Amath 482 homework 5: Neural Networks for Classifying Fashion MNIST

Andy Zhang

March 13, 2020

Abstract

This report demonstrates the classification of fashion MNIST dataset using neural networks. It is separated into 2 parts: using only fully-connected neural networks and additionally using convolutional neural networks. To generate a network that is not only accurate in training data, but also in validating and test data, we are going to adjust on various hyper parameters.

1 Introduction

Since the development of technology brings increasing amount of image data, neural network is then a good tool for classifying them. In this report, we are going to focus on classification of fashion MNIST dataset, with 10 different classes of fashion items: 0-T-shirt, 1-Trouser, 2-Pullover, 3-Dress, 4-Coat, 5-Sandal, 6-Shirt, 7-Sneaker, 8-Bag, and 9-Ankle Boot being defined respectively.

In this report, we are going to analyse on 55,000 training data and apply the neural network on 5,000 validating data. If we've got a good result, then this model is then applied to 10,000 test data. In order not to over-fit the data, that is, the loss on training data is significantly less than that of validating and test data, we have to choose proper **L2 Regularization** value, and add the two-norm squared of the weight vector $\vec{w}$ multiplies **L2 Regularization** to the loss function. Other ways including reducing widths of neurons and using larger data set could also be considered.

2 Theoretical Background

2.1 Artificial Neural Network

Artificial Neural Network is the computing system inspired by the biological neural network in animal brains. It's based on collection of neurons that transmit information to each other, like those in animal brains. Applying artificial neural networks aimed to solve problems same as a human would do when classifying and currently is used in broader fields.

The output of the neural network is computed by non-linear functions that considers the dot product of input and weight vectors, adding bias, against threshold values. If we add several hidden layers between the input and output, then a deep learning neural network can be created.

2.2 Fully connected neural network

Fully connected neural networks are broadly applicable, but may perform more weakly than the convolutional neural networks. It consists of several layers that all of the neurons are fully connected. We may define the output $\vec{y}$ from input $\vec{x}$ by:

$$\vec{y} = \sigma(A\vec{x} + \vec{b}) \tag{1}$$

, where A denotes the matrix that contains all weights from input to output and b means the added bias. The function σ is a step function that checks if the weighted sum is larger than the threshold or not.

If we're using a deep-learning neural network with one hidden layer, we would have it defined as:

$$\vec{y} = \sigma(A_2\sigma(A_1\vec{x_1} + \vec{b_1} + \vec{b_2}) \tag{2}$$

. If we are dealing with more hidden layers, then we can just apply it for several times.

We use width and depth to represent the number of neurons per layer and number of layers respectively, and they are called hyper parameters of the network.

2.3 Convolutional neural network

A convolutional neural network (CNN) is a kind of deep learning neural network that are mostly applied to analyse images, which matches the goal of this homework: imagery classification. Unlike fully-connected neural network that may easily run into over-fitting, we may take advantage of the multi-layer pattern of CNN and combine simpler features to form complex features. Since the same weight matrix is used for all neurons in the same layer, the result should be similar to the original image data but only slightly shifted so that we won't lose information. This weight matrix is called a filter and we call layer of neurons the feature map.

We may operate on more parameters when using CNN, including the size and number of filters, the stride (distance to shift on the receptive field), padding, pooling methods, and pool sizes. Padding can help to generate the same size layer from the image and pooling allows us to subsample the layer. A commonly-used CNN structure is Lenet-5 that first implements the alteration between convolution layer and subsampling, then it uses a few fully-connected layers for classification.

3 Algorithm Implementation and Development

Part 1 Fully-connected neural network

Prepare data

We are going to use a fully-connected neural network for image classification in part 1. To start with it, we load the dataset of fashion MNIST, then reshape and permute it in the order we want. We choose the first 5,000 out of the 60,000 pieces of data as validation and the rest 55,000 for training. We should also call the built-in **categorical** function to the output to make them categorical data.

Choose different parameters

I tried to adjust on the depth, width, learning rate, and L2 regularization, max epochs in this part. First, I use the three-layer structure from the **MNIST-Classifier** code and it results in an accuracy about 87% on validating data. I then change the width to be 500, which is about 2/3 of the input size and change its depth to be 3, accompanied with RELU activation function. The resulting 5-layer structure (500-500-500-100-10) improves the accuracy for a bit.

