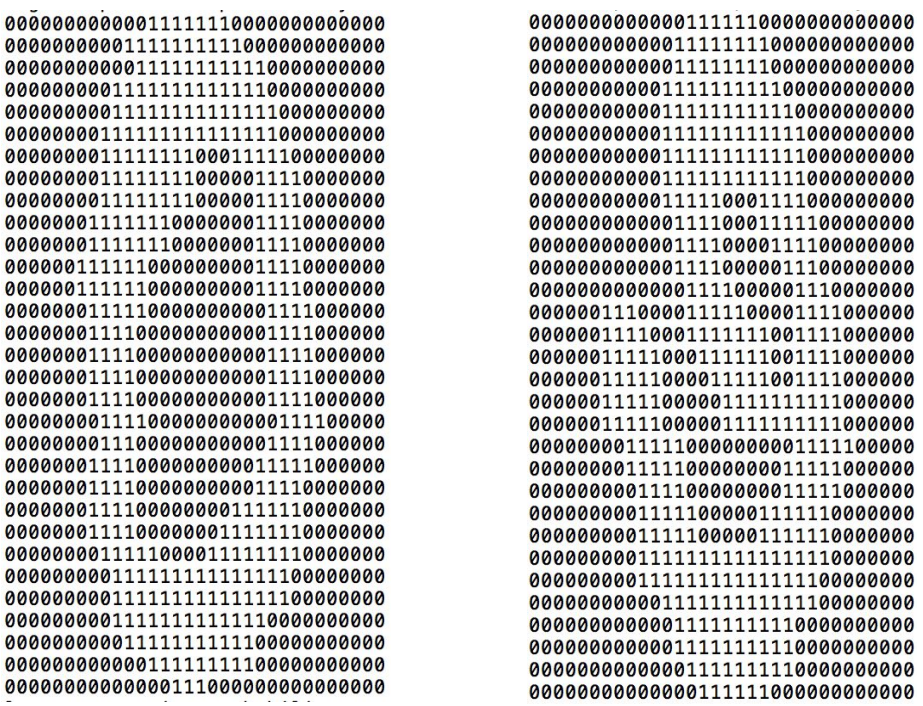
We first created a 10 x 32 x 32 x 2 frequency table, where each cell holds the count of a pixel being 0 or 1 given its true class in the training data set. Then we calculate the $P(class)$ and $P(F_{ij} | class)$ table using laplace smoothing. We tested with all values between 0.1 and 10 (with interval of 0.1) to find the best laplace coefficient. Our test results showed that when the coefficient is between 5.2 and 6.5, our accuracies on the test dataset are the highest, so we chose 5.8 as our coefficient since it's a number roughly in the middle.

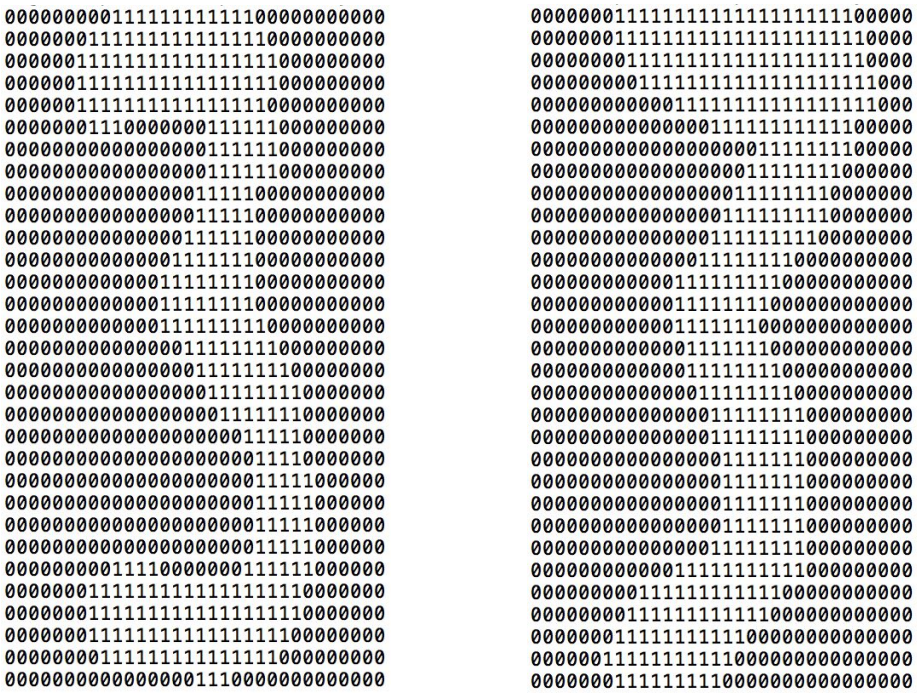
As a result, we get the following confusion table:

