Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation

Ross Girshick, Jeff Donahue, Trevor Darrell, Jitendra Malik
PASCAL VOC detection history

[Source: http://pascallin.ics.uci.edu/challenges/VOC/voc20{07,08,09,10,11,12}/results/index.html]
A rapid rise in performance

[Source: http://pascallin.ecs.soton.ac.uk/challenges/VOC/voc20{07,08,09,10,11,12}/results/index.html]
Complexity and the plateau

PASCAL VOC challenge dataset

- DPM
- HOG+BOW
- DPM, MKL
- Selective Search
- DPM++, MKL

Top competition results (2007 - 2012)

[Source: http://pascallin.ecs.soton.ac.uk/challenges/VOC/voc20{07,08,09,10,11,12}/results/index.html]
SIFT, HOG, LBP, ...

mAP (%)

PASCAL VOC challenge dataset

[Regionlets. Wang et al. ICCV’13]  [SegDPM. Fidler et al. CVPR’13]
R-CNN: Regions with CNN features

![Graph showing mAP (%) for R-CNN on PASCAL VOC challenge dataset from 2007 to 2012. Post-competition results (2013-present) and Top competition results (2007-2012).]
Feature learning with CNNs

Fukushima 1980
Neocognitron
Feature learning with CNNs

Fukushima 1980
Neocognitron

Rumelhart, Hinton, Williams 1986
“T” versus “C” problem
Feature learning with CNNs

Fukushima 1980
Neocognitron

Rumelhart, Hinton, Williams 1986
“T” versus “C” problem

LeCun et al. 1989-1998
Handwritten digit reading / OCR
Feature learning with CNNs

Fukushima 1980
Neocognitron

Rumelhart, Hinton, Williams 1986
“T” versus “C” problem

LeCun et al. 1989-1998
Handwritten digit reading / OCR

Krizhevksy, Sutskever, Hinton 2012
ImageNet classification breakthrough
“SuperVision” CNN
CNNs for object detection

Vaillant, Monrocq, LeCun 1994
Multi-scale face detection

LeCun, Huang, Bottou 2004
NORB dataset

Cireşan et al. 2013
Mitosis detection

Sermanet et al. 2013
Pedestrian detection

Szegedy, Toshev, Erhan 2013
PASCAL detection (VOC’07 mAP 30.5%)
Can we break through the PASCAL plateau with feature learning?
**R-CNN: Regions with CNN features**

- **Input image**
- **Extract region proposals (~2k / image)**
- **Compute CNN features**
- **Classify regions (linear SVM)**

- aeroplane? no.
- person? yes.
- tvmonitor? no.
R-CNN at test time: Step 1

**Input image**

- Extract region proposals (~2k / image)

**Proposal-method agnostic, many choices**
- Selective Search [van de Sande, Uijlings et al.] (Used in this work)
- Objectness [Alexe et al.]
- Category independent object proposals [Endres & Hoiem]
- CPMC [Carreira & Sminchisescu]

**Active area, at this CVPR**
- BING [Ming et al.] – fast
- MCG [Arbelaez et al.] – high-quality segmentation
R-CNN at test time: **Step 2**

- **Input image**
- **Extract region proposals (~2k / image)**
- **Compute CNN features**

- **CNN**
  - aeroplane? no.
  - person? yes.
  - tvmonitor? no.
R-CNN at test time: **Step 2**

Input image → Extract region proposals (~2k / image) → Compute CNN features

Dilate proposal

- aeroplane? no.
- person? yes.
- tvmonitor? no.
R-CNN at test time: **Step 2**

- **Input image**
- **Extract region proposals (~2k / image)**
- **Compute CNN features**

- **a. Crop**

  - aeroplane? no.
  - person? yes.
  - tvmonitor? no.
R-CNN at test time: Step 2

Input image
Extract region proposals (~2k / image)
Compute CNN features

- a. Crop
- b. Scale (anisotropic)

aeroplane? no.
person? yes.
tvmonitor? no.

227 x 227
R-CNN at test time: **Step 2**

1. **Crop**
2. **Scale (anisotropic)**

**Input image**

**Extract region proposals (~2k / image)**

**Compute CNN features**

- aeroplane? no.
- person? yes.
- tvmonitor? no.

**c. Forward propagate**

Output: “fc7” features
R-CNN at test time: **Step 3**

**CNN**

**Input image**

**Extract region proposals (~2k / image)**

**Compute CNN features**

**Classify regions**

- aeroplane? no.
- person? yes.
- tvmonitor? no.