Then I notice the accuracy still shows increasing trend in epoch 5 so I increase the max epochs to 8 and decrease the Learning rate to 1e-4 correspondingly. To avoid over-fitting, I increase L2 Regularization to 1e-3. Searching on different optimizers tells that Adam ranks the top so I continue to use it.

Part 2 Convolutional neural network

Prepare data

I apply the same procedure as in **Part 1**.

Choose different parameters

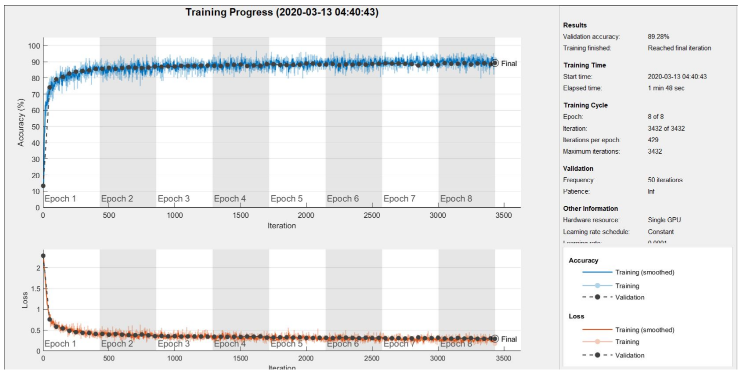
In this case, I tried on adjusting number of filters, filter size, strides, pool size, and those tried in **Part 1**. Likewise, I try the conventional Lenet-5 structure at first and it results in about 88% accuracy. I then choose the same L2 Regularization but a Learning rate of 1e-3. I also try on different activation functions and find out that hyperbolic tangent seems to perform the best. Increasing the number of filters doesn't really improve accuracy, but slows down my algorithm significantly so I use the same structure as Lenet-5. Besides, changing the stride doesn't affect the result a lot.

To get a good result on accuracy, I decrease the filter size to 3*3 and increase the number of filters to 32, 64, and 128 for the 3 convolution layers. A max pooling layer is used instead of average pooling to extract the characteristic values, with pool size of 3*3. I also add a batch Normalization Layer and then use hyperbolic tangent activation function. Then I add two fully-connected layers at last, with width of 100 and apply RELU activation function instead.

4 Computational Results

Part 1 Fully-connected neural network

Using the hyper parameters mentioned in **Algorithm Implementation and Development**, a result shown in **Figure 1** below is generated, which



Results
Validation accuracy: 89.28%
Training finished: Reached final iteration

Training Time
Start time: 2020-03-13 04:40:43
Elapsed time: 1 min 48 sec

Training Cycle
Epoch: 8 of 8
Iteration: 3432 of 3432
Iterations per epoch: 429
Maximum iterations: 3432

Validation
Frequency: 50 iterations
Patience: Inf

Other Information
Hardware resource: Single GPU
Learning rate schedule: Constant
Learning rate: 0.0001

Figure 1: Training Progress of Fully-connected NN

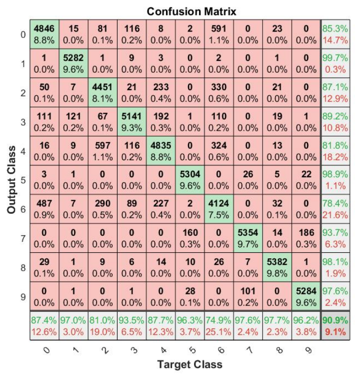
seems to be the best of all attempts. It reaches 89.2% on validation data and there's not excessive over-fitting by checking on the accuracy and loss.

Looking into the confusion matrix of training and validating data that are shown in **Figure 2**, in which we can see 1.6% difference in the generated neural network. Besides, we can see that shirts and T-shirts, coats and pullovers are commonly misclassified.

Applying this network onto the testing data yields an accuracy of only 88%, which is a little worse than I expect from the validating data. We can see that it only gets 84.1% correct on T-shirts (0), 76.7% on pullovers (2), 82.9% accurate on coats (4), and 68.2 % on shirts (6). This makes sense because these tops look similar to each other on a 28 pixels*28 pixels image.

