CZ4042 Neural Networks

Project 2: Deep CNN & Autoencoders

Chaitanya Joshi (U1522971F)

Matouš Skála (N1602866H)

Part 1: Deep Convolutional Neural Network

The goal of this part of the project is to use deep convolutional neural networks (CNN) for object recognition in images. We use Python (using Tensorflow and TFLearn, a high-level API for tensorflow) to train all our neural networks. We start by creating a one hidden layer deep CNN to classify 28 x 28 px images from MNIST dataset into the 10 different classes of digits (0-9). We experiment with various hyperparameters for this model to select the optimum learning parameters for a two hidden layer deep CNN. We further experiment to find the optimum number of convolutional filters for this model, to maximise classification accuracy.

1.1. Data Preprocessing

The dataset given consists of images in JPG format, with folder names used as labels. We load the images into tensorflow arrays of size 28 x 28 x 1, where 1 is the number of channels. (For MNIST data, the images only have one black/white channel. Coloured images have three channels - Red, Green and Blue.) We also load the category label (digits 0 to 9) for each image into a 10 dimensional one-hot tensorflow array. We are given two sets of data, one for training (5000 images) and one for testing (1000 images).

1.2. One Hidden Layer Deep CNN

The network architecture consists of an input layer with dimensions 28 x 28, a convolutional layer consisting of 20 filters of 9 x 9 receptive fields, a mean pooling layer of pooling dimension 2 x 2, a fully connected layer connected to a softmax regression layer, followed by an output classification layer over the 10 categories. The network learns to minimise categorical cross-entropy loss through stochastic gradient descent with momentum. We employ L2-norm regularization with a coefficient of 0.001 on the model weights to prevent overfitting.

We conduct exhaustive experiments to determine model hyperparameters like learning rate, momentum parameter and decay term in gradient descent learning, or different batch sizes for stochastic gradient descent.

We trained models with different permutations of hyperparameters on the training data for 25 epochs each and chose the best parameters based on test dataset accuracy. The data was always shuffled before training and testing. The results are described in the sections below.

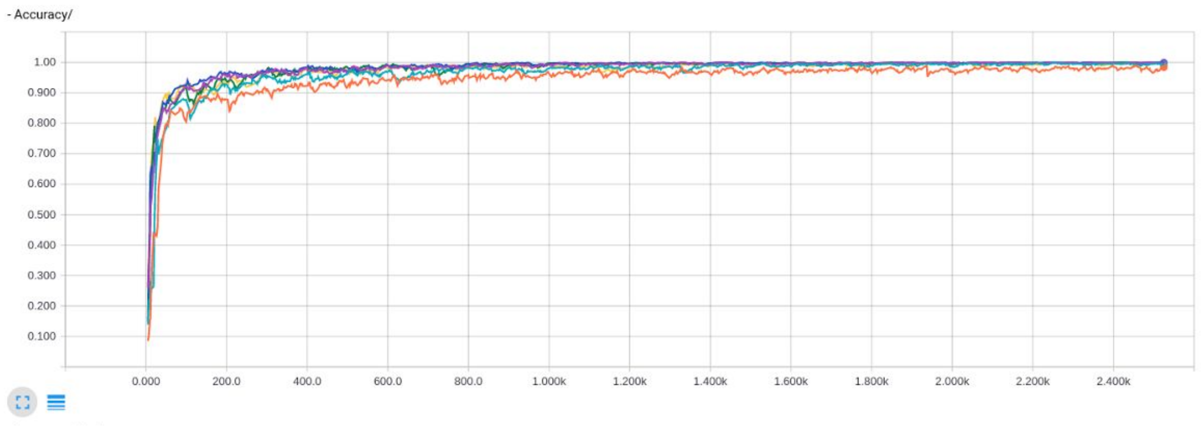
1.2.1 Learning Rate

We tested six different learning rates for the described CNN architecture, keeping all other model hyperparameters the same. We kept the momentum parameter as 0.95, the decay term as 0.95 and the batch size as 50. The results are shown in Table 1.

Momentum = 0.95, Decay Term = 0.95, Batch Size = 50	
Learning Rate	**Test Set Accuracy**
0.01	0.961
0.02	0.970
0.05	0.972
0.1	0.973
0.15	0.965
0.2	0.967

Table 1. Test Set Accuracy for various learning rates

Tensorboard screenshots for all six runs can be found in /img/1layer/learning-rate. We have provided graphs for accuracy and loss on both training and testing data. It is clear from Graph 1 that training converges for all six models.



Graph 1. Training Accuracy for models in Table 1 (according to color of plot).

We see from Table 1 that a learning rate of 0.1 gives the highest accuracy on the test set. Setting learning rate as 0.05 or 0.02 leads to marginally worse results with all the other hyperparameters fixed.

