Computer Vision and Image Processing
Homework 4: Autoencoders for Image Classification

50248987
Satya Chembolu

Q1) How does an autoencoder detect errors?

- An autoencoder is an 'unsupervised' neural network which tries to replicate the input by encoding to a lower dimension (thereby learning the important features of the input) and decoding/reconstructing back to original dimension. While training, the output layer of the autoencoder is compared to the input data layer (for example, by L2 norm or squared error) and the difference builds the reconstruction error. We train our encoder by reducing this error. Here, a *mean squared error* for sparse encoder is used, which is

$$E = \underbrace{\frac{1}{N}\sum_{n=1}^{N}\sum_{k=1}^{K}\left(x_{kn} - \hat{x}_{kn}\right)^2}_{\text{mean squared error}} + \lambda * \underbrace{\Omega_{weights}}_{\substack{L_2 \\ \text{regularization}}} + \beta * \underbrace{\Omega_{sparsity}}_{\substack{\text{sparsity} \\ \text{regularization}}},$$

Q2) The network starts out with 28*28 = 784 inputs. Why do subsequent layers have fewer nodes?

- The goal of an autoencoder is to learn the important features of the input in the encoding layers (the hidden layers) and then try to generate the input in the output layer. If the hidden layers have same or more number of nodes as the input layer, the encoder might learn the 'identity function' and the output just completely mirrors the input, thereby not learning any important features of the input. This could result in 'overfitting' of the train data and results in high testing-errors.

Q3) Why are autoencoders trained one hidden layer at a time?

- The autoencoders are trained in a greedy, unsupervised manner where the weights for a hidden layer are learnt/trained by minimizing the reconstruction error from itself. This property of independent layer-training helps in the following ways:
  - By training each hidden layer independently, the complexity of training is largely reduced.
  - Also, the layers become independent entities which can be stacked on to form the whole network. Each layer will not require a fresh training on the addition of new layers to the network. Further, the stacked complete network is simply fine-tuned using a supervised approach (labels in case of digits).
  - This unsupervised nature of training helps us train the model on more amount of data as 'unlabelled' data is more readily available than 'labelled' data.

Q4) How were the features from hidden layer obtained? Compare the method of identifying features here with the method of HW1. What are a few pros and cons of these methods?

- Every neuron of an autoencoder is associated with a vector of weights which are trained to respond to a particular visual feature from the input. When visualised, these weights represent the 'features' learnt by each of the neurons. In this case shown in the pdf, we have 100 nodes in the first hidden layer so we get 100 feature-output from the layer, which is shown in the given figure.
- In the case of HW1, we needed a predefined filter bank using which we obtained the features. Whereas in the current case, we wouldn't need a predefined filter bank and this makes the filters represent the data better. Also, the current model is more flexible to be extended or implemented on other classification problems by tuning the hyper parameters like number of layers, number of nodes etc. Also, the accuracy achieved by our current model is far higher (~98%) on both synthetic and MNIST datasets compared to the accuracy achieved by the HW1 approach (~52%). Although, this can be attributed the more complex nature of scene recognition and lesser train data in that case.

Q5) What does the function plot confusion do?

- Plotconfusion gives the *confusion matrix* on the test data tested over the constructed model. A Confusion matrix gives the accuracy of the classification model built (test accuracy in our case) by comparing the predicted labels to the actual labels. The rows give the predicted label while the columns give the actual label. So, the diagonal contains the true positives count (and %) while the off diagonal elements give the wrong predictions count (and %). The final cell of the table gives the overall accuracy. This table also helps us build the intuition behind the wrong predictions by comparing the predicted labels to the original labels.

Q6) What activation function is used in the hidden layers of the MATLAB tutorial?

- Sigmoid is the activation function used in the hidden layers. It is the default activation function for the autoencoders in the neural networks toolbox.

Q7) In training deep networks, the ReLU activation function is generally preferred to the sigmoid activation function. Why might this be the case?

- The sigmoid function limits the activation output below 1. Therefore, as the input to sigmoid increases, the gradient tends to zero. This problem is called vanishing gradient problem. Where, ReLU which is given by max (0, z) (where z is the input to the activation), avoids the vanishing gradient problem. This is an important factor which makes ReLU more preferable compared to sigmoid.

Q8) The MATLAB demo uses a random number generator to initialize the network. Why is it not a good idea to initialize a network with all zeros? How about all ones, or some other constant value? (Hint: Consider what the gradients from backpropagation will look like.)

- First, neural networks tend to get stuck in local minima, so it's a good idea to give them many different starting values, which can be done by randomizing.
- Second, if the neurons start with the same weights, then all the neurons will follow the same gradient, and will always end up doing the same thing as one another, which makes the network highly redundant.
  Therefore, it is better and logical to randomly initialize the weights rather than at some constant value.

Q9) Give pros and cons for both stochastic and batch gradient descent. In general, which one is faster to train in terms of number of epochs? Which one is faster in terms of number of iterations?

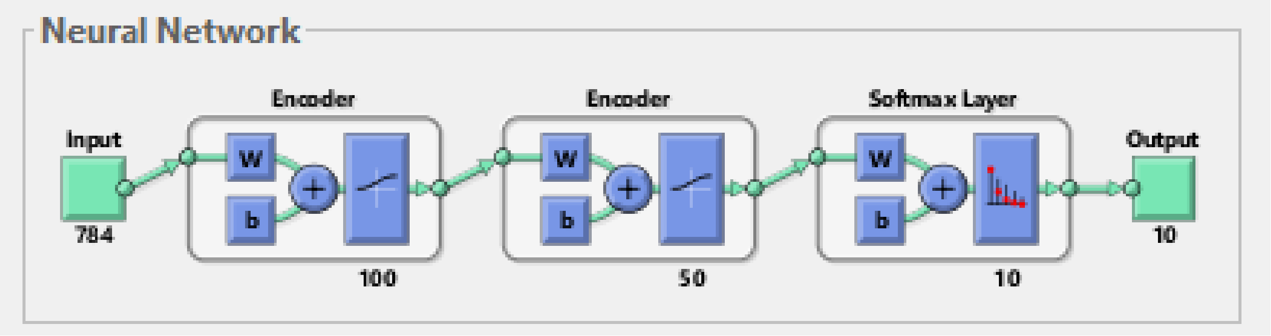
- *Batch gradient descent (BGD)*:
  - Weights are updated only after all the training errors are computed. This makes it computationally very heavy. Also, loading the whole training data into memory at a time isn't always possible.
  - For BGD, number of epochs is equivalent to number of iterations.
  - One advantage of BGD is that individual training noises are overlooked and the optimal solution is eventually reached.
  - Weight updates are limited to 1 per epoch. *Therefore, the time taken for a single epoch in BGD is lesser than that of SGD*.
- *Stochastic gradient descent (SGD)*:
  - SGD Converges faster than BGD as the weights are updated more frequently in SGD. *Therefore, SGD uses lesser number of epochs than BGD.*
  - In SGD, the weights are updated over each and every training example. *Therefore, SGD uses higher number of iterations than BGD.*
  - Conversely*, the time taken for one iteration in SGD is lesser than that of BGD.*
  - Mini-batch SGD is a better version of SGD where the weights are updated after every batch of training data. This avoids the high noise each training input could have while still updating the weights frequently making the computation very fast.

Q10) Try playing around with some of the parameters specified in the tutorial. Perhaps the sparsity parameters or the number of nodes; or number of layers. Report the impact of slightly modifying the parameters. Is the tutorial presentation robust or fragile with respect to parameter settings?

