Neural Networks for Classifying Fashion MNIST

Yinxi Pan

AMATH482, Homework5, Winter2020

March 13, 2020

Abstract

The report talks about how to classify different images by neural networks. The main method used is the deep learning neural network. The report classifies 10 different fashion images, each representing some fashion elements such as cloth, and shoes. The first part uses the fully-connected neural network to classify the image and achieves an accuracy of 87%. The second part uses the convolutional neural network and achieves an accuracy of 88.9%.

1 Introduction and Overview

Artificial neural networks are a machine learning tool used to perform tasks such as classification, regression, and dimensionality reduction. In this homework, we work on implementing different neural networks to classify the Fashion MNIST data. The Fashion MNIST data is similar to the popular MNIST dataset of handwritten digits. This report follows basic skills in classifying the MNIST data and apply them to the Fashion MNIST data. Later sections include basic theories about neural networks, how to implement the model and results.

2 Theoretical Background

There are now many different types of neural networks, some of which are quite different from the multilayer perceptron. The multilayer perceptron, typically paired with a sigmoid or ReLU activation function, is what we would refer to as a feed-forward neural network because everything flows in one direction from the inputs to the outputs. For a given layer, every neuron is connected to every neuron in the previous layer. We call those types of layers fully-connected or dense. When all of the layers are of that type, we say that it is a fully-connected network. The number of neurons in each layer is called the width of the layer. The number of layers is called the depth of the network. Deep learning or deep neural networks refers to neural networks with many layers (it used to be that any network with at least two hidden layers was considered deep, but now networks can have hundreds of hidden layers so what is "deep" is not so clear)[1].

In deep learning, a convolutional neural network (CNN, or ConvNet) is a class of deep neural networks, most commonly applied to analyzing visual imagery. They are also known as shift invariant or space invariant artificial neural networks (SIANN), based on their shared-weights architecture and translation invariance characteristics. They have applications in image and video recognition, recommender systems, image classification, medical image analysis, natural language processing,[4] and financial time series[2].

A convolution is a weighted sum of the pixel values of the image, as the window slides across the whole image. Turns out, this convolution process throughout an image with a weight matrix produces another image (of the same size, depending on the convention). Convolving is the process of applying a convolution. The sliding-window shenanigans happen in the convolution layer of the neural network. A typical CNN has multiple convolution layers. Each convolutional layer typically generates many alternate convolutions, so the weight matrix is a tensor of $5 \times 5 \times$ n, where n is the number of convolutions.

The name "convolutional neural network" indicates that the network employs a mathematical operation called convolution. Convolution is a specialized kind of linear operation. Convolutional networks are simply neural networks that use convolution in place of general matrix multiplication in at least one of their layers[3].

Convolutional Neural Networks (CNNs) is the most popular neural network model being used for image classification problem. The big idea behind CNNs is that a local understanding of an image is good enough. The practical benefit is that having fewer parameters greatly improves the time it takes to learn as well as reduces the amount of data required to train the model. Instead of a fully connected network of weights from each pixel, a CNN has just enough weights to look at a small patch of the image. It's like reading a book by using a magnifying glass; eventually, you read the whole page, but you look at only a small patch of the page at any given time[4].

3 Algorithm Implementation and Development

Both parts uses similar code but different types of layers. Here is the algorithm implementation for part 1.[5]

1. Load the data and convert it into double precision.
2. Reshape and randomize the data for building the model.
3. Create the layers and set up options for the model. When we set up layers, we have to try different parameters included in the set up such as the number of layers and the width

[1]Lecture file

[2]https://en.wikipedia.org/wiki/Convolutional_neural_network

[3]https://en.wikipedia.org/wiki/Convolutional_neural_network

[4]https://towardsdatascience.com/the-4-convolutional-neural-network-models-that-can-classify-your-fashion-images-9fe7f3e5399d

[5]Matlab code please check Appendix.B

of layers. In this way, we could later on test if we have a better model by changing the parameters.

4. Train the model.

5. Get confusion matrix for the training and testing data.

4 Computational Results

1. (Part1) Fully-connected neural network:
   The final model has an architecture of the following:

   (a) The depth of the network: 3
   (b) The width of the layers: 200,50,50
   (c) The learning rate: 1e-3
   (d) The regularization parameters: 1e-4
   (e) The activation functions: reluLayer
   (f) The optimizer: 'adam'

   Highest accuracy achieved on the testing data is 87 %.
   Highest accuracy achieved on the training data is 92.3 %.
   Highest accuracy achieved on the validation data is 88.44 %.