Part 2 Convolutional neural network

While using the convolutional neural network, we would see noticeable increase in accuracy, shown in **Figure 4**. Though we may see slight over-fitting



Confusion Matrix

Output Class	0	1	2	3	4	5	6	7	8	9	
0	4846 8.8%	15 0.0%	81 0.1%	116 0.2%	8 0.0%	2 0.0%	591 1.1%	0 0.0%	23 0.0%	0 0.0%	85.3% 14.7%
1	1 0.0%	5282 9.6%	1 0.0%	9 0.0%	3 0.0%	0 0.0%	2 0.0%	0 0.0%	1 0.0%	0 0.0%	99.7% 0.3%
2	50 0.1%	7 0.0%	4451 8.1%	21 0.0%	233 0.4%	0 0.0%	330 0.6%	0 0.0%	21 0.0%	0 0.0%	87.1% 12.9%
3	111 0.2%	121 0.2%	67 0.1%	5141 9.3%	192 0.3%	1 0.0%	110 0.2%	0 0.0%	19 0.0%	1 0.0%	89.2% 10.8%
4	16 0.0%	9 0.0%	597 1.1%	116 0.2%	4835 8.8%	0 0.0%	324 0.6%	0 0.0%	13 0.0%	0 0.0%	81.8% 18.2%
5	3 0.0%	1 0.0%	0 0.0%	0 0.0%	0 0.0%	5304 9.6%	0 0.0%	26 0.0%	5 0.0%	22 0.0%	98.9% 1.1%
6	487 0.9%	7 0.0%	290 0.5%	89 0.2%	227 0.4%	2 0.0%	4124 7.5%	0 0.0%	32 0.1%	0 0.0%	78.4% 21.6%
7	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	160 0.3%	0 0.0%	5354 9.7%	14 0.0%	186 0.3%	93.7% 6.3%
8	29 0.1%	1 0.0%	9 0.0%	6 0.0%	14 0.0%	10 0.0%	26 0.0%	7 0.0%	5382 9.8%	1 0.0%	98.1% 1.9%
9	0 0.0%	1 0.0%	0 0.0%	1 0.0%	0 0.0%	28 0.1%	0 0.0%	101 0.2%	0 0.0%	5284 9.6%	97.6% 2.4%
	87.4% 12.6%	97.0% 3.0%	81.0% 19.0%	93.5% 6.5%	87.7% 12.3%	96.3% 3.7%	74.9% 25.1%	97.6% 2.4%	97.7% 2.3%	96.2% 3.8%	**90.9% 9.1%**