- The default parameters used in the tutorial are as follows:

Architecture:

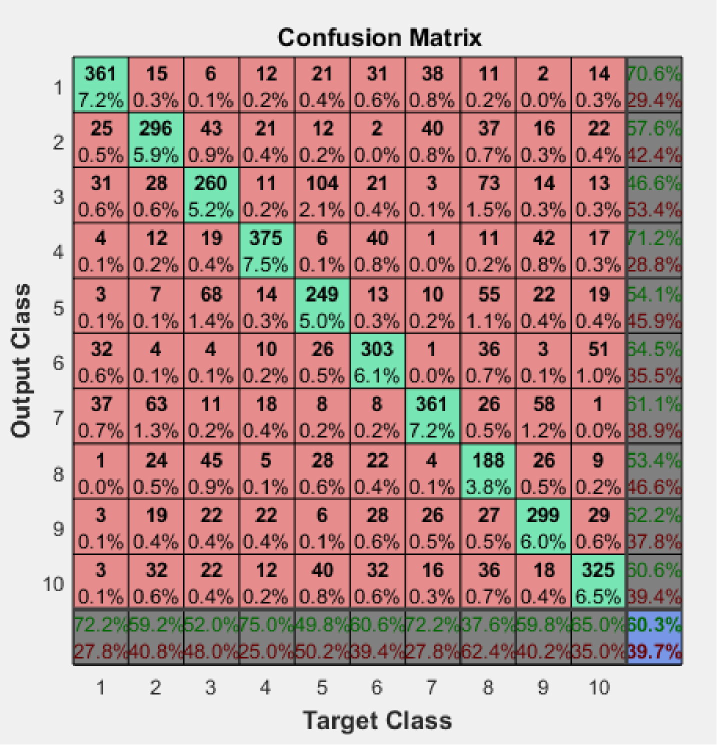
- Number of inputs nodes: 784 (28*28)
- Number of hidden layers: 2



	Hidden layer 1	Hidden layer 2
Number of Nodes	100	50
Maximum number of epochs	400	100
L2WeightRegularization	0.004	0.002
Sparsity Regularization	4	4
Sparsity Proportion	0.15	0.1
Scale Data	false	false

Confusion Matrix:

1) Pre fine-tuned:



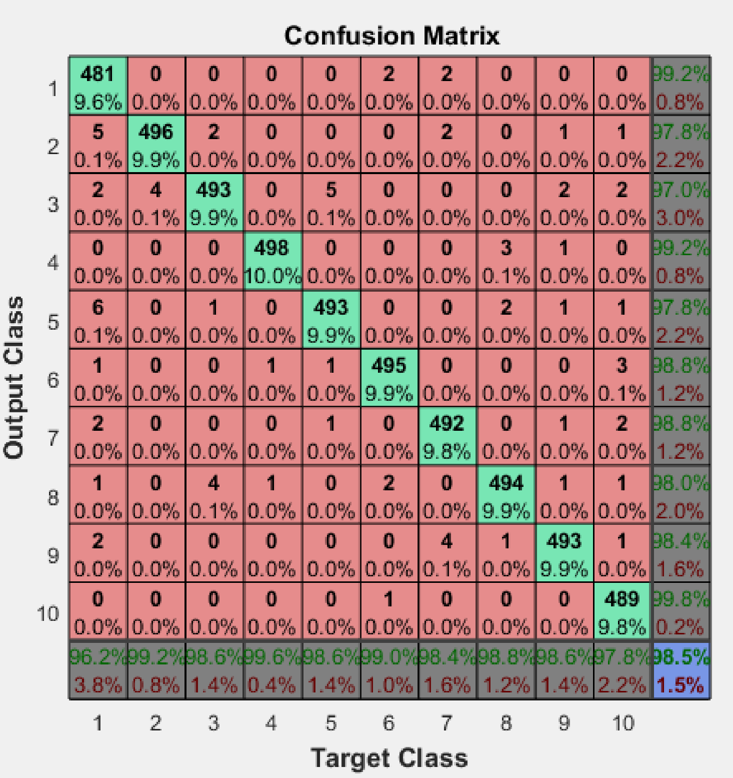
Confusion Matrix

Output Class

	1	2	3	4	5	6	7	8	9	10	
1	361 7.2%	15 0.3%	6 0.1%	12 0.2%	21 0.4%	31 0.6%	38 0.8%	11 0.2%	2 0.0%	14 0.3%	70.6% 29.4%
2	25 0.5%	296 5.9%	43 0.9%	21 0.4%	12 0.2%	2 0.0%	40 0.8%	37 0.7%	16 0.3%	22 0.4%	57.6% 42.4%
3	31 0.6%	28 0.6%	260 5.2%	11 0.2%	104 2.1%	21 0.4%	3 0.1%	73 1.5%	14 0.3%	13 0.3%	46.6% 53.4%
4	4 0.1%	12 0.2%	19 0.4%	375 7.5%	6 0.1%	40 0.8%	1 0.0%	11 0.2%	42 0.8%	17 0.3%	71.2% 28.8%
5	3 0.1%	7 0.1%	68 1.4%	14 0.3%	249 5.0%	13 0.3%	10 0.2%	55 1.1%	22 0.4%	19 0.4%	54.1% 45.9%
6	32 0.6%	4 0.1%	4 0.1%	10 0.2%	26 0.5%	303 6.1%	1 0.0%	36 0.7%	3 0.1%	51 1.0%	64.5% 35.5%
7	37 0.7%	63 1.3%	11 0.2%	18 0.4%	8 0.2%	8 0.2%	361 7.2%	26 0.5%	58 1.2%	1 0.0%	61.1% 38.9%
8	1 0.0%	24 0.5%	45 0.9%	5 0.1%	28 0.6%	22 0.4%	4 0.1%	188 3.8%	26 0.5%	9 0.2%	53.4% 46.6%
9	3 0.1%	19 0.4%	22 0.4%	22 0.4%	6 0.1%	28 0.6%	26 0.5%	27 0.5%	299 6.0%	29 0.6%	62.2% 37.8%
10	3 0.1%	32 0.6%	22 0.4%	12 0.2%	40 0.8%	32 0.6%	16 0.3%	36 0.7%	18 0.4%	325 6.5%	60.6% 39.4%
	72.2% 27.8%	59.2% 40.8%	52.0% 48.0%	75.0% 25.0%	49.8% 50.2%	60.6% 39.4%	72.2% 27.8%	37.6% 62.4%	59.8% 40.2%	65.0% 35.0%	60.3% 39.7%

Target Class

2) Fine-tuned with max-epochs = 400:



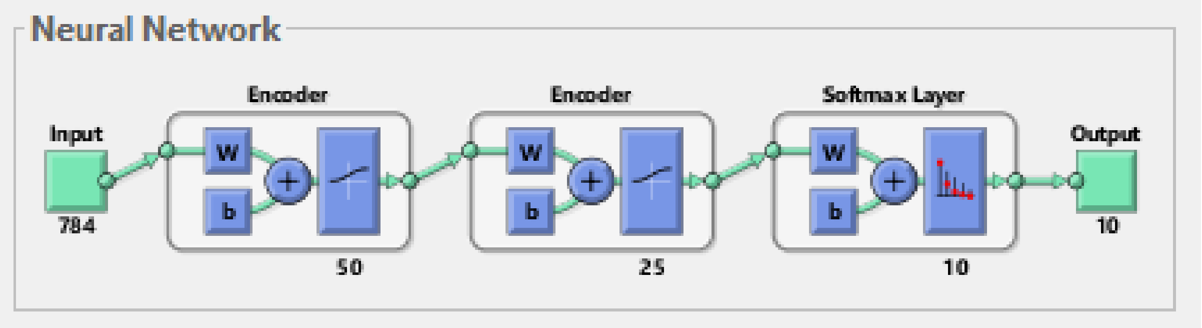
	1	2	3	4	5	6	7	8	9	10	
1	481 9.6%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	2 0.0%	2 0.0%	0 0.0%	0 0.0%	0 0.0%	99.2% 0.8%
2	5 0.1%	496 9.9%	2 0.0%	0 0.0%	0 0.0%	0 0.0%	2 0.0%	0 0.0%	1 0.0%	1 0.0%	97.8% 2.2%
3	2 0.0%	4 0.1%	493 9.9%	0 0.0%	5 0.1%	0 0.0%	0 0.0%	0 0.0%	2 0.0%	2 0.0%	97.0% 3.0%
4	0 0.0%	0 0.0%	0 0.0%	498 10.0%	0 0.0%	0 0.0%	0 0.0%	3 0.1%	1 0.0%	0 0.0%	99.2% 0.8%
5	6 0.1%	0 0.0%	1 0.0%	0 0.0%	493 9.9%	0 0.0%	0 0.0%	2 0.0%	1 0.0%	1 0.0%	97.8% 2.2%
6	1 0.0%	0 0.0%	0 0.0%	1 0.0%	1 0.0%	495 9.9%	0 0.0%	0 0.0%	0 0.0%	3 0.1%	98.8% 1.2%
7	2 0.0%	0 0.0%	0 0.0%	0 0.0%	1 0.0%	0 0.0%	492 9.8%	0 0.0%	1 0.0%	2 0.0%	98.8% 1.2%
8	1 0.0%	0 0.0%	4 0.1%	1 0.0%	0 0.0%	2 0.0%	0 0.0%	494 9.9%	1 0.0%	1 0.0%	98.0% 2.0%
9	2 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	4 0.1%	1 0.0%	493 9.9%	1 0.0%	98.4% 1.6%
10	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	1 0.0%	0 0.0%	0 0.0%	0 0.0%	489 9.8%	99.8% 0.2%
	96.2% 3.8%	99.2% 0.8%	98.6% 1.4%	99.6% 0.4%	98.6% 1.4%	99.0% 1.0%	98.4% 1.6%	98.8% 1.2%	98.6% 1.4%	97.8% 2.2%	98.5% 1.5%

Therefore, we can see that an accuracy of 98.5% was achieved on the synthetic data with given architecture.

