Classification Through Neural Networks

Aidan Sleavin

The goal is to train a computer algorithm to be able to classify pictures of objects into nine different categories. Through machine learning with neural networks a program will naturally learn the classification of different objects and be able to categorize future pictures fed to it.

Overview

Classifying data into different categories is a large portion of data science. With large quantities of data it becomes unreasonable for a human to parse through all of the information and sort by hand. This is where a classification algorithm comes into place to quickly sort through data. Whereas many classification algorithms are user defined, neural networks are not. That is to say non-neural network classifiers take a range of specific user inputs to figure out how to sort data. Neural networks discover these differences for themselves through trial and error.

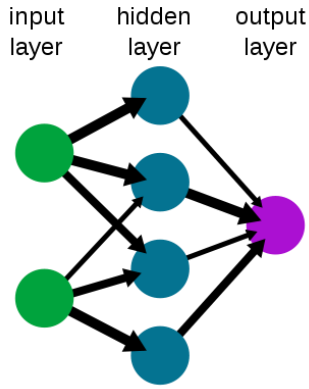
To classify, neural networks need data to "train" on. It needs to look an hundreds of images stamped with the correct classification and learn from what it sees. As the neural network trains if goes through many iterations, each time refining its ability to classify and becoming more accurate. As it iterates it tests itself on the data already trained on.

Through neural networks we want to classify pictures of objects into nine different categories, T-shirt/top, Trouser, Pullover, Dress, Coat, Sandal, Shirt, Sneaker, Bag, and Ankle boot. Given in the data are 60,000 images to train on and an additional 10,000 to test the final neural network with. Each image is 28x28 pixels which makes the training quicker but the images themselves are less accurate.

Theory

As the name implies neural networks are modeled after an animal's brain and nervous system. Our brains are made of many neurons that interact with each other constantly sending information to the next. Each one is connected to many others to create a complex network. This is the same for the neural networks made on a computer. In a neural network there are multiple layers of what can be

A simple neural network *Figure 1*



described as neurons. There are three main categories of layers, the input layer where the data is given to the network, the output layer where the network gives out its response, and everything in between are the hidden layers where the network runs through the data to give the final answer (a diagram can be seen in (figure 1).

The two types of neural networks used in this project are Convolutional Neural Networks (CNN), and Fully Connected Neural Networks (FCN). A FCN will have each piece of a layer send information to each piece in the next layer. A CNN will only have each piece send its information to some the pieces on the next layer. A CNN is more like what is seen in brains, each neuron has a finite amount of connection to the neurons around it and will be clustered into clumps that deal with one piece of information. Where if and FCN was in a brain that would be like each neuron connecting to every other neuron in the body.

When creating the neural network there are multiple factors which change how accurate after it is done training. A key one is the amount of hidden layers used (depth) and how and how many "neurons" are in each layer (width). One might think that just having a larger depth and width would always wind up with more accuracy, albeit at a computing efficiency cost, but if there is too much of either the data starts to get over fit. This means that the neural network will do a great job classifying what it was trained on but wont be able to effectively classify outside its training data.

Another obvious factor is how long the data is allowed to train for, or the number of epochs. As a network trains its increase in ability is logarithmic an eventually reaches a point where with the already given parameters wont get any better. So through trial and error this amount must be found, if the network is only allowed to run for a couple generation chances are it will not be very accurate.

Implementation

To begin the data that is to be trained on must be fed into the program MATLAB and structured into a readable format for the neural network. This happens from line 2 through 13. The variables given in the data are, pictures X_train and X_test, to be trained and tested on respectively. And 2 vectors y_train and y_test, which for each picture contain a value 0-9 that corresponds to a different item. The first 5000 variables in the training data are taken out to be used later as validation data. The validation data is used to see how well adjustments to the parameters of the neural network affect the results.

