6.867 Final Project: An overview of convolutional neural networks with the SUN397 scene recognition database

Michele Pratusevich

December 5, 2014

1 Introduction

Collaboration between artificial intelligence, neuroscience, and machine learning resulted in the invention of neural networks for use in classification or prediction tasks. With roots from single-layer perceptrons [14] to multi-layer and kernel perceptrons [2], increasing success on non-linear classification tasks (especially related to vision or images) have seen the revival of their popularity among the computer vision community.

A popular success story is the development of a back-propagation convolutional neural network (CNN) for handwritten digit recognition [10]. The original data used was approximately 10,000 handwritten digits, classified into one of 10 categories (one category for each digit). This original dataset was later expanded into the MNIST handwritten digit database with over 60,000 examples [11], [13]. The network used by LeCunn et. al. used the idea of convolution layers - described essentially as features extracted from parts of the image by kernels of varying sizes, generally getting smaller and smaller. The general idea is that these $n \times n$ kernels will 'find' specific features in the image. Each kernel is basically a linear combination of the pixels in it's 'receptive field' in some way. In between, there are layers that perform averaging or weighting of the features that it receives. The final layer is fully-connected to the receptive fields of the previous layer and outputs a probability estimate for the class the image belongs to, meaning it essentially performs a weighted sum of all the inputs it sees for each parameter, with different weight parameters for each predicted class. Improvements to this network have been made incrementally over time, but the error rates on the test set for this initial network was 3.7 percent.

The interesting insights from LeCun's simplified architecture [10] compared to a previous network that had more connections (also designed by LeCun) [9] were two-fold: (1) that images could be fed directly into the neural network rather than extracting features, then feeding the features into the network, and (2) knowing a-priori the nature of the classification task can give hints about how to simplify the network to contain fewer parameters to optimize over. In general, these ideas of simplification of neural networks can be applied to larger and more complex neural networks [12].

Since LeCun's successful use of CNNs for handwritten digit classification, CNNs became a popular technique for completing computer vision tasks. Krizhevsky et. al. [8] constructed an 8-layer CNN that achieved 17 percent top-5 error rates on the ImageNet database [15], taking advantage of optimizations using GPUs and a few other numerical improvements cited in the paper. Since then, very deep CNNs have been used for nearly every computer vision task with a standard dataset, and countless other non-standardized tasks. Since the original AlexNet was published, fine-tunings of the original network, in addition to changes to the architecture, were performed with the hopes of creating a neural network that can be used as a generalized feature extractor for all kinds of image tasks [5]. Software packages to facilitate this and the use of GPUs for speed-up were created [7].

One big problem with using these extremely deep CNNs for feature extraction, fine-tuning, and general image classification tasks is the time needed to train these networks. With GPU speedups and optimized code, this task is made easier, but it still takes approximately 3 days on a GPU to train on the entirety of the ImageNet database (6 million images). Fine-tuning for a specific task using a pre-trained network takes less time, but depending on the desired result takes a few hours. Another big problem with deep

CNNs for vision tasks is the need for extremely large quantities of labeled training data. For a large network with 10000s of parameters, having only 50 labeled training images is not enough to optimize over. AlexNet was only able to train from scratch because the training set for ImageNet is incredibly large, with approximately 10,000 images per class and a total of 1000 classes.

Training neural networks still requires the fine-tuning of hyperparameters dictating the learning rate and the weight decay, but some have offered suggestions for how to improve back-propagation and the choosing of hyperparameters [4]. However, simultaneously, there have been a number of papers questioning the need for the depth and complexity of the recently-used CNNs [3]. LeCun provided suggestions for the simplification of neural networks when working with the original handwritten digit classification network [12]. Another interesting area of research has been the idea of using Gaussian processes to optimize parameters in a neural network for a specific task.