- Now, we try to change the number of nodes in each layer to observe the changes in accuracies.

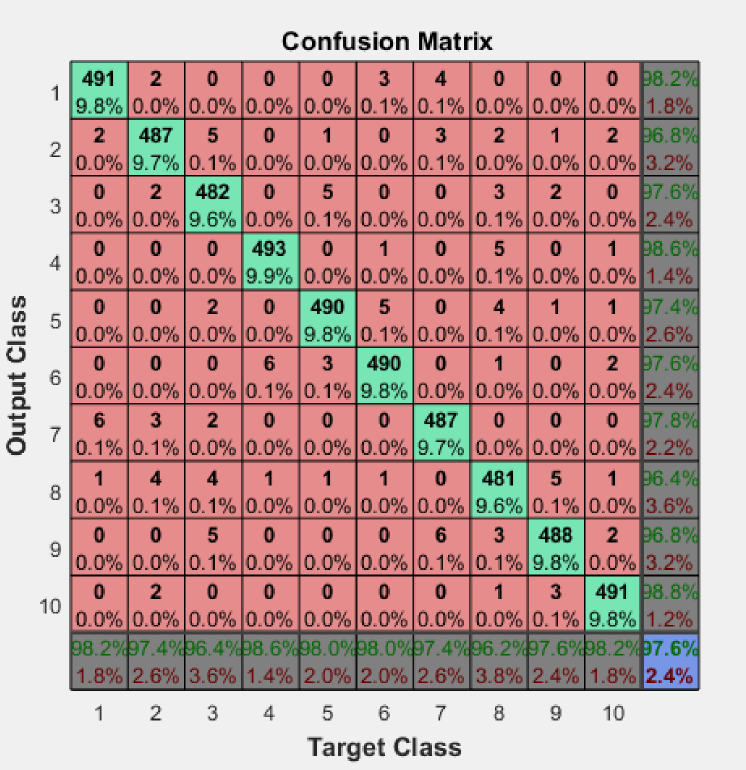
Architecture:

- Number of inputs nodes: 784 (28*28)
- Number of hidden layers: 2



	Hidden layer 1	Hidden layer 2
Number of Nodes	**50**	**25**
Maximum number of epochs	400	100
L2WeightRegularization	0.004	0.002
Sparsity Regularization	4	4
Sparsity Proportion	0.15	0.1
Scale Data	false	false

Confusion matrix (fine-tuned with max-epochs = 400):



Confusion Matrix

Output Class

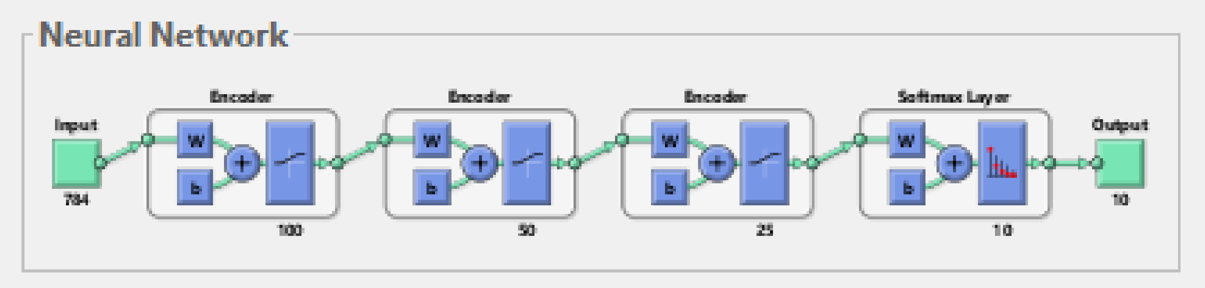
	1	2	3	4	5	6	7	8	9	10	
1	491 9.8%	2 0.0%	0 0.0%	0 0.0%	0 0.0%	3 0.1%	4 0.1%	0 0.0%	0 0.0%	0 0.0%	98.2% 1.8%
2	2 0.0%	487 9.7%	5 0.1%	0 0.0%	1 0.0%	0 0.0%	3 0.1%	2 0.0%	1 0.0%	2 0.0%	96.8% 3.2%
3	0 0.0%	2 0.0%	482 9.6%	0 0.0%	5 0.1%	0 0.0%	0 0.0%	3 0.1%	2 0.0%	0 0.0%	97.6% 2.4%
4	0 0.0%	0 0.0%	0 0.0%	493 9.9%	0 0.0%	1 0.0%	0 0.0%	5 0.1%	0 0.0%	1 0.0%	98.6% 1.4%
5	0 0.0%	0 0.0%	2 0.0%	0 0.0%	490 9.8%	5 0.1%	0 0.0%	4 0.1%	1 0.0%	1 0.0%	97.4% 2.6%
6	0 0.0%	0 0.0%	0 0.0%	6 0.1%	3 0.1%	490 9.8%	0 0.0%	1 0.0%	0 0.0%	2 0.0%	97.6% 2.4%
7	6 0.1%	3 0.1%	2 0.0%	0 0.0%	0 0.0%	0 0.0%	487 9.7%	0 0.0%	0 0.0%	0 0.0%	97.8% 2.2%
8	1 0.0%	4 0.1%	4 0.1%	1 0.0%	1 0.0%	1 0.0%	0 0.0%	481 9.6%	5 0.1%	1 0.0%	96.4% 3.6%
9	0 0.0%	0 0.0%	5 0.1%	0 0.0%	0 0.0%	0 0.0%	6 0.1%	3 0.1%	488 9.8%	2 0.0%	96.8% 3.2%
10	0 0.0%	2 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	1 0.0%	3 0.1%	491 9.8%	98.8% 1.2%
	98.2% 1.8%	97.4% 2.6%	96.4% 3.6%	98.6% 1.4%	98.0% 2.0%	98.0% 2.0%	97.4% 2.6%	96.2% 3.8%	97.6% 2.4%	98.2% 1.8%	97.6% 2.4%

Target Class

An accuracy of 97.6% is obtained for the network with two hidden layers of sizes 50 and 25 respectively.

- Now we try to increase the number of layers and check the accuracy again:

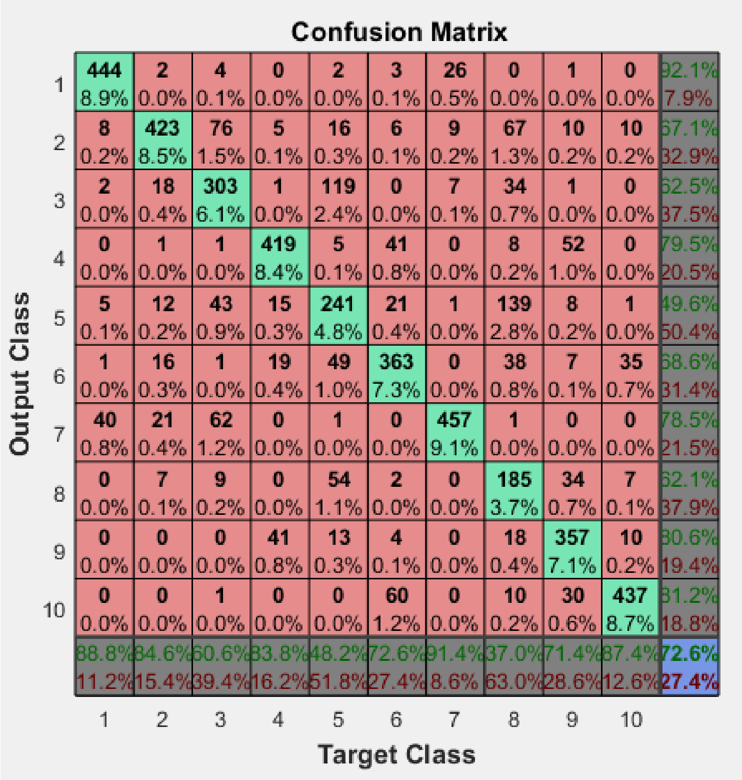
  Architecture:
  - Number of inputs nodes: 784 (28*28)
  - Number of hidden layers: **3**