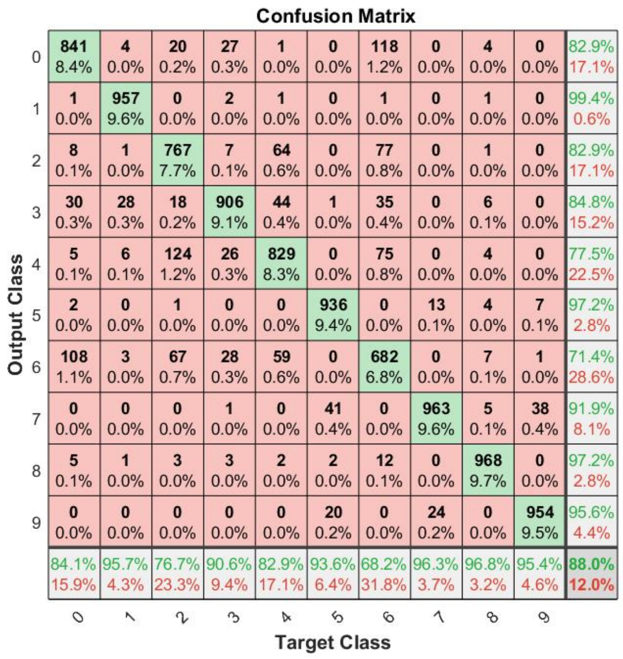
Target Class

Confusion Matrix

Output Class	0	1	2	3	4	5	6	7	8	9	
0	390 7.8%	3 0.1%	9 0.2%	13 0.3%	2 0.0%	0 0.0%	63 1.3%	0 0.0%	2 0.0%	0 0.0%	80.9% 19.1%
1	1 0.0%	539 10.8%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	1 0.0%	0 0.0%	0 0.0%	0 0.0%	99.6% 0.4%
2	4 0.1%	2 0.0%	387 7.7%	3 0.1%	28 0.6%	0 0.0%	33 0.7%	0 0.0%	1 0.0%	0 0.0%	84.5% 15.5%
3	17 0.3%	10 0.2%	7 0.1%	465 9.3%	22 0.4%	0 0.0%	16 0.3%	0 0.0%	5 0.1%	0 0.0%	85.8% 14.2%
4	1 0.0%	1 0.0%	65 1.3%	12 0.2%	419 8.4%	0 0.0%	39 0.8%	0 0.0%	0 0.0%	0 0.0%	78.0% 22.0%
5	0 0.0%	0 0.0%	1 0.0%	0 0.0%	0 0.0%	474 9.5%	0 0.0%	5 0.1%	1 0.0%	3 0.1%	97.9% 2.1%
6	40 0.8%	1 0.0%	31 0.6%	7 0.1%	16 0.3%	0 0.0%	336 6.7%	0 0.0%	2 0.0%	0 0.0%	77.6% 22.4%
7	0 0.0%	0 0.0%	1 0.0%	0 0.0%	0 0.0%	15 0.3%	1 0.0%	494 9.9%	1 0.0%	16 0.3%	93.6% 6.4%
8	4 0.1%	0 0.0%	3 0.1%	1 0.0%	1 0.0%	0 0.0%	4 0.1%	1 0.0%	475 9.5%	2 0.0%	96.7% 3.3%
9	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	4 0.1%	0 0.0%	12 0.2%	3 0.1%	485 9.7%	96.2% 3.8%
	85.3% 14.7%	96.9% 3.1%	76.8% 23.2%	92.8% 7.2%	85.9% 14.1%	96.1% 3.9%	68.2% 31.8%	96.5% 3.5%	96.9% 3.1%	95.8% 4.2%	**89.3% 10.7%**

Target Class

Figure 2: Confusion matrix of Training data (left) and Validating data (right)



Confusion Matrix

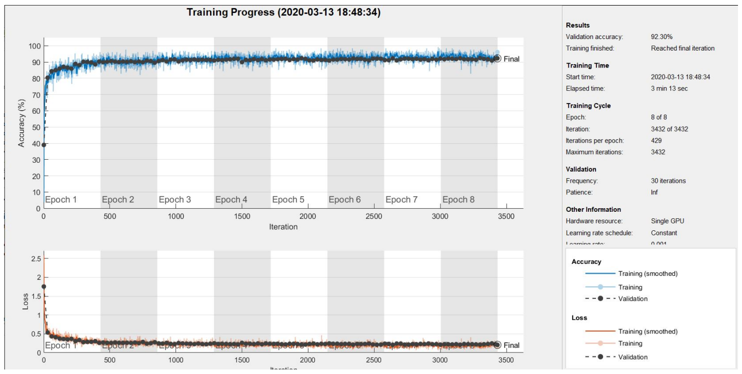
Output Class	0	1	2	3	4	5	6	7	8	9	
0	841 8.4%	4 0.0%	20 0.2%	27 0.3%	1 0.0%	0 0.0%	118 1.2%	0 0.0%	4 0.0%	0 0.0%	82.9% 17.1%
1	1 0.0%	957 9.6%	0 0.0%	2 0.0%	1 0.0%	0 0.0%	1 0.0%	0 0.0%	1 0.0%	0 0.0%	99.4% 0.6%
2	8 0.1%	1 0.0%	767 7.7%	7 0.1%	64 0.6%	0 0.0%	77 0.8%	0 0.0%	1 0.0%	0 0.0%	82.9% 17.1%
3	30 0.3%	28 0.3%	18 0.2%	906 9.1%	44 0.4%	1 0.0%	35 0.4%	0 0.0%	6 0.1%	0 0.0%	84.8% 15.2%
4	5 0.1%	6 0.1%	124 1.2%	26 0.3%	829 8.3%	0 0.0%	75 0.8%	0 0.0%	4 0.0%	0 0.0%	77.5% 22.5%
5	2 0.0%	0 0.0%	1 0.0%	0 0.0%	0 0.0%	936 9.4%	0 0.0%	13 0.1%	4 0.0%	7 0.1%	97.2% 2.8%
6	108 1.1%	3 0.0%	67 0.7%	28 0.3%	59 0.6%	0 0.0%	682 6.8%	0 0.0%	7 0.1%	1 0.0%	71.4% 28.6%
7	0 0.0%	0 0.0%	0 0.0%	1 0.0%	0 0.0%	41 0.4%	0 0.0%	963 9.6%	5 0.1%	38 0.4%	91.9% 8.1%
8	5 0.1%	1 0.0%	3 0.0%	3 0.0%	2 0.0%	2 0.0%	12 0.1%	0 0.0%	968 9.7%	0 0.0%	97.2% 2.8%
9	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	20 0.2%	0 0.0%	24 0.2%	0 0.0%	954 9.5%	95.6% 4.4%
	84.1% 15.9%	95.7% 4.3%	76.7% 23.3%	90.6% 9.4%	82.9% 17.1%	93.6% 6.4%	68.2% 31.8%	96.3% 3.7%	96.8% 3.2%	95.4% 4.6%	**88.0% 12.0%**