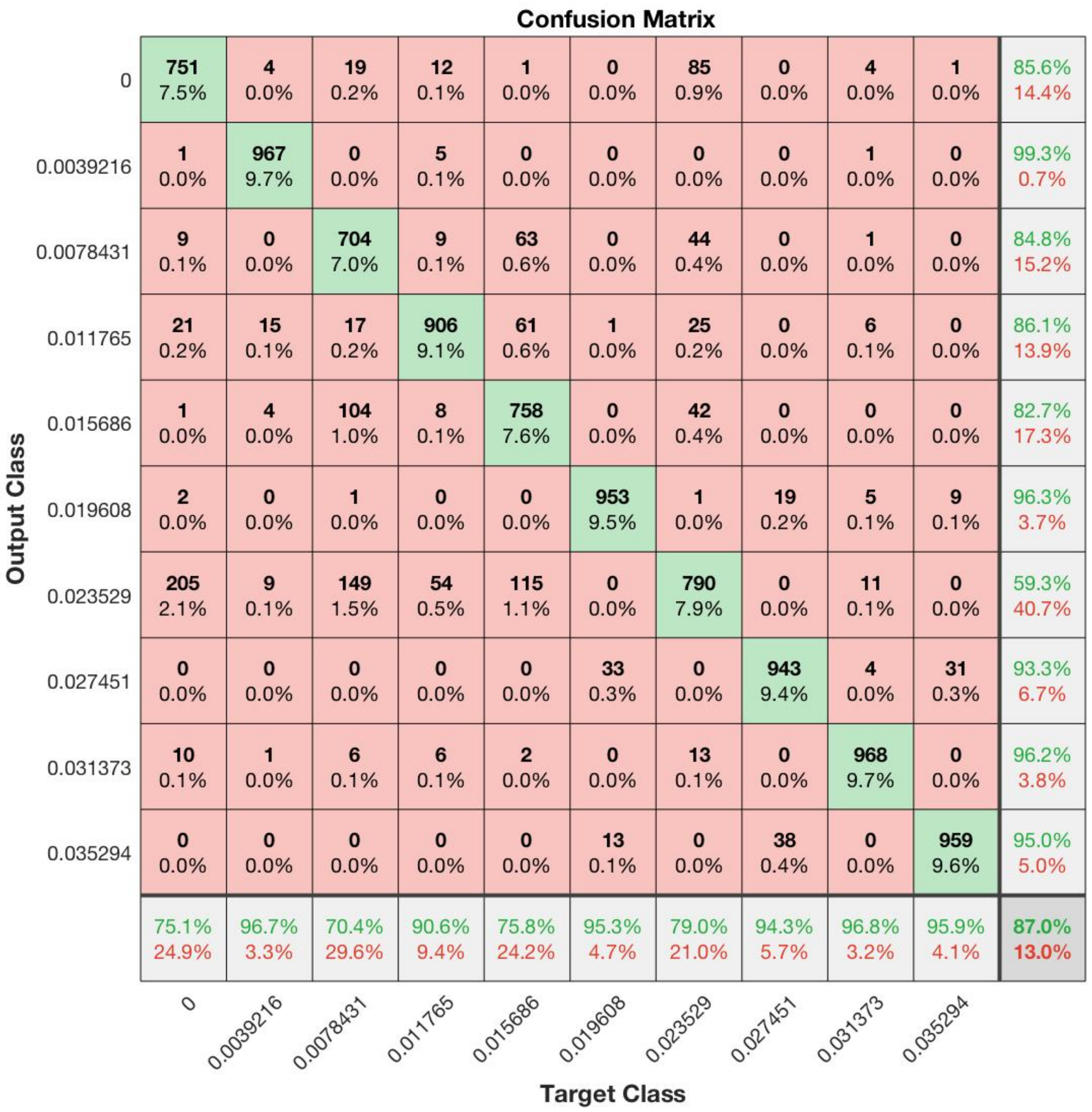
   It's really hard to make the validation accuracy higher than 90%. I tried to increase the number of layers, and the width of each layer. Both of them does not affect much on the accuracy rate. For the learning rate and regularization, both increasing and decreasing would cause a decrease in accuracy. Different optimizers also don't affect much on accuracy. Figure(1) is the confusion matrix for the test data in my final model.

