Understanding the DeepLearnToolbox CNN Example

10 Jan 2015

In this post, I provide a detailed description and explanation of the Convolutional Neural Network example provided in Rasmus Berg Palm's DeepLearnToolbox for MATLAB. His example code applies a relatively simple CNN with 2 hidden layers and only 18 neurons to the MNIST dataset. The CNN's accuracy is 98.92% on the test set, which seems very impressive to me given the small number of neurons.

The code is neat and well written, and I think it's a huge service to the engineering community when people share their code like this. That said, there's almost no documentation, explanation, or comments provided, so I'm attempting to partially rectify that here. Down at the bottom of the post you can find a link to my documented / commented versions of many of the CNN functions.

Should You Use DeepLearnToolbox?

Before we dive in, I think it's worth pointing out that the CNN code in the DeepLearnToolbox is a few years old now, and deep learning has been evolving rapidly. Other than the fact that it's a multi-layered Convolutional Neural Network, this code doesn't appear to use any of the innovations typically associated with “Deep Learning”. For example, there is no unsupervised feature learning going on here, or layer-wise pre-training. It’s just back-propagation over labeled training examples. Also, it still uses the sigmoid activation function, and it seems to me that a lot of current deep networks appear to use other activation functions such as “ReLU”.

That said, I’ve found the code very informative for helping me understand some of the basics of CNNs. At only 18 neurons, the example network is fairly small and simple (relatively speaking–even small CNNs can feel like a rat’s nest of parameters and connections!). So if your goal is, like me, to learn more about CNNs, keep reading on.

If you really just want a good library for building state-of-the-art CNNs, you might look elsewhere. There’s an extensive list of libraries here, but it doesn’t seem to be very well organized by quality or relevance. You might find some helpful guidance from these discussions on reddit in /r/machinelearning here and here.

Where I’m Coming From

I learned some of the basics of Machine Learning from Andrew Ng’s Machine Learning course on Coursera, including Neural Networks (the Multi-Layer Perceptron). After that, I learned a lot of the fundamentals of deep learning from Stanford’s Deep Learning Tutorial (which Ng also contributed to).

The Stanford tutorial was very helpful, and you even get to build a CNN in the last exercise. However, it uses different terminology and coding styles than some of the other important work out there, and doesn’t show you how to build a CNN with multiple hidden layers. I found this

DeepLearnToolbox MATLAB code to be very informative for filling in some of my missing knowledge.

Terminology

One of the most helpful aspects of this exercise for me has been the chance to learn some of the terminology used by Convolutional Neural Networks. There are *a lot* of redundant terms here. It's not a terrible thing, though; each different label can help you interpret the same concept in a different way. The key is just knowing what they all mean.

Convolution, filter mask, and kernel

Some of the language in CNNs seems to be taken from image processing. In image processing, a common operation is to apply a filter to an image. The filter has a small "filter mask" or "kernel" which is "convolved" with the image. (If you're unfamiliar with filters and convolutions, I have a post on it here, though it's not my best work :)).

If you think about the operation performed between the filter mask and a single patch of the image, it's the same basic operation as an MLP neuron–each of the corresponding components are multiplied together, then summed up. It's the same as taking the dot product between the image patch and the filter mask if you were to unwind them into vectors. The difference in terminology really just comes from the 2D structure of the image.

So when you see the words "filter" or "kernel", these just correspond to the weights of a single neuron.

Feature

Another term I like to use for these same weight values is “feature”. The techniques in Unsupervised Feature Learning show us that we can really think of a neuron’s weights as a particular “feature” that the neuron is responding to.

Maps

But wait, there’s more! CNNs introduce another term, which is very helpful in reasoning about them. The result of a convolution between a single filter mask and an image is referred to as a “map”. A single map is a 2D matrix which is the result of applying a filter over the entire image. Note that the map will be a little smaller than the image (by the width of the kernel), because the filter can’t be applied past the edges of the image.