	Hidden layer 1	Hidden layer 2	Hidden layer 3
Number of Nodes	**100**	**50**	**25**
Maximum number of epochs	400	100	400
L2WeightRegularization	0.004	0.002	0.002
Sparsity Regularization	4	4	4
Sparsity Proportion	0.15	0.1	0.1
Scale Data	false	false	false

- The results were a bit peculiar in this case. The accuracy achieved here after fine-tuning once was comparatively low (72.6%). It was observed that maximum epoch was reached before the minimum gradient reached. So, the model is fined tuned multiple times (effectively increasing the number of fine-tune epochs) and the accuracies improved to 96.1%.

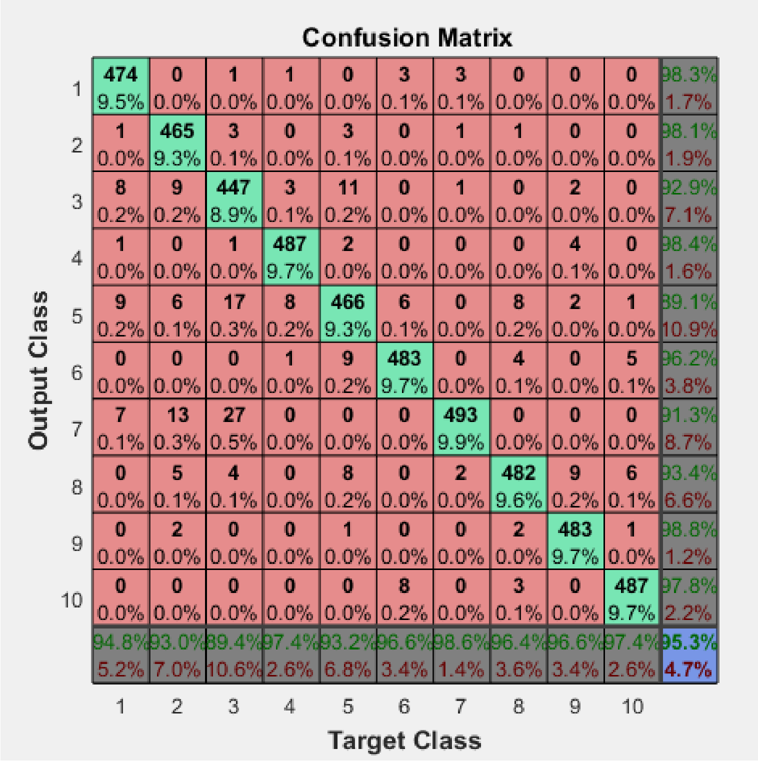
Confusion matrix (fine-tuned with max-epochs: 400):



Confusion Matrix

Output Class \ Target Class	1	2	3	4	5	6	7	8	9	10	
1	444 8.9%	2 0.0%	4 0.1%	0 0.0%	2 0.0%	3 0.1%	26 0.5%	0 0.0%	1 0.0%	0 0.0%	92.1% 7.9%
2	8 0.2%	423 8.5%	76 1.5%	5 0.1%	16 0.3%	6 0.1%	9 0.2%	67 1.3%	10 0.2%	10 0.2%	67.1% 32.9%
3	2 0.0%	18 0.4%	303 6.1%	1 0.0%	119 2.4%	0 0.0%	7 0.1%	34 0.7%	1 0.0%	0 0.0%	62.5% 37.5%
4	0 0.0%	1 0.0%	1 0.0%	419 8.4%	5 0.1%	41 0.8%	0 0.0%	8 0.2%	52 1.0%	0 0.0%	79.5% 20.5%
5	5 0.1%	12 0.2%	43 0.9%	15 0.3%	241 4.8%	21 0.4%	1 0.0%	139 2.8%	8 0.2%	1 0.0%	49.6% 50.4%
6	1 0.0%	16 0.3%	1 0.0%	19 0.4%	49 1.0%	363 7.3%	0 0.0%	38 0.8%	7 0.1%	35 0.7%	68.6% 31.4%
7	40 0.8%	21 0.4%	62 1.2%	0 0.0%	1 0.0%	0 0.0%	457 9.1%	1 0.0%	0 0.0%	0 0.0%	78.5% 21.5%
8	0 0.0%	7 0.1%	9 0.2%	0 0.0%	54 1.1%	2 0.0%	0 0.0%	185 3.7%	34 0.7%	7 0.1%	62.1% 37.9%
9	0 0.0%	0 0.0%	0 0.0%	41 0.8%	13 0.3%	4 0.1%	0 0.0%	18 0.4%	357 7.1%	10 0.2%	80.6% 19.4%
10	0 0.0%	0 0.0%	1 0.0%	0 0.0%	0 0.0%	60 1.2%	0 0.0%	10 0.2%	30 0.6%	437 8.7%	81.2% 18.8%
	88.8% 11.2%	84.6% 15.4%	60.6% 39.4%	83.8% 16.2%	48.2% 51.8%	72.6% 27.4%	91.4% 8.6%	37.0% 63.0%	71.4% 28.6%	87.4% 12.6%	72.6% 27.4%

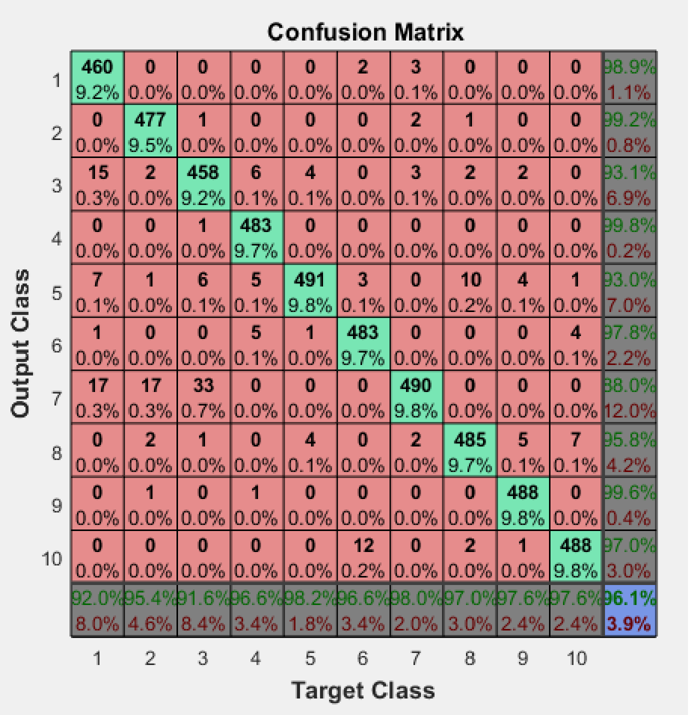
Confusion matrix (fine-tuned with max-epochs = 800):