In this project, I will explore the idea of simplified CNNs for the scene recognition vision task, drawing on ideas from recent literature about the effectiveness of using pre-trained networks, simplifying existing networks, and using extracted features to do various tasks. The main concepts I will explore are:

1. The effect of network size on training time
2. The effect of kernel size on the learned receptive field and on training time
3. The process of choosing hyperparameters for learning

The main guiding this exploration is: can you construct a significantly simpler neural network for scene classification that has comparable results to an extremely deep neural network? Decisions in training and parameter tuning will be made to create the smallest network that can be trained in a reasonable amount of time (say 2 hours) to achieve positive results. A general answer to this question is a philosophical question in the field of computer vision, since it touches on the question of whether a highly-specific system that is quick to construct or a highly-general system that is versatile a better outcome. I claim that it depends on the task, but the idea of constructing a simple neural network for complex tasks like scene classification has not been addressed in the literature.

2 Dataset, Tools, and Related Work

The task of scene recognition is different from object recognition in that scene recognition relies on clues from the entire image and the general layout rather than specific qualities of objects. The standard dataset for scene recognition is the SUN397 database [17], which has 397 scene categories, with at least 100 images in both categories. The standard train / test splits have 50 images per category in both training and testing. The benchmark classification results on this dataset using hand-crafted features is reported by Xiao, et. al. [17], [16]. To build on the SUN397 database is the PLACES database [18], which benchmarks itself against SUN397, and notably uses CNNs for scene classification, rather than hand-crafted features. Because SUN397 has a manageable amount of data to use for training, I will use that database to do tests, benchmarking against the classification rates reported in the SUN papers. The PLACES database and results will provide a second state-of-the-art benchmark for scene classification as well.

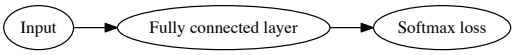
To construct the neural networks and take advantage of GPU capabilities, I will use the Caffe software package [7], optimized for GPU performance and slowly becoming the standard in academic work for building CNNs. There are bindings for C++, Python, and MATLAB, allowing for the use of various interfaces for analysis. The added benefit of using this system is that the PLACES database provides a trained CNN model that can be used for fine-tuning with SUN. Additionally, the Vision Group has a cluster of GPU machines that can effectively run Caffe with some persuasion, so the training times for the networks will be drastically improved over running on a small laptop PC CPU. One of the bottleneck steps in using training data is reading data from disk, so the training and testing sets are converted into LMDB databases optimized for multi-threaded querying. All images are resized to 227 x 227 pixels, in accordance with comparable computer vision experiments. The reason for the small size in images

is three-fold: (1) consistent higher-resolution images were only available as standard datasets relatively recently; (2) it has been shown in prior experiments from both neuroscience and computer vision that image-level features do not change significantly when shrunk to this size [5], and (3) smaller images mean performance increases when reading from disk and computing features over the entire image.

3 Method

The first step in the neural network analysis is to set up Caffe to run on both my personal computer (for small-scale testing and development) and the Vision Cluster (for running experiments). Both proved to be a challenging task because of the need to link highly optimized numerical libraries, resulting in my writing an addendum for how to install Caffe on OSX 10.9 on the Caffe Github wiki.

To set up a single network in Caffe is complicated without knowing the interface, so as a benchmark to start, I constructed a basic logistic regression classifier as a neural network. In the default neural network, activation is computed using the sigmoid function and the loss function is the softmax function.



(a) Simplified neural network architecture for logistic regression

(b) General layer structure of a convolutional neural network

The setup for (linear) logistic regression as a neural network is then represented as shown in Figure 1a. A fully-connected layer is equivalent to calculating weighted sum of dot products between the inputs to the layer, and the softmax function takes these 397 functions and calculates probabilities between them. There are as many layers in the fully-connected network as there are training images, but there are only 397 outputs, meaning there are 397 linear-combinations of functions calculated using weights from the training images. Using stochastic gradient descent to compute the parameters, the optimal learning weight (by grid search and suggestions from Bottou [4]) was found to be approximately 1×10^{-8}. On a GPU for 20000 iterations, this takes approximately 15 minutes and results in a top-1 accuracy prediction of 4 percent and a top-5 accuracy prediction of 13 percent. Because simple logistic regression learns relatively few parameters and is only capable of linear combinations of inputs, the idea is that adding any sort of more complicated network will increase this base classification rate. Methods involving more complicated networks were only considered if they yielded results better than this baseline 13 percent top-5 accuracy with logistic regression.

