HARISH GANESAN

UB Person No: 50249122

Computer Vision and Image Processing
CSE 573

Homework 4

Autoencoders for Image Classification

1. Let us say the autoencoder uses the following formula for encoding -
   $$\mathbf{z} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b})$$

   And the following formula for decoding –

   $$\mathbf{x}' = \sigma'(\mathbf{W}'\mathbf{z} + \mathbf{b}')$$

   Where W and W' are weight matrices, b and b' are offsets, x is the input, x' is the output and z is the hidden layer. Both sigma and sigma prime are activation functions.

   We find the error by calculating the squared difference between the input and the output (we take the square for the absolute value).

   $$\mathcal{L}(\mathbf{x}, \mathbf{x}') = \|\mathbf{x} - \mathbf{x}'\|^2 = \|\mathbf{x} - \sigma'(\mathbf{W}'(\sigma(\mathbf{W}\mathbf{x} + \mathbf{b})) + \mathbf{b}')\|^2$$

   As we can see from the above equation, the error is calculated by substituting the first two equations into ||x-x'||^2

2. The network starts with each pixel as an input, that is each pixel in the 28x28 image (784 pixels) is one input. Then we reduce the number of layers as we need to encode all this information into fewer dimensions. This is similar to "compressing" the data. In the hidden layer, the autoencoder learns to select the nodes over which the input varies most heavily, and we focus on these. If we have a large number of nodes in the hidden layer, this ends up becoming a trivial reconstruction of the input.

3. We train the autoencoders one layer at a time because we take a “greedy” approach for calculating the weights at each layer. This involves training layers one by one and then at each step, take the old network with k-1 layers and add an additional kth hidden layer that takes the input as the previous layer’s output. We also give no labels in the input, which means this is an unsupervised learning algorithm. This helps us because there is more data available to us that is not labelled than labelled data.

   The main reason why this works well is because it finds a better local optima. Due to training the network on unlabeled data, the weights are staring in a better position than randomly initializing the weights. This is because the unlabeled data already provides us with a significant amount of prior information about what patterns exist in the input data.

4. The features in Figure 3 were obtained after the input data (digitTrainTestArray) is fed to a single layered autoencoder, where the hidden layer has a 100 nodes. Each of the nodes learnt a different feature from the data, and therefore there are a 100 such features present in Figure 3.
   In HW1, we used k-means classification to find out the dictionary, which was out feature set. The con of using that method as opposed to autoencoders is that k-means can only be used for classification whereas autoencoders can be used to “compress” information into a lesser number of dimensions, as well as can be used for classification using a softmax layer. This leads to the autoencoders having a better run-time as opposed to k-means, which takes a very long time to generate the dictionary when the data size is large. Autoencoders also have on average, a lesser error rate than k-means. On the other hand, one advantage of using k-means is that it is far easier to implement and understand as opposed to any neural network or autoencoder. Since k-means uses clustering, some key elements or patterns which are present in the data also might be overlooked, and this will not happen in an autoencoder.

5. The function *plotconfusion* is used to return a confusion matrix plot for the target and output data. The rows correspond to a predicted class and the columns to the actual/true class.

6. The default activation function in MATLAB is the *logsig* or "logistic sigmoid function".

7. ReLu is basically a Rectifier which is also known as ramp function. The y value is 0 for all negative values of x and increases linearly from x=0.
   The problem with sigmoid is that it's gradient vanishes to 0 after a certain point whereas ReLu has a constant slope of 1. ReLu also gives us Sparsity in the encoding, which means it gives a value of 0 anytime $W*x < 0$, whereas other activation functions do not do so. This makes our calculation much faster.

8. There are two main reasons why we shouldn't set the initial weights to zero, and they are as follows :
   a. If all nodes start with the same weights, then all the nodes will follow the same gradient during backpropogation, and will always end up doing the same thing as all the other nodes. This is called the "symmetry" problem – when the weights are all updated symmetrically due to initial values being equal
   b. Neural Networks usually get stuck in the local minima, so it makes sense to give them all different values.

9. Batch Gradient Descent computes the gradient using the whole dataset. This works well for relatively smooth error manifolds. We eventually move toward a local or global optimal solution. The pro of using this method is that since it uses the entire dataset, it provides a very accurate result. One disadvantage of using this method is that it is very computationally expensive to use the entire dataset, sometimes the dataset will be so large it cannot be held in the RAM.