**Proposal**

**4096-dimensional fc7 feature vector**

**Linear classifiers**

(SVM or softmax)

- person? 1.6
- horse? -0.3
- ...
Step 4: **Object proposal refinement**

- **Original proposal**
- **Predicted object bounding box**

Linear regression on CNN features

Bounding-box regression
## R-CNN results on PASCAL

<table>
<thead>
<tr>
<th>Method</th>
<th>VOC 2007</th>
<th>VOC 2010</th>
</tr>
</thead>
<tbody>
<tr>
<td>DPM v5 (Girshick et al. 2011)</td>
<td>33.7%</td>
<td>29.6%</td>
</tr>
<tr>
<td>UVA sel. search (Uijlings et al. 2013)</td>
<td>35.1%</td>
<td></td>
</tr>
<tr>
<td>Regionlets (Wang et al. 2013)</td>
<td>41.7%</td>
<td>39.7%</td>
</tr>
<tr>
<td>SegDPM (Fidler et al. 2013)</td>
<td>40.4%</td>
<td></td>
</tr>
</tbody>
</table>

**Reference systems**

- **metric**: mean average precision (higher is better)
## R-CNN results on PASCAL

<table>
<thead>
<tr>
<th>Method</th>
<th>VOC 2007</th>
<th>VOC 2010</th>
</tr>
</thead>
<tbody>
<tr>
<td>DPM v5 (Girshick et al. 2011)</td>
<td>33.7%</td>
<td>29.6%</td>
</tr>
<tr>
<td>UVA sel. search (Uijlings et al. 2013)</td>
<td>35.1%</td>
<td></td>
</tr>
<tr>
<td>Regionlets (Wang et al. 2013)</td>
<td>41.7%</td>
<td>39.7%</td>
</tr>
<tr>
<td>SegDPM (Fidler et al. 2013)</td>
<td></td>
<td>40.4%</td>
</tr>
<tr>
<td><strong>R-CNN</strong></td>
<td>54.2%</td>
<td>50.2%</td>
</tr>
<tr>
<td><strong>R-CNN + bbox regression</strong></td>
<td>58.5%</td>
<td>53.7%</td>
</tr>
</tbody>
</table>

**metric:** mean average precision (higher is better)
ImageNet detection

New challenge introduced in 2013

PASCAL-like image complexity

200 object categories
R-CNN and OverFeat

R-CNN
- developed using PASCAL VOC
- tested on PASCAL VOC: s-o-t-a
- no results on ILSVRC 2013 (until now)

OverFeat [Sermanet et al. 2014]
- developed using ILSVRC 2013
- tested on ILSVRC 2013: s-o-t-a
- no results on PASCAL VOC

No apples-to-apples comparison
R-CNN on ImageNet detection

ILSVRC2013 detection test set mAP

- **R-CNN BB**: 31.4%
- **OverFeat (2)**: 24.3%
- **UvA-Euvisio**n: 22.6%
- **NEC-MU**: 20.9%
- **OverFeat (1)**: 19.4%
- **Toronto A**: 11.5%
- **SYSU_Vision**: 10.5%
- **GPU_UCLA**: 9.8%
- **Delta**: 6.1%
- **UIUC-IFP**: 1.0%

## Detection speed & scalability

<table>
<thead>
<tr>
<th></th>
<th>20 classes</th>
<th>200 classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Selective search</td>
<td>2.1s</td>
<td>2.1s</td>
</tr>
<tr>
<td>Cropping &amp; resizing</td>
<td>2.6s</td>
<td>2.6s</td>
</tr>
<tr>
<td>CNN feature computation (GPU)</td>
<td>5.9s</td>
<td>5.9s</td>
</tr>
<tr>
<td>Region classification</td>
<td>0.005s</td>
<td>0.001s</td>
</tr>
<tr>
<td>NMS</td>
<td>0.026s</td>
<td>0.012s</td>
</tr>
</tbody>
</table>

**Hardware:** Intel Core i7-3930K 3.2Ghz and NVIDIA Tesla K20c

We thank NVIDIA for generous hardware donations.
Detection speed & scalability

<table>
<thead>
<tr>
<th>20 classes</th>
<th>200 classes</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1s cpu</td>
<td>2.1s</td>
</tr>
<tr>
<td>2.6s cpu</td>
<td>2.6s</td>
</tr>
<tr>
<td>5.9s gpu</td>
<td>5.9s</td>
</tr>
</tbody>
</table>

Selective search 0.005s
Cropping & resizing 0.001s
CNN feature computation (GPU) 0.026s
Region classification 0.12s
NMS 30ms!