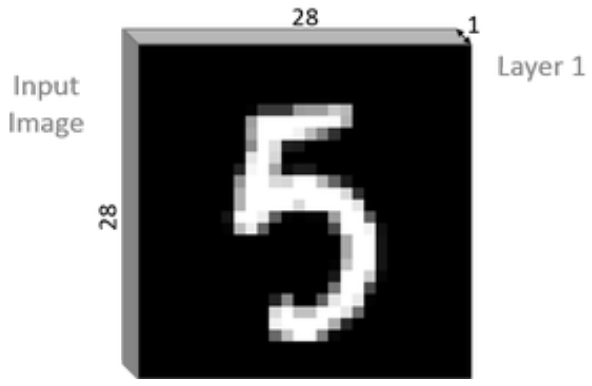
Each layer of a CNN will output multiple maps, one for each of the “neurons” / “kernels” / “filters” / “features” in that layer.

Each layer of the CNN applies its filters to the maps output by the previous layer.

Architecture In Detail

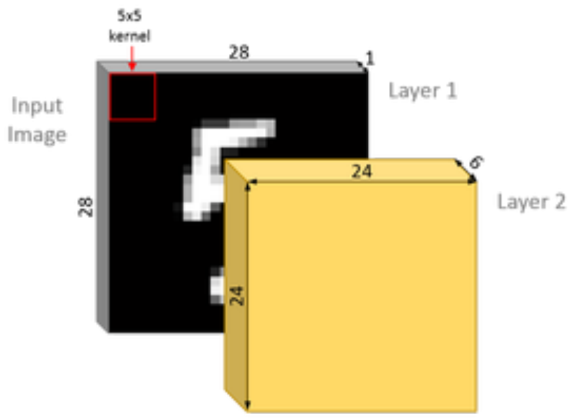
To understand a particular network architecture, I think it makes the most sense to start by looking at the feed-forward operation.

Layer 1



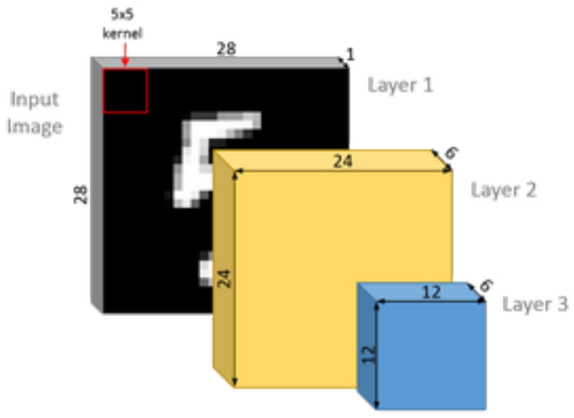
For this example, we're working with the MNIST hand-written digit dataset. The input to the CNN, therefore, is a 28x28 pixel grayscale image. This is the input to the network, but we also treat it in the code as the "output of layer 1".

Layer 2



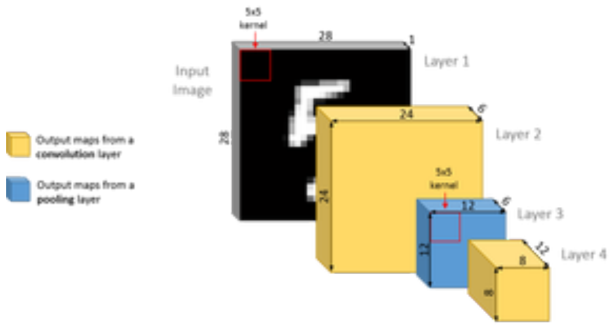
The first hidden layer of the network, "Layer 2", is going to perform some convolutions over the image. The filter mask will be 5x5 pixels in size. Convolving a 5x5 filter with a 28x28 pixel image yields a 24x24 filtered image. This layer has 6 distinct filters (or 6 neurons, if you like) that we're going to convolve with the input image. These six convolutions will generate 6 separate output maps, giving us a 24x24x6 matrix as the output of layer 2.

Layer 3



Layer 3 is a pooling layer. The pooling operation we're using here is just averaging, and the pooling regions are very small: just 2x2 pixels. This has the effect of subsampling the output maps by a factor of 2 in both dimensions, so we get a 12x12x6 matrix.

Layer 4



Layer 4 is another convolution layer. Again we'll use a kernel size of 5x5. In Layer 4, we have 12 distinct filters that we will apply. That is, it contains 12 neurons.