Confusion Matrix

Output Class \ Target Class	1	2	3	4	5	6	7	8	9	10	
1	474 9.5%	0 0.0%	1 0.0%	1 0.0%	0 0.0%	3 0.1%	3 0.1%	0 0.0%	0 0.0%	0 0.0%	98.3% 1.7%
2	1 0.0%	465 9.3%	3 0.1%	0 0.0%	3 0.1%	0 0.0%	1 0.0%	1 0.0%	0 0.0%	0 0.0%	98.1% 1.9%
3	8 0.2%	9 0.2%	447 8.9%	3 0.1%	11 0.2%	0 0.0%	1 0.0%	0 0.0%	2 0.0%	0 0.0%	92.9% 7.1%
4	1 0.0%	0 0.0%	1 0.0%	487 9.7%	2 0.0%	0 0.0%	0 0.0%	0 0.0%	4 0.1%	0 0.0%	98.4% 1.6%
5	9 0.2%	6 0.1%	17 0.3%	8 0.2%	466 9.3%	6 0.1%	0 0.0%	8 0.2%	2 0.0%	1 0.0%	89.1% 10.9%
6	0 0.0%	0 0.0%	0 0.0%	1 0.0%	9 0.2%	483 9.7%	0 0.0%	4 0.1%	0 0.0%	5 0.1%	96.2% 3.8%
7	7 0.1%	13 0.3%	27 0.5%	0 0.0%	0 0.0%	0 0.0%	493 9.9%	0 0.0%	0 0.0%	0 0.0%	91.3% 8.7%
8	0 0.0%	5 0.1%	4 0.1%	0 0.0%	8 0.2%	0 0.0%	2 0.0%	482 9.6%	9 0.2%	6 0.1%	93.4% 6.6%
9	0 0.0%	2 0.0%	0 0.0%	0 0.0%	1 0.0%	0 0.0%	0 0.0%	2 0.0%	483 9.7%	1 0.0%	98.8% 1.2%
10	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	8 0.2%	0 0.0%	3 0.1%	0 0.0%	487 9.7%	97.8% 2.2%
	94.8% 5.2%	93.0% 7.0%	89.4% 10.6%	97.4% 2.6%	93.2% 6.8%	96.6% 3.4%	98.6% 1.4%	96.4% 3.6%	96.6% 3.4%	97.4% 2.6%	95.3% 4.7%

Confusion matrix (fine-tuned with max-epochs = 1200):



Confusion Matrix

Output Class

	1	2	3	4	5	6	7	8	9	10	
1	460 9.2%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	2 0.0%	3 0.1%	0 0.0%	0 0.0%	0 0.0%	98.9% 1.1%
2	0 0.0%	477 9.5%	1 0.0%	0 0.0%	0 0.0%	0 0.0%	2 0.0%	1 0.0%	0 0.0%	0 0.0%	99.2% 0.8%
3	15 0.3%	2 0.0%	458 9.2%	6 0.1%	4 0.1%	0 0.0%	3 0.1%	2 0.0%	2 0.0%	0 0.0%	93.1% 6.9%
4	0 0.0%	0 0.0%	1 0.0%	483 9.7%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	99.8% 0.2%
5	7 0.1%	1 0.0%	6 0.1%	5 0.1%	491 9.8%	3 0.1%	0 0.0%	10 0.2%	4 0.1%	1 0.0%	93.0% 7.0%
6	1 0.0%	0 0.0%	0 0.0%	5 0.1%	1 0.0%	483 9.7%	0 0.0%	0 0.0%	0 0.0%	4 0.1%	97.8% 2.2%
7	17 0.3%	17 0.3%	33 0.7%	0 0.0%	0 0.0%	0 0.0%	490 9.8%	0 0.0%	0 0.0%	0 0.0%	88.0% 12.0%
8	0 0.0%	2 0.0%	1 0.0%	0 0.0%	4 0.1%	0 0.0%	2 0.0%	485 9.7%	5 0.1%	7 0.1%	95.8% 4.2%
9	0 0.0%	1 0.0%	0 0.0%	1 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	488 9.8%	0 0.0%	99.6% 0.4%
10	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	12 0.2%	0 0.0%	2 0.0%	1 0.0%	488 9.8%	97.0% 3.0%
	92.0% 8.0%	95.4% 4.6%	91.6% 8.4%	96.6% 3.4%	98.2% 1.8%	96.6% 3.4%	98.0% 2.0%	97.0% 3.0%	97.6% 2.4%	97.6% 2.4%	96.1% 3.9%

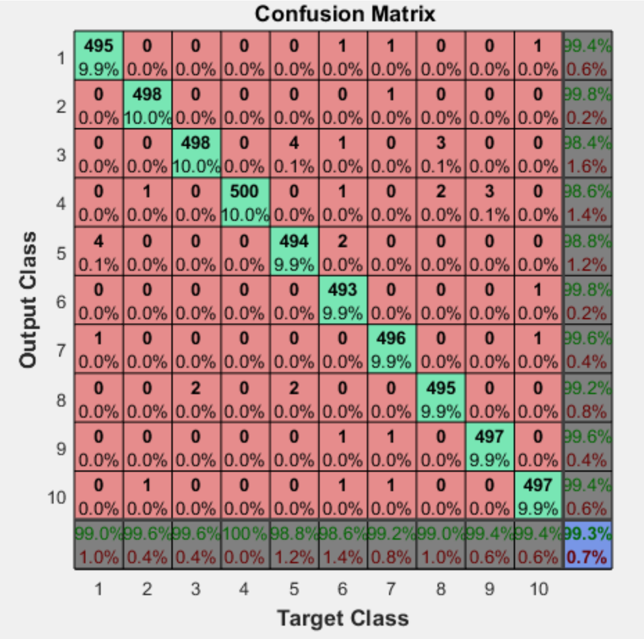
Target Class

Thus, the final accuracy achieved was 96.1% after 1200 epochs for the network training, in this case of 3 hidden layers. (Further increase in the number of epochs did not result in accuracy improvement)

- It is also observed that increasing the *sparsity proportions* on hidden layer 1 and 2 from 0.15 and 0.1 respectively to 0.4 each improves the accuracy to 99.3%.

	Hidden layer 1	Hidden layer 2
Number of Nodes	100	50
Maximum number of epochs	400	100
L2WeightRegularization	0.004	0.002
Sparsity Regularization	4	4
Sparsity Proportion	**0.4**	**0.4**
Scale Data	false	false

Confusion matrix (fine-tuned):



Confusion Matrix

Output Class

	1	2	3	4	5	6	7	8	9	10	
1	495 9.9%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	1 0.0%	1 0.0%	0 0.0%	0 0.0%	1 0.0%	99.4% 0.6%
2	0 0.0%	498 10.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	1 0.0%	0 0.0%	0 0.0%	0 0.0%	99.8% 0.2%
3	0 0.0%	0 0.0%	498 10.0%	0 0.0%	4 0.1%	1 0.0%	0 0.0%	3 0.1%	0 0.0%	0 0.0%	98.4% 1.6%
4	0 0.0%	1 0.0%	0 0.0%	500 10.0%	0 0.0%	1 0.0%	0 0.0%	2 0.0%	3 0.1%	0 0.0%	98.6% 1.4%
5	4 0.1%	0 0.0%	0 0.0%	0 0.0%	494 9.9%	2 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	98.8% 1.2%
6	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	493 9.9%	0 0.0%	0 0.0%	0 0.0%	1 0.0%	99.8% 0.2%
7	1 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	496 9.9%	0 0.0%	0 0.0%	1 0.0%	99.6% 0.4%
8	0 0.0%	0 0.0%	2 0.0%	0 0.0%	2 0.0%	0 0.0%	0 0.0%	495 9.9%	0 0.0%	0 0.0%	99.2% 0.8%
9	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	1 0.0%	1 0.0%	0 0.0%	497 9.9%	0 0.0%	99.6% 0.4%
10	0 0.0%	1 0.0%	0 0.0%	0 0.0%	0 0.0%	1 0.0%	1 0.0%	0 0.0%	0 0.0%	497 9.9%	99.4% 0.6%
	99.0% 1.0%	99.6% 0.4%	99.6% 0.4%	100% 0.0%	98.8% 1.2%	98.6% 1.4%	99.2% 0.8%	99.0% 1.0%	99.4% 0.6%	99.4% 0.6%	99.3% 0.7%

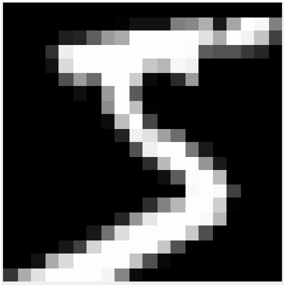
Target Class

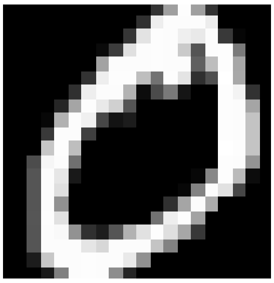
	Number of hidden layers	Number of nodes: Hidden layer 1 (Sparsity proportion in ())	Number of nodes: Hidden layer 2 (Sparsity proportion in ())	Number of nodes: Hidden layer 3	Final Accuracy (%)
Model1	2	100 (0.15)	50 (0.1)	NA	98.5
Model2	2	50 (0.15)	25 (0.1)	NA	97.6
Model3	2	100 (0.4)	50 (0.4)	NA	99.3
Model4	3	100 (0.15)	50 (0.1)	25	96.1

It can be observed that the accuracies vary with any changes in the parameters like the number of nodes or layers. It should also be noted that, while the accuracies vary, the variation is not too high (see the table above).