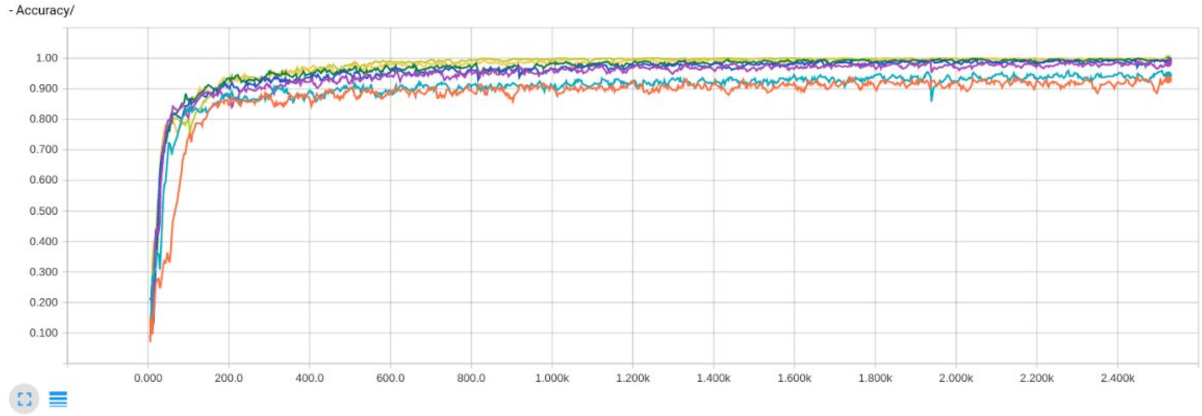
1.2.2 Momentum Parameter

We tested seven different values for the momentum parameter, keeping all other model hyperparameters the same. We kept the learning rate as 0.01, the decay term as 0.95 and the batch size as 50. The results are shown in Table 2.

Learning Rate = 0.01, Decay Term = 0.95, Batch Size = 50	
Momentum Parameter	**Test Set Accuracy**
0.50	0.902
0.75	0.932
0.90	0.961
0.93	0.964
0.95	0.966
0.97	0.968
0.99	0.970

Table 2. Test Set Accuracy for various momentum parameters

Tensorboard screenshots for all seven runs can be found in /img/1layer/momentum. We have provided graphs for accuracy and loss on both training and testing data. It is clear from Graph 2 that training converges for all seven models.



Graph 2. Training Accuracy for models in Table 2 (according to color of plot).

We see from Table 2 that setting momentum parameter as 0.99 gives the highest accuracy on the test set, keeping all the other hyperparameters fixed.

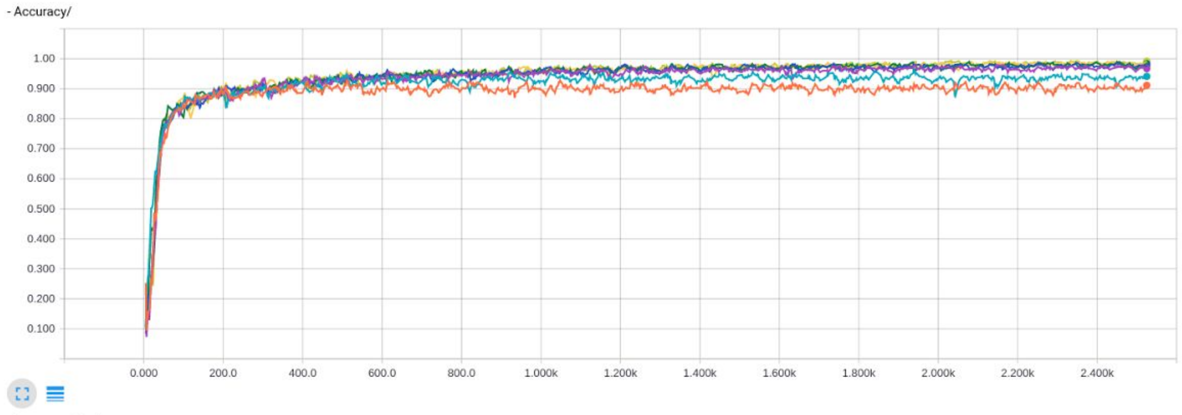
1.2.3 Decay Term

We tested seven different values for the decay term, keeping all other model hyperparameters the same. We kept the learning rate as 0.01, the momentum parameter as 0.9 and the batch size as 50. The results are shown in Table 3.

Learning Rate = 0.01, Momentum = 0.9, Batch Size = 50	
Decay Term	**Test Set Accuracy**
0.50	0.894
0.75	0.921
0.90	0.954
0.93	0.961
0.95	0.961
0.97	0.964
0.99	0.958

Table 3. Test Set Accuracy for various decay terms

Tensorboard screenshots for all seven runs can be found in /img/1layer/decay. We have provided graphs for accuracy and loss on both training and testing data. It is clear from Graph 3 that training converges for all seven models.



Graph 3. Training Accuracy for models in Table 3 (according to color of plot).

We see from Table 3 that setting decay term as 0.97 gives the highest accuracy on the test set, keeping all the other hyperparameters fixed. Results are only marginally worse for values such as 0.95 and 0.93.

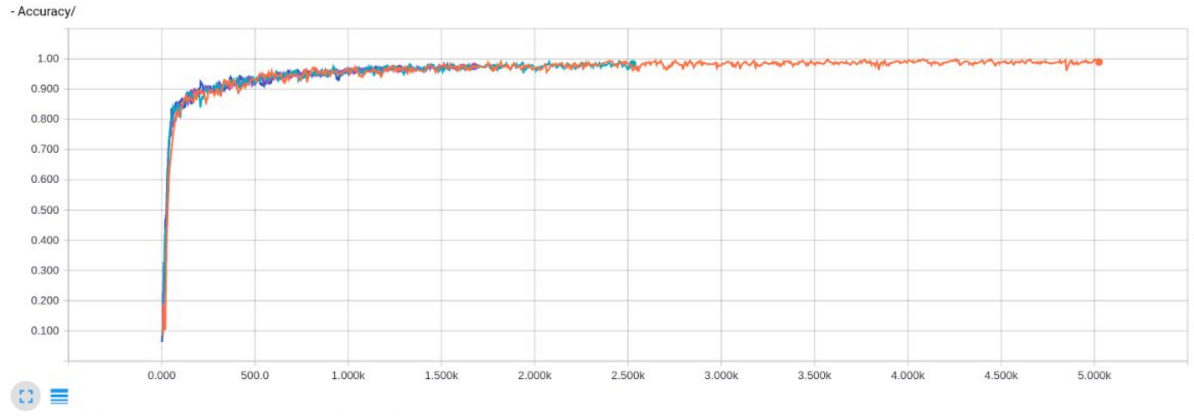
1.2.4 Batch Size

We tested four different batch sizes, keeping all other model hyperparameters the same. We kept the learning rate as 0.01, the momentum parameter as 0.9 and the decay term as 0.95. The results are shown in Table 4.