There is an important detail here, though, that you don't want to miss. When we performed our convolutions on the original input image, the image only had a depth of 1 (because it's grayscale). The output of Layer 3, though, has a depth of 6.

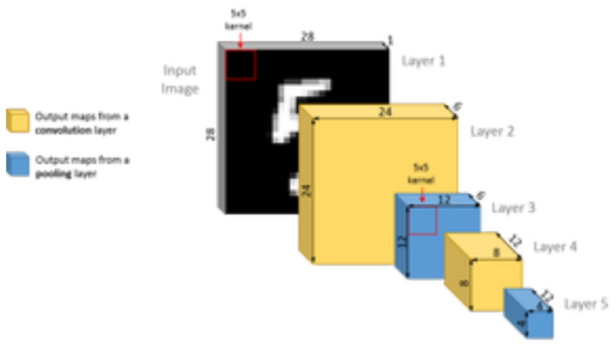
A single neuron in layer 4 is going to be connected to all 6 maps in layer 3.

There are different ways of thinking about how this is handled.

One way is to say that our Layer 4 filters have a size of 5x5x6. That is, each filter in Layer 4 has 150 unique weight values in it.

In the code, however, convolutions are only done with two-dimensional filters, and it becomes a little hairy to describe. Instead of having 12 5x5x6 filters, we have 72 5x5x1 filters. For each of the 12 Layer 4 filters, there are 6 separate 5x5x1 kernels. To apply a single Layer 4 filter, we actually perform 6 convolutions (one for each output map in Layer 3), and then sum up all of the resulting maps to make a single 8x8x1 output map for that filter. This is done for each of the 12 filters to create the 8x8x12 output of Layer 4.

Layer 5



Finally, we perform one last pooling operation that's identical to the one in Layer 3–we just subsample by a factor of 2 in each dimension. The resulting output maps are unwound into our final feature vector containing 192 values (4x4x12 = 192).

This feature vector is then classified using simple linear classifiers (one per category) on the output.

Some final notes on the code:

- The CNN code doesn't quite handle color images. The implementation of the convolutional layer is abstracted to handle a variable number of input maps (and a color image is just an input with 3 maps), but the functions are hardcoded to assume that input layer (the first layer) only has one map.
- The pooling operation requires that the scaling factor divide evenly into the map size. This is a problem if you had, for example, a 32x32 pixel image with a 6x6 kernel. This produces a map size of 27x27 which you can't divide in half evenly.
- In other code I have worked with (the Coursera Machine Learning course, and in Stanford's deep learning tutorial) image patches are unwound into vectors, and neuron activations are calculated as matrix products. Here, the code instead retains the 2-dimensional structure of the image and uses MATLAB's convolution operator 'convn' to calculate the neuron activations.

The Code

To get the example code, first download the DeepLearnToolbox from its GitHub page. It's just a directory of MATLAB functions, so there's nothing special you need to do to install it other than adding it to your path.

Then, you can download my commented / documented code here and replace the corresponding files.

Specifically, here's what my zip file contains:

- A documented and slightly-reworked version of the MNIST CNN example, 'test_example_cnn.m'.
- Documented and commented versions of 'cnnff.m', 'cnnsetup.m', 'cnntest.m', and 'cnntrain.m'
    - I did not document the key training function 'cnnbp.m', which calculates the gradients, or 'cnnapplygrads.m', which updates the parameter values using the gradients. Sorry!
- 'CNN_Illustrations.pptx' - The powerpoint slides containing the illustrations in this post.
- 'cnn_100epochs.mat' - A CNN trained on the MNIST dataset for 100 epochs. It gets 98.92% on the test set.

19 Comments mccormickml.com 1 Login

Recommend 3 Share Sort by Best

Join the discussion…

Computer Eng. • 6 months ago

Dear,

I have the same question. How to use my own dataset to extract the features. when i tried i received this

Index exceeds matrix dimensions.