   Stochastic Gradient Descent computes the gradient using a single sample, and this is useful for data with lots of maximas and minimas. This is because the single sample tends to jerk the model out of a local minima into a region that might be more optimal. One advantage of using this method is that it is much faster computationally than BGD. One con is that taking a single sample can lead to a noise. So, a solution to this problem is to use mini-batches where we take a few samples at once, and keep this number relatively small compared to the dataset size. We can also perform more iterations of Minibatch SGD to get more accurate results.

   Batch Gradient Descent is faster than Stochastic Gradient Descent when it comes to the number of epochs, as BGD has to consider the entire dataset, and hence it will go through an Epoch all at once whereas SGD has to consider only one point or a few.

   When it comes to number of iterations, Batch Gradient Descent is slower because every time the weights are updates, it needs to calculate the gradient for the entire dataset once again, whereas in every iteration, Stochastic Gradient Descent only has to consider one value to update the weights.

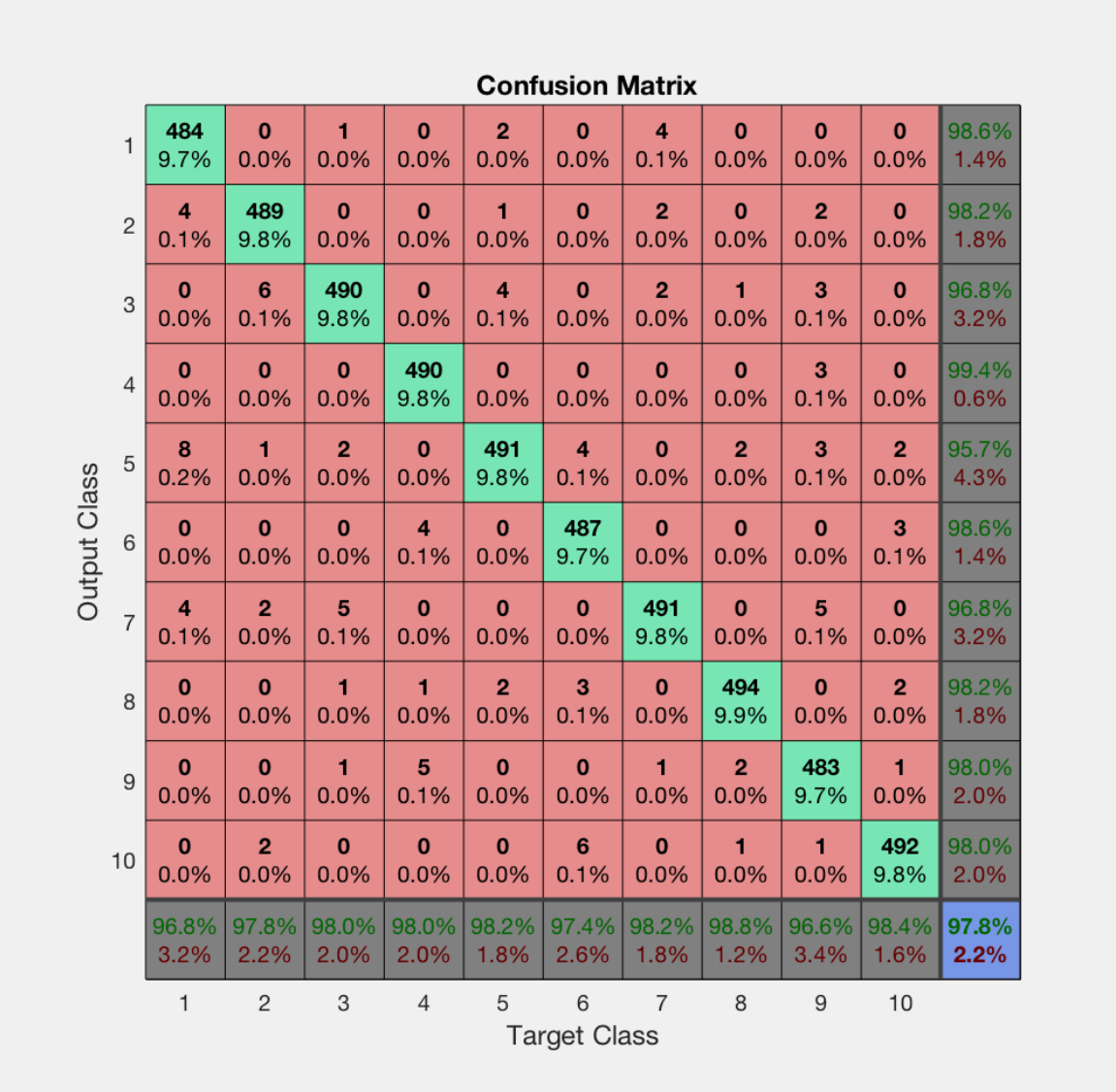
10. Upon changing the parametes such as maximum epoch size, number of hidden layers and L2WeightRegularization, we get varying results for the confusion matrix.