Learning Rate = 0.01, Momentum = 0.9, Decay Term = 0.95	
Batch Size	**Test Set Accuracy**
25	0.9624
50	0.9610
75	0.9620
100	0.9500

Table 4. Test Set Accuracy for various batch sizes

Tensorboard screenshots for all four runs can be found in /img/1layer/batch-size. We have provided graphs for accuracy and loss on both training and testing data. It is clear from Graph 4 that training converges for all four models.



Graph 4. Training Accuracy for models in Table 4 (according to color of plot).

We see from Table 4 that the smaller the batch sizes, the higher the accuracy on the test set, keeping all the other hyperparameters fixed. However, smaller batch sizes take linearly longer to train. Hence, we balance the marginal increase in accuracy with the extra time taken to train and select 50 as the optimum batch size.

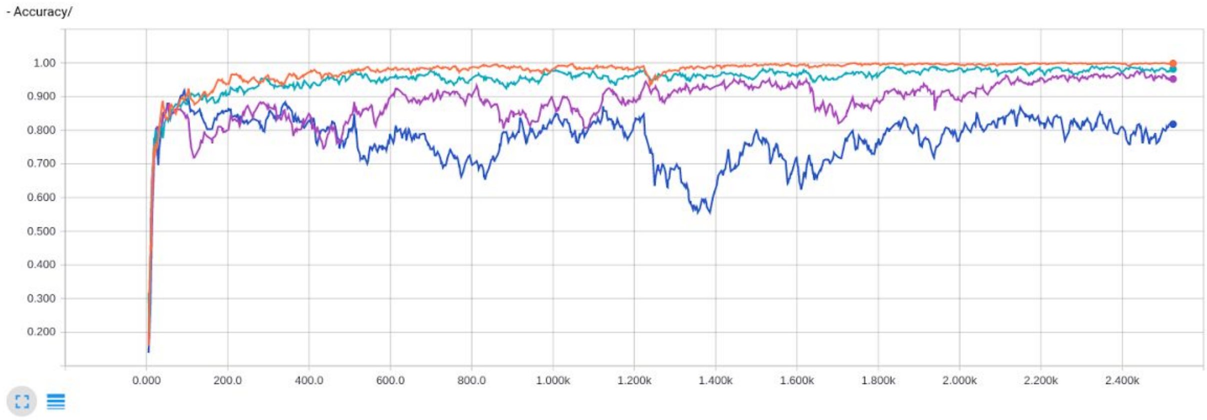
1.2.5 In Combination

Based on the four individual experiments, we ran further tests on hyperparameter values in combination. Our aim was to find the permutation of parameters to give the highest accuracy. The results are shown in Table 5.

Learning Rate = 0.1, Batch Size = 50		
Momentum Parameter	**Decay Term**	**Test Set Accuracy**
0.95	0.95	0.969
0.97	0.97	0.953
0.99	0.99	0.825
0.99	0.95	0.940

Table 5. Test Set Accuracy for various combinations of hyperparameters

Tensorboard screenshots for all four runs can be found in /img/1layer/combo1. We have provided graphs for accuracy and loss on both training and testing data. It is clear from Graph 5 that training converges for the orange and sky blue plots, but is unable to reach the global minima for the purple and blue plots.



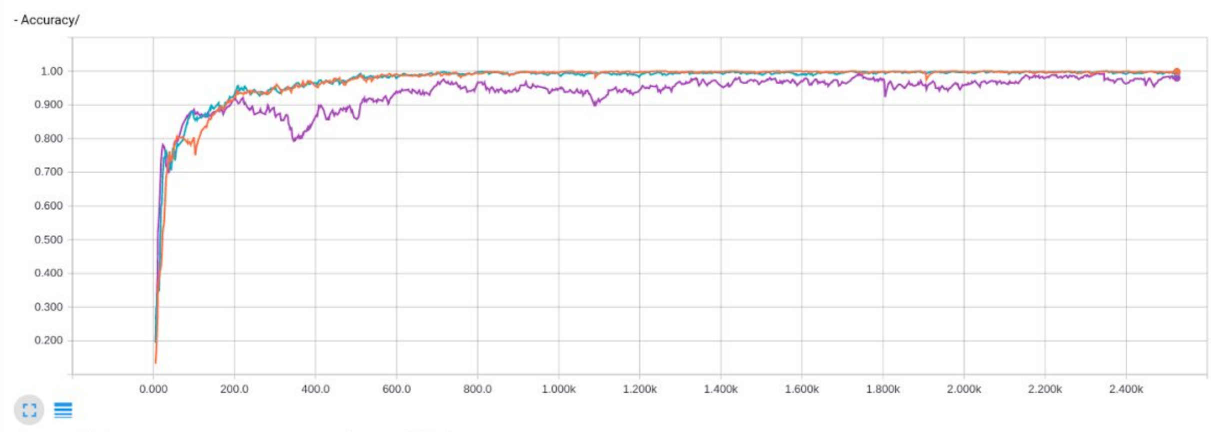
Graph 5. Training Accuracy for models in Table 5 (according to color of plot).