O(1) Selective search
O(N) Cropping & resizing
O(N) CNN feature computation (GPU)
O(N) Region classification
O(N) NMS

Hardware: Intel Core i7-3930K 3.2Ghz and NVIDIA Tesla K20c
We thank NVIDIA for generous hardware donations.
Top bicycle FPs (AP = 72.8%)

**Figure 7:** False positive trends with rank. Left: stacked area plot showing fraction of FP of each type as the total number of FP increase. Right: same, but shown as line plots.

**Figure 8:** Examples of top bicycle false positives.
False positive type distribution

R–CNN FT fc7 BB: vehicles
R–CNN FT fc7 BB: furniture
R–CNN FT fc7 BB: animals

Loc = localization
Sim = similar classes
Oth = other / dissimilar classes
BG = background

Top bicycle FPs (AP = 72.8%)
False positive #15

bicycle (bg): ov=0.00 1−r=0.44
False positive #15

Unannotated bicycle

bicycle (bg): ov=0.00 1−r=0.44
False positive #15

1949 French comedy by Jacques Tati
Training R-CNN

Bounding-box labeled detection data is scarce

Key insight:
Use *supervised* pre-training on a data-rich auxiliary task and *transfer* to detection
R-CNN training: Step 1

Supervised pre-training
Train a SuperVision CNN* for the 1000-way ILSVRC image classification task

*Network from Krizhevsky, Sutskever & Hinton. NIPS 2012
Also called “AlexNet”
R-CNN training: **Step 2**

Fine-tune the CNN for detection
Transfer the representation learned for ILSVRC classification to PASCAL (or ImageNet detection)

**Target task:**
PASCAL VOC detection
(~25k object labels)
Fine-tune the CNN for detection
Transfer the representation learned for ILSVRC classification to PASCAL (or ImageNet detection)

Target task:
PASCAL VOC detection
(~25k object labels)

Try Caffe!  http://caffe.berkeleyvision.org
- Clean & fast CNN library in C++ with Python and MATLAB interfaces
- Used by R-CNN for training, fine-tuning, and feature computation
R-CNN training: Step 3

Train detection SVMs
(With the softmax classifier from fine-tuning mAP decreases from 54% to 51%)
What did the network learn?

Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

The second convolutional layer takes as input the (response-normalized and pooled) output of the first convolutional layer and filters it with $256$ kernels of size $5 \times 5 \times 48$. The third, fourth, and fifth convolutional layers are connected to one another without any intervening pooling or normalization layers. The third convolutional layer has $384$ kernels of size $3 \times 3 \times 256$ connected to the (normalized, pooled) outputs of the second convolutional layer. The fourth convolutional layer has $384$ kernels of size $3 \times 3 \times 192$, and the fifth convolutional layer has $256$ kernels of size $3 \times 3 \times 192$. The fully-connected layers have $4096$ neurons each.

4 Reducing Overfitting

Our neural network architecture has 60 million parameters. Although the 1000 classes of ILSVRC make each training example impose 10 bits of constraint on the mapping from image to label, this turns out to be insufficient to learn so many parameters without considerable overfitting. Below, we describe the two primary ways in which we combat overfitting.

4.1 Data Augmentation

The easiest and most common method to reduce overfitting on image data is to artificially enlarge the dataset using label-preserving transformations (e.g., [25, 4, 5]). We employ two distinct forms of data augmentation, both of which allow transformed images to be produced from the original images with very little computation, so the transformed images do not need to be stored on disk. In our implementation, the transformed images are generated in Python code on the CPU while the GPU is training on the previous batch of images. So these data augmentation schemes are, in effect, computationally free.

The first form of data augmentation consists of generating image translations and horizontal reflections. We do this by extracting random $224 \times 224 \times 3$ patches (and their horizontal reflections) from the $256 \times 256$ images and training our network on these extracted patches. This increases the size of our training set by a factor of 2048, though the resulting training examples are, of course, highly interdependent. Without this scheme, our network suffers from substantial overfitting, which would have forced us to use much smaller networks. At test time, the network makes a prediction by extracting five $224 \times 224 \times 3$ patches (the four corner patches and the center patch) as well as their horizontal reflections (hence ten patches in all), and averaging the predictions made by the network's softmax layer on the ten patches.