    We can observe from the overall accuracy that if we keep the rest of the parameters same and decrease the number of hidden layers, the accuracy decreases. If we increase double the number of hidden layers, the accuracy increases marginally.

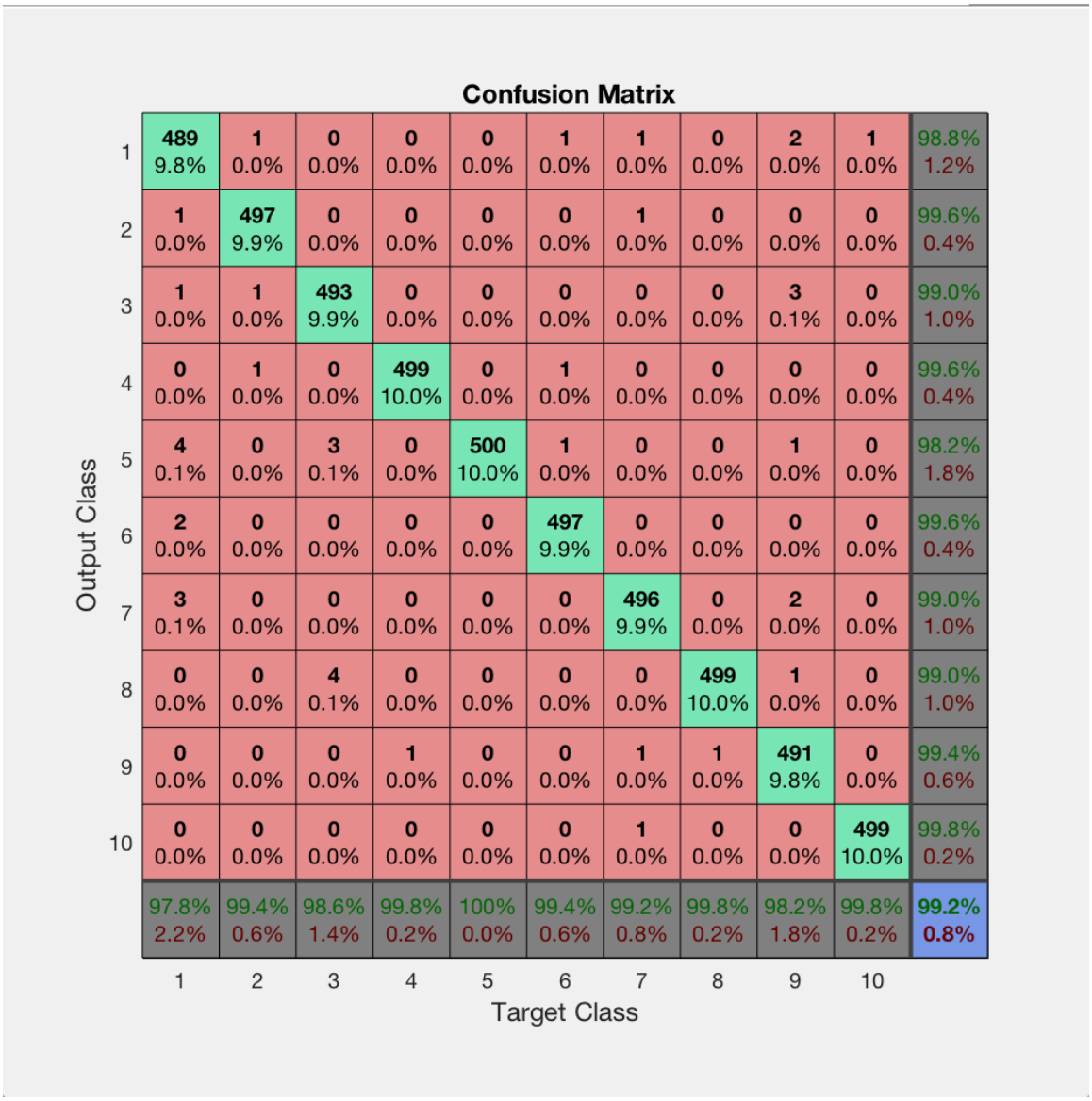
    In general, when the epoch size is more, the accuracy is higher, because it goes through more iterations.
    A lesser value of L2WeightRegularization leads to an increase in accuracy and vice-versa.