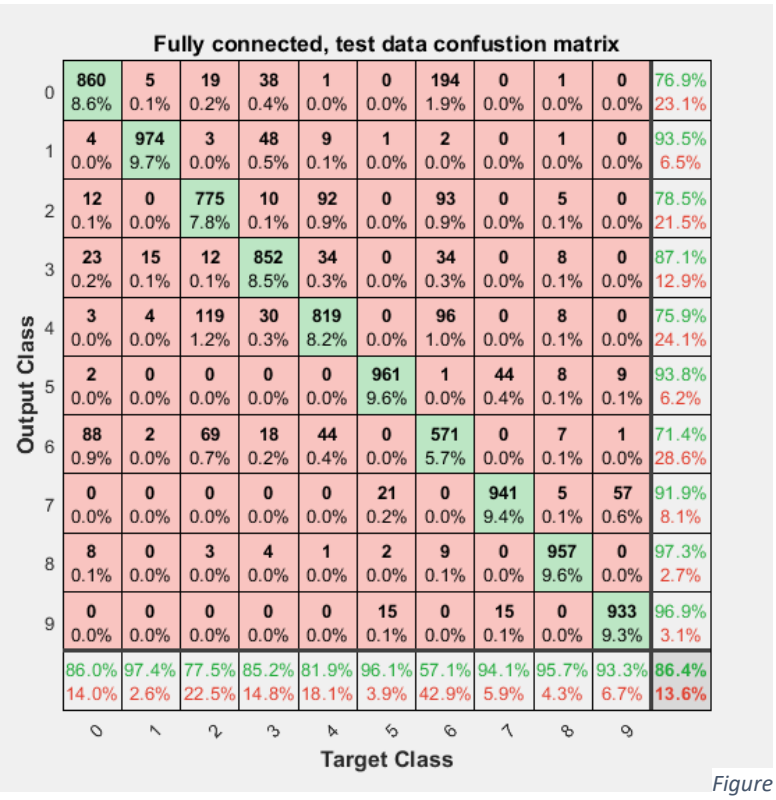
For line 15-34 is the creation of the FCN. It starts with the imageInputLayer and uses four fully connected layers. The width of the final layer (the output) needs to be 10 because what is being output is a percentage of confidence that the image belongs to each of the 10 categories. The input layer needs to be equal to the size of the picture 28x28. For the hidden layers is the trial and error. The layers shown in Appendix B are the final form of layering used as it is what gave best results. From line 27-24 are the other option used in the neural network. The variable being changed are, the amount of epochs, and the learn rate. After around 4 epochs the training was slow enough it was not necessary to go further. And around a training rate of 1.4*e-3 was best.

Lines 37-59 implement the same creation for the neural network but for the CNN. Again the output layer is a fully connected layer with 10 outputs. The main changes to make to find the best neural network with a CNN is the stride component which is how much of the image is overlapped for each iteration.

To see what it is like when the neural network is being trained, figure 3 shows a graph over each iteration of how accurate the network is.

Results

At the beginning of the training the neural networks gain a lot of accuracy but quickly peter out. After around 4 epochs the networks have almost nothing to gain by further training. A learn rate of 1.4e-3 is the best through trial and error, as well as a 5e-4 L2 regularization. For the CNN a stride of 2x2 was the best at it is very precise and 1x1 would take to long to effectively run.



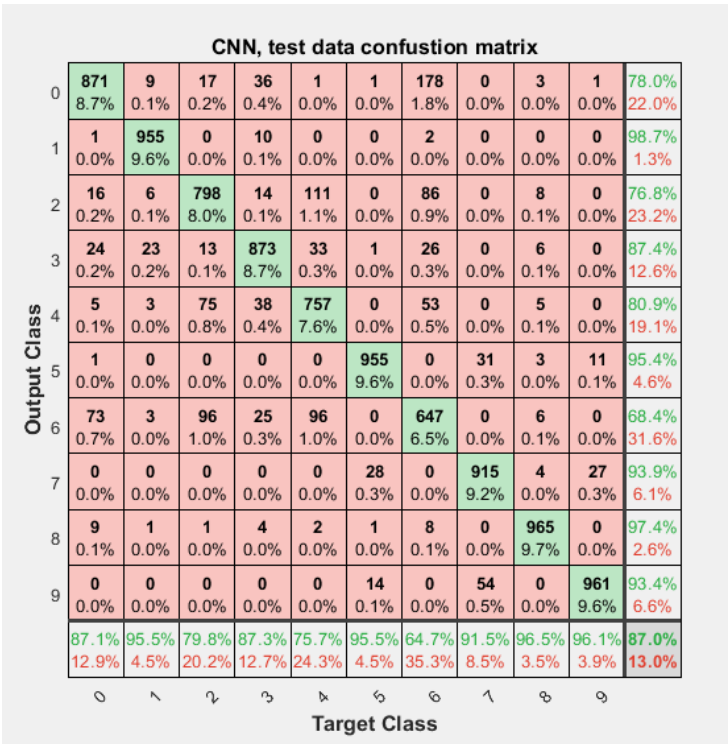


Figure 2

Overall both of the neural networks ended up with an accuracy of around %86.5. when randomly giving the networks parameters they would have an accuracy at about %84, but with optimization went up. Both once completed with testing on the validation data each network was run on the test data. In figure 2 the confusion matrix for each network can be seen. The confusion matrix shows for each item, 0-9, the percentage the network thought is was each item, 0-9. It can be seen the network always had the highest percentage for the correct one, but not always correct. The CNN ended up with the best total accuracy of %87.0 and the FCN ended up with %86.4. From both confusion matrix it can be seen that the item with the index of 6 (Shirt) was confused the most. This could be because a shirt in many aspects looks similar to pullovers, dresses, and coats. Similar in the regards that it would have edges where the other do, in the body and the arms. In the output and target class shirts had less than a %65 percent chance of being identified for both networks, pretty poor.

But because shirts took the average so far down most of the other items had a high 80 percentile and into the 90s of being classified correctly.

Conclusion

Neural networks can accurately and quickly classify data from multiple categories to reduce human work. They have the ability to self-determine the important pieces that stand out for recognition which might be difficult to explicitly define for a computer. Convolutional networks have a bit of the edge on classification though the time it takes to run can be quite a bit slower. This is because they more accurately model neurons an how they interact with the neurons around.