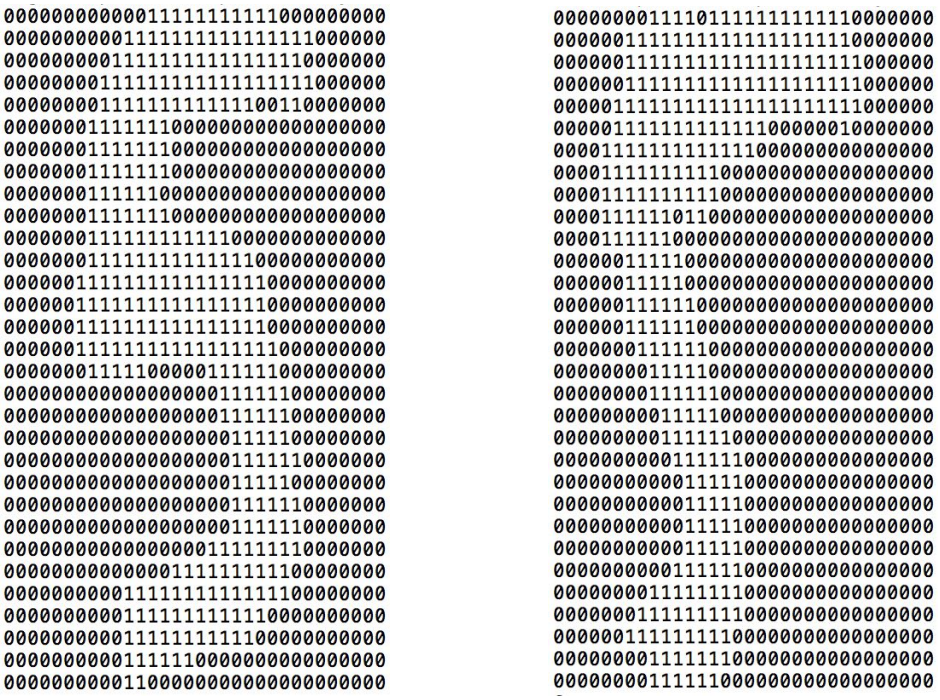
0.972222	0.000000	0.000000	0.000000	0.027778	0.000000	0.000000	0.000000	0.000000	0.000000
0.000000	0.933333	0.000000	0.000000	0.000000	0.000000	0.000000	0.022222	0.022222	0.022222
0.000000	0.000000	0.853659	0.000000	0.000000	0.000000	0.000000	0.024390	0.097561	0.024390
0.000000	0.000000	0.000000	0.909091	0.000000	0.000000	0.000000	0.030303	0.000000	0.060606
0.000000	0.000000	0.000000	0.000000	0.881356	0.000000	0.000000	0.067797	0.050847	0.000000
0.000000	0.000000	0.000000	0.000000	0.000000	0.931034	0.000000	0.000000	0.000000	0.068966
0.000000	0.000000	0.000000	0.000000	0.023256	0.000000	0.976744	0.000000	0.000000	0.000000
0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.978723	0.021277	0.000000
0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	1.000000	0.000000
0.000000	0.000000	0.000000	0.047619	0.000000	0.000000	0.000000	0.023810	0.000000	0.928571