Inputs to all tests scaled the image down to 227 by 227 pixels, consistent with the format proposed by AlexNet and used by competitors in the Large-Scale Visual Recognition Challenge [8], [15]. The mean image of the training set was calculated and subtracted from all the images used (both training and test), effectively centering the data.

The standard metric for measuring success against the SUN database is the top-1 classification accuracy rate for correctly predicting with maximum likelihood the correct class. However, because many categories in the SUN397 database are similar (like for example 'building exterior' and 'apartment building outdoor'), I will also report a top-5 accuracy rate.

After exploring this pipeline using Caffe, for each set of experiments I had to do the following steps:

1. Define the network representation

2. Define the solver parameters
3. Define the validation / hyperparameter search procedure
4. Train the network on the GPU using the training set
5. Test the network on the GPU using the test set

For all experiments, the same training and test sets were used.

4 Experiments

Because I was limited by computational power and time, for each experiment I chose a few token representations and ran tests on those. The next section will summarize the findings as general conclusions, while here I will report the results and give some observations for specific experiments. A summary of the results of the experiments with best results are given in Table 1, along with a comparison of state-of-the-art results. Note that the best result reported in the PLACES-CNN paper [18] does not use fine-tuning, but rather features extracted using the CNN fed through an SVM.

Procedure	Top-1 Accuracy	Top-5 Accuracy
Logistic regression	4 %	14 %
Best single-layer network	6.7 %	18.2 %
Best two-layer network	12.6 %	31.1 %
Fine-tuned PLACES CNN	51.3 %	80.1 %
PLACES CNN best SUN397 result	54.3 %	not reported

Table 1: Summary of top-1 and top-5 accuracy results on the test set

There is a significant gap in the performance of smaller networks trained on fewer images than the fine-tuned PLACES network that was originally trained on 6 million auxiliary images.

4.1 Single-layer networks of varying kernel size

For the first set of experiments, single-layer networks with varying kernel size were used. A convolutional layer in a neural network in the context of a computer vision task is similar to training a 'receptive field' of an eye. The idea is that some number of $n \times m$ kernels will be computed over the entire image, acting like 'filters', then passed to higher layers in the network. In the single-layer network case, the output of the kernel filters are then passed to a fully-connected layer that does a weighted sum of the filters, then passed to a softmax function to predict the probabilities.

There were a number of numerical issues when approaching the convolution layer problem. For one, having a large number of output filters means that there are an incredibly large vector of numbers passed to the fully-connected layer of the network. Having too many parameters in the vector many means that there is a strong chance that there will not be enough memory to pass these variables around to optimize. Therefore, we use our knowledge of the domain space and of kernels to add an additional layer to the network that both improves performance and decreases training time. Instead of passing all the computed kernels for the entire image to the fully-connected layer, only pass the kernel result that maximally fires for that particular image. Based on how many of the calculated kernels you want to pass to the next layer, this technique constructs a 'feature map' for the image that is passed to the next layer. This is done in Caffe with 'max-pooling'. In general, the structure of networks used is shown in Figure 1b.

The kernels that were tested were 8×8, 12×12, and 20×20. AlexNet, PlacesCNN, LeNet, and many other successful CNNs for image tasks use 12×12 kernels in their first convolutional layer. The kernel that performed the best for a single layer was the 20×20 kernel. In all cases, there were a total of 96 kernels computed and a max-pooling layer was then fed into the fully-connected layer. Results for all single-kernel experiments are given in Table 2.