Appendix A: functions

- Im2Double: takes a pixel of intiger from 0-255 and converts to double precision from 0-1
- Reshape: changes the array to and array of the same size but different side lengths
- Permute: changes the order in which each piece in an array are indexed
- imageInputLayer: creates the neural network taking information of a desired size. Within this function can be called many other functions such as:
    - fullyConnectedLayer: creates a new fully connected layer of desired size
    - convolution2DLayer: creates a convoluted layer for 2d data
- trainingOptions: defines the options used for training the neural network
- plotConfusion: takes in two vectors of the same length with different variables n and gives and nxn plot on how often each was accurate to each other.

Appendix B:

```
1. close; clear all; clc;

2. load('fashion_mnist.mat');

3. X_train = im2double(X_train);
4. X_test = im2double(X_test);

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

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

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

11.     y_valid = categorical(y_train(1:5000))';
12.     y_train = categorical(y_train(5001:end))';
13.     y_test = categorical(y_test)';
```

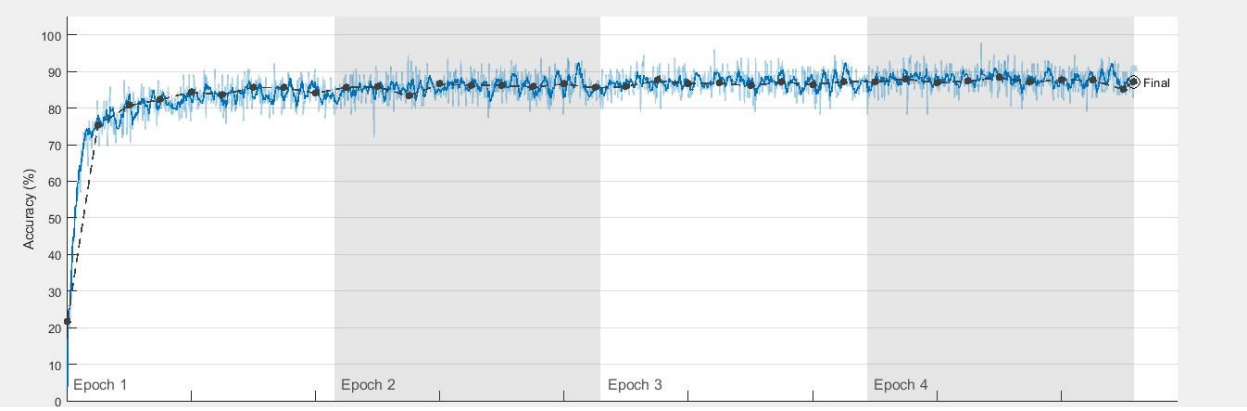


Figure 3

```
14.     %%
15.     layers = [imageInputLayer([28 28 1])
16.     fullyConnectedLayer(100)
17.     reluLayer
18.     fullyConnectedLayer(80)
19.     reluLayer
20.     fullyConnectedLayer(50)
21.     reluLayer
22.     fullyConnectedLayer(20)
23.     reluLayer
24.     fullyConnectedLayer(10)
25.     softmaxLayer
26.     classificationLayer];

27.     options = trainingOptions('adam',...
28.     'MaxEpochs',4,...
29.     'InitialLearnRate', 1.4e-3,...
30.     'L2Regularization', 5e-4,...
31.     'ValidationData', {X_valid, y_valid},...
32.     'Verbose',false,...
33.     'Plots','training-progress')

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

35.     %%
36.     %convolutional
37.     layers = [
38.     imageInputLayer([28 28 1],"Name","imageinput")
39.     convolution2dLayer([5 5],6,"Name","conv_1","Padding","same")
40.     tanhLayer("Name","tanh_1")
41.     averagePooling2dLayer([3
   3],"Name","avgpool2d_1","Padding","same","Stride",[2 2])
42.     convolution2dLayer([5 5],16,"Name","conv_2")
43.     tanhLayer("Name","tanh_2")
44.     averagePooling2dLayer([2
   2],"Name","avgpool2d_2","Padding","same","Stride",[2 2])
45.     convolution2dLayer([5 5],120,"Name","conv_3")
46.     tanhLayer("Name","tanh_3")
47.     fullyConnectedLayer(84,"Name","fc_1")
48.     tanhLayer("Name","tanh_4")
49.     fullyConnectedLayer(10,"Name","fc_2")
50.     softmaxLayer("Name","softmax")
51.     classificationLayer("Name","classoutput")];

52.     options = trainingOptions('adam',...
53.     'MaxEpochs',5,...
54.     'InitialLearnRate', 1.4e-3,...
55.     'L2Regularization', 5e-4,...
```

```
    'ValidationData', {X_valid, y_valid},...
    'Verbose',false,...
    'Plots','training-progress')

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

    %%
    y_pred = classify(net,X_test);
    figure(3)
    plotconfusion(y_test,y_pred);
    title("CNN, test data confustion matrix")
```