Target Class

Figure 3: Confusion matrix of Testing data

in epoch 7 and 8, it guarantees a validation accuracy of about 92.3%.

The confusion matrix of training and validating data are shown below in



Results

Validation accuracy:	92.30%
Training finished:	Reached final iteration

Training Time

Start time:	2020-03-13 18:48:34
Elapsed time:	3 min 13 sec

Training Cycle

Epoch:	8 of 8
Iteration:	3432 of 3432
Iterations per epoch:	429
Maximum iterations:	3432

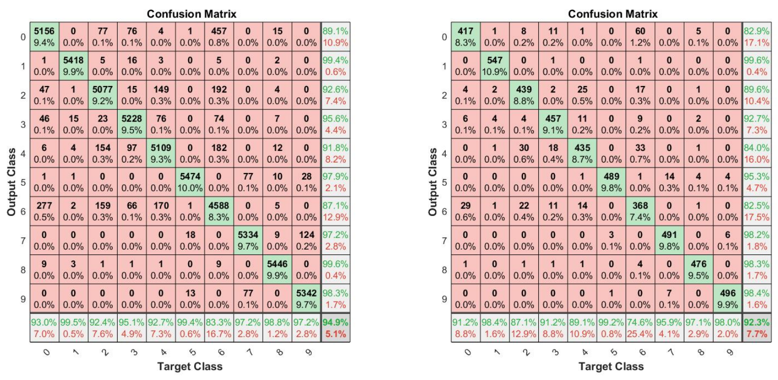
Validation

Frequency:	30 iterations
Patience:	Inf

Other Information

Hardware resource:	Single GPU
Learning rate schedule:	Constant

Figure 4: Training Progress of CNN


Output Class
Target Class

Confusion Matrix

	0	1	2	3	4	5	6	7	8	9	
0	5156 9.4%	0 0.0%	77 0.1%	76 0.1%	4 0.0%	1 0.0%	457 0.8%	0 0.0%	15 0.0%	0 0.0%	89.1% 10.9%
1	1 0.0%	5418 9.9%	5 0.0%	16 0.0%	3 0.0%	0 0.0%	5 0.0%	0 0.0%	2 0.0%	0 0.0%	99.4% 0.6%
2	47 0.1%	1 0.0%	5077 9.2%	15 0.0%	149 0.3%	0 0.0%	192 0.3%	0 0.0%	4 0.0%	0 0.0%	92.6% 7.4%
3	46 0.1%	15 0.0%	23 0.0%	5228 9.5%	76 0.1%	0 0.0%	74 0.1%	0 0.0%	7 0.0%	0 0.0%	95.6% 4.4%
4	6 0.0%	4 0.0%	154 0.3%	97 0.2%	5109 9.3%	0 0.0%	182 0.3%	0 0.0%	12 0.0%	0 0.0%	91.8% 8.2%
5	1 0.0%	1 0.0%	0 0.0%	0 0.0%	0 0.0%	5474 10.0%	0 0.0%	77 0.1%	10 0.0%	28 0.1%	97.9% 2.1%
6	277 0.5%	2 0.0%	159 0.3%	66 0.1%	170 0.3%	1 0.0%	4588 8.3%	0 0.0%	5 0.0%	0 0.0%	87.1% 12.9%
7	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	18 0.0%	0 0.0%	5334 9.7%	9 0.0%	124 0.2%	97.2% 2.8%
8	9 0.0%	3 0.0%	1 0.0%	1 0.0%	1 0.0%	0 0.0%	9 0.0%	0 0.0%	5446 9.9%	0 0.0%	99.6% 0.4%
9	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	13 0.0%	0 0.0%	77 0.1%	0 0.0%	5342 9.7%	98.3% 1.7%
	93.0% 7.0%	99.5% 0.5%	92.4% 7.6%	95.1% 4.9%	92.7% 7.3%	99.4% 0.6%	83.3% 16.7%	97.2% 2.8%	98.8% 1.2%	97.2% 2.8%	94.9% 5.1%