Error in minFuncSGD (line 58)

mb_data = data(:,:,:,rp(s:s+minibatch-1));

Error in cnnTrain (line 28)

opttheta = minFuncSGD(@(x,y,z)
cnnCost(x,y,z,cnnConfig,meta),theta,images,labels,options);
5 • Reply • Share ›

Abdelhak Ziouani • 7 months ago
I cant run this code. my error is
Undefined function 'expand' for input arguments of type 'double'. in file cnnbp.m
1 • Reply • Share ›

Ismail T Ahmed • 3 months ago
hello
when i run on my dataset i have this error

Error using cnnsetup (line 9)
Layer 3 size must be integer. Actual: 19198 -1.5
• Reply • Share ›

Khaoula Khaoula Hadjer • 6 months ago
thank you for your explanation. I run the code with a faces database and I get 98.33%. But I want to understand how to choose the rate of error and what 0.12 means exactly? Another question is how I determine the number of kernel. I try with 6 filter in the 1st convolution layer and 6 also in the second I obtain 98% and when I try with 6 in the first and 12 in the second I get 95%
• Reply • Share ›

Moahaimen Talib • 6 months ago
dear sir i have 20 binary images with *.pbm and each image has 16 feuture i need to classify these images in unsupervised deep neural network please could you help me in what i have to code as input ?
• Reply • Share ›

Abdelhak Ziouani • 7 months ago
I cant run this code.

• Reply • Share ›

bongda • 10 months ago
I cant run this code. Can you tell me why. I use matlab 2014a. i downloaded it and extract to DeepLearnToolbox folder.

• Reply • Share ›

Chris McCormick Mod bongda • 10 months ago

Can you share the error you're getting?

• Reply • Share ›

bongda Chris McCormick • 10 months ago

this is my error
>> load('mnist_uint8.mat')
>> test_example_CNN
Undefined function or variable 'cnnsetup'.

Error in test_example_CNN (line 28)
cnn = cnnsetup(cnn, train_x, train_y);

• Reply • Share ›

Chris McCormick Mod bongda • 10 months ago

Probably just a path issue? Make sure CNN/cnnsetup.m is in your matlab path. Run 'path' to see your current path, and use the 'addpath' function to add directories.

• Reply • Share ›

bongda Chris McCormick • 10 months ago

i did it. but i have more error
>> load('mnist_uint8.mat')
>> test_example_CNN
Undefined function or variable 'isOctave'.

Error in cnnsetup (line 2)
assert(~isOctave() || compare_versions(OCTAVE_VERSION, '3.8.0', '>='), ['Octave 3.8.0 or greater is required for CNNs
as there is a bug in convolution in previous versions. See http://savannah.gnu.org/bug.... Your version is ' myOctaveVersion]);

Error in test_example_CNN (line 28)
cnn = cnnsetup(cnn, train_x, train_y);

• Reply • Share ›

Chris McCormick Mod bongda • 9 months ago

You're using Matlab, not Octave, so that assert doesn't apply to you-

-you could simply remove that line. Or even better, you could define a small function 'isOctave' that just returns false.

• Reply • Share ›

dung • a year ago

How do u custom code for use images color?

• Reply • Share ›

numlengg • a year ago

Dear all, I read and run the above example of CNN. It's very nice and informative. I need some guidance. If I have a data set and instead of splitting them into training and testing, I am interesting to find the features using CNN and later on classify using SVM or any other classifier. What should I do with this example. In other words how can I get the features only (With out classification).

• Reply • Share ›

Computer Eng. numlengg • 6 months ago

Dear,

I have the same question. How to use my own dataset to extract the features. when i tried i received this

Index exceeds matrix dimensions.

Error in minFuncSGD (line 58)

mb_data = data(:,:,:,rp(s:s+minibatch-1));

Error in cnnTrain (line 28)

opttheta = minFuncSGD(@(x,y,z) cnnCost(x,y,z,cnnConfig,meta),theta,images,labels,options);

1 • Reply • Share ›

ttt • a year ago

Thank you very much for the above nice explain.

• Reply • Share ›

Chris McCormick Mod ttt • a year ago

Thanks!

• Reply • Share ›

Tim • a year ago

very clearly explained, really helpful thanks!

• Reply • Share ›

Chris McCormick Mod Tim • a year ago

Thanks, glad it was helpful!

• Reply • Share ›

Related posts

Product Quantizers for k-NN Tutorial Part 1 13 Oct 2017

k-NN Benchmarks Part I - Wikipedia 08 Sep 2017

Concept Search on Wikipedia 22 Feb 2017