    We can see the complete set of parameters and the subsequent results in the following figures :-

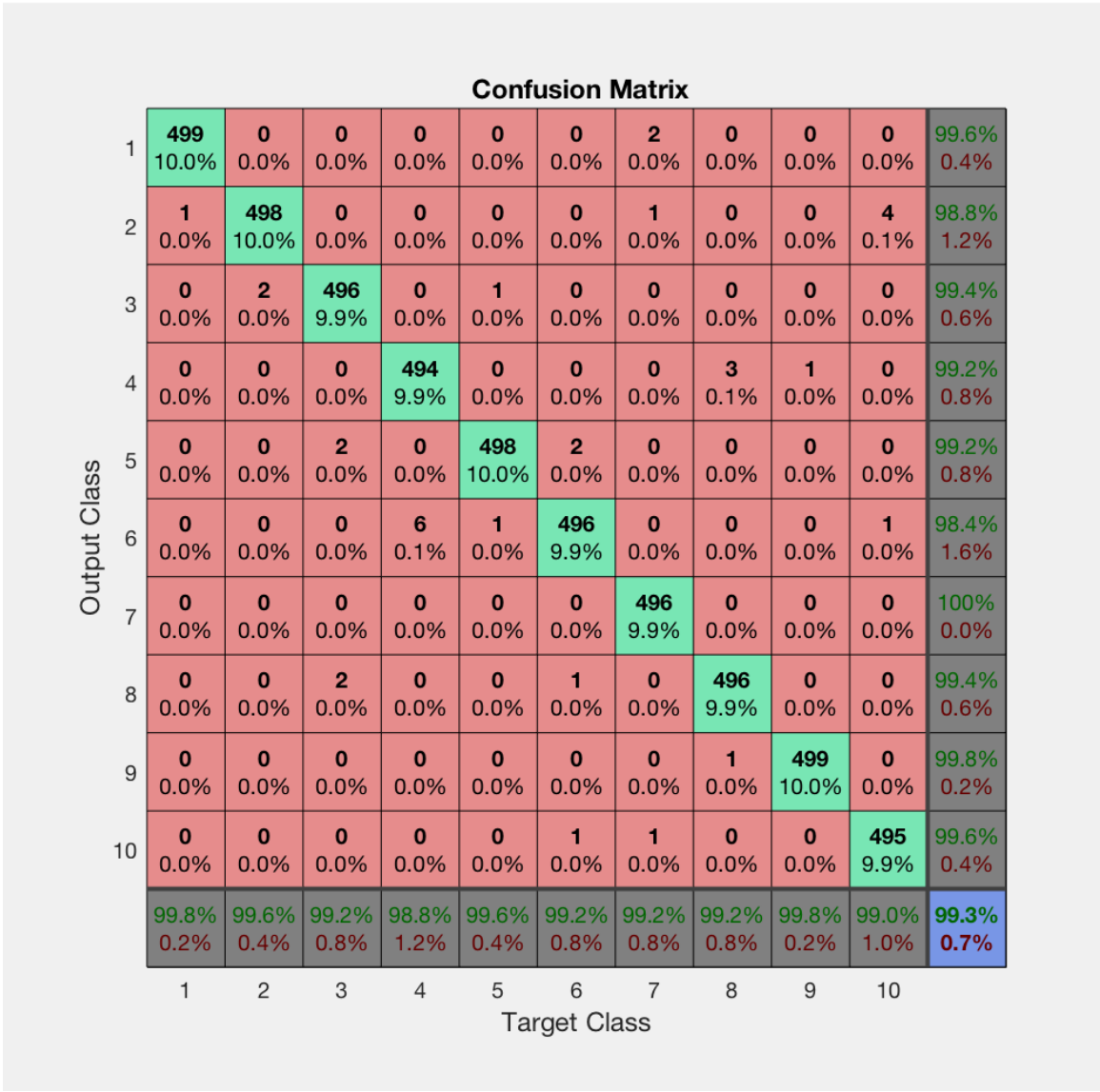
a. Maximum Epochs (1/2) :400/100 | Hidden Layers (1/2) : 100/50 | L2WeightRegularization: 0.008/0.004