2. (Part2) Convolutional neural network:
   The final model has an architecture of the following:

   (a) The number of filters for your convolutional layers: 10/16/120/100/84
   (b) The filter sizes: [5,5]
   (c) The strides: [2,2]
   (d) The padding options: No padding
   (e) The pool sizes for your pooling layers: half

   Highest accuracy achieved on the testing data is 88.9 %.
   Highest accuracy achieved on the training data is 86.9 %.
   Highest accuracy achieved on the validation data is 88.64 %.

It's also really hard to make the validation accuracy higher than 90%. I tried to increase the number of layers, and the accuracy increases a little bit. I have errors when changing the size of padding, filter, and pool so I stay with the demo code. Changing the number of epochs doesn't affect much on accuracy. Variations in the learning rate and regularization decrease accuracy. Figure(2) is the confusion matrix for the test data in my final model.

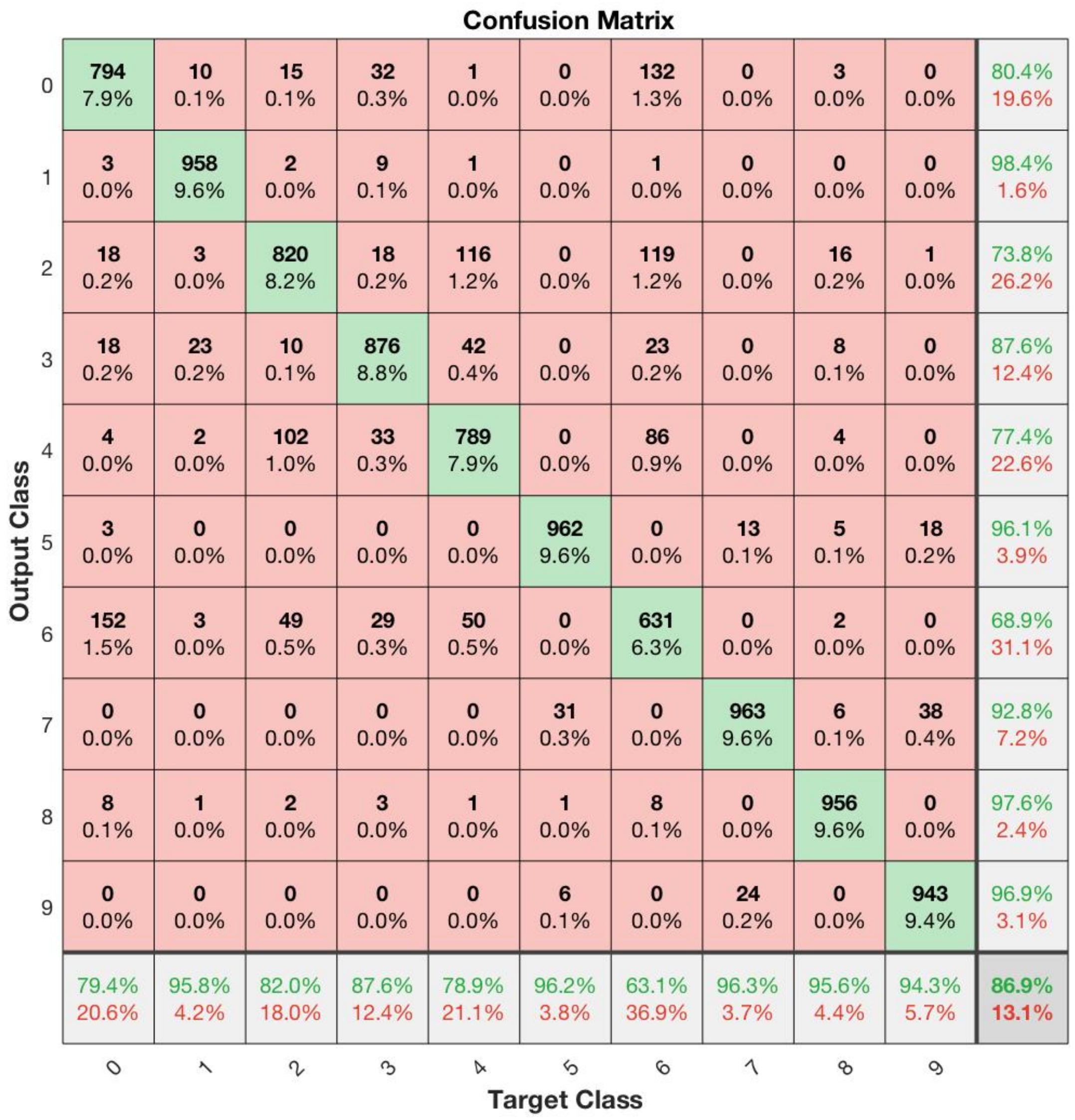


Confusion Matrix

Output Class (rows) / Target Class (columns)

	0	0.0039216	0.0078431	0.011765	0.015686	0.019608	0.023529	0.027451	0.031373	0.035294	
0	751 7.5%	4 0.0%	19 0.2%	12 0.1%	1 0.0%	0 0.0%	85 0.9%	0 0.0%	4 0.0%	1 0.0%	85.6% 14.4%
0.0039216	1 0.0%	967 9.7%	0 0.0%	5 0.1%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	1 0.0%	0 0.0%	99.3% 0.7%
0.0078431	9 0.1%	0 0.0%	704 7.0%	9 0.1%	63 0.6%	0 0.0%	44 0.4%	0 0.0%	1 0.0%	0 0.0%	84.8% 15.2%
0.011765	21 0.2%	15 0.1%	17 0.2%	906 9.1%	61 0.6%	1 0.0%	25 0.2%	0 0.0%	6 0.1%	0 0.0%	86.1% 13.9%
0.015686	1 0.0%	4 0.0%	104 1.0%	8 0.1%	758 7.6%	0 0.0%	42 0.4%	0 0.0%	0 0.0%	0 0.0%	82.7% 17.3%
0.019608	2 0.0%	0 0.0%	1 0.0%	0 0.0%	0 0.0%	953 9.5%	1 0.0%	19 0.2%	5 0.1%	9 0.1%	96.3% 3.7%
0.023529	205 2.1%	9 0.1%	149 1.5%	54 0.5%	115 1.1%	0 0.0%	790 7.9%	0 0.0%	11 0.1%	0 0.0%	59.3% 40.7%
0.027451	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	33 0.3%	0 0.0%	943 9.4%	4 0.0%	31 0.3%	93.3% 6.7%
0.031373	10 0.1%	1 0.0%	6 0.1%	6 0.1%	2 0.0%	0 0.0%	13 0.1%	0 0.0%	968 9.7%	0 0.0%	96.2% 3.8%
0.035294	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	13 0.1%	0 0.0%	38 0.4%	0 0.0%	959 9.6%	95.0% 5.0%
	75.1% 24.9%	96.7% 3.3%	70.4% 29.6%	90.6% 9.4%	75.8% 24.2%	95.3% 4.7%	79.0% 21.0%	94.3% 5.7%	96.8% 3.2%	95.9% 4.1%	87.0% 13.0%

Figure 1: Part1: Confusion matrix for the test data.


Confusion Matrix

Output Class

Target Class

	0	1	2	3	4	5	6	7	8	9	
0	794 7.9%	10 0.1%	15 0.1%	32 0.3%	1 0.0%	0 0.0%	132 1.3%	0 0.0%	3 0.0%	0 0.0%	80.4% 19.6%
1	3 0.0%	958 9.6%	2 0.0%	9 0.1%	1 0.0%	0 0.0%	1 0.0%	0 0.0%	0 0.0%	0 0.0%	98.4% 1.6%
2	18 0.2%	3 0.0%	820 8.2%	18 0.2%	116 1.2%	0 0.0%	119 1.2%	0 0.0%	16 0.2%	1 0.0%	73.8% 26.2%
3	18 0.2%	23 0.2%	10 0.1%	876 8.8%	42 0.4%	0 0.0%	23 0.2%	0 0.0%	8 0.1%	0 0.0%	87.6% 12.4%
4	4 0.0%	2 0.0%	102 1.0%	33 0.3%	789 7.9%	0 0.0%	86 0.9%	0 0.0%	4 0.0%	0 0.0%	77.4% 22.6%
5	3 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	962 9.6%	0 0.0%	13 0.1%	5 0.1%	18 0.2%	96.1% 3.9%
6	152 1.5%	3 0.0%	49 0.5%	29 0.3%	50 0.5%	0 0.0%	631 6.3%	0 0.0%	2 0.0%	0 0.0%	68.9% 31.1%
7	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	31 0.3%	0 0.0%	963 9.6%	6 0.1%	38 0.4%	92.8% 7.2%
8	8 0.1%	1 0.0%	2 0.0%	3 0.0%	1 0.0%	1 0.0%	8 0.1%	0 0.0%	956 9.6%	0 0.0%	97.6% 2.4%
9	0 0.0%	0 0.0%	0 0.0%	0 0.0%	0 0.0%	6 0.1%	0 0.0%	24 0.2%	0 0.0%	943 9.4%	96.9% 3.1%
	79.4% 20.6%	95.8% 4.2%	82.0% 18.0%	87.6% 12.4%	78.9% 21.1%	96.2% 3.8%	63.1% 36.9%	96.3% 3.7%	95.6% 4.4%	94.3% 5.7%	86.9% 13.1%