We suspected that the plots not converging is due to the learning rate being too high for the specific values of momentum and decay terms. We conducted further experiments to investigate this because none of the above models outperformed the best performing models from Section 1.2.1 and Section 1.2.2. The results are shown in Table 6.

Learning Rate	**Momentum Parameter**	**Decay Term**	**Batch Size**	**Test Set Accuracy**
0.01	0.99	0.95	50	0.970
0.02	0.99	0.95	50	0.973
0.05	0.99	0.95	50	0.960

Table 6. Test Set Accuracy for various combinations of hyperparameters (second experiment)

Tensorboard screenshots for all three runs can be found in /img/1layer/combo2. We have provided graphs for accuracy and loss on both training and testing data. It is clear from Graph 6 that training converges for all three models.



Graph 6. Training Accuracy for models in Table 6 (according to color of plot).

Decreasing the learning rate lead to considerable improvements in performance of the models. We describe our final model in the next section.

1.2.6 Final Model

Our final model is a one hidden layer deep CNN as described in Section 1.2, with learning rate set as 0.02, momentum parameter of 0.99, decay term of 0.95 and batch size at 50. It reaches an accuracy of 97.3% on the 1000 testing images set.

1.3 Two Hidden Layer Deep CNN

The network architecture consists of an input layer with dimensions 28 x 28, a convolutional layer consisting of 30 filters of 5 x 5 receptive fields, a mean pooling layer of pooling dimension 2 x 2, a second convolutional layer of 50 filters of 5 x 5 receptive fields, followed by another mean pooling layer of pooling dimension 2 x 2. Finally, we have a fully connected layer connected to a softmax regression layer, followed by an output classification layer over the 10 categories. The network learns to minimise categorical cross-entropy loss through stochastic gradient descent with momentum. We employ L2-norm regularization with a coefficient of 0.001 on the model weights to prevent overfitting.

We use the hyperparameters obtained experimentally in Section 1.2.6 (learning rate = 0.02, momentum parameter = 0.99, decay term = 0.95, batch size = 50) for the model.

We perform further experiments to determine the best configuration of number of filters in each of the two convolutional layers. All models were trained for 20 epochs on the training dataset and our final model was the one with the highest accuracy on the test dataset. The data was always shuffled before training and testing. The results are described in the sections below.

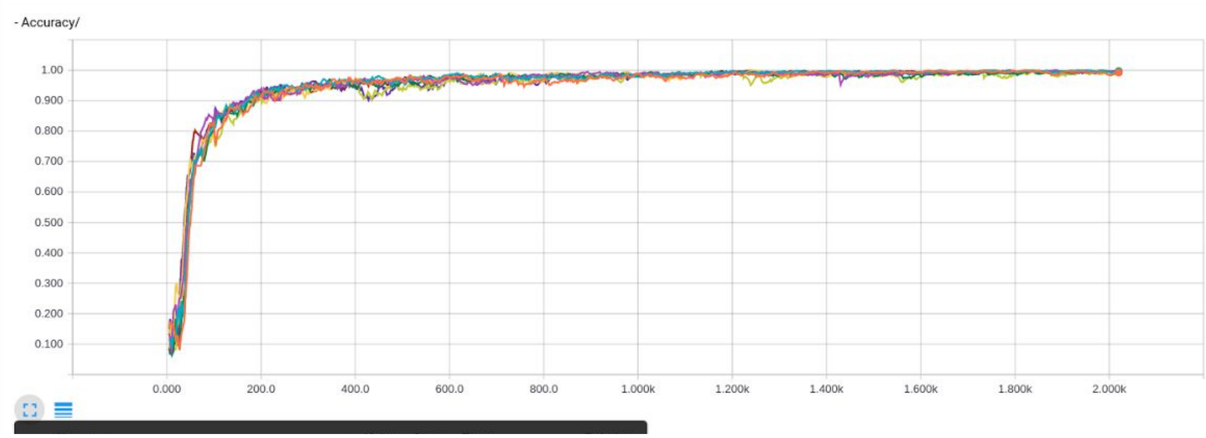
1.3.1 Number of Filters

We experimented with different permutations of number of filters in each of the two convolutional layers. We take three different values for number of filters (30, 50 and 65) and run nine different experiments to narrow down on the best configuration. The results are are shown in Table 7.

Learning Rate = 0.02, Momentum = 0.99, Decay Term = 0.95, Batch Size = 50		
Number of Filters (Hidden Layer 1)	**Number of Filters (Hidden Layer 2)**	**Test Set Accuracy**
30	30	0.970
30	50	0.980
30	65	0.981
50	30	0.980
50	50	0.975
50	65	0.978
65	30	0.974
65	50	0.974
65	65	0.977

Table 7. Test Set Accuracy for different filter permutations

Tensorboard screenshots for all nine runs can be found in /img/2layer/initial. We have provided graphs for accuracy and loss on both training and testing data. It is clear from Graph 7 that training converges for all nine models.