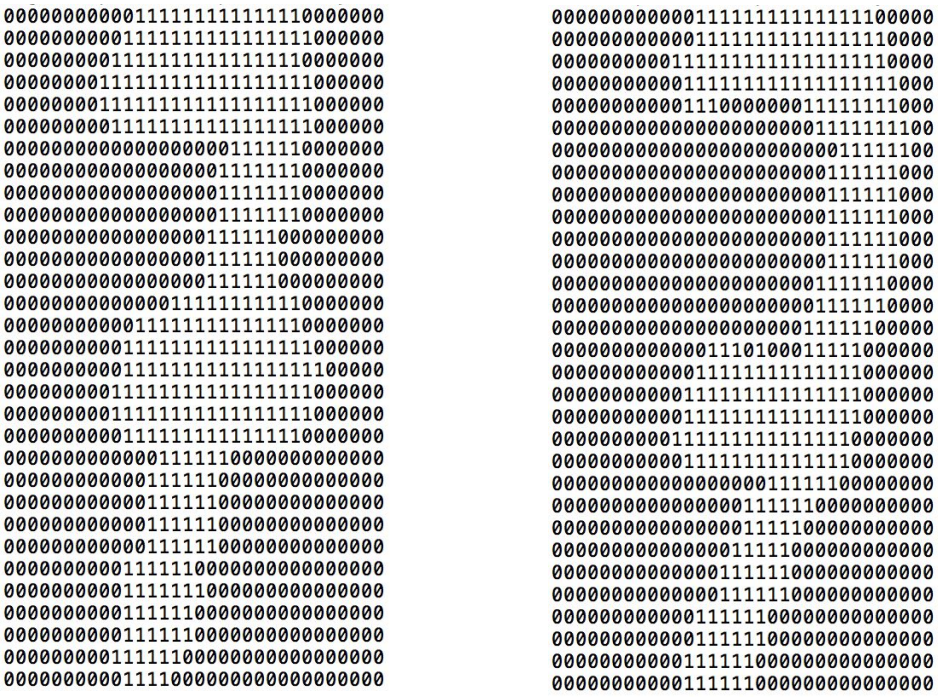
The row represents a picture's true class, and the column represents its label generated by our classifier.

Test tokens that have the highest or lowest posterior probabilities are (highest on the left, lowest on the right):