MNIST DATA (*Extra credit*):

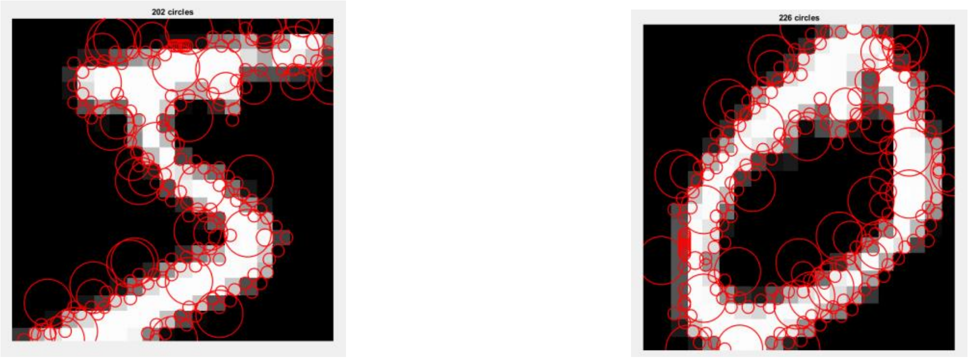
- Load the data to MATLAB and inspect a few of your images using tools you learned about this semester. Upscale the images you inspect so that you can actually see them.
  - The MNIST data is loaded and processed (trimmed to 20x20 and normalized as in the tutorial provided). Below are some examples. (up scaled to be visible)





  - To check out interesting features on these images, few tools were tried out on these images.

    Some of the results are:



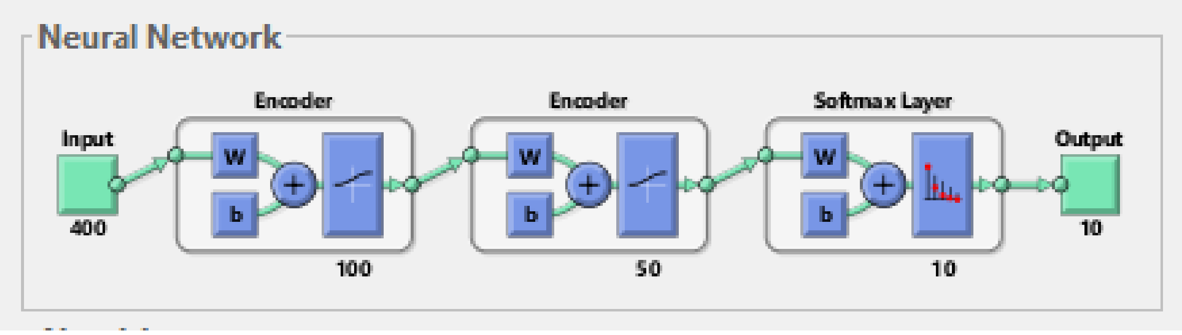
Blob detection on digits

  - This gives us the corners of the digit which can further be used to identify it. One idea would be to train directly based on the blob locations rather than pixel intensities.

- Repeat the steps of the tutorial with “real" images.
  - The MNIST data is downloaded, unzipped and loaded. Now this data (train images, train labels, test images, test labels) is fed to the steps followed in the tutorial. All the steps are repeated on the MNIST data and the results are as shown below.

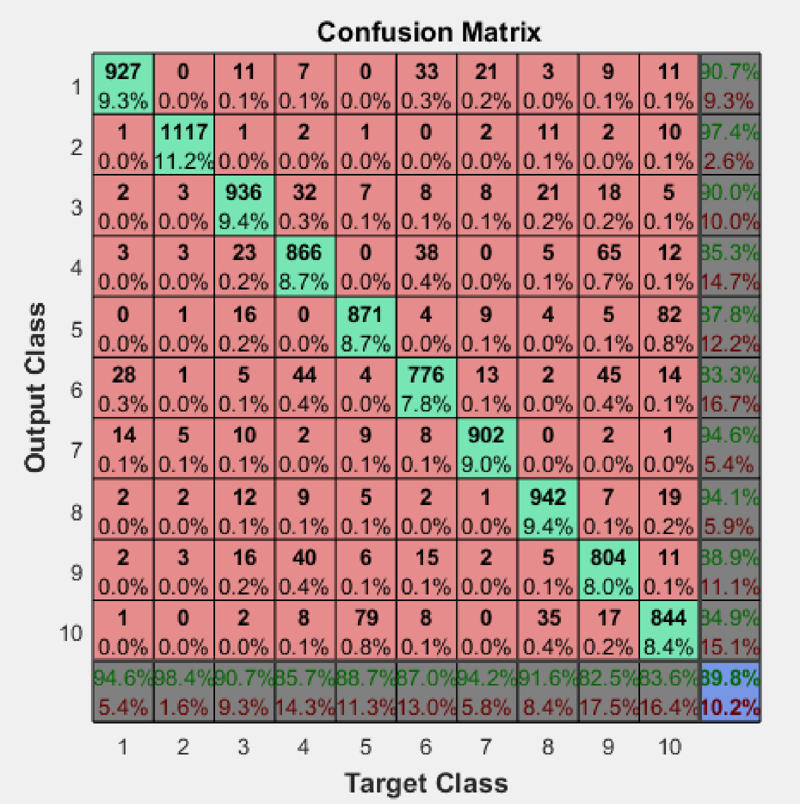
Architecture 1:

- Number of inputs nodes: 400 (20*20)
- Number of hidden layers: 2



	Hidden layer 1	Hidden layer 2
Number of Nodes	**100**	**50**
Maximum number of epochs	400	100
L2WeightRegularization	0.004	0.002
Sparsity Regularization	4	4
Sparsity Proportion	0.15	0.1
Scale Data	false	false

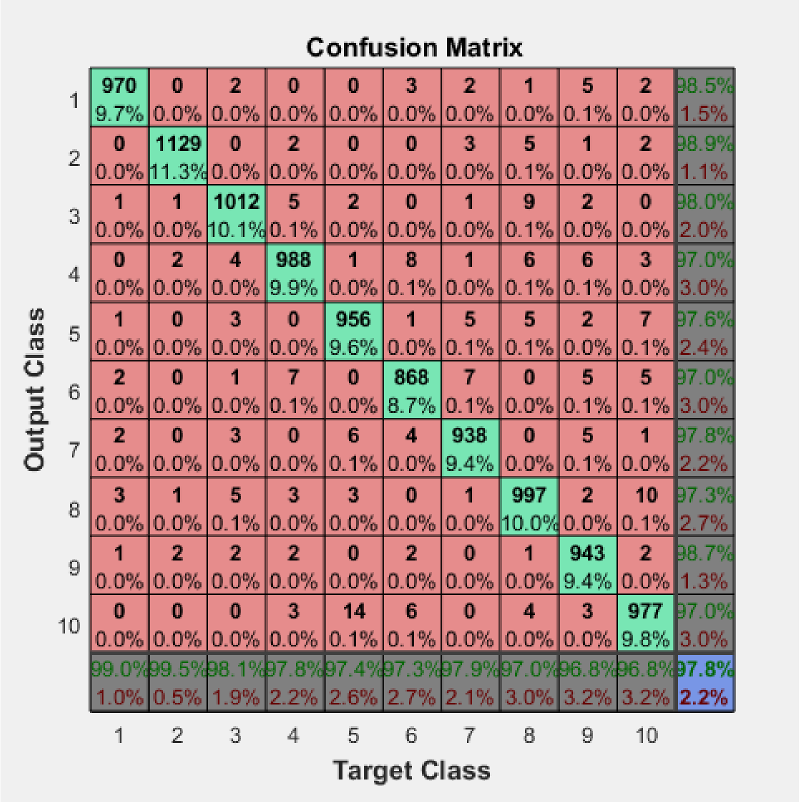
Confusion matrix (pre fine-tuned):