The second form of data augmentation consists of altering the intensities of the RGB channels in training images. Specifically, we perform PCA on the set of RGB pixel values throughout the ImageNet training set. To each training image, we add multiples of the found principal components.
What did the network learn?

Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000 neurons in a kernel map). The second convolutional layer takes as input the (response-normalized and pooled) output of the first convolutional layer and filters it with 256 kernels of size $5 \times 5 \times 48$. The third, fourth, and fifth convolutional layers are connected to one another without any intervening pooling or normalization layers. The third convolutional layer has 384 kernels of size $3 \times 3 \times 256$ connected to the (normalized, pooled) outputs of the second convolutional layer. The fourth convolutional layer has 384 kernels of size $3 \times 3 \times 192$, and the fifth convolutional layer has 256 kernels of size $3 \times 3 \times 192$. The fully-connected layers have 4096 neurons each.

4 Reducing Overfitting

Our neural network architecture has 60 million parameters. Although the 1000 classes of ILSVRC make each training example impose 10 bits of constraint on the mapping from image to label, this turns out to be insufficient to learn so many parameters without considerable overfitting. Below, we describe the two primary ways in which we combat overfitting.

4.1 Data Augmentation

The easiest and most common method to reduce overfitting on image data is to artificially enlarge the dataset using label-preserving transformations (e.g., [25, 4, 5]). We employ two distinct forms of data augmentation, both of which allow transformed images to be produced from the original images with very little computation, so the transformed images do not need to be stored on disk. In our implementation, the transformed images are generated in Python code on the CPU while the GPU is training on the previous batch of images. So these data augmentation schemes are, in effect, computationally free.

The first form of data augmentation consists of generating image translations and horizontal reflections. We do this by extracting random $224 \times 224$ patches (and their horizontal reflections) from the $256 \times 256$ images and training our network on these extracted patches. This increases the size of our training set by a factor of 2048, though the resulting training examples are, of course, highly interdependent. Without this scheme, our network suffers from substantial overfitting, which would have forced us to use much smaller networks. At test time, the network makes a prediction by extracting five $224 \times 224$ patches (the four corner patches and the center patch) as well as their horizontal reflections (hence ten patches in all), and averaging the predictions made by the network's softmax layer on the ten patches.

The second form of data augmentation consists of altering the intensities of the RGB channels in training images. Specifically, we perform PCA on the set of RGB pixel values throughout the ImageNet training set. To each training image, we add multiples of the found principal components.

This is the reason why the input images in Figure 2 are $224 \times 224 \times 3$-dimensional.
Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

4 Reducing Overfitting

Our neural network architecture has 60 million parameters. Although the 1000 classes of ILSVRC make each training example impose 10 bits of constraint on the mapping from image to label, this turns out to be insufficient to learn so many parameters without considerable overfitting. Below, we describe the two primary ways in which we combat overfitting.

4.1 Data Augmentation

The easiest and most common method to reduce overfitting on image data is to artificially enlarge the dataset using label-preserving transformations (e.g., [25, 4, 5]). We employ two distinct forms of data augmentation, both of which allow transformed images to be produced from the original images with very little computation, so the transformed images do not need to be stored on disk. In our implementation, the transformed images are generated in Python code on the CPU while the GPU is training on the previous batch of images. So these data augmentation schemes are, in effect, computationally free.

The first form of data augmentation consists of generating image translations and horizontal reflections. We do this by extracting random 224⇥224 patches (and their horizontal reflections) from the 256⇥256 images and training our network on these extracted patches. This increases the size of our training set by a factor of 2048, though the resulting training examples are, of course, highly interdependent. Without this scheme, our network suffers from substantial overfitting, which would have forced us to use much smaller networks. At test time, the network makes a prediction by extracting five 224⇥224 patches (the four corner patches and the center patch) as well as their horizontal reflections (hence ten patches in all), and averaging the predictions made by the network's softmax layer on the ten patches.

The second form of data augmentation consists of altering the intensities of the RGB channels in training images. Specifically, we perform PCA on the set of RGB pixel values throughout the ImageNet training set. To each training image, we add multiples of the found principal components, 4This is the reason why the input images in Figure 2 are 224⇥224⇥3-dimensional.

4 What did the network learn?

Visualize images that activate pool₅ a feature
Semantic segmentation

CPMC segments (Carreira & Sminchisescu)

<table>
<thead>
<tr>
<th></th>
<th>VOC 2011 test</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bonn second-order pooling (Carreira et al.)</td>
<td>47.6%</td>
</tr>
<tr>
<td>R-CNN fc_6 full+fg (no fