Imagenet Classification with Deep Convolutional Neural Networks

1 Citation

Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems. 2012.

http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

2 Abstract

We use 5 convolutional layers (some with max pooling) followed by 3 fully-connected layers and a softmax to win ImageNet 2012 with top-5 error of 15.3%.

3 Introduction

CNNs are pretty good for image classification, but they are hard to train. We fix this by using two GTX 580 3GB GPUs and an efficient convolution operation. We also have tricks to avoid overfitting. We take 5-6 days to train model.

4 The Dataset

ImageNet has 15M images in 22K categories. The classification challenge (ImageNet Large Scale Visual Recognition Challenge - ILSVRC) uses 1.2M/50K/150K training/validation/testing images from 1K categories.

We scale the image so shorter side has length 256 pixels, extract the central 256x256 crop, and subtract the mean activity over training set from each pixel.

5 Architecture

To reduce train error more quickly, we feed neuron outputs through a Rectified Linear Unit (ReLU). A ReLU computes $f(x) = \max(x, 0)$.

We train using two GPUs, so each GPU gets half the kernels from each layer. So, the kernel maps from layer K only use kernel maps from layer $K-1$ that are on the same GPU. The exception is for layer 3, where we use all kernel maps from layer 2.

We use Local Response Normalization (LRN), where activity of a neuron at (x, y) in kernel i (call this $a^i_{x,y}$) is normalized to $b^i_{x,y}$ by comparing it to n neighboring kernel maps (there are N kernel maps total in the layer) as follows:

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

Where $(k, n, \alpha, \beta) = (2, 5, 10^{-4}, 0.75)$. LRN forces neighboring kernel maps to compete for activation.

We use $z \times z$ (where $z = 3$) pooling with a stride of $s = 2$. Having $z > s$ means pooling windows overlap, which improves performance.

We have 5 convolutional layers, 3 fully connected layers, and finally a 1K softmax. We minimize the multinomial logistic regression objective (average log probability of correct label under prediction distribution).

Kernels connect to previous layer kernels only on same GPU (except for third convolutional layer and fully connected layers). LRN follows first and second convolutional layers. Max pooling follows LRN and the fifth convolutional layer.

Here's what the model does:

1. $224 \times 224 \times 3$ image convolved with 96 kernels of size $11 \times 11 \times 3$ with a stride of 4.
2. LRN + pooling.
3. 256 kernels of size $5 \times 5 \times 48$.
4. LRN + pooling.
5. 384 kernels of size $3 \times 3 \times 256$ (use all kernels from previous layer).
6. 384 kernels of size $3 \times 3 \times 192$.
7. 256 kernels of size $3 \times 3 \times 192$.
8. 3 fully connected layers with 4096 neurons each.
9. Softmax.

6 Reducing Overfitting

To avoid overfitting the 60M network parameters, we use data augmentation and dropout. For data augmentation, we do the following:

1. Sample 224×224 patches of the 256×256 image.
2. Take horizontal reflections.
3. Do PCA on RGB values of training images and add multiples of principal components. So, for pixel $I_{x,y} = [I^R_{x,y}, I^G_{x,y}, I^B_{x,y}]$, add $[\mathbf{p_1}, \mathbf{p_2}, \mathbf{p_3}][\alpha_\mathbf{1}\lambda_\mathbf{1}, \alpha_\mathbf{2}\lambda_\mathbf{2}, \alpha_\mathbf{3}\lambda_\mathbf{3}]^\mathbf{T}$ where $\mathbf{p_i}$ and λ_i are the i^th eigenvector and eigenvalue of the 3×3 covariance matrix of RGB pixel values and α_i is sampled from a zero-mean Gaussian with standard deviation of 0.1.

At test time, we extract the four 224×224 corner patches, center patch, and all horizontal reflections. We average our prediction over each of them.

For dropout, we drop a hidden neuron with probability 0.5 and we multiply neuron output by 0.5. Dropout doubles the number of iterations until convergence.

7 Details of Learning

Set (batch size, momentum, weight decay) $= (128, 0.9, 0.0005)$. The update rule is:

$$v_{i+1} := 0.9v_i - 0.0005\epsilon w_i - \frac{\epsilon}{128}\sum_{k=1}^{128}[\frac{\partial L}{\partial w}]_{w_i} \tag{2}$$

$$w_{i+1} := w_i + v_{i+1} \tag{3}$$

Weights are initialized with zero-mean Gaussians with $\sigma = 0.01$. Biases in the fully connected layers and the second, third, and fifth convolutional layers with 1 (ensures ReLUs start with positive input). Other biases are initialized with 0.

Learning rate (ϵ) began at 0.01 and was divided by 10 each time validation error stopped improving (3 times total). Ran 90 epochs. Took 6 days.

8 Results

On ILSVRC 2010, we get top-1 (top-5) error rate of 37.5% (17.0%), which is 10% (10%) better than previous best.

The first GPU learns to be color agnostic while the second uses color. Looking at a few samples, network makes reasonable guesses even when it is wrong. Similar objects (e.g. purple flowers, elephants, ships) have similar (via Euclidean distance) activations on last fully connected hidden layer.

9 Discussion

Removing any convolutional layer hurts performance. We didn't use unsupervised pre-training (maybe that would help).