Confusion Matrix

	0	1	2	3	4	5	6	7	8	9	
0	417 8.3%	1 0.0%	8 0.2%	11 0.2%	1 0.0%	0 0.0%	60 1.2%	0 0.0%	5 0.1%	0 0.0%	82.9% 17.1%
1	0 0.0%	547 10.9%	0 0.0%	1 0.0%	0 0.0%	0 0.0%	1 0.0%	0 0.0%	0 0.0%	0 0.0%	99.6% 0.4%
2	4 0.1%	2 0.0%	439 8.8%	2 0.0%	25 0.5%	0 0.0%	17 0.3%	0 0.0%	1 0.0%	0 0.0%	89.6% 10.4%
3	6 0.1%	4 0.1%	4 0.1%	457 9.1%	11 0.2%	0 0.0%	9 0.2%	0 0.0%	2 0.0%	0 0.0%	92.7% 7.3%
4	0 0.0%	1 0.0%	30 0.6%	18 0.4%	435 8.7%	0 0.0%	33 0.7%	0 0.0%	1 0.0%	0 0.0%	84.0% 16.0%
5	0 0.0%	0 0.0%	0 0.0%	0 0.0%	1 0.0%	489 9.8%	1 0.0%	14 0.3%	4 0.1%	4 0.1%	95.3% 4.7%
6	29 0.6%	1 0.0%	22 0.4%	11 0.2%	14 0.3%	0 0.0%	368 7.4%	0 0.0%	1 0.0%	0 0.0%	82.5% 17.5%
7	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	3 0.1%	0 0.0%	491 9.8%	0 0.0%	6 0.1%	98.2% 1.8%
8	1 0.0%	0 0.0%	1 0.0%	1 0.0%	1 0.0%	0 0.0%	4 0.1%	0 0.0%	476 9.5%	0 0.0%	98.3% 1.7%
9	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	1 0.0%	0 0.0%	7 0.1%	0 0.0%	496 9.9%	98.4% 1.6%
	91.2% 8.8%	98.4% 1.6%	87.1% 12.9%	91.2% 8.8%	89.1% 10.9%	99.2% 0.8%	74.6% 25.4%	95.9% 4.1%	97.1% 2.9%	98.0% 2.0%	92.3% 7.7%

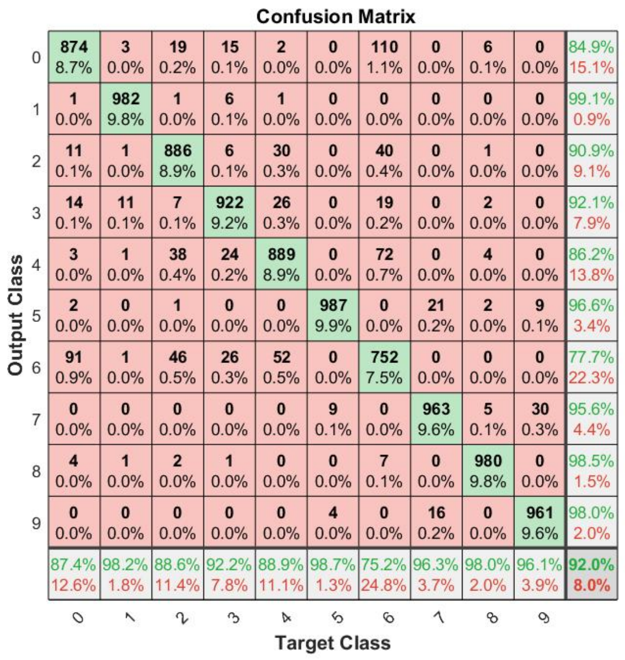
Figure 5: Confusion matrix of Training data (left) and Validating data (right)

Figure 5, we may see 94.9% and 92.3% accurate on training and validating data, which implies some over-fitting. However, there's significant increase in accuracy of the tops mentioned above besides shirts (6).

