Building on ILSVRC2012-Winning AlexNet for Tiny ImageNet

Benjamin Au
Stanford University
bau227@stanford.edu

Abstract

We review the state-of-the-art algorithms in convolutional neural networks for computer vision object recognition. In particular, we implement an 9-layer convolutional neural network inspired by Krichevsky's ILSVRC2012-winning CNN. We compare the two networks on Tiny Imagenet, a small slice 10% the size of the official ImageNet database. We leverage Caffe, an open-source convolutional network library to assist in network implementation. Our CNN achieves top-1 error rate of 69.6%..

1. Introduction

Khrichevsky et al revolutionize the field of computer vision object recognition research with an implementation of convolutional neural networks, achieving a 15.3% top-5 error rate at the ILSVRC2012 competition, 10% better than peer algorithms [1]. Since the inception of Khrichevsky's Convolutional Net debut at the ILSVRC2012 computer vision competition, advancements in convolutional neural networks have rapidly pushed the boundaries of performance in computer vision object recognition. These advancements have been caused in part from larger datasets datasets to train from, as well as better hardware to build larger nets, but also, and perhaps more importantly, due to innovation in algorithms and models that provide more efficiently-trainable and better performing machine learning algorithms. Recently, researcher's at Google have pushed computer vision to similar to human vision performance, reaching a 6.67% 5-attempt classification error rate on the ImageNet dataset [2].

We build on Khrichevsky's AlexNet, and optimize it based on our smaller dataset,, the Tiny Imagenet. We leverage Caffe, an open-source convolutional neural network library, written in C++, optimized for GPU-based computation based on the CUDA library, and modularized to efficiently develop neural network models [3]. We trained our models on a single system consisting of 10GB Ram, 5GB GPU and 1500 cores at 800MHz.

Please follow the steps outlined below when submitting your manuscript to the IEEE Computer Society Press. This style guide now has several important modifications (for example, you are no longer warned against the use of sellotape to attach your artwork to the paper), so all authors should read this new version.

1.1. Dataset

As a training dataset, we use the Tiny Imagenet, a subset of the larger Imagenet. ImageNet is a database of 1.2 million images, partitioned equally into 1000 classes of images. The Tiny ImageNet, in comparison is a database of 110,000 images, partitioned equally into 200 classes of images. Of this, we separate the Tiny Imageset into a training set of 100,000 images, and validation and testing sets of 5,000 images each.

Given the variable resolution of images, we down-sampled images into a fixed resolution of 227 X 227 pixels. In addition we performed mean-subtraction on each pixel of images, calculated based on the training set. Finally we performed data augmentation by random mirror-image flipping.

1.2. Background Concepts

ReLU: Rectified linear units achieve non-linearity through the equation max(0, x). Because they do not saturate unlike TanH and sigmoid nonlinearities, they significantly reduce time to train a neural network.

Fully-connected layer: The standard neural network layer, in which each neural in the layer is connected to every neuron in the previous layer.

Convolutional Layers: Convolutional layers share weights across all neurons in a filter. Each neuron in a filter takes as input, a local region from the previous region, in our case 3x3 and 5x5 height and width field size, along with the entire depth of the previous layer. Convolution layers allow the learning of localize field information, while massively reducing dimensionality.

Local Response Normalization: While ReLU neurons do not saturate, Khrichevsky et al showed that applying Local Response Normalization after convolutional

ReLU's still boosted generalization and performance [1]. As such, we apply LRN and find similar modest (~2%) boost in performance. LRN is calculated as follows: if a(i, x, y) is the activation of a neuron in kernel i at position (x, y), then, response-normalized activity b(i, x, y) is:

$$b^i_{x,y} = a^i_{x,y} / \left(k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} (a^j_{x,y})^2 \right)^{\beta}$$

Local Response Normalization activation equation (1)

Pooling Layers take a small (in our case 2x2) field for each filter as input and pool the values into singular output. Pooling layers typically follow convolution layers and allow for significant dimensionality reduction. In particular, we leverage max-pooling in our neural network design.

Nesterov's Accelerating Gradient. Neural network's typically leverage stochastic gradient descent for learning. Instead, we use Nesterov's Accelerating Gradient, which leverages changes in momentum of the gradient to accelerate convergence, in theory in O(1/t^2) time as opposed to O(1/t) of SGD [4]. W(t+1) represents the change in weight from NAG.

$$V_{t+1} = \mu V_t - \alpha \nabla L(W_t + \mu V_t)$$

$$W_{t+1} = W_t + V_{t+1}$$

Nesterov's Accelerating Gradient equation (1)

1.3. Neural Network Architecture

We take inspiration after Khrichevsky's seminal ILSVRC2012-winning convolutional neural network and build a similar net to train in the Tiny ImageNet Challenge. Our net is 9-layer-deep convolutional neural network. The first 5 layers are convolution/pooling layers, followed by 3 fully-connected layers, with a final softmax layer to measure loss. We apply ReLU non-linearity after every layer, given it's usefulness in speeding up training. We apply Local Normalization layer after convolution layer's 1, 2 and 5 at the recommendation of Khrichevsky, expecting marginal improvement in performance. We apply 50% dropout after each FC layer, in order to mitigate issues of overfitting. Due to GPU memory constraints, we limit the number of filters in each convolution layer to 96. We attempt a strategy of maintaining as much pixel information by keeping field size small in each convolution layer. After balancing GPU memory requirements, we choose 5X5 for convolution one and two, and 3x3 for convolution layers 3 - 5. Zero padding and stride are chosen to maintain input height and width through each convolution layer. Training Parameters

Following recommended best practices, we use learning rate of 0.01, with a stepwise learning-rate reduction policy of 10% per epoch. We leverage nesterov's accelerating gradient instead of SGD, allowing us to achieve more rapid convergence, using momentum parameter of 0.9 and weight decay of 0.0005. During prediction, we oversample by averaging predictions over 10 sample predictions based on cropping the sample image in each corner as well as the center, and then performing similar to the mirror image. Modest parameter tuning was performed, but did not provide substantial improvements to the above.

1.4. Performance and Measurement

To measure performance, we measure single-guess accuracy on the testing set. Our best parameters achieved top-1 error rate of 69.6% accuracy on the test set. While this is significantly lower than Khrichevsky's model, we also had a fraction of testing data to train on. In addition, our net is able to achieve learning significantly above the random guess of 99.5%.

1.5. Conclusion

Convolutional neural networks are also powerful machine learning paradigm for computer vision object classification. In this study, we take after the seminal ILSVRC-winning convolutional network designed by Khrichevsky et al. We achieved substantial performance at 69.6%, though not near the state-of-the-art (GoogleNet has achieved top-5 error rate of 6.67%). We leverage Caffe, an open-source, GPU-enabled CNN development library that allows for rapid design and testing of CNN models.

References

[1] Alex Krizhevsky, Ilya Sutskever, and Geoff Hinton. Imagenet classification with deep convolutional neural networks. In Advances in Neural Information Processing Systems 25, pages 1106–1114, 2012.

[2] Szegedy, Christian, et al. "Going deeper with convolutions." arXiv preprint arXiv:1409.4842 (2014).

[3] Jia et al. Caffe: Convolutional Architecture for Fast Feature Embedding. arXiv1408.5093. June 2014.

[4] Karpathy et al. CS231n: Convolutional Neural Networks for Visual Recognition. 2015. http://cs231n.github.io/convolutional-networks/