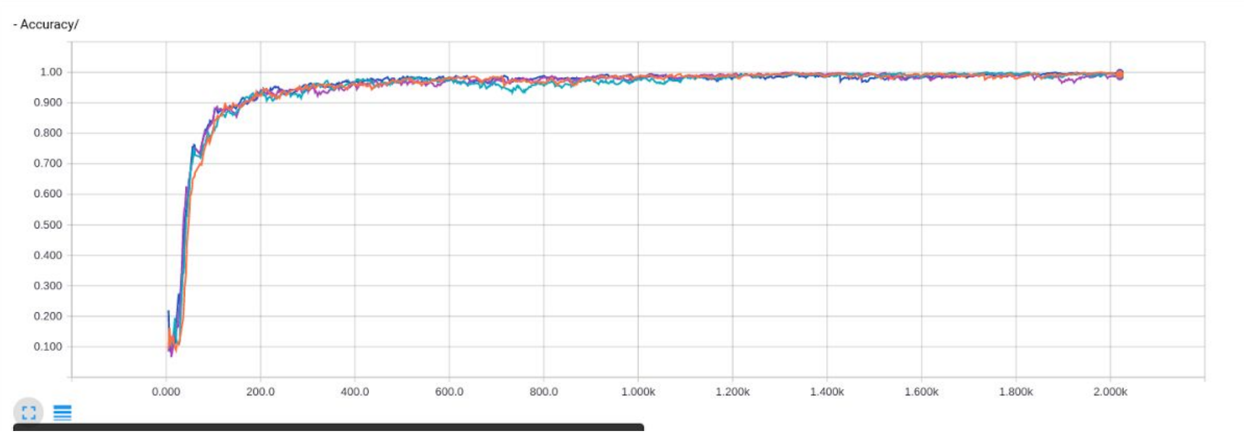
Graph 7. Training Accuracy for models in Table 7 (according to color of plot).

The results in Table 7 can be used as a heuristic to further experiment with the number of filters. We decided to keep second hidden layer filters fixed at 65 and experimented with number of filters in the first hidden layer within the range of 25 to 45. The results are shown in Table 8.

Learning Rate = 0.02, Momentum = 0.99, Decay Term = 0.95, Batch Size = 50		
Number of Filters (Hidden Layer 1)	**Number of Filters (Hidden Layer 2)**	**Test Set Accuracy**
25	65	0.973
35	65	0.962
40	65	0.970
45	65	0.981

Table 8. Test Set Accuracy for different filter permutations

Tensorboard screenshots for all four runs can be found in /img/2layer/testing. We have provided graphs for accuracy and loss on both training and testing data. It is clear from Graph 9 that training converges for all four models.



Graph 8. Training Accuracy for models in Table 8 (according to color of plot).

From Table 7 and Table 8, we find that the best accuracy on the test set (98.1%) is obtained by two combinations of number of filters (30-65 and 45-65). We chose the model with fewer filters because our findings from Table 8 show that there is no advantage of increasing number of filters in the first hidden layer.

1.3.2 Final Model

Our final model is a two hidden layer deep CNN as described in Section 1.3, with learning rate set as 0.02, momentum parameter of 0.99, decay term of 0.95 and batch size at 50. The number of filters in the first and second hidden layers are 30 and 65 respectively. It reaches an accuracy of 98.1% on the 1000 testing images set.

1.4 Running the Code

The folder contains two python files- cnn.py for the one hidden layer network and cnn_multilayer.py for the two hidden layer network. They can be run using command line flags such as python cnn.py --learning_rate=0.1 to modify model hyperparameters.

For each run, a Tensorboard summary is saved in /summaries, a snapshot of the model's weights is saved every 50 steps in /checkpoints, and snapshots at which the previous best test set accuracy is beaten are saved in /best_checkpoints. The summary can be used to visualize the results of the run on Tensorboard. The checkpoints can also be loaded to use on new data by following TFLearn documentation.

Part 2: Autoencoders

The goal of this part of the project is to make use of neural networks as content addressable memory. For this task, Matlab and its Neural Network Toolbox is used. We start by creating an autoencoder for encoding 28 x 28 px images from MNIST dataset into 100 hidden neurons. Then, we add the second layer of autoencoder for encoding extracted features into 50 neurons. Eventually, we add a softmax layer to recognize the encoded images.

2.1. Data Preprocessing

The dataset given consists of images in JPG format, with folder names used as labels. To be able to further process data in Matlab, we need to load the image datastore, decode images, convert pixel values to range of <0, 1> and store them as matrices in a cell array. Data preprocessing is implemented in **autoencoderProcess.m**. We get 4 files in total on output. **dataTrain.mat** and **dataTest.mat** contain train and test images in the cell array of 28 x 28 matrices. **labelsTrain.mat** and **labelsTest.mat** contain train and test labels used for training and evaluation of the softmax layer in the last step.

2.2. The First Autoencoder

First we trained an autoencoder with 100 hidden neurons, encoding images from the train dataset. Only the images without labels have been used, as autoencoder is a neural network based on unsupervised learning.