Using CNN instead, the result on testing data shown in **Figure 6** looks much better than that of fully-connected NN and reaches 92% accuracy. And in all 10 categories, the accuracy either improves or remains the same. However, this model is still having some difficulties in accurately classifying shirts (only 75.2% correctly classified).

5 Summary and Conclusion

To conclude, CNN performs much better in imagery classification than only using fully-connected neural networks. The results shown in this report may be



Confusion Matrix

Output Class / Target Class

	0	1	2	3	4	5	6	7	8	9	
0	874 8.7%	3 0.0%	19 0.2%	15 0.1%	2 0.0%	0 0.0%	110 1.1%	0 0.0%	6 0.1%	0 0.0%	84.9% 15.1%
1	1 0.0%	982 9.8%	1 0.0%	6 0.1%	1 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	99.1% 0.9%
2	11 0.1%	1 0.0%	886 8.9%	6 0.1%	30 0.3%	0 0.0%	40 0.4%	0 0.0%	1 0.0%	0 0.0%	90.9% 9.1%
3	14 0.1%	11 0.1%	7 0.1%	922 9.2%	26 0.3%	0 0.0%	19 0.2%	0 0.0%	2 0.0%	0 0.0%	92.1% 7.9%
4	3 0.0%	1 0.0%	38 0.4%	24 0.2%	889 8.9%	0 0.0%	72 0.7%	0 0.0%	4 0.0%	0 0.0%	86.2% 13.8%
5	2 0.0%	0 0.0%	1 0.0%	0 0.0%	0 0.0%	987 9.9%	0 0.0%	21 0.2%	2 0.0%	9 0.1%	96.6% 3.4%
6	91 0.9%	1 0.0%	46 0.5%	26 0.3%	52 0.5%	0 0.0%	752 7.5%	0 0.0%	0 0.0%	0 0.0%	77.7% 22.3%
7	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	9 0.1%	0 0.0%	963 9.6%	5 0.1%	30 0.3%	95.6% 4.4%
8	4 0.0%	1 0.0%	2 0.0%	1 0.0%	0 0.0%	0 0.0%	7 0.1%	0 0.0%	980 9.8%	0 0.0%	98.5% 1.5%
9	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	4 0.0%	0 0.0%	16 0.2%	0 0.0%	961 9.6%	98.0% 2.0%
	87.4% 12.6%	98.2% 1.8%	88.6% 11.4%	92.2% 7.8%	88.9% 11.1%	98.7% 1.3%	75.2% 24.8%	96.3% 3.7%	98.0% 2.0%	96.1% 3.9%	92.0% 8.0%

Figure 6: Confusion matrix of Testing data (CNN)

further improved by applying more complicated structures, which could make the algorithm much more slower. Thus, the tradeoff between efficiency and accuracy should be considered before implementing the structure.

6 Appendix A

xtrain = permute(xtrain, order) changes xtrian in the given order.

layer = imageInputLayer(inputSize) defines an image input layer. InputSize is the size of the input images for the layer.

categorical() creates an categorical array from cell array.

layer = fullyConnectedLayer(outputSize) creates a fully connected layer. OutputSize specifies the size of the output for the layer. A fully connected layer will multiply the input by a matrix and then add a bias vector.

layer = reluLayer() creates a rectified linear unit layer. This type of layer performs a simple threshold operation.

options = trainingOptions(solverName) creates a set of training options for the solver specified by solverName.

trainedNet = trainNetwork(xtrain, ytrain layers, options) trains and returns a network trainedNet for a classification problem. xtrain and ytrain are input data for training, layers is an array of network layers, and options is a set of training options.