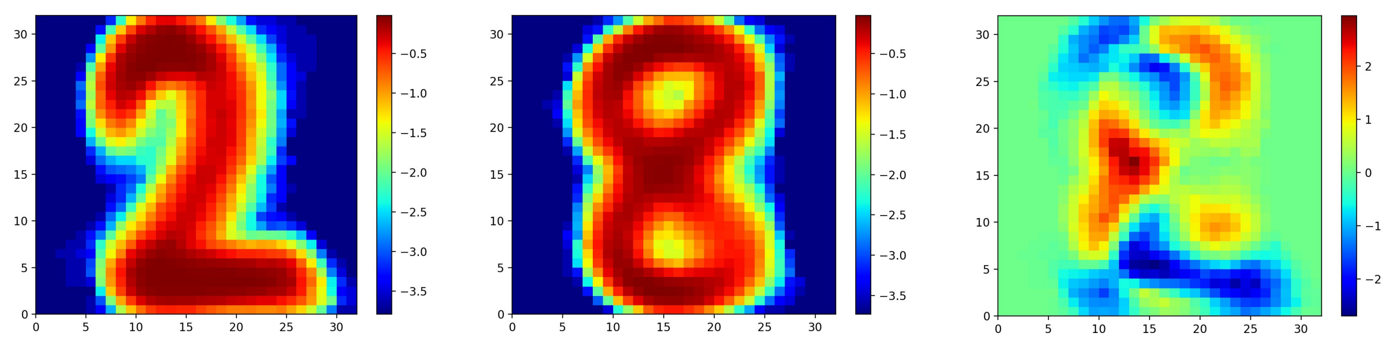


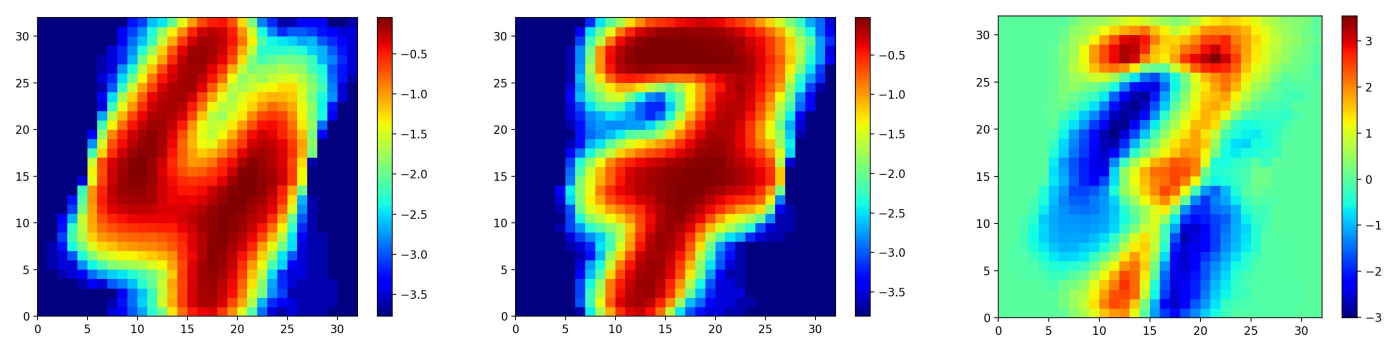


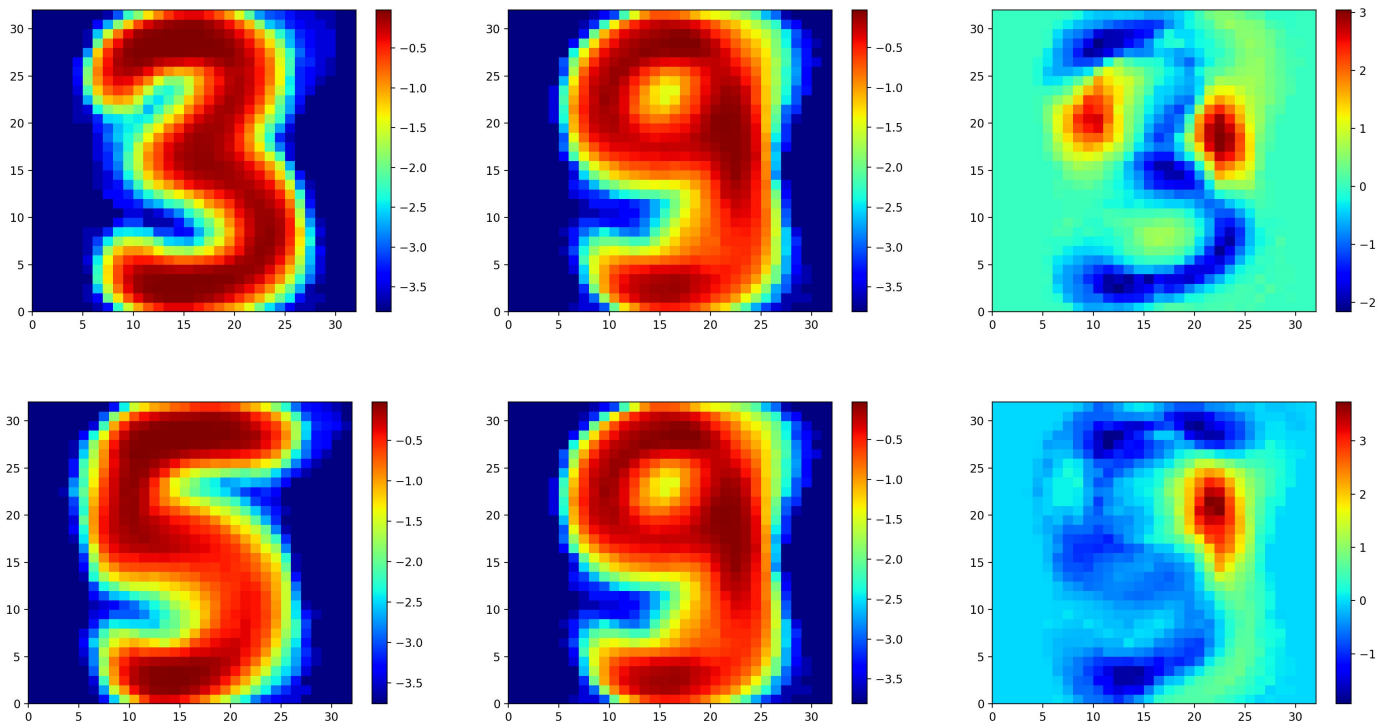
```
00000000000011111110000000000000
00000000011111111110000000000000
00000000011111111111100000000000
00000000111111111111100000000000
00000001111111111111111100000000
00000001111111000111111100000000
00000011111110000111111100000000
00000011111100000011111110000000
00000011111100000011111110000000
00000001111100000011111110000000
00000001111111111111111111000000
00000001111111111111111111000000
00000001111111111111111111000000
00000000111111111110011111000000
00000000011111111000011111000000
00000000000000000000011111000000
00000000000000000000011111000000
00000000000000000000011111000000
00000000000000000000011111000000
00000000000000000000001111000000
00000000000000000000011111000000
00000000000000000000011111000000
00000000000000000000001111000000
00000000000000000000001111000000
00000000000000000000111111000000
00000000001111111111111111000000
00000000001111111111111111000000
00000000011111111111111100000000
00000000011111111111111100000000
00000000001111111111111100000000
00000000000001111100000000000000
```

```
00000000000111000000000000000000
00000000001111110000000000000000
00000000011111111100000000000000
00000000111111111110000000000000
00000001111111111110000000000000
00000000111111111111000000000000
00000000111100001111110000000000
00000000111100001111111000000000
00000000011110000111111000000000
00000000011110000011111100000000
00000000011110000011111100000000
00000000001111101111111110000000
00000000001111111111111110000000
00000000001111111111111110000000
00000000000011111111111111000000
00000000000001111111111111000000
00000000000000011111111111000000
00000000000000000000000111100000
00000000000000000000000111100000
00000000000000000000000111100000
00000000000000000000000011110000
00000000000000000000000011110000
00000000000000000000000011110000
00000000000000000000000001111000
00000000000000000000000011111000
00000000000000000000011111111000
00000000000110000011111111111100
00000000001111111111111111111000
00000000001111111111111111111000
00000000000001111111111111110000
00000000000000011111111110000000
```