Confusion Matrix

Output Class \ Target Class	1	2	3	4	5	6	7	8	9	10	
1	927 9.3%	0 0.0%	11 0.1%	7 0.1%	0 0.0%	33 0.3%	21 0.2%	3 0.0%	9 0.1%	11 0.1%	90.7% 9.3%
2	1 0.0%	1117 11.2%	1 0.0%	2 0.0%	1 0.0%	0 0.0%	2 0.0%	11 0.1%	2 0.0%	10 0.1%	97.4% 2.6%
3	2 0.0%	3 0.0%	936 9.4%	32 0.3%	7 0.1%	8 0.1%	8 0.1%	21 0.2%	18 0.2%	5 0.1%	90.0% 10.0%
4	3 0.0%	3 0.0%	23 0.2%	866 8.7%	0 0.0%	38 0.4%	0 0.0%	5 0.1%	65 0.7%	12 0.1%	85.3% 14.7%
5	0 0.0%	1 0.0%	16 0.2%	0 0.0%	871 8.7%	4 0.0%	9 0.1%	4 0.0%	5 0.1%	82 0.8%	87.8% 12.2%
6	28 0.3%	1 0.0%	5 0.1%	44 0.4%	4 0.0%	776 7.8%	13 0.1%	2 0.0%	45 0.4%	14 0.1%	83.3% 16.7%
7	14 0.1%	5 0.1%	10 0.1%	2 0.0%	9 0.1%	8 0.1%	902 9.0%	0 0.0%	2 0.0%	1 0.0%	94.6% 5.4%
8	2 0.0%	2 0.0%	12 0.1%	9 0.1%	5 0.1%	2 0.0%	1 0.0%	942 9.4%	7 0.1%	19 0.2%	94.1% 5.9%
9	2 0.0%	3 0.0%	16 0.2%	40 0.4%	6 0.1%	15 0.1%	2 0.0%	5 0.1%	804 8.0%	11 0.1%	88.9% 11.1%
10	1 0.0%	0 0.0%	2 0.0%	8 0.1%	79 0.8%	8 0.1%	0 0.0%	35 0.4%	17 0.2%	844 8.4%	84.9% 15.1%
	94.6% 5.4%	98.4% 1.6%	90.7% 9.3%	85.7% 14.3%	88.7% 11.3%	87.0% 13.0%	94.2% 5.8%	91.6% 8.4%	82.5% 17.5%	83.6% 16.4%	89.8% 10.2%

Confusion matrix (fine-tuned):

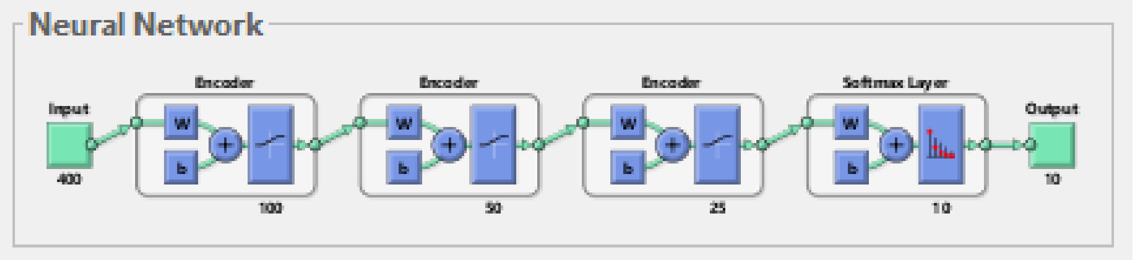


Confusion Matrix

Output Class \ Target Class	1	2	3	4	5	6	7	8	9	10	
1	970 9.7%	0 0.0%	2 0.0%	0 0.0%	0 0.0%	3 0.0%	2 0.0%	1 0.0%	5 0.1%	2 0.0%	98.5% 1.5%
2	0 0.0%	1129 11.3%	0 0.0%	2 0.0%	0 0.0%	0 0.0%	3 0.0%	5 0.1%	1 0.0%	2 0.0%	98.9% 1.1%
3	1 0.0%	1 0.0%	1012 10.1%	5 0.1%	2 0.0%	0 0.0%	1 0.0%	9 0.1%	2 0.0%	0 0.0%	98.0% 2.0%
4	0 0.0%	2 0.0%	4 0.0%	988 9.9%	1 0.0%	8 0.1%	1 0.0%	6 0.1%	6 0.1%	3 0.0%	97.0% 3.0%
5	1 0.0%	0 0.0%	3 0.0%	0 0.0%	956 9.6%	1 0.0%	5 0.1%	5 0.1%	2 0.0%	7 0.1%	97.6% 2.4%
6	2 0.0%	0 0.0%	1 0.0%	7 0.1%	0 0.0%	868 8.7%	7 0.1%	0 0.0%	5 0.1%	5 0.1%	97.0% 3.0%
7	2 0.0%	0 0.0%	3 0.0%	0 0.0%	6 0.1%	4 0.0%	938 9.4%	0 0.0%	5 0.1%	1 0.0%	97.8% 2.2%
8	3 0.0%	1 0.0%	5 0.1%	3 0.0%	3 0.0%	0 0.0%	1 0.0%	997 10.0%	2 0.0%	10 0.1%	97.3% 2.7%
9	1 0.0%	2 0.0%	2 0.0%	2 0.0%	0 0.0%	2 0.0%	0 0.0%	1 0.0%	943 9.4%	2 0.0%	98.7% 1.3%
10	0 0.0%	0 0.0%	0 0.0%	3 0.0%	14 0.1%	6 0.1%	0 0.0%	4 0.0%	3 0.0%	977 9.8%	97.0% 3.0%
	99.0% 1.0%	99.5% 0.5%	98.1% 1.9%	97.8% 2.2%	97.4% 2.6%	97.3% 2.7%	97.9% 2.1%	97.0% 3.0%	96.8% 3.2%	96.8% 3.2%	97.8% 2.2%

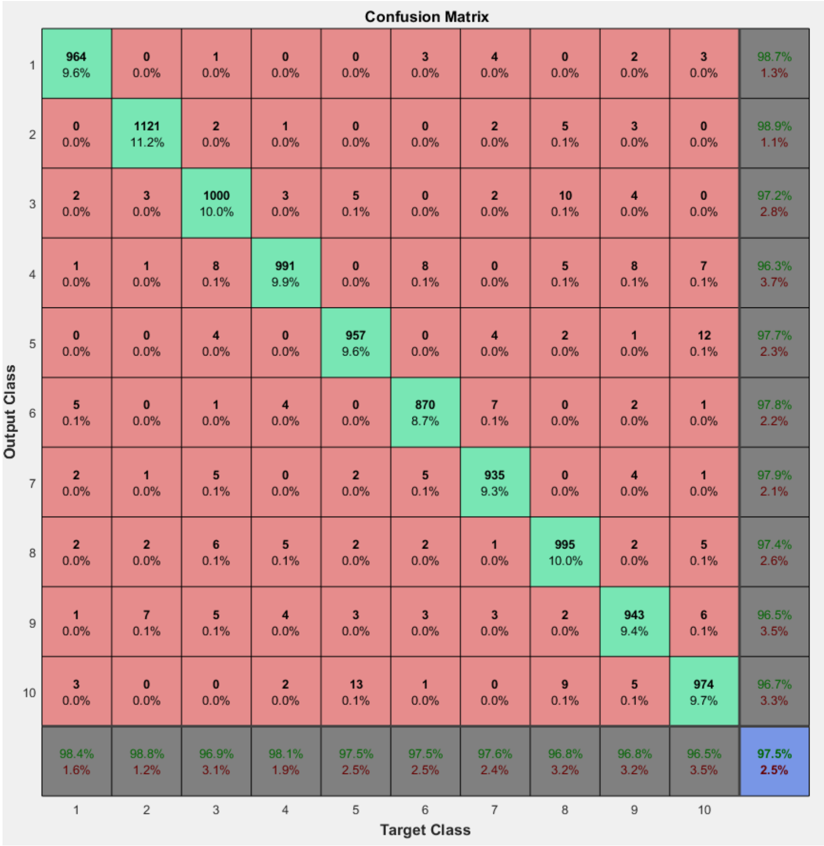
Architecture 2:

