Scalable Deep Learning for Image Classification with K-Means and SVM

Alexandre Vilcek
(vilcek@gmail.com)

Abstract. Deep Learning has got a lot of attention recently in the specialized machine learning community and also in common media – the latter mainly due to research activities of large technology companies. A large part of that research is applied to image classification tasks, and is based on techniques such as Deep Belief Networks and Convolutional Neural Networks. Although those techniques yield state of the art results, they are known to be hard to tune and scale and, as of today, there is very few software packages available to allow for a simple and quick implementation. In this report I will show an implementation of a deep learning framework based on recent research [1],[3], proposing K-Means as the algorithm for the unsupervised learning step. I will also show some techniques that allows for implementation of scalable versions of the proposed framework.

1. Introduction

The proposed framework will be implemented in R (The R Project for Statistical Computing) running on a standard Linux desktop. Specifically, for the unsupervised feature learning task, I will implement the Spherical K-Means algorithm in R, porting it from MATLAB code used in [1]. For the image classification task, I will use an SVM implementation in R provided by [5].

To test the implementation, I will use the MNIST dataset [4]. The goal is to learn a new feature representation for the images that is suitable for a linear classification task. In the standard pixel representation a good classification is possible, but demands a much more complex non-linear classifier, such as a deep neural network or an SVM with a non-linear kernel. A non-linear classifier turns out to be harder to implement in a scalable way.

I will present this work as follows: In chapter 2 I depict an overview of the proposed architecture, showing its main processing steps and how they are composed to build a complete deep learning framework[1]. In chapter 3 I present results obtained by applying this framework for a handwritten digits classification task using the MNIST data set. This data set was chosen due to its low dimensionality and large number of data examples. Also, it is extensively used in image recognition benchmarks, which helps when comparing this solution to alternatives. In chapter 4 I discuss possible implementations that allows this proposed framework to scale and be suitable for very large datasets. Finally, in chapter 5 I conclude this report and suggest further investigation.

2. Architecture Overview

The architecture for the proposed framework is shown in Figure 1. It is implemented according to the following steps:

1. Pre-processing of the data set:
Here we need to perform brightness and contrast normalization, as well as whitening, so that all the feature remapping processing provide good results. I followed the corresponding procedures as described in [1].

2. Extracting random patches from the data set:
These patches, extracted from the original data set, are the input for the unsupervised learning procedure (next step). Again, I followed the suggestions presented in [1] and [3] for the necessary amount of patches and the patch size. In my setup, I extracted 500,000 7x7 random patches from 60,000 28x28 images from the training data set. Each one of those are an image of a handwritten digit (0 to 9).

[1] Due to lack of computational resources available, I implement only a single-layer of the framework

3. Learning a dictionary of features:

Here we perform the unsupervised learning step through K-Means, which attempts to learn a dictionary of image features that will be further used to build a new representation from the original dataset. The setup for this step followed the suggestions from [1] and [3], concerning K-Means implementation (an implementation known as Spherical K-Means in this case), necessary number of clusters and processing steps. My initial intention was to use an of-the-shelf implementation of the Spherical K-Means algorithm, such as [6], but it turned out to demand too much memory, surpassing what I had available for the tests. So I decided to port to R an implementation in MATLAB provided by [1], which performed much better with limited memory resources. For this setup I run it with 1,600 centroids. Figure 2 shows two learned feature dictionaries. On the left, we see the dictionary learned without pre-processing of the data set. Clearly we see that it fails in learning good "stokes" representations of the handwritten digits and it is full of empty clusters. On the other hand, the image on the right shows another dictionary, now learned after pre-processing of the data set as described earlier and yielding much better results much more similar to actual pen strokes with clear edges.

4. Mapping the original features (pixels) of the dataset into a new feature space, which is a function of the learned dictionary:

Now we compute a new feature representation of the original data set, which will hopefully turn it linearly separable. As suggested in [1], I implemented the soft threshold encoding scheme for K-Means. In this step, each original 28 x 28 pixel image, represented in a 1 x 784 vector, is now encoded in a 1 x 6400 vector.