The four pairs of digit types that have the highest confusion rates are as below (from left to right, feature likelihood class 1, feature likelihood class 2, odds ratio class 1 / class 2):







Part 2.1

We used applied multi-class perceptron learning rule for digit classification. We decided to set epoch to 5 since there is no big difference in accuracy rate after 5th train. For learning rate, we chose 0.2. For decision rule, we followed the equation given in lecture which was $c = argmax(weighted\ vector\ *\ data)$. After running decision rule, we compared the result with actual value. If the result is different from actual digit, we updated weight value by using equations, $w = w + (learning\ rate * data)$ and $w' = w' - (learning\ rate\ *\ data)$. If the result from decision rule is same as actual digit, weight value does not change. For every training, we updated learning rate using equation $learning\ rate\ /(epoch\ + 1)$.

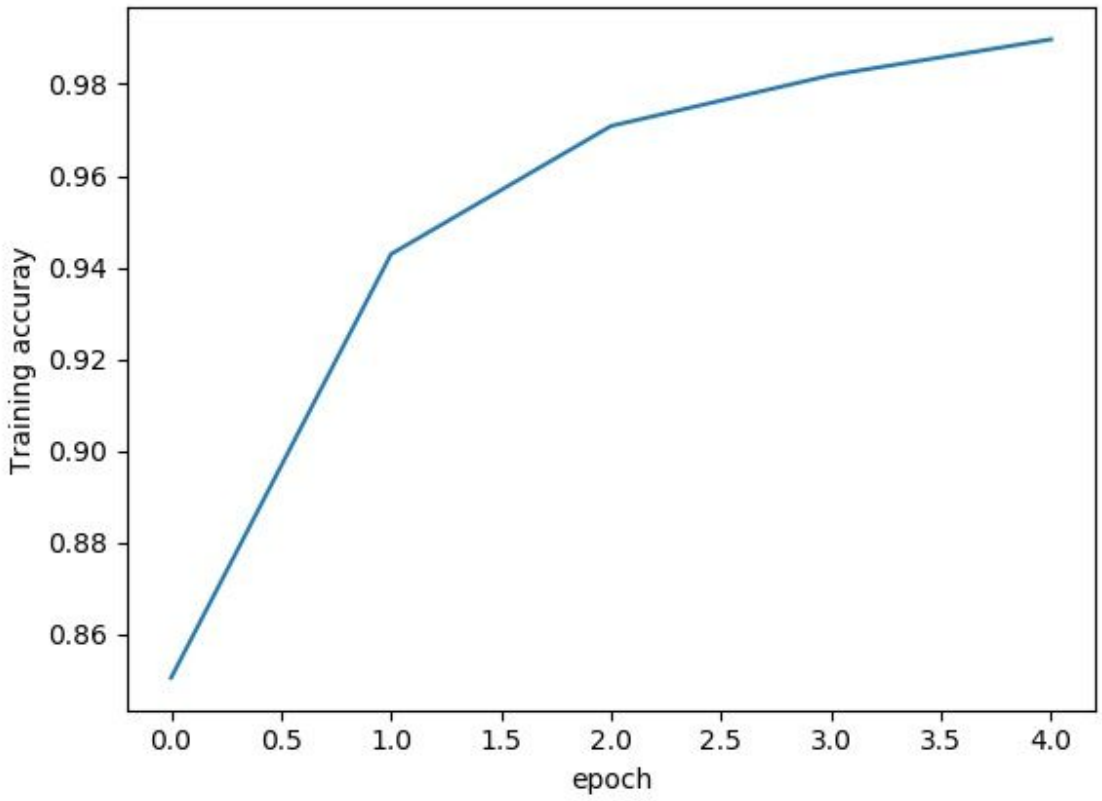
```
training accuracy =  0.8505747126436781
learning rate =  0.2
training accuracy =  0.9429392446633826
learning rate =  0.2
training accuracy =  0.9708538587848933
learning rate =  0.1
training accuracy =  0.9819376026272578
learning rate =  0.03333333333333333
training accuracy =  0.9897372742200329
learning rate =  0.008333333333333333


tesing accuracy =  0.9617117117117117
```

Above image shows the training accuracy for every training and it shows the testing accuracy after finishing training.

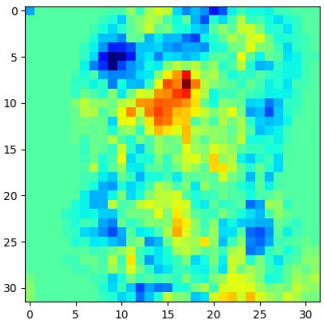
Confusion Matrix:

```
confusion matrix =
array([[0.98, 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.  , 0.01],
       [0.  , 0.92, 0.  , 0.  , 0.01, 0.01, 0.  , 0.01, 0.03, 0.02],
       [0.  , 0.01, 0.96, 0.  , 0.  , 0.  , 0.  , 0.  , 0.01, 0.  ],
       [0.  , 0.  , 0.01, 0.95, 0.  , 0.01, 0.  , 0.  , 0.  , 0.02],
       [0.  , 0.01, 0.  , 0.  , 0.95, 0.  , 0.01, 0.  , 0.  , 0.02],
       [0.  , 0.  , 0.01, 0.01, 0.  , 0.95, 0.  , 0.  , 0.01, 0.02],
       [0.  , 0.01, 0.  , 0.  , 0.  , 0.  , 0.97, 0.  , 0.01, 0.  ],
       [0.  , 0.  , 0.  , 0.01, 0.  , 0.  , 0.  , 0.97, 0.  , 0.01],
       [0.01, 0.02, 0.01, 0.01, 0.01, 0.01, 0.01, 0.  , 0.91, 0.01],
       [0.01, 0.02, 0.  , 0.02, 0.02, 0.01, 0.  , 0.01, 0.02, 0.9 ]])
```

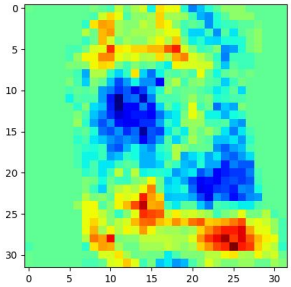


This is the training curve which shows the overall accuracy on the training set.