Procedure	Top-1 Accuracy	Iterations	Top-5 Accuracy
96 8×8 kernels, no dropout	50000	4.8 %	14.8 %
96 8×8 kernels, with dropout	20000	5.7 %	15.2 %
96 8×8 kernels, with dropout and RELU	50000	.8 %	2 %
96 12×12 kernels, no dropout	20000	4.4 %	13.5 %
96 12×12 kernels, with dropout	50000	5.5 %	15.8 %
96 20×20 kernels, with dropout	20000	6.7 %	18.2 %
30 20×20 kernels, with dropout	20000	6.5 %	17.9 %

Table 2: Summary of top-1 and top-5 accuracy results on single-layer networks after 20000 iterations

4.2 Single-layer networks with rectifying linear units

It has often been cited in successful CNN papers like AlexNet or LeNet [8], [10] that instead of using a sigmoid activation function that a rectified linear function of the form $\max(0, x)$ leads to faster training and improved results, but in 20000 iterations of training on multiple single-layer networks like 8×8 and 12×12, the training took approximately double the time than similar networks using sigmoid activation functions and yielded poorer results. So for the rest of the tests, using rectified linear units were not used.

4.3 Using dropout

Experiments were also conducted for single-layer networks using dropout layers. It was cited by AlexNet [8] that dropout layers reduced overfitting. At the output of every convolutional layer, a dropout layer that takes the sample and with probability 0.5 includes it in the resulting output vector. This is equivalent to doing averaging over a number of models to produce a final result. From Table 2, it is clear that dropout layers improve the performance without changing the hyperparameters. Therefore, with all further experiments, dropout layers will be added after every convolutional layer to prevent overfitting. It is especially important to do this when there is not a lot of labeled training data, as with the SUN397 database setup we are using, where there are only 50 labeled images per class.

4.4 Multi-layer networks

After the hyperparameters were roughly optimized for single-layer networks, two-layer networks of varying sizes were tried. An additional variable in network design is the decision for how many outputs from each convolutional layer to include. Because each new output adds an additional set of parameters for the network to optimize, the hypothesis was that smaller numbers of outputs of correctly-chosen output sizes would be both more effective during testing and faster to train. A summary of two-layer results is given in Table 3.

Procedure	Iterations	Top-1 Accuracy	Top-5 Accuracy
30 12×12 kernels, 30 6×6 kernels	30000	10.0 %	25.24 %
30 20×20 kernels, 30 6×6 kernels	50000	7.5 %	21.4 %
30 12×12 kernels, 30 3×3 kernels	100000	6.1 %	18.9 %
30 20×20 kernels, 30 3×3 kernels	50000	5.1 %	16.3 %
60 20×20 kernels, 30 6×6 kernels	20000	9.6 %	26.0 %
96 12×12 kernels, 30 6×6 kernels	30000	12.6%	31.1%

Table 3: Summary of best top-1 and top-5 accuracy results on two-layer networks

In addition to tuning a few 2-layer network, one 3-layer network was tuned for 50000 iterations, with 30 12×12 filters, then 30 6×6 filters, then 30 3×3 filters, with pooling and dropout layers after each convolutional layer. However, there was most likely something wrong with the solver parameters or there

was not enough training done, since the top-1 accuracy for the 3-layer network was 1.6 percent and the top-5 accuracy was 7.4 percent. There was not enough time to do more 3-layer network experiments, but if there was time, training or grid search with more parameters would have been tried.

4.5 Fine-tuning on PLACES-CNN

To both be consistent with state-of-the-art algorithms for classification and reproduce results from the PLACES database paper [18], I took the PLACES-CNN model that was developed and distributed with the paper and fine-tuned it to classify the SUN397 database. The procedure for fine-tuning is to take the weights from a pre-trained network that was trained on an auxiliary database, changing the final softmax layer to a layer that predicts the correct number of classes for the SUN397 task, and running approximately 20000 iterations. This PLACES database network has a total of 8 layers, 3 of them fully connected and the first 5 convolutional layers with filters of varying sizes. There are over a million parameters in this model, and the original one takes 7 days to train on a GPU from scratch for all of the PLACES database. After fine-tuning, the results are astounding in how much more accurate it is than simple networks - comparable to the best performance on SUN397 ever. There was a top-1 accuracy of 51.3 percent and a top-5 accuracy of 80.1 percent.