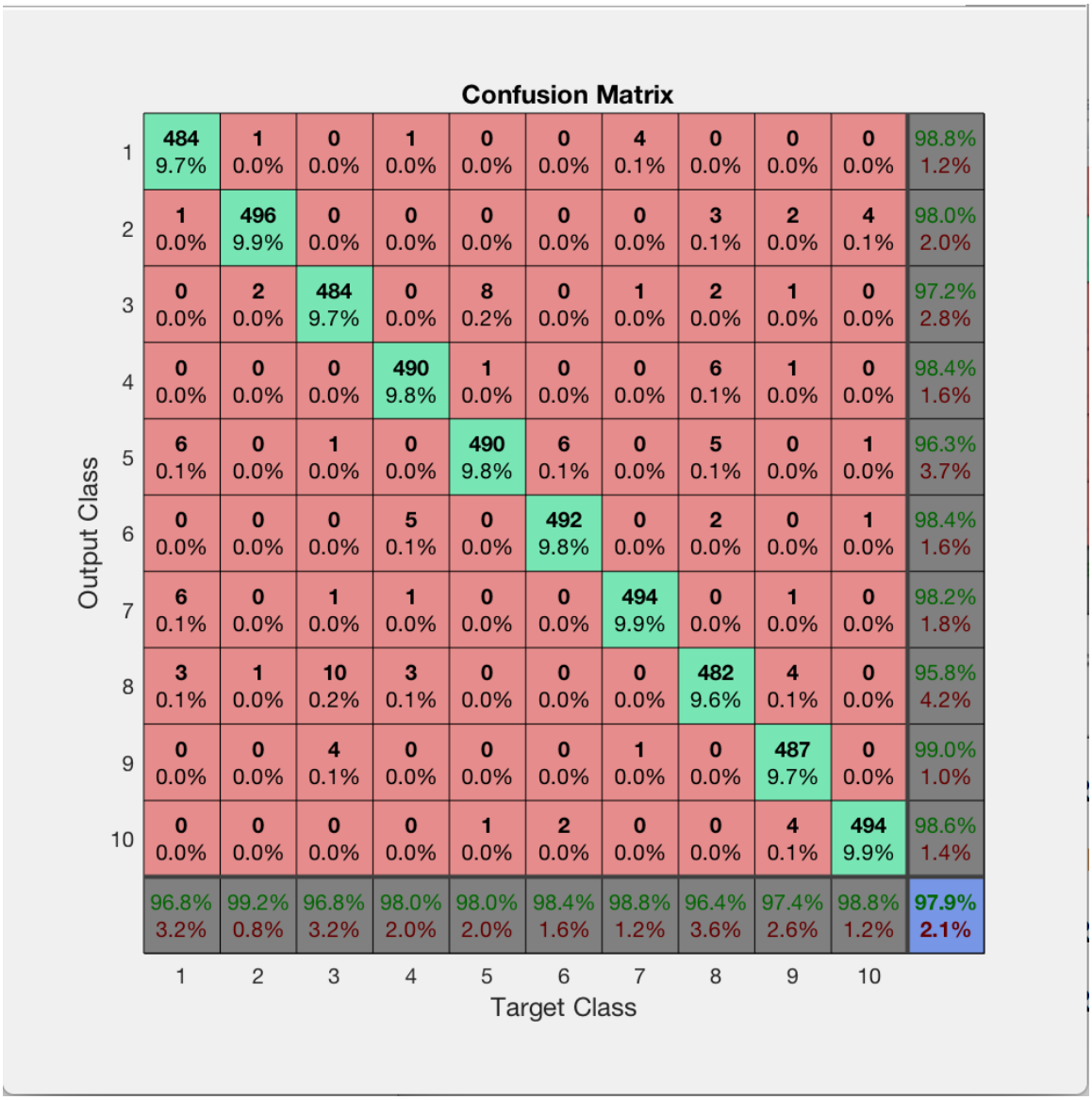
b. Maximum Epochs (1/2) :400/100 | Hidden Layers (1/2) : 200/100 | L2WeightRegularization(1/2) : 0.004/0.002