We tried to find the best parameters for sparse autoencoder by changing **sparsity proportion** which indicates the desired proportion of training samples a neuron should react to, and **sparsity regularization coefficient** that controls the impact of sparsity regularizer in the cost function. After training the model with different parameters shown in *Table 1* and evaluating mean square error (MSE) of the test set reconstruction, the best combination has been chosen with sparsity proportion 0.3 and regularization coefficient 1.

		sparsity regularization coefficient		
		1	**2**	**4**
sparsity proportion	**0.1**	0.0153	0.0163	0.0187
	0.2	0.0124	0.0135	0.0156
	0.3	**0.0109**	0.0114	0.0133

Table 1. MSE in relation to sparsity proportion and regularization coefficient after 200 epochs

Subsequently, we tried running the training with increasing number of epochs. After looking at *Table 2* or *Figure 1*, we can easily observe that the MSE of test set reconstruction is continuously decreasing with each additional epoch, but the improvement is not much significant after epoch 200.

epoch	**MSE**
100	0.0151
200	0.0109
300	0.0103
400	0.01
500	0.0096
600	0.0094
700	0.0092
800	0.0092
900	0.009
1000	**0.0089**

Table 2. Test set reconstruction MSE in relation to number of epochs

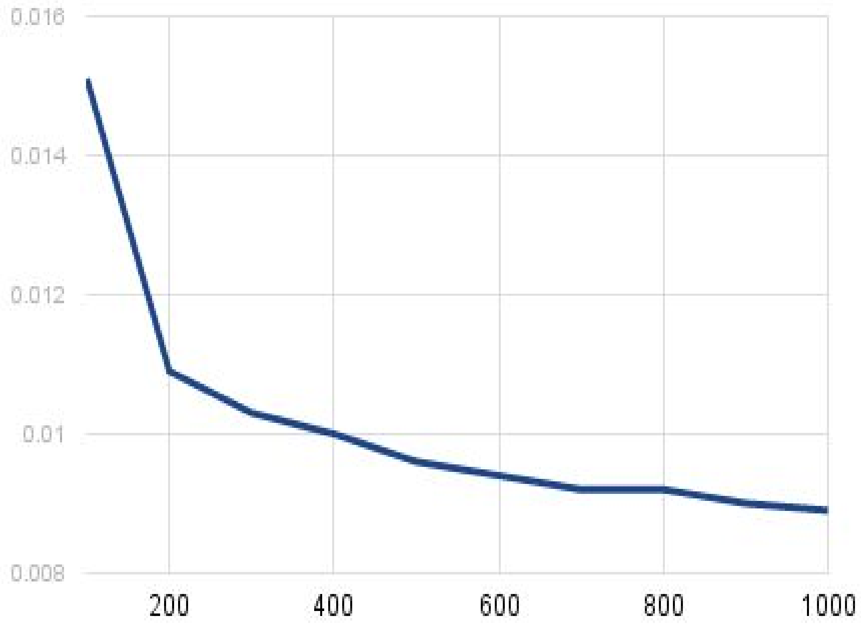
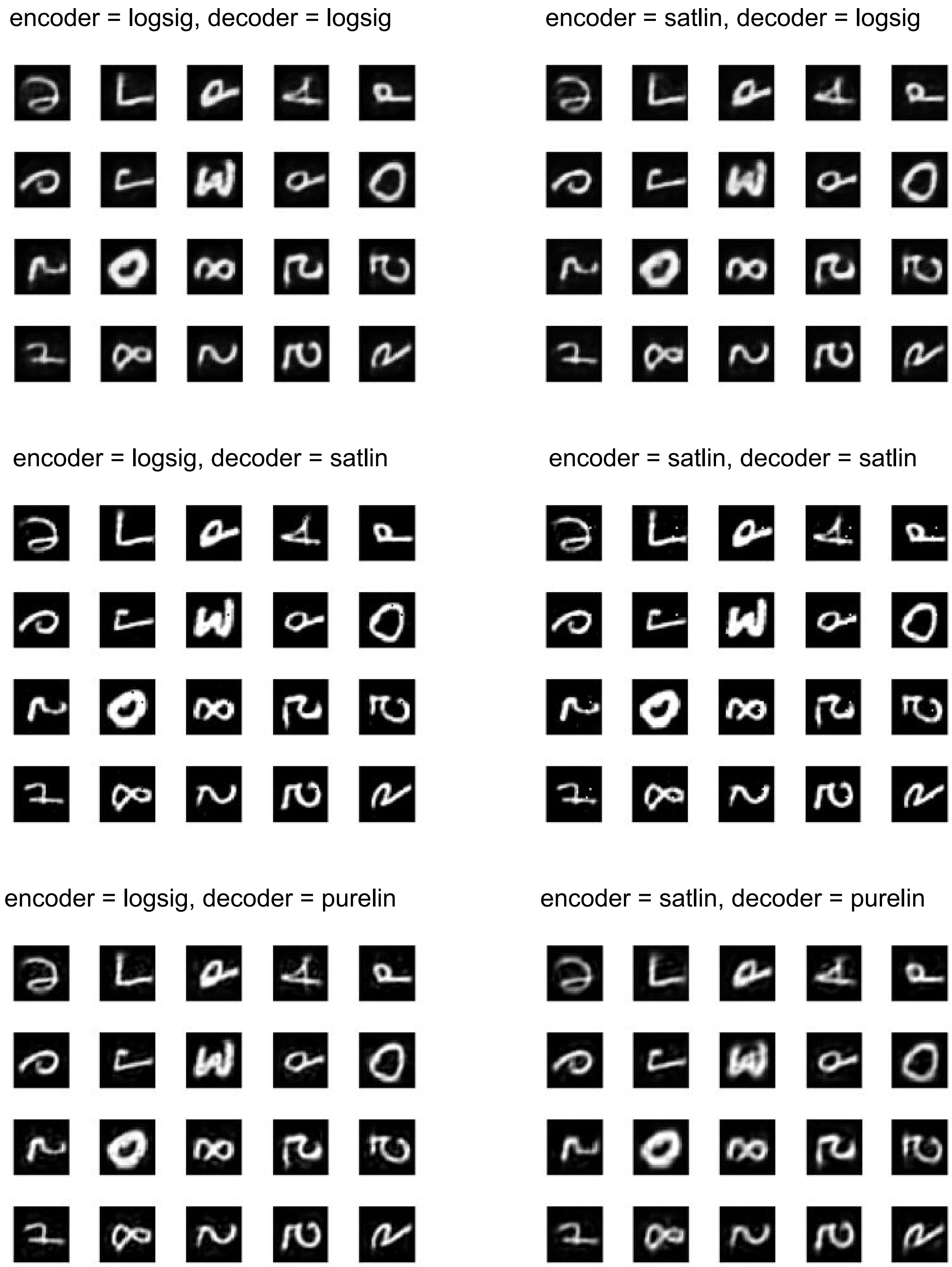