Figure 2: Part2: Confusion matrix for the test data.

5 Summary and Conclusions

For the first part, we achieve the highest accuracy as 87%. For the second part, we achieve the highest accuracy as 88.9%. We should be able to get better results by trying other parameters if we have a terminal computer to work on. Matlab has crushed my laptop several times. So I believe my result is not the best. But 89% is still a pretty good result of classifying fashion images.

Appendix A.MATLAB functions used[6]

1. categorical: creates a categorical array from the array
2. permute: rearranges the dimensions of an array in the order specified by the vector dimorder.
3. im2double: converts the image I to double precision.
4. trainingOptions: returns training options for the optimizer specified by solverName.

[6]Other functions applied in this report can be found in the first report. The full description of each function below can be foun on : "MATLAB." MATLAB Documentation, www.mathworks.com/help/matlab/.

Appendix B. MATLAB codes

```matlab
% Part1
clc; close; clear all
%%
load('fashion_mnist.mat');

% image to double
X_test = im2double(X_test);
X_train = im2double(X_train);

y_test = im2double(y_test);
y_train = im2double(y_train);

% reshape and reorder
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(200)
        reluLayer
        fullyConnectedLayer(50)
        reluLayer
        fullyConnectedLayer(100) %
        reluLayer %
        fullyConnectedLayer(10)
        softmaxLayer
        classificationLayer];

options = trainingOptions('adam', ...
    'MaxEpochs',15,...
    'InitialLearnRate',1e-3, ...
    'L2Regularization',1e-4, ...
    'ValidationData',{X_valid,y_valid}, ...
    'Verbose',false, ...
    'Plots','training-progress');

net = trainNetwork(X_train,y_train,layers,options);
%% Confusion for training
figure(2)
y_pred = classify(net,X_train);
plotconfusion(y_train,y_pred)

```

```
%% Test classifier
figure(3)
y_pred = classify(net,X_test);
plotconfusion(y_test,y_pred)
```

```
%% MNIST Classifier with convolutional neural network
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([5 5],10,"Name","conv_1","Padding","same")
    tanhLayer("Name","tanh_1")
    averagePooling2dLayer([2 2],"Name","avgpool2d_1","Padding","same","Stride",[2 2])
    convolution2dLayer([5 5],16,"Name","conv_2")
    tanhLayer("Name","tanh_2")
    averagePooling2dLayer([2 2],"Name","avgpool2d_2","Padding","same","Stride",[2 2])
    convolution2dLayer([5 5],120,"Name","conv_3")
    tanhLayer("Name","tanh_3")
    averagePooling2dLayer([2 2],"Name","avgpool2d_3","Padding","same","Stride",[2 2])
    convolution2dLayer([5 5],100,"Name","conv_4","Padding","same")
    tanhLayer("Name","tanh_5")
    fullyConnectedLayer(84,"Name","fc_1")
    tanhLayer("Name","tanh_4")
    fullyConnectedLayer(10,"Name","fc_2")
    softmaxLayer("Name","softmax")
    classificationLayer("Name","classoutput")];

options = trainingOptions('adam', ...
    'MaxEpochs',5,...
    'InitialLearnRate',1e-3, ...
    'L2Regularization',1e-4, ...
    'ValidationData',{X_valid,y_valid}, ...
    'Verbose',false, ...
    'Plots','training-progress');

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

%% Confusion for training
```

```
figure(1)
y_pred = classify(net,X_train);
plotconfusion(y_train,y_pred)

%% Test classifier
figure(2)
y_pred = classify(net,X_test);
plotconfusion(y_test,y_pred)
```