5 Conclusions

5.1 Network size and training time

The training time for two-layer networks that are smaller or comparable in size to single-layer networks take a modest amount of time to train - in 40 minutes, a 60-neuron network can be trained, with approximately 100000 parameters between the various kernels. When the network grows in size, like the 96-filter 12×12 kernel followed by 30 6×6 kernels takes approximately 2 hours to train.

The size dimension that is most important is the number of parameters in the network rather than the number of layers or the number of kernels. Adding layers like a max-pooling layer gives a lot more flexibility in the gradients of various kernels, therefore causing a shorter training time. Larger kernel computations have more parameters, therefore taking longer to train.

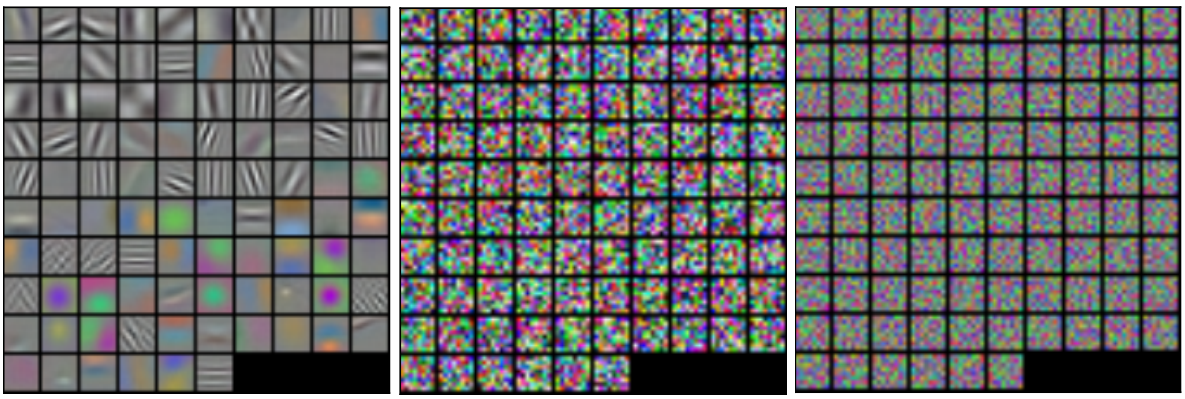
5.2 Kernel size and receptive fields

Looking at the first-layer neurons in the single-layer and multi-layer networks (including the fine-tuned PLACES CNN), it is clear that the neurons correspond to specific structures or filters in the image, as shown in Figure 2a. However, for smaller networks and (relatively) low numbers of training iterations, this is not the case. We can see the effect of adding a dropout clearly in Figures 2b and 2c, where one is darker than the other (corresponding to when images are excluded from the training set, meaning they are black).

Even looking at the best-performing two-layer network (which had 12.6 percent top-1 accuracy), the neurons in the first layer do not resemble any kind of human-readable structure. There are a few neurons that are starting to resemble structure with clusterings of color or pattern, as shown in Figure 3c, but it is a stretch. In fact, they look similar (and similarly unstructured) to the first-layer neurons in a similar single-layer 12 by 12 kernel network, as seen in Figures 3a and 3b.