Figure 1. Test set reconstruction MSE in relation to number of epochs

Using chosen parameters, we tried running the training with different encoder and decoder transfer functions. Apart from default **logistic sigmoidal function**, we applied **linear transfer function** and **positive saturating linear function**. As seen in *Table 2,* the best result in terms of reconstruction accuracy was achieved with logistic sigmoidal function for encoder and positive saturating linear function for decoder. See *Figure 2* for visual comparison of the reconstructed patterns. We can observe that positive saturating linear function produces sharper edges. The results of logistic sigmoidal and linear functions are pretty similar.

		decoder transfer function		
		logsig	**satlin**	**purelin**
encoder transfer function	**logsig**	0.0089	**0.0044**	0.0089
	satlin	0.0105	0.0049	0.0122

Table 3. MSE for different encoder and decoder transfer functions after 1000 epochs



encoder = logsig, decoder = logsig

encoder = satlin, decoder = logsig

encoder = logsig, decoder = satlin

encoder = satlin, decoder = satlin

encoder = logsig, decoder = purelin

encoder = satlin, decoder = purelin

Figure 2. Test pattern reconstructions

2.3. The Second Autoencoder

We train another autoencoder to predict outputs of the first encoder using 50 hidden neurons. Therefore, we extract the second set of features. For training, the same parameters as for the first autoencoder were used. After stacking both autoencoders, the reconstruction MSE for test patterns equals to **0.0279**. It’s clearly greater than **0.0044** when using only one autoencoder*,* because we lose some information by compressing features into only 50 neurons.

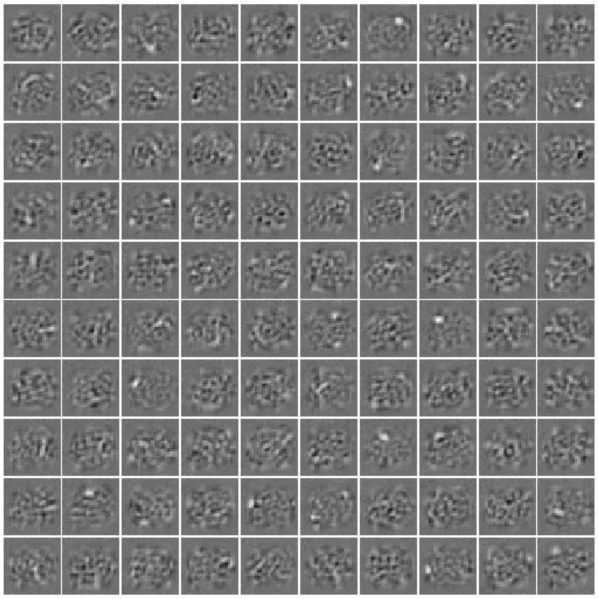


Figure 3. Features learned by the first autoencoder

Figure 4. Features learned by the second autoencoder

2.4. Softmax Layer

Finally, we add softmax layer and train it to recognize digits in output of the second autoencoder. The final neural network thus consists of three stacked layers, as seen on *Figure 5*.

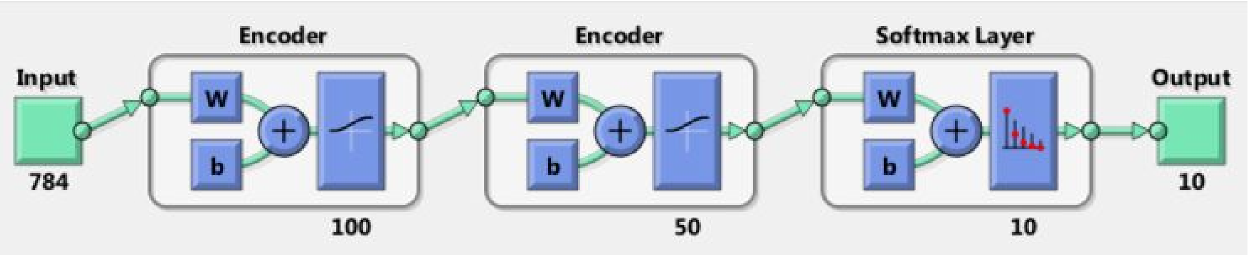


Figure 5. Neural network with 2 stacked encoders and softmax layer

We use logistic sigmoidal transfer functions for both autoencoders all layers are trained for 200 epochs. We conduct a few more experiments with different number of neurons in hidden layers. According to Table 4, we can conclude that the best configuration in terms of classification accuracy is **150** hidden neurons for the first autoencoder and **100** hidden neurons for the second autoencoder.