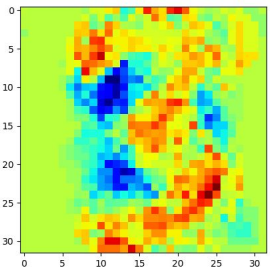
2.1 Extra Credit



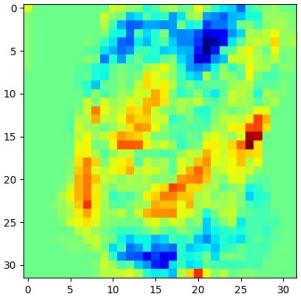
Perceptron Weight Array for digit 1



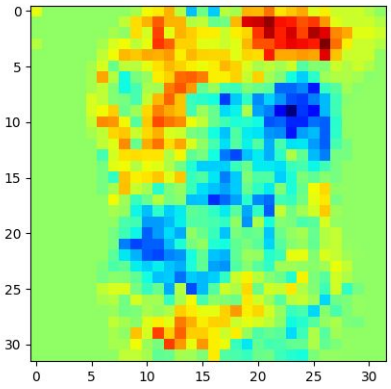
Perceptron Weight Array for digit 2



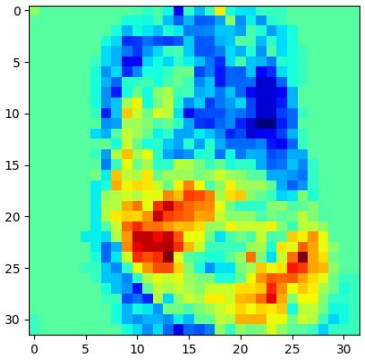
Perceptron Weight Array for digit 3



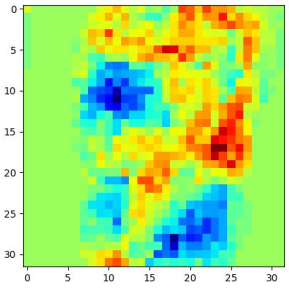
Perceptron Weight Array for digit 4



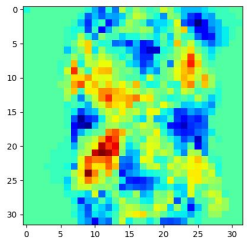
Perceptron Weight Array for digit 5



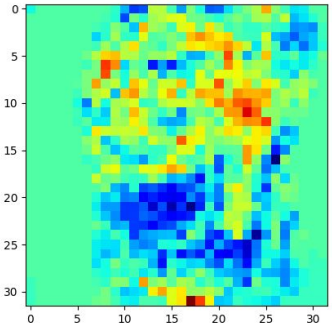
Perceptron Weight Array for digit 6



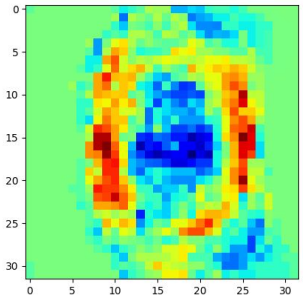
Perceptron Weight Array for digit 7



Perceptron Weight Array for digit 8



Perceptron Weight Array for digit 9



Perceptron Weight Array for digit 0

For this visualization, I choose to use heatmap to represent each trained weight array, because it can show the heat (importance) of each pixel (element) in the weight array. From the result, we can see that the weight array somehow has the same shape as the digit, since the color of pixels with high importance have more yellowish color. The negative signs tell us that the weight has been updated with the formula $w' = w' - (learning\ rate * data)$ because the perceptron incorrectly classified the training data. It is purposed to push the data more to the correct value.