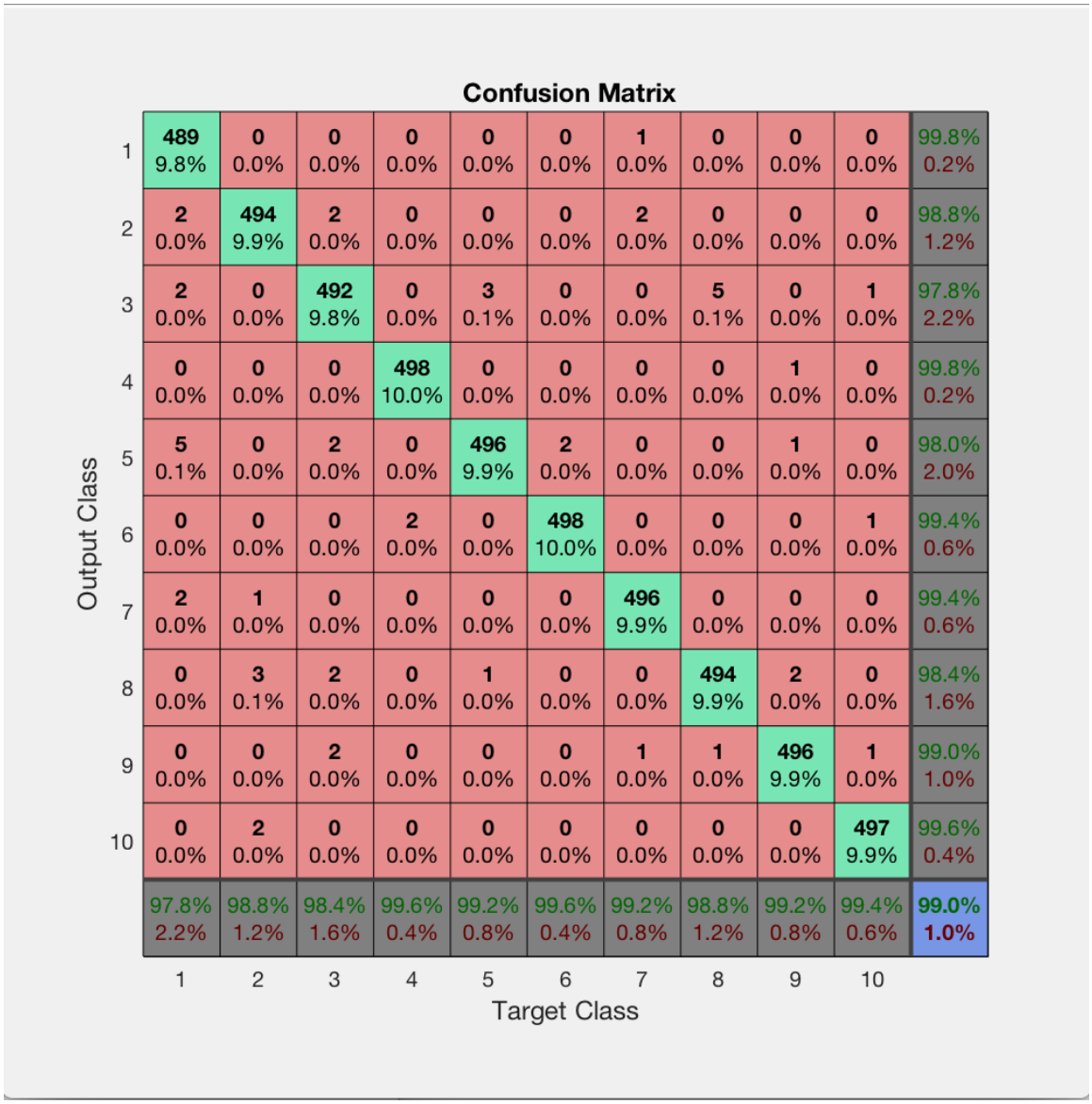
c. Maximum Epochs (1/2) :400/100 | Hidden Layers (1/2) : 100/50 | L2WeightRegularization(1/2) : 0.002/0.001