CLASS = classify(trainedNet, xtrain) classifies each row of the xtrain data according to trained network.

plotconfusion(targets,outputs) plots a confusion matrix, using target (true) and output (predicted) labels. Specify the labels as categorical vectors.

layer = convolution2dLayer(filterSize, numFilters) creates a layer for 2D convolution. filterSize specifies the height and width of the filters and numFilters denotes the number of filters applied.

layer = maxPooling2dLayer(poolSize) creates a layer that performs max pooling.

layer = tanhLayer() creates a hyperbolic tangent layer.

layer = softmaxLayer() creates a softmax layer. This layer is useful for classification problems.

layer = classificationLayer() creates a classification output layer for a neural network. The classification output layer holds the name of the loss function that is used for training the network, the size of the output, and the class labels.

7 Appendix B

```
%% Part 1 Fully-connected neural network
load('fashion_mnist.mat')

X_train = im2double(X_train);
X_test = im2double(X_test);

X_train = reshape(X_train,[60000 28 28 1]);
X_train = permute(X_train,[2 3 4 1]);

X_test = reshape(X_test,[10000 28 28 1]);
X_test = permute(X_test,[2 3 4 1]);

X_valid = X_train(:,:,:,1:5000);
X_train = X_train(:,:,:,5001:end);

y_valid = categorical(y_train(1:5000))';
y_train = categorical(y_train(5001:end))';
y_test = categorical(y_test)';


layers = [imageInputLayer([28 28 1])
        fullyConnectedLayer(500)
        reluLayer
        fullyConnectedLayer(500)
        reluLayer
        fullyConnectedLayer(500)
        reluLayer
        fullyConnectedLayer(100)
        reluLayer
        fullyConnectedLayer(10)
        softmaxLayer
        classificationLayer];

options = trainingOptions('adam', ...
    'MaxEpochs',8,...
    'Shuffle','every-epoch',...
    'InitialLearnRate',1e-4, ...
    'L2Regularization',1e-3, ...
    'ValidationData',{X_valid,y_valid}, ...
    'Verbose',false, ...
    'Plots','training-progress')

net = trainNetwork(X_train,y_train,layers,options);
```

```
%%
y_pred = classify(net,X_train);
plotconfusion(y_train,y_pred)
%%
y_pred = classify(net,X_valid);
plotconfusion(y_valid,y_pred)
%%
y_pred = classify(net,X_test);
plotconfusion(y_test,y_pred)



%%
clear; close all; clc;
load('fashion_mnist.mat')


X_train = im2double(X_train);
X_test = im2double(X_test);

X_train = reshape(X_train,[60000 28 28 1]);
X_train = permute(X_train,[2 3 4 1]);

X_test = reshape(X_test,[10000 28 28 1]);
X_test = permute(X_test,[2 3 4 1]);

X_valid = X_train(:,:,:,1:5000);
X_train = X_train(:,:,:,5001:end);

y_valid = categorical(y_train(1:5000))';
y_train = categorical(y_train(5001:end))';
y_test = categorical(y_test)';

layers = [
    imageInputLayer([28 28 1],"Name","imageinput")
    convolution2dLayer([3 3],32,"Name","conv_1","Padding","same")
    batchNormalizationLayer
    tanhLayer
    maxPooling2dLayer([3 3],"Name","avgpool2d_1","Padding","same","Stride",[2 2])
    convolution2dLayer([3 3],64,"Name","conv_2")
    batchNormalizationLayer
    tanhLayer
    maxPooling2dLayer([3 3],"Name","avgpool2d_2","Padding","same","Stride",[2 2])
    convolution2dLayer([3 3],128,"Name","conv_3")
    batchNormalizationLayer
    tanhLayer
    fullyConnectedLayer(100)
    reluLayer
    fullyConnectedLayer(100)
    reluLayer
    fullyConnectedLayer(10)
    softmaxLayer
    classificationLayer("Name","classoutput")];

options = trainingOptions('adam', ...
    'MaxEpochs',8,...
    'Shuffle','every-epoch', ...
    'ValidationFrequency',30, ...
    'InitialLearnRate',1e-3, ...
    'L2Regularization',1e-3, ...
    'ValidationData',{X_valid,y_valid}, ...
    'Verbose',false, ...
    'Plots','training-progress')

net = trainNetwork(X_train,y_train,layers,options);
%%
y_pred = classify(net,X_train);
plotconfusion(y_train,y_pred)
%%
y_pred = classify(net,X_valid);
plotconfusion(y_valid,y_pred)
%%
y_pred = classify(net,X_test);
plotconfusion(y_test,y_pred)
```