AlexNet


Djordje Slijepcevic
Introduction

• Convolutional Neural Network (CNN)
• Winner of ImageNet Large Scale Visual Recognition Challenge (ILSVRC) 2012
  • first successful CNN application for such a big dataset
  • top-5 test error rate of 15.3% (+10.9% compared to 2nd)
• Relatively simple layout (compared to modern architectures)
  • 5 conv. layers
  • 3 fully connected layers
  • max-pooling layers
  • dropout layers
Dataset

- **ImageNet:**
  - 15+ million labeled high-resolution images
  - 22000 categories

- **ILSVRC uses a subset of ImageNet:**
  - ~ 1000 images per category
  - 1000 categories
  - 1.2 million training images | 50000 validation images | 150000 testing images

- **AlexNet:**
  - images were down-sampled and cropped to 256×256 pixels
  - subtraction of the mean activity over the training set from each pixel
Task

Dataset

[https://cs.stanford.edu/people/karpathy/cnnembed/, 30.11.2017]

Architecture

5 convolutional layers

3 fully connected layers

1000-way softmax

Activation function

• Traditionally, saturating nonlinearities:
  • hyperbolic tangent function: \( f(x) = \tanh(x) = 2 \times \frac{1}{1+e^{-2x}} - 1 \)
  • sigmoid function: \( f(x) = \frac{1}{1+e^{-x}} \)  
    \( \rightarrow \) slow to train

• Non-saturating nonlinearity:
  • Rectified Linear Unit (ReLU): \( f(x) = \max(0, x) \)  
    \( \rightarrow \) quick to train
Activation function

- Traditionally, saturating nonlinearities:
  - Saturated neurons facilitate vanishing of gradients
  - exp function is a bit compute expensive
    → slow to train

- Non-saturating nonlinearity:
  - Does not saturate (in the + region)
  - Very computationally efficient
    → quick to train
Activation function

- Dataset: CIFAR-10
- Experiment:
  - CNN (4 layers) + ReLUs (solid line) vs. CNN (4 layers) + tanh (dashed line)
  - ReLUs six times faster

Activation function
Training on Multiple GPUs

- Half of the neurons of an certain layer are on each GPU
- GPUs communicate only in certain layers
- Improvement (as compared with a net with half as many kernels in each convolutional layer trained on one GPU):
  - top-1 error rate by 1.7%
  - top-5 error rate by 1.2%
Training on Multiple GPUs

Intra-GPU connections

Inter-GPU connections
Local Response Normalization

• ReLUs do not require input normalization to prevent them from saturating
• However, Local Response Normalization aids generalization

Activity of a neuron by applying kernel \(i\) at position \((x,y)\)

\[
b_{x,y}^i = a_{x,y}^i / \left( k + \alpha \sum_{j=\max(0,i-n/2)}^{\min(N-1,i+n/2)} (a_{x,y}^j)^2 \right)^\beta
\]

• Improvement:
  • top-1 error rate by 1.4%
  • top-5 error rate by 1.2%

\(k = 2\)
\(n = 5\)
\(\alpha = 10^{-4}\)
\(\beta = 0.75\)

sum runs over \(n\) “adjacent” kernel maps at the same spatial position
Local Response Normalization
Overlapping Pooling

- Pooling layers summarize the outputs of neighboring neurons in the same kernel map.
- Overlapping pooling $\Rightarrow s < z$

- Improvement using MaxPooling:
  - top-1 error rate by 0.4%
  - top-5 error rates by 0.3%
Overlapping Pooling
Overall Architecture

- **96 kernels (11x11x3)**
- **256 kernels (5x5x48)**
- **384 kernels (3x3x192)**
- **4096 neurons**

Stride of 4
Max pooling

- **128 kernels (3x3x192)**
- **256 kernels (3x3x192)**
- **4096 neurons**
Reducing Overfitting - Data Augmentation

• 1\textsuperscript{st} : image translations and horizontal reflections
  • random 224x224 patches + horizontal reflections from the 256x256 images
  • Testing: five 224x224 patches + horizontal reflections → averaging the predictions over the ten patches

• 2\textsuperscript{nd} : change the intensity of RGB channels
  • PCA on the set of RGB pixel values throughout the ImageNet training set
  • To each RGB image pixel \( I_{xy} = [I_{xy}^R, I_{xy}^G, I_{xy}^B] \) following is added

\[
[p_1, p_2, p_3][\alpha_1 \lambda_1, \alpha_2 \lambda_2, \alpha_3 \lambda_3]^T \quad |\alpha_i \sim N(0, 0.1)\
\]

• Improvement:
  • top-1 error rate by 1%
Reducing Overfitting - Dropout

- Output of each hidden neuron is set to zero with probability 0.5
- Learning more robust features
- Doubles the number of iterations required to converge
- Applied in the first two fully connected layers

[N. Srivastava et al., Dropout: A Simple Way to Prevent Neural Networks from Overfitting, 2014]
Reducing Overfitting - Dropout
Stochastic Gradient Descent

• Training process
  • Minimizing the cross-entropy loss function:

\[
L(w) = \sum_{i=1}^{N} \sum_{c=1}^{1000} -y_{ic} \log f_c(x_i) + \epsilon ||w||^2
\]

- predicted probability of class c for image x
- indicator that example i has label c
Stochastic Gradient Descent

• SGD with a batch size of 128
• Learning rate initialized at 0.01; divided by 10 if validation error rate stopped improving
• Update rule for weight $w$:
  
  $v_{i+1} := 0.9 * v_i - 0.0005 * \epsilon * w_i - \epsilon * \left( \frac{\partial L}{\partial w} \right)_{w_i} D_i$

  $w_{i+1} := w_i + v_{i+1}$

• ~90 cycles $\rightarrow$ five to six days on two NVIDIA GTX 580 3GB GPUs
Results: ILSVRC 2012

<table>
<thead>
<tr>
<th>Model</th>
<th>Top-1 (val)</th>
<th>Top-5 (val)</th>
<th>Top-5 (test)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SIFT + FVs [7]</td>
<td>—</td>
<td>—</td>
<td>26.2%</td>
</tr>
<tr>
<td>1 CNN</td>
<td>40.7%</td>
<td>18.2%</td>
<td>—</td>
</tr>
<tr>
<td>5 CNNs</td>
<td>38.1%</td>
<td>16.4%</td>
<td>16.4%</td>
</tr>
<tr>
<td>1 CNN*</td>
<td>39.0%</td>
<td>16.6%</td>
<td>—</td>
</tr>
<tr>
<td>7 CNNs*</td>
<td>36.7%</td>
<td>15.4%</td>
<td>15.3%</td>
</tr>
</tbody>
</table>

Comparison of error rates on ILSVRC-2012 validation and test sets. The first row is the best result achieved by others. Models with an asterisk* were “pre-trained“ to classify the entire ImageNet 2011 Fall release.

Results

• The filters of the first convolutional layer \(\rightarrow\) one GPU generates high-frequency grayscale features and the other low-frequency color features

Results

[D. Wei et al., http://vision03.csail.mit.edu/cnn_art/]
Conclusion

• Depth is very important
  • network’s performance degrades if a single convolutional layer is removed
• The use of ReLUs is essential for improving training runtime
• Training on multiple GPUs
• Tricks: overlapping pooling, dropout, data augmentation, weight decay...
• Strong influence ➔ cited by 17596 (December 6th, 2017)
Tensorflow – Code

• Can be found here:
  • https://github.com/tensorflow/models/blob/master/research/slim/nets/alexnet.py
  • https://github.com/kratzert/finetune_alexnet_with_tensorflow/blob/master/alexnet.py