Object Detection using Convolutional Neural Networks

Shawn McCann
Stanford University
sgmccann@stanford.edu

Jim Reesman
Stanford University
jreesman@cs.stanford.edu

Abstract

We implement a set of neural networks and apply them to the problem of object classification using well-known datasets. Our best classification performance is achieved with a convolutional neural network using ZCA whitened grayscale images. We achieve good results as measured by Kaggle leaderboard ranking.

Introduction

NOTE: this project was originally conceived as a vehicle detection in roadway images problem. We were not successful in processing the raw dataset sufficiently to allow meaningful results using the techniques described in this paper. As a result, we've focused the paper on our successful implementation of neural networks for object detection using known-good datasets. On advice from Sammy, we've submitted the paper with the same title used for the milestone report for compatibility with the submission system, but have changed the title internally to reflect the modified scope.

Our primary interest in this project was to gain experience implementing deep learning techniques.

Deep learning refers, loosely, to representational techniques that learn features in layers, learning higher level features as functions of lower level features learned from data. Deep learning techniques are used in many problem areas, including object classification [1], speech recognition [2], and NLP [3].

We approached the project as a sequence of independent steps. We first implemented a simple multi-layer perceptron with features learned in one case by stacked autoencoders and in once case by stacked sparse autoencoders. Using the MNIST dataset, this allowed us to validate basic capabilities, our ability to get GPU-based code running on the computing systems available to us, and achieve an understanding of standard datasets.

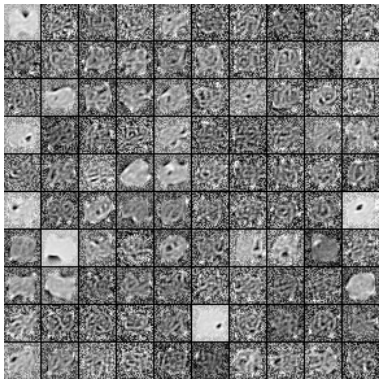
Then we moved to implementing a Convolutional Neural Network, and applying it to a multi-class classification problem using known datasets (MNIST, CIFAR-10). We submitted our results to two different Kaggle contests, and include our results.

Background and Related Work

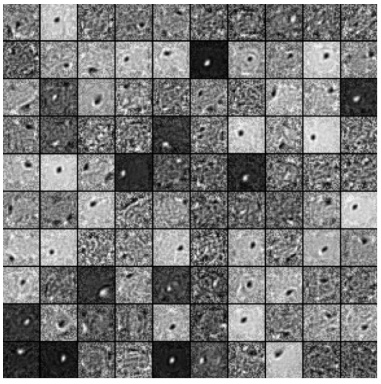
There are many examples of work on the general problem of object classification. For example, the Pascal Visual Objects Challenge (VOC) website lists 20 publications that use the VOC dataset specifically [4]. The classification of objects in small images using deep belief networks based on Restricted Boltzman Machines (RBMs) is discussed in [5]. The use of GPU systems to scale object detection performance is described in [6]. An analysis of feature representations is presented in [7]. Techniques for improving feature representations by *denoising* specifically for improving the performance of stacked autoencoders are discussed in [8]. Recent work showing very high performance uses Convolutional Neural Networks (CNN) for object classification [1].

Classification using the MNIST dataset

The first phase of the project focussed on developing a neural network classifier. We made use of the deeplearning.net tutorial and the Stanford UFLDL tutorial[9, 10], implemented a number of different network architectures and compared the results using the MNIST dataset[11].



(a) Features learned by the Stacked Autoencoder



(b) Features learned by the Denoising Autoencoder

Figure 1: Learned Features

The MNIST data consists of 50,000 training images, 10,000 validation images, and 10,000 test images. Each image is a 28x28 pixel grayscale image.

Multi-Layer Perceptron

For the Multi-Layer Perceptron, the initial architecture we tested consisted of an input layer of size 784 pixels (28x28), one hidden layer of size 500 units and an output layer of size 10 units (one for each digit type).

Autoencoders

Once the Multi-Layer Perceptron was working, the next step was to incorporate an autoencoder into the network to support feature learning. We used both denoising autoencoders and sparse autoencoders. In the case of sparse autoencoders, we used two techniques for imposing sparsity: restricting the size of the hidden layer (500 units) and implementing the sparsity parameter as described in the UFLDL tutorial (KL divergence).

Figure 1b shows an image of the features learned by the denoising autoencoder (using a corruption factor of 0.3). These results compare favourably with those in Vincent [8].

While Figure 1a shows an image of the features learned by the sparse autoencoder using a sparsity factor of 0.05.

For natural images, the tutorials indicate that we should expect to see the network learning edge detector filters. However, it is our understanding that in the case of a dataset like MNIST, it is more common to see brush stroke patterns emerging in the filters.

Multi-Layer Perceptron with Autoencoder

Our next step then combined the autoencoder with the multi-layer perceptron. This configuration used three hidden layers with 484, 400, and 324 units respectively. As before, the output was a 10 unit softmax layer.

Pre-training was done on each of the autoencoder layers to allow it to learn relevant features in an unsupervised manner. Once the pre-training was complete, a fine-tuning step was done to train the network using supervised learning.

The stacked autoencoder exhibits different performance based on layer sizes and noise levels shown in Table 1.

Table 1: Training Time and Test Error - MNIST data

Hidden Layer Sizes	Noise	System	Training Time	Test Error
500, 500, 500	[0.1, 0.2, 0.3]	corn	280 min	1.99%
500, 500, 500	[0.3, 0.3, 0.3]	AWS	229 min	1.58%
484, 400, 324	[0.3, 0.3, 0.3]	AWS	172 min	1.65%

Convolutional Network

As a final step, we switched to a convolutional network and tested that on the MNIST data. This network used four layers where layers 1 and 2 were are convolutional, max-pooling layers, and layers 3 and 4 form a multi-layer perceptron with 500 hidden units and 10 outputs. The details for the convolutional layers are given in Table 2.

Table 2: CNN architecture - MNIST data

Layer	Filter	Pooling	Feature Maps	Output Size
1	5x5	2x2	20	12x12
2	5x5	2x2	50	4x4

Results

Table 3 shows the training times experienced for each network. Note that the testing was done on the rye machines in the Stanford Computing Cluster using the GPU optimized theano library. Performance of the GPU optimized code was found to be an order of magnitude faster than running on the CPU (corn machines).

Table 3: Training Time

Model	Epochs	Training Time
MLP	1000	147 min
MLP + AE		
Pre-Training	14 for each layer	64 min
Training	14	21 min
CNN	200	33 min

Table 4 shows the testing error rates that were observed using the various networks.

Table 4: Performance - MNIST data

Model	Test Error
MLP	1.65%
MLP+AE	1.87%
CNN	0.92%

Kaggle Competition

As a test of our implementation, we wrote a custom classifier using the convolutional network and ran it on the dataset for the Kaggle Digit Recognizer competition. Our network scored 99.76% accuracy which ranked 3rd place on the leaderboard.

Classification using the CIFAR-10 dataset

Once we had the convolutional network working on the MNIST dataset, the next step was to adapt it to work with imagery from the CIFAR-10 dataset. Examples of the CIFAR-10 images are shown in Figure 2.

Since the CIFAR-10 data contains color images, whereas the MNIST images were grayscale, we converted the CIFAR images to grayscale for use with the convolutional network. The initial results with no pre-processing gave us an accuracy of around 55% (45% error rate) and we see that the network learned a set of edge detector filters as expected.

Preprocessing

To improve the results, we implemented ZCA whitening as described in the UFLDL tutorial. Rerunning the tests with this modification improved the results to 65% accuracy (35% error)

Examples of the conversion from RGB to grayscale images, and from grayscale to ZCA whitened images can be seen in Figure 3.

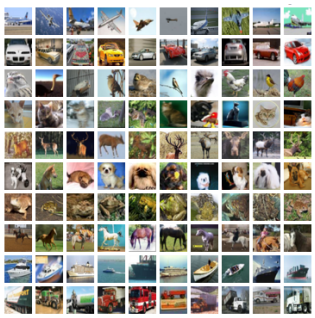


Figure 2: CIFAR-10 image examples

However, one challenge that we found with the preprocessing was that the covariance matrix Σ was computed on a batch of images instead of a single image. This caused difficulties when testing the network as we were testing with smaller batch sizes (100 images) than was used during training (10,000 images) which changed the covariance matrix and resulted in degraded performance. Our solution was to use the same batch size for both training and testing, however further work is required to better understand the whitening process and it's associated constraints.

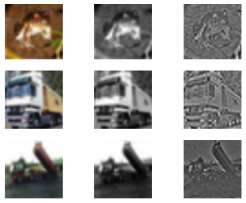


Figure 3: Image Preprocessing

Extending to multiple channels

To improve the results further, we extended the network to work with multiple channels so as to avoid having to convert the RGB images to grayscale. However, testing of this configuration did not result in any improvement of the results. Upon investigation, it was found that only a small number of useful filters were being learned by the network (see Figure 4) . Various experiments were run to try and improve the filter learning (different number of feature maps, different filter sizes), but they all proved ineffective.

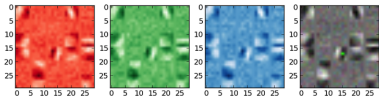


Figure 4: Features learned by RGB CNN

Results

Table 5 shows the validation and test error achieved with these models.

Table 5: CNN performance - CIFAR-10 data

Input	Validation Error	Test Error
Grayscale	47.00%	47.02%
Grayscale, whitened	34.65%	35.85%

We initially configured the training to run for 200 epochs, but eventually cut off training at 80 epochs. The error didn't meaninfully decrease beyond 40 epochs.

Figure 5 shows the confusion matrix resulting from testing with the grayscale CIFAR-10 images. The confusion matrix is consistent with our intuition, showing that a common confusion is between Auto and Truck, and for example Autos are never confused with Birds or Dogs.

Kaggle Competition

As a test of our convolutional network, we ran it on the dataset for the Kaggle CIFAR-10 competition and scored 61.16%, which put us in 12th place on the leaderboard.

	Plane	Auto	Bird	Cat	Deer	Dog	Frog	Horse	Ship	Truck	Total	Hit	Miss	Miss Ratio
Plane	63	3	7	6	4	0	2	1	11	3	100	63	37	37.0%
Auto	0	59	1	2	0	1	3	1	6	9	82	59	23	28.0%
Bird	6	0	55	12	6	7	8	4	2	2	102	55	47	46.1%
Cat	6	3	8	48	6	11	5	6	0	2	95	48	47	49.5%
Deer	3	0	3	10	51	5	4	7	1	1	85	51	34	40.0%
Dog	3	0	16	11	5	47	5	9	0	1	97	47	50	51.5%
Frog	2	6	6	5	3	6	82	2	2	3	117	82	35	29.9%
Horse	3	0	0	7	12	8	3	71	0	1	105	71	34	32.4%
Ship	9	4	4	0	2	0	0	1	79	6	105	79	26	24.8%
Truck	8	14	0	2	1	1	0	0	5	81	112	81	31	27.7%

Figure 5: CIFAR-10 CNN Confusion Matrix

Conclusions

In this project we successfully implemented several neural network architectures, and applied them to the problem of object classification in images. While we did not achieve the full results we had originally hoped for, we did successfully implement multiple neural network architectures (including CNN) and successfully applied them to object classification problems with good results.

Our primary results are the Kaggle contest results, shown in Table 6.

Table 6: Kaggle Contest Results

Kaggle Contest	Accuracy	Place
Digit Recognizer	99.76%	3^{rd}
CIFAR-10	61.16%	12^{th}

References

[1] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *arXiv preprint arXiv:1311.2524*, 2013.

[2] George E Dahl, Dong Yu, Li Deng, and Alex Acero. Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition. *Audio, Speech, and Language Processing, IEEE Transactions on*, 20(1):30–42, 2012.

[3] Richard Socher, Yoshua Bengio, and Christopher D Manning. Deep learning for nlp (without magic). In *Tutorial Abstracts of ACL 2012*, pages 5–5. Association for Computational Linguistics, 2012.

[4] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman. The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results. http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

[5] Alex Krizhevsky. Learning multiple layers of features from tiny images. Master's thesis, University of Toronto.

[6] Adam Coates, Paul Baumstarck, Quoc Le, and Andrew Y Ng. Scalable learning for object detection with gpu hardware. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 4287–4293. IEEE, 2009.

[7] Adam Coates, Andrew Y Ng, and Honglak Lee. An analysis of single-layer networks in unsupervised feature learning. In *International Conference on Artificial Intelligence and Statistics*, pages 215–223, 2011.

[8] Pascal Vincent, Hugo Larochelle, Yoshua Bengio, and Pierre-Antoine Manzagol. Extracting and composing robust features with denoising autoencoders. In *Proceedings of the 25th international conference on Machine learning*, pages 1096–1103. ACM, 2008.

[9] Theano Development Team. Deep learning tutorials. http://deeplearning.net/tutorial/contents.html.

[10] A. Ng, J. Ngiam, C.Y. Foo, Y. Mai, and C. Suen. Ufldl tutorial. http://ufldl.stanford.edu/wiki/index.php.

[11] Yann LeCun, Corinna Cortes, and Christopher J.C. Burges. The mnist database of handwritten digits. http://yann.lecun.com/exdb/mnist/index.html.