5.3 Choosing hyperparameters

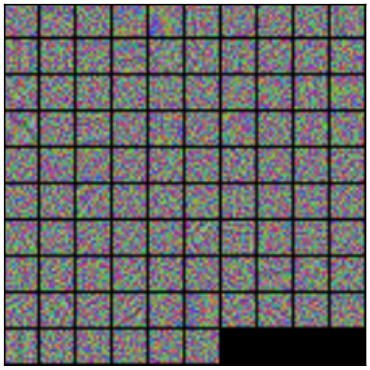
To start the hyperparameter optimization process, the parameters most commonly used in training AlexNet and recommended by Caffe [7], [8] were used. However, because those networks were almost 10 orders of magnitude in the number of parameters than the small networks used in this project, the learning rate for training had to be significantly decreased for numerical stability. It was also recommended [4], [1] to include a dropoff factor of 10 for the learning rate to prevent overfitting and numerical instability later in the process, so this was followed and not played with too much. Another important aspect



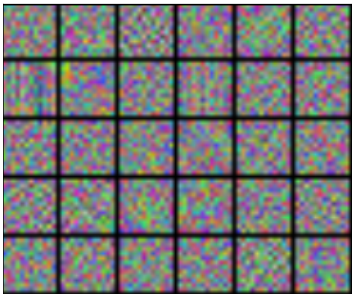
(a) Layer 1 neurons in fine-tuned PLACES network

(b) Neurons in single-layer 8 by 8 kernel network without a dropout layer

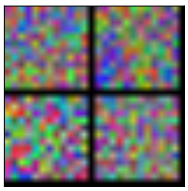
(c) Neurons in single-layer 8 by 8 kernel network with a dropout layer



(a) First-layer neurons in the best-performing two-layer network



(b) First-layer neurons in a 12x12 single-layer network



(c) Two neurons showing some color or pattern structure

of using stochastic gradient descent is initializing the weights. While AlexNet uses Gaussian weight initialization [8], using the xavier algorithm for determining the initialization of weights based on how many parameters there are in the layer [6]. This led to better performance results, so was used instead of Gaussian weights for all the initialization.

The method I used for choosing parameters was primarily grid search, mainly varying learning rate and dropoff rate. However, using a Gaussian process would have been an easier process than spending time running experiments that did not yield good results.

One potential improvement in choosing hyperparameters like network and kernel size is knowledge of the domain space. The experiments and kernel sizes chosen for this project were chosen based on knowledge about scene recognition. Since the input images were scaled to 227 by 227 pixels, the initial kernels were chosen to be large, to cover larger features in the space. Because scene recognition essentially analyzes larger features in the image, larger kernels tended to perform better at all network types, seen in Tables 2 and 3. It would have also been easier to use something like a Gaussian process to estimate various parameters in the network, but this is later work that can be done with the preliminary results in this project.

Another suggestion from the AlexNet experiments [8] was 'fine-tune' a network rather than tune it from scratch. In other words, if your network shares any structures in common from networks that have already been tuned, use those weights to initialize the layers in the network that are shared. This can be seen with the discussion of fine-tuning the PLACES architecture in Section 4.5.

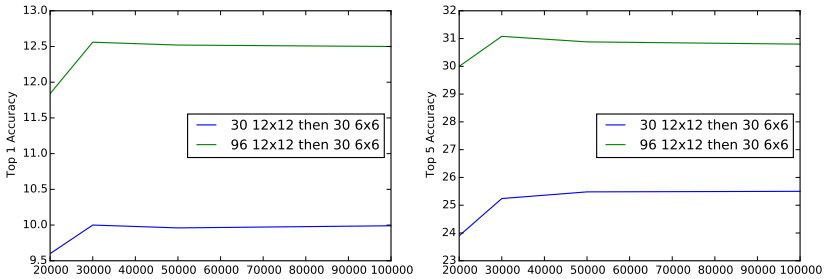
In future work, it would be interesting to construct a Gaussian process model with various network parameters correlated with the number of network parameters to automate or improve the hyperparameter

estimation and network design process.