5. Optionally repeat steps 1 to 4, if more than one "layer" of feature representation is desirable:

In this experiment, due to limited computational resources available, I tested only a new feature representation in one layer.

6. Classify the data set, now represented in the new learned feature space, using standard linear classification algorithms:

In this experiment, I tested it with the SVM implementation provided by [5]. I performed 10-fold cross-validation to choose the C parameter, which turned out to be C=50 the best choice.

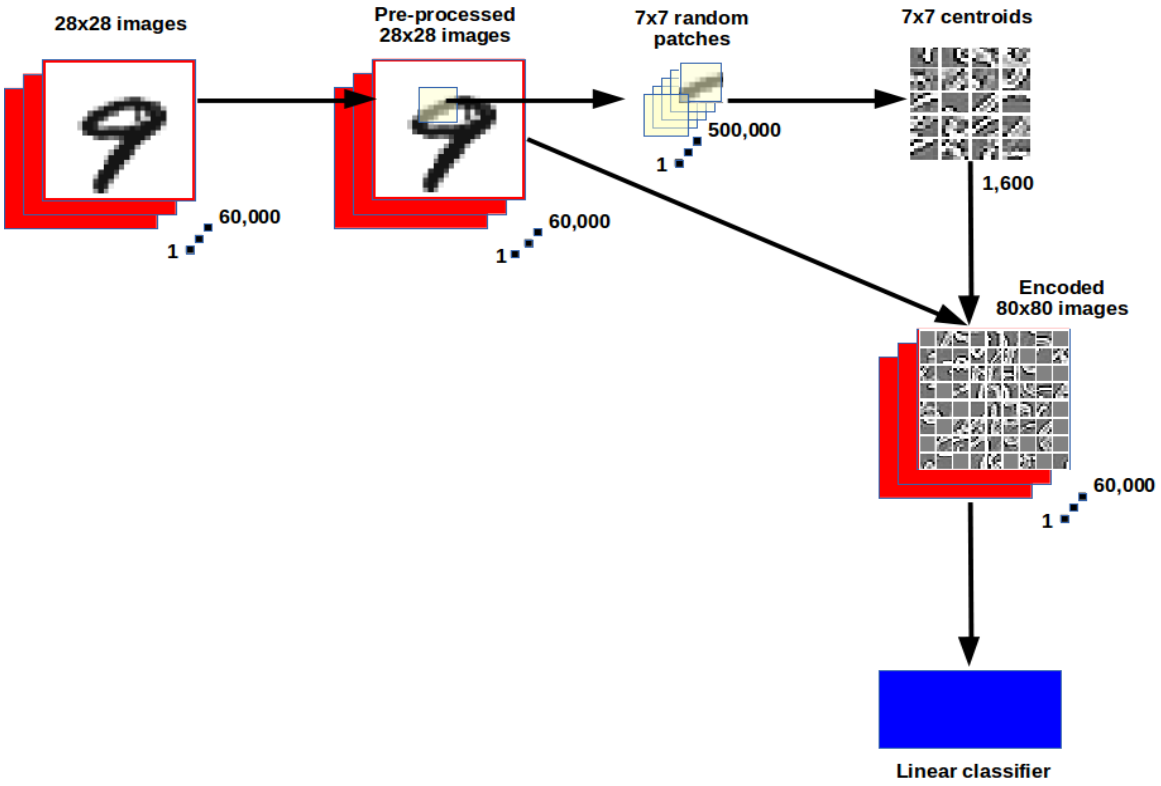

Fig. 1: Overview of the proposed architecture for the framework implementation

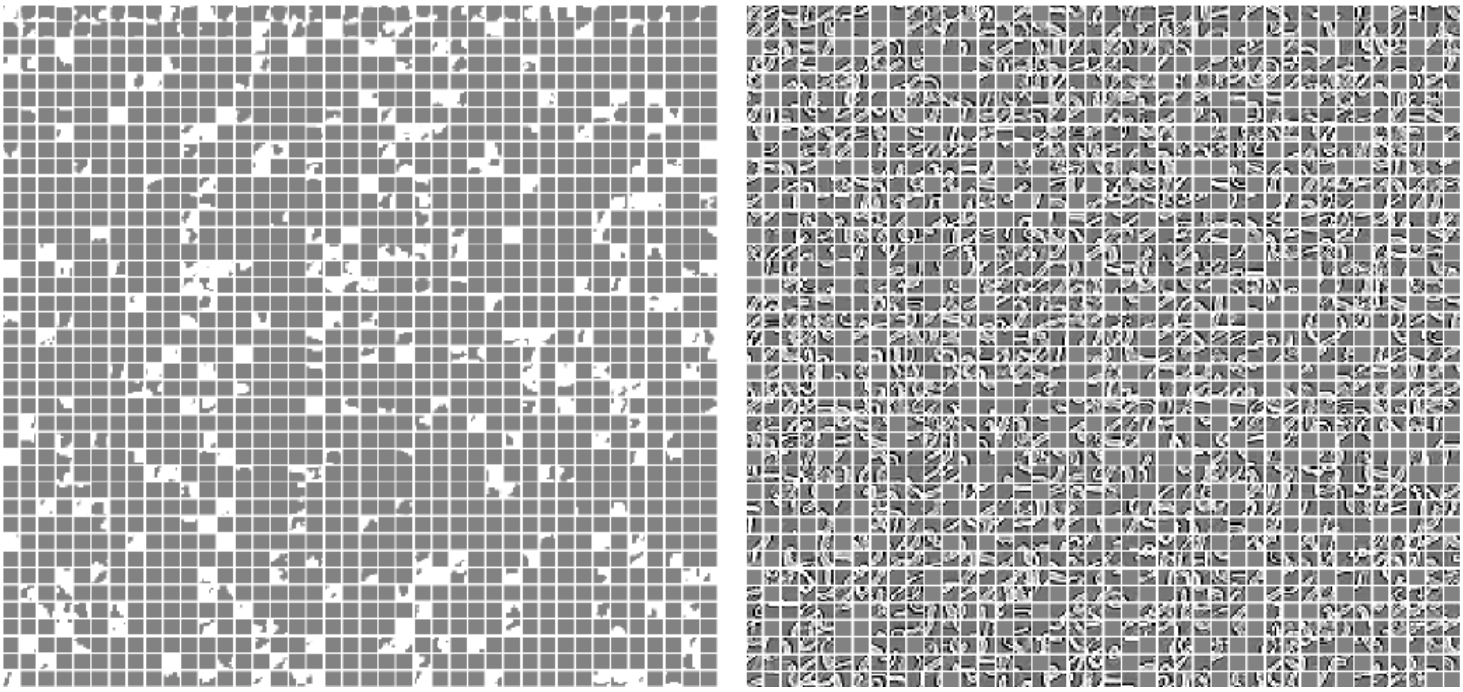

Fig. 2: 1,600 centroids learned from 500,000 7x7 pixel patches. Left: no pre-processing; Right: brightness and contrast normalization plus whitening

3. Tests and Results

The MNIST database of handwritten digits, used to test this framework, is composed by 60,000 labeled training images and 10,000 labeled test images. K-Means was tested using 500,000 7x7 patches extracted from all available training data set. The encoding and classification steps used 30% of both training and test sets, drawn randomly from their respective complete sets.

After running an SVM classifier using a standard linear kernel, I got the results depicted by the following confusion matrix:

				true labels						
prediction	0	1	2	3	4	5	6	7	8	9
0	**275**	0	0	1	0	0	2	1	0	2
1	0	**335**	1	0	0	0	0	2	1	0
2	0	1	**283**	4	0	1	0	0	3	0
3	0	0	1	**327**	0	1	1	3	1	0
4	0	1	1	0	**302**	0	5	0	3	2
5	3	0	2	7	0	**253**	3	0	2	0
6	1	0	0	0	0	3	**296**	0	0	0
7	0	1	1	0	0	0	0	**315**	2	1
8	2	0	0	1	0	4	0	0	**262**	2
9	0	1	0	0	3	0	0	4	0	**271**

This result corresponds to a test error rate of 2.7% , which is better than all results published for 2 or 3-layer neural networks trained without cross-entropy loss function and without deskewing pre-processing. It suggests that, if training the system with 100% of the data set, and applying deskewing to it, it could improve significantly.

