Convolutional Neural Networks

Joseph Suarez, Matthew Bardoe
Choate Rosemary Hall, Wallingford, CT
{jsuarez15, mbardoe}@choate.edu

Abstract

Convolutional neural networks (CNN) have been shown to yield much better performance on many image datasets than standard feed-forward artificial neural networks (FFANN). Though they also have the *potential* to run more quickly than FFANN, taking advantage of this speedup requires a highly optimized GPU or parallelized CPU implementation of the convolution operation, which accounts for nearly all of the computational cost. We show that *reasonably* optimized code of the type produceable by non-specialists can yield passable results on CNN. We implement several simple tricks to reduce runtime. We also present a selection of successful CNN trials that illustrate how the network behaves with a small number of layers. The program that accompanies this paper is extremely flexible and allows users to very quickly build custom CNN and FFANN.

1.0 Processing Time

While it is computationally feasible to cross-validate any one of the CNN's parameters, to our knowledge, there is no "best" value for any given parameter over all possible combinations of other parameters, as parameters do not appear to be independent. We find empirically that networks with certain numbers of neurons suddenly stop working when training data is added or removed. Intuitively, the best results should be obtainable using the largest number of train samples available (around 50,000 for the MNIST handwritten dataset). Tuning parameters to any one number of train samples requires a few dozen tests, thus it is not practical to run experiments with the full dataset due to time constraints. However, training with a few hundred examples is also not practical, as the training results do not generalize well to the test set.

We implement learning using *minibatches*: we train with a small number of samples (approximately 100) per iteration, but each iteration, a new minibatch of samples is randomly selected from the entire MNIST training set. In addition to reducing computation time, this procedure also reduces the effects of overfitting.

We also note the amount of processing power allotted to each line of code using Matlab's profile viewer (see Table. 1). Most compute time is spent on the convolution operation (during the forward pass). By preallocating certain data structures outside of the convolution operation, we reduce runtime by an order of magnitude; however, the convolution operation still accounts for most of runtime, as shown in Table 1.

Table 1: Output of Matlab's profile viewer after three iterations similar to trial 3 in Table 2.

Profile Summary
Generated 30-May-2014 03:00:06 using cpu time.

Function Name	Calls	Total Time	Self Time*	Total Time Plot (dark band = self time)
CNNForwardProp	1100	193.464 s	193.011 s	
squeeze	173400	11.910 s	11.910 s	
TrainGeneralCNN>@(x)1./(1+exp(-x))	12100	0.453 s	0.453 s	
strcat	12	0.018 s	0.018 s	
num2str	12	0.010 s	0.000 s	
num2str>handleNumericPrecision	9	0.010 s	0.005 s	
num2str>convertUsingRecycledSprintf	9	0.005 s	0.005 s	
int2str	3	0 s	0.000 s	
blanks	12	0 s	0.000 s	

Because of limitations in processing power, we constrain our CNN to two or three layers. We show that the network is indeed learning, but we do not train a full scale CNN.

2.0 Choosing Hyperparameters: Data

We present the set of trials that allowed us to optimize the CNN in Table 2, 3. The maximum accuracy of 88.1% on the test set is achieved with a two-layer CNN in trial 5.

Table 2: General data for CNN trials. Train and test accuracies are similar because use of minibatches essentially eliminates the overfitting problem.

Trial Number	Minibatch Size	Iterations	Train Accuracy	Test Accuracy
1	100	30	0.758	0.773
2	200	50	0.693	0.737
3	200	500	0.850	0.861
4	300	1000	0.867	0.874
5	**300**	**2000**	**0.859**	**0.881**
6	500	2000	0.864	0.865
7	100	150	0.71	0.65
8	100	250	0.78	0.78
9	300	2000	0.719	0.792

In Table 2, trials 1-6 were performed using a two layer CNN, trial 7-8 with a three layer CNN, and trial 9 with a FFANN. While trials 8-9 yielded lower accuracy, they prove that our three layer CNN is learning, and simply requires more training iterations, potentially with a larger minibatch size, to yield good results; trial 8 was the longest running trial, taking over an hour to complete.

Table 3: infoMatrix values for corresponding CNN trials in Table 2. The first row of infoMatrix denotes layer types (0=input, 1=convolution, 2=subsampling, 3=FFANN). The second row denotes the dimensions of convolutional/ subsampling boxes and the number of neurons for FFANN layers. The last row is used only for convolution layers and represents the number of different weight matrices used to examine the previous layer.

Trial Number	Info Matrix

1-6	`infoMatrix=[...`
	`0, 1;`
	`1, 28;`
	`1, 10;];`

7-8	`infoMatrix=[...`
	`0, 1, 1;`
	`1, 5, 24;`
	`1, 10, 1];`

9	`infoMatrix=[...`
	`0, 3;`
	`1, 10;`
	`1, 1];`

3.0 Visualizing Convolution Filters

Preliminary trials included no FFANN layers to ensure that the algorithm was performing error backpropagation through the convolution layer correctly. These initial trials possess an interesting property: as the input image was examined with ten different weight matrices with the same dimensions as the input, each weight matrix learned to recognize one number. Two visualizations of the weights are shown in Fig. 2.

Fig. 2: Visualization of weights in a two-layer CNN. Note that each portion of the figure contains ten, concatenated weight matrices that are normally stored in a block matrix.

4.0 CNN vs. FFANN

We compare trials 5, 9 in Table 2 in order to show that, allotted the same minibatch size, number of iterations, and number of weights, CNN can outperform FFANN. Train time for these two networks is similar, as small CNN with few convolution operations are not computationally expensive to train. Trial 5 also surpasses all of our FFANN experiments in previous projects.

5.0 Conclusion

We show that small CNN can outperform FFANN with the same number of weights, discuss runtime optimization, present data from various CNN and FFANN experiments, and demonstrate that CNN filters can be meaningfully visualized. In our best result, we achieve 88.1% accuracy on the MNIST dataset using a two-layer CNN. Train time for this run is only a few minutes, but larger CNN require a drastic increase in processing power; we hypothesize that more processing power or further optimization of our program would yield improved results. The program we present in this paper is very general: its usage of infoMatrix allows the user to quickly build a custom CNN or FFANN, and could serve as a valuable testing tool with some optimization.