- ○ Number of inputs nodes: 400 (20*20)
- ○ Number of hidden layers: 3



	Hidden layer 1	Hidden layer 2	Hidden layer 3
Number of Nodes	**100**	**50**	**25**
Maximum number of epochs	400	100	400
L2WeightRegularization	0.004	0.002	0.002
Sparsity Regularization	4	4	4
Sparsity Proportion	0.15	0.1	0.1
Scale Data	false	false	false

Figure C: Confusion matrix (fine-tuned; final):

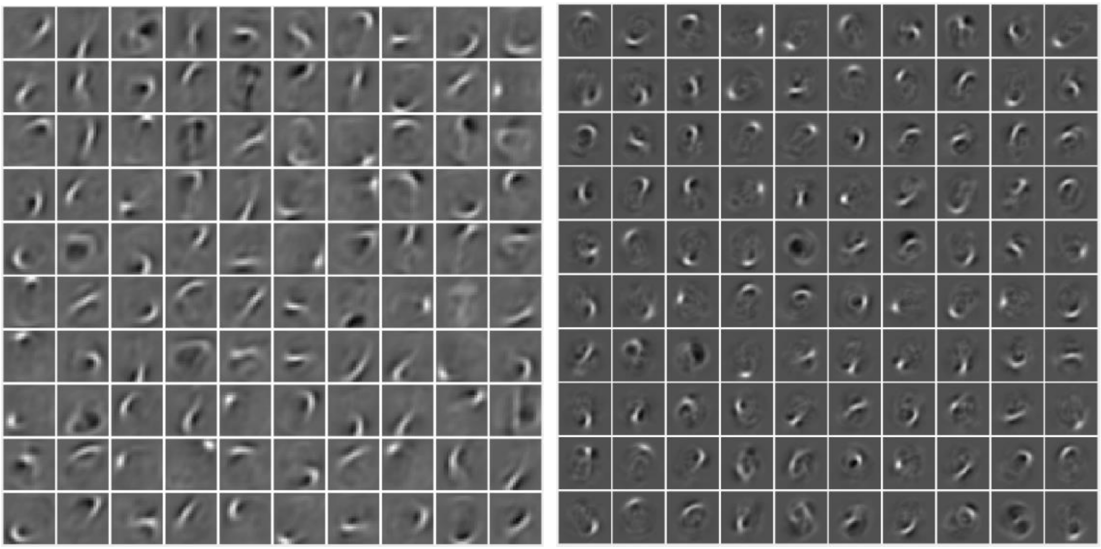


Confusion Matrix

Output Class / Target Class

	1	2	3	4	5	6	7	8	9	10	
1	964 9.6%	0 0.0%	1 0.0%	0 0.0%	0 0.0%	3 0.0%	4 0.0%	0 0.0%	2 0.0%	3 0.0%	98.7% 1.3%
2	0 0.0%	1121 11.2%	2 0.0%	1 0.0%	0 0.0%	0 0.0%	2 0.0%	5 0.1%	3 0.0%	0 0.0%	98.9% 1.1%
3	2 0.0%	3 0.0%	1000 10.0%	3 0.0%	5 0.1%	0 0.0%	2 0.0%	10 0.1%	4 0.0%	0 0.0%	97.2% 2.8%
4	1 0.0%	1 0.0%	8 0.1%	991 9.9%	0 0.0%	8 0.1%	0 0.0%	5 0.1%	8 0.1%	7 0.1%	96.3% 3.7%
5	0 0.0%	0 0.0%	4 0.0%	0 0.0%	957 9.6%	0 0.0%	4 0.0%	2 0.0%	1 0.0%	12 0.1%	97.7% 2.3%
6	5 0.1%	0 0.0%	1 0.0%	4 0.0%	0 0.0%	870 8.7%	7 0.1%	0 0.0%	2 0.0%	1 0.0%	97.8% 2.2%
7	2 0.0%	1 0.0%	5 0.1%	0 0.0%	2 0.0%	5 0.1%	935 9.3%	0 0.0%	4 0.0%	1 0.0%	97.9% 2.1%
8	2 0.0%	2 0.0%	6 0.1%	5 0.1%	2 0.0%	2 0.0%	1 0.0%	995 10.0%	2 0.0%	5 0.1%	97.4% 2.6%
9	1 0.0%	7 0.1%	5 0.1%	4 0.0%	3 0.0%	3 0.0%	3 0.0%	2 0.0%	943 9.4%	6 0.1%	96.5% 3.5%
10	3 0.0%	0 0.0%	0 0.0%	2 0.0%	13 0.1%	1 0.0%	0 0.0%	9 0.1%	5 0.1%	974 9.7%	96.7% 3.3%
	98.4% 1.6%	98.8% 1.2%	96.9% 3.1%	98.1% 1.9%	97.5% 2.5%	97.5% 2.5%	97.6% 2.4%	96.8% 3.2%	96.8% 3.2%	96.5% 3.5%	97.5% 2.5%

- Compare the results of the synthetic images with the results of “real” images. Are the features similar in appearance? Is performance better or worse than observed with the synthetic dataset?
    - The features from the first hidden layer (with 100 nodes) are compared for both synthetic and MNIST datasets. (shown below)



The Features from hidden layer (Left: MNIST data Right: Synthetic data)

- It can be seen that the features from the MNIST data hidden layer are different from those from Synthetic data hidden layer. This could be attributed to the higher variation in the ‘real’ dataset compared to the limited variation in the synthetic dataset. The synthetic data features mostly represent digits skewed at different angles while the MNIST data features represent more randomness of handwritten digits.

- The accuracies on both datasets are compared.

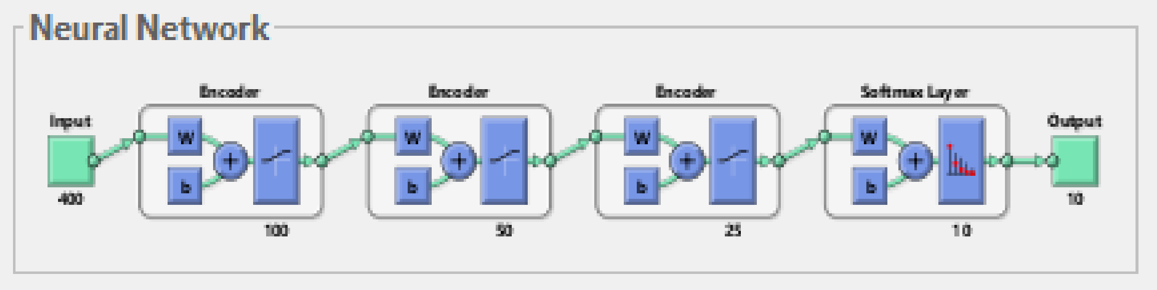
Accuracies (%)	2-layer Network	3-layer Network
Synthetic dataset (28*28)	98.5	96.1*
MNIST (20*20)	97.8	97.5

* - More number of epochs to reach optimal accuracy (Max epochs = 1200); rest = 400

- The accuracy for MNIST data is a little higher in this case of 3-layer network which can be attributed to the more training data available. For the 2-layer network, which has the default parameter settings in the tutorial, the synthetic data has better accuracy which can be attributed to better tuning of hyper parameters like number of nodes etc. with respect to that dataset.

Network, Parameter Details (Table D):

Data used	MNIST		
Number of Training Images	60000		
Number of testing Images	10000		
Number of Layers	3		
Initial weights	Random; non-zero; unequal		
Minimum Gradient (for early stopping)	10^-6		
Reconstruction error type	Mean squared error		
Number of nodes (Layer wise)	100,50,25		
Backpropagation	Scaled conjugate gradient		
Accuracy (final in %)	97.5		
	Hidden layer 1	Hidden layer 2	Hidden layer 3
Number of Nodes	**100**	**50**	**25**
Maximum number of epochs	400	100	400
L2WeightRegularization	0.004	0.002	0.002
Sparsity Regularization	4	4	4
Sparsity Proportion	0.15	0.1	0.1
Scale Data	false	false	false



- Provide a confusion matrix for your final classifier.
  - *Provided above*. (Figure C)
- Document your network architecture, parameters settings, and number of training and test images used.
  - *Table D*