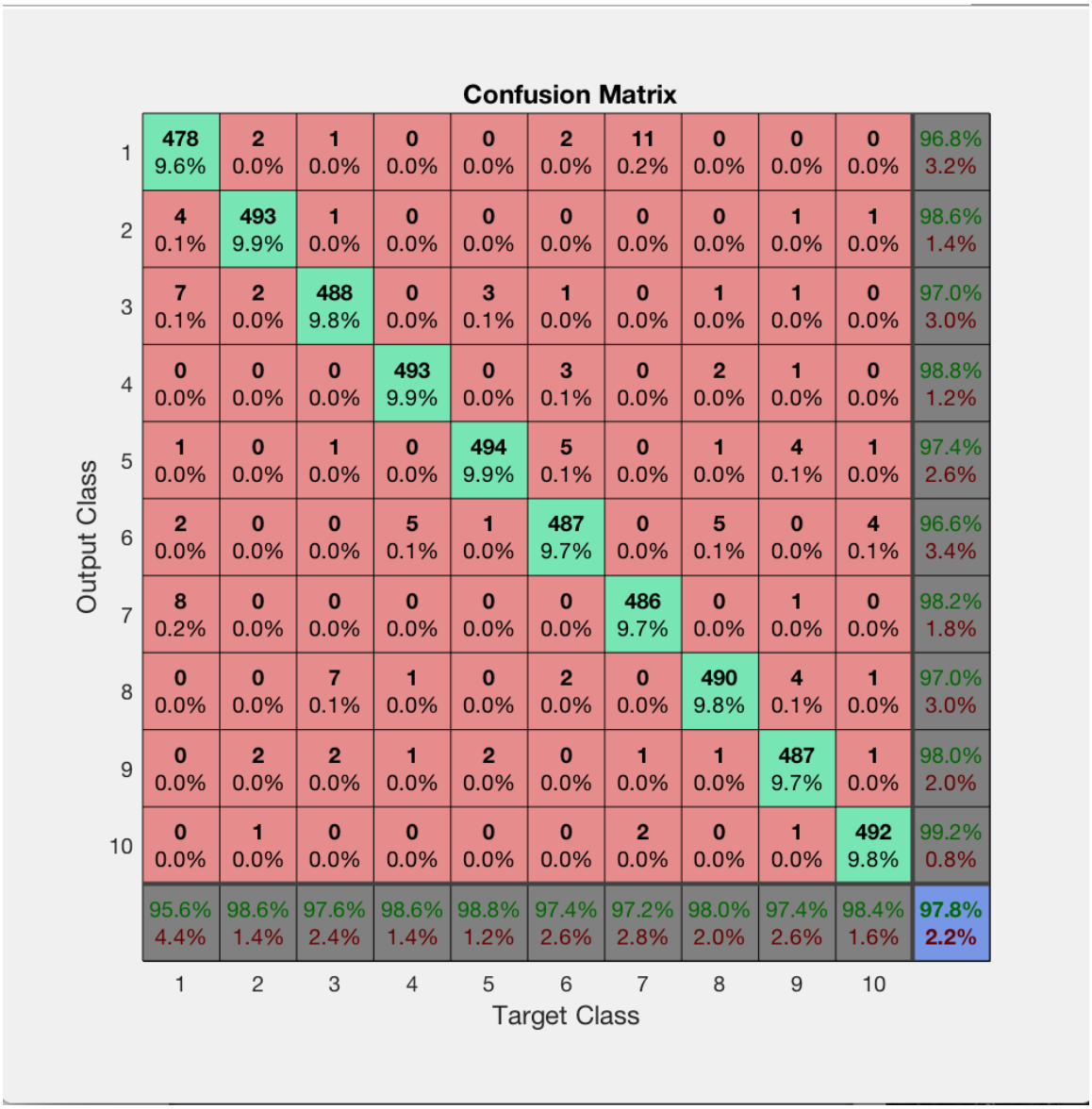
d. Maximum Epochs (1/2) :400/100 | Hidden Layers (1/2) : 50/25 | L2WeightRegularization(1/2) : 0.004/0.002



e. Maximum Epochs (1/2) :500/200 | Hidden Layers (1/2) : 100/50 | L2WeightRegularization(1/2) : 0.004/0.002



f. Maximum Epochs (1/2) :200/50 | Hidden Layers (1/2) : 100/50 | L2WeightRegularization(1/2) : 0.004/0.002



g. Maximum Epochs (1/2) :400/100 | Hidden Layers (1/2) : 100/50 | L2WeightRegularization(1/2) : 0.004/0.002