5.4 Mitigating Overfitting

Additionally, larger networks were less susceptible to overfitting. For example, the 8×8 single-layer kernel network with 96 filters after 50000 iterations had decreased the training set loss to 0.5, but on the test set showed loss values closer to 6. Networks with more parameters tend to overfit less, and also using strategies like dropout mitigated this risk. However, the same 8×8 network from above with a dropout layer after the convolutional layer after only 20000 iterations showed a nearly 0 loss on the training set, but reported a loss of over 34 on the test set. The results were 15 percent top-5 accuracy despite the large test loss, but it is clear that there was an overfitting of the training data. This could have been caused by either a lack of training data, or trying to fit too many parameters in the model.

Sometimes overfitting also happens with a large number of iterations. Training the same small network for 100000 iterations, it is easy to test at various points and see where the error on the test set levels out. In Figure 4a, it is clear that both the top-1 and top-5 accuracy reach a peak at some point (in both of these cases, approximately 30000 iterations) and then level off or even decrease slightly. The slight decrease might be a result of the overfitting, or of the slightly randomized nature of running the accuracy test. It will save both training time and accuracy if training is stopped once the network is 'saturated', or does not change in accuracy on the test set.



(a) Top-1 and Top-5 accuracy over iterations

5.5 Overall

In all, CNNs are very powerful for computer vision tasks, and much more comprehensive work needs to be done in optimizing both shapes of networks and hyperparameter optimization to make neural networks even more powerful than they are today. As technology currently stands, it seems more effective for specific vision tasks to fine-tune a pre-trained network than re-train a simple network to do the same task, since the results are dramatically better.

References

[1] Pulkit Agrawal, Ross Girshick, and Jitendra Malik. Analyzing the performance of multilayer neural networks for object recognition. In *ECCV*, pages 329–344. Springer, 2014.

[2] Mark A Aizerman, E Braverman, and L Rozonoer. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, 25:821–837, 1964.

[3] Lei Jimmy Ba and Rich Caurana. Do deep nets really need to be deep? *arXiv preprint arXiv:1312.6184*, 2013. Eventually in NIPS.

[4] Lon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010*, pages 177–186. Springer, 2010.

[5] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *Proceedings of the International Conference on Machine Learning*, Beijing, China, 2014.

[6] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *International Conference on Artificial Intelligence and Statistics*, pages 249–256, 2010.

[7] Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

[8] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[9] Yann LeCun, B. Boser, J. S. Denker, D. Henderson, and L. D. Jackel. Backpropagation applied to handwritten zip code recognition y. LeCun. *Neural Computation*, 1(4):541–551, 1989.

[10] Yann LeCun, B Boser, J.S. Denker, D. Henderson, R.E. Howard, W. Hubbard, and L.D. Jackel. Handwritten digit recognition with a back-propagation network. *NIPS*, 1990.

[11] Yann LeCun, Lon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[12] Yann LeCun, John S. Denker, Sara A. Solla, Richard E. Howard, and Lawrence D. Jackel. Optimal brain damage. In *NIPS*, volume 2, pages 598–605, 1989.

[13] Li Deng. The MNIST database of handwritten digit images for machine learning research [best of the web]. *IEEE Signal Processing Magazine*, 29(6):141–142, November 2012.

[14] Frank Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological review*, 65(6):386, 1958.

[15] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, and others. ImageNet large scale visual recognition challenge (ILSVRC). *arXiv preprint arXiv:1409.0575*, 2014.

[16] Jianxiong Xiao, Krista A. Ehinger, James Hays, Antonio Torralba, and Aude Oliva. SUN database: Exploring a large collection of scene categories. *International Journal of Computer Vision*, August 2014.

[17] Jianxiong Xiao, James Hays, Krista A. Ehinger, Aude Oliva, and Antonio Torralba. Sun database: Large-scale scene recognition from abbey to zoo. In *Computer vision and pattern recognition (CVPR), 2010 IEEE conference on*, pages 3485–3492. IEEE, 2010.

[18] Bolei Zhou, Agata Lapedriza, Jianxiong Xiao, Antonio Torralba, and Aude Oliva. Learning deep features for scene recognition using places database. In *NIPS*, 2014.