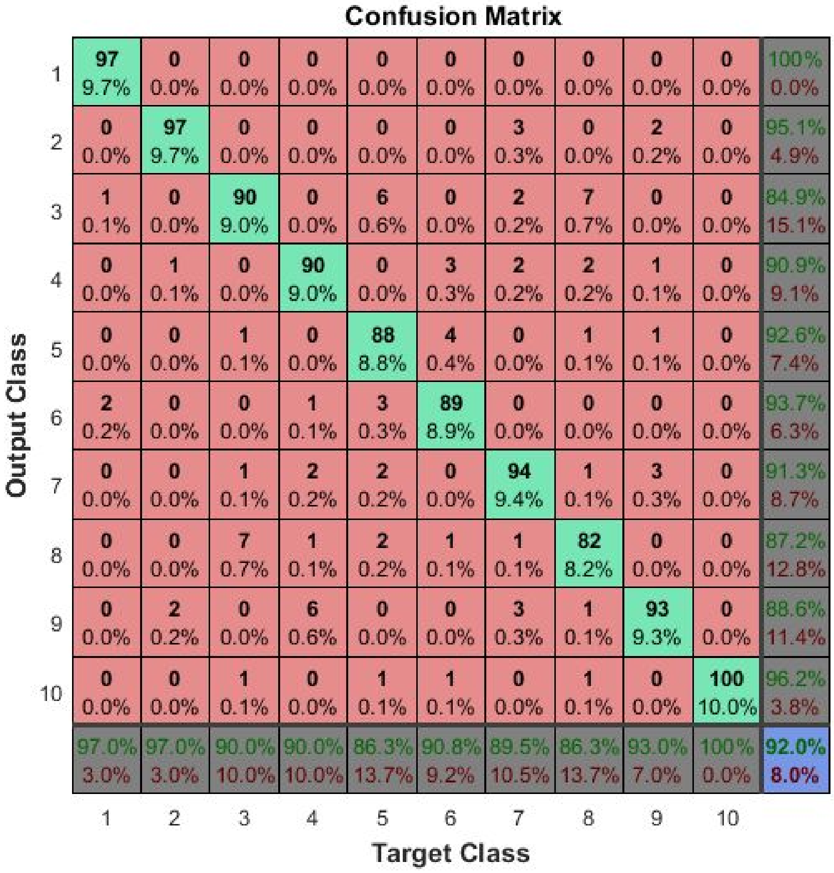
After determining the best configuration, we did final experiments with sparsity parameters. We found out that the maximum classification accuracy doesn't necessarily equal minimum reconstruction error. The greatest accuracy achieved was 92.0%, with sparsity regularization coefficient **2** and sparsity proportion **0.2**. For the final source code please refer to **autoencoder.m** file. The final result in form of confusion matrix is presented in *Figure 6*.

		first autoencoder neurons			
		50	**100**	**150**	**200**
second autoencoder neurons	**25**	87.1% / 0.0257	87.3% / 0.0263	86.8% / 0.0240	87.1% / 0.0236
	50		89.5% / 0.0208	89.6% / 0.0182	89.4% / 0.0171
	100			**90.5%** / 0.0157	89.9% / 0.0147
	150				89.6% / **0.0136**

Table 4. Classification accuracy and MSE in relation to number of neurons in hidden layer

		sparsity regularization coefficient		
		1	**2**	**4**
sparsity proportion	**0.1**	90.8% / 0.0276	91.7% / 0.0294	90.8% / 0.0333
	0.2	91.2% / 0.0190	**92.0%** / 0.0203	89.9% / 0.0225
	0.3	90.5% / **0.0157**	88.7% / 0.0162	88.6% / 0.0191

Table 5. Classification accuracy and MSE in relation to sparsity parameters



1	97 9.7%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	100% 0.0%
2	0 0.0%	97 9.7%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	3 0.3%	0 0.0%	2 0.2%	0 0.0%	95.1% 4.9%
3	1 0.1%	0 0.0%	90 9.0%	0 0.0%	6 0.6%	0 0.0%	2 0.2%	7 0.7%	0 0.0%	0 0.0%	84.9% 15.1%
4	0 0.0%	1 0.1%	0 0.0%	90 9.0%	0 0.0%	3 0.3%	2 0.2%	2 0.2%	1 0.1%	0 0.0%	90.9% 9.1%
5	0 0.0%	0 0.0%	1 0.1%	0 0.0%	88 8.8%	4 0.4%	0 0.0%	1 0.1%	1 0.1%	0 0.0%	92.6% 7.4%
6	2 0.2%	0 0.0%	0 0.0%	1 0.1%	3 0.3%	89 8.9%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	93.7% 6.3%
7	0 0.0%	0 0.0%	1 0.1%	2 0.2%	2 0.2%	0 0.0%	94 9.4%	1 0.1%	3 0.3%	0 0.0%	91.3% 8.7%
8	0 0.0%	0 0.0%	7 0.7%	1 0.1%	2 0.2%	1 0.1%	1 0.1%	82 8.2%	0 0.0%	0 0.0%	87.2% 12.8%
9	0 0.0%	2 0.2%	0 0.0%	6 0.6%	0 0.0%	0 0.0%	3 0.3%	1 0.1%	93 9.3%	0 0.0%	88.6% 11.4%
10	0 0.0%	0 0.0%	1 0.1%	0 0.0%	1 0.1%	1 0.1%	0 0.0%	1 0.1%	0 0.0%	100 10.0%	96.2% 3.8%
	97.0% 3.0%	97.0% 3.0%	90.0% 10.0%	90.0% 10.0%	86.3% 13.7%	90.8% 9.2%	89.5% 10.5%	86.3% 13.7%	93.0% 7.0%	100% 0.0%	92.0% 8.0%
	1	2	3	4	5	6	7	8	9	10	

Figure 6. Confusion matrix of the test set classification with final parameters