The complete source code in R is available in [7].

4. Scalable Implementations

The implementation presented here is suitable for running in a commodity desktop computer in relative few time (few hours) for relative small dimensional data (1,000 to 3,000 pixels per image) and about 50,000 to 100,000 images. It was not possible, due to lack of enough computational resources, to apply this framework to more complex images, such as those used in [8].

In that case, a scalable implementation would be necessary. A possible approach for implement one would be to use a scale-out processing framework, such as Hadoop [9]. There are several ways to implement it on top of Hadoop, all using the MapReduce [10] programming model. An option for such an implementation in R would be to use the Hadoop Streaming framework, or several available products that allows for writing map and Reduce function in R code.

There are several processing blocks that would benefit from the distributed, parallel processing of such an implementation:

1. Data pre-processing: for both brightness and contrast normalization as well as whitening, the parallelization is straightforward. A Map-only Hadoop job can split the data matrix in several chunks, each chunk being processed independently and in parallel with each other.

2. Patch extraction: this is also straightforward. Again, a Map-only Hadoop job taking care of several independent chunks of the data matrix in parallel.

3. K-Means processing: there is a classic and also straightforward K-Means algorithm expressed in MapReduce: in the Map step, perform the centroid assignment to the data set. And in the Reduce step, compute new centroids and check for convergence.

4. Feature encoding: again, an straightforward implementation through a Map-only job that processes several chunks of the data matrix in parallel.

5. Liner classifier: an alternative to SVM which is easier to implement in MapReduce is Logistic Regression through Gradient Descent. In this implementation, several partial gradients would be computed in parallel, for independent chunks (mini-batch approach) of the data matrix. Then, a Reduce process would aggregate those partial gradients for the updating of the parameters.

5. Conclusion and Further Work

In this work, I've shown an working implementation, in R, of the framework proposed in [1] and [3] to construct a deep learning architecture for classification. As future work, it would be interesting to implement an scalable version to run on top of a Hadoop cluster that could be expanded to two or three layers of features. Such a framework could then be applied for more complex image classification tasks, such as [8].

References

[1] Coates, A., Ng, A.Y.: Learning Feature Representations with K-means, G. Montavon, G. B. Orr, K.-R. Mαuller (Eds.), Neural Networks: Tricks of the Trade, 2nd edn, Springer LNCS 7700, 2012

[2] Coates, A., Ng, A.Y.: Selecting receptive fields in deep networks. In: Advances in Neural Information Processing Systems 24. pp. 2528–2536 (2011)

[3] Coates, A., Lee, H., Ng, A.Y.: An analysis of single-layer networks in unsupervised feature learning. In: 14th International Conference on AI and Statistics. pp. 215–223 (2011)

[4] The MNIST database of handwritten digits: Yann LeCun, Courant Institute, NYU; Corinna Cortes, Google Labs, New York; Christopher J.C. Burges, Microsoft Research, Redmond
http://yann.lecun.com/exdb/mnist/

[5] http://cran.stat.unipd.it/web/packages/e1071/vignettes/svmdoc.pdf

[6] Kurt Hornik, Ingo Feinerer, Martin Kober, Christian Buchta (2012). Spherical k-Means Clustering. Journal of Statistical Software, 50(10), 1-22. URL http://www.jstatsoft.org/v50/i10/.

[7] https://github.com/vilcek/Image_Classification

[8] http://www.kaggle.com/c/dogs-vs-cats

[9] http://hadoop.apache.org/

[10] MapReduce: Simplified Data Processing on Large Clusters, by Jeffrey Dean and Sanjay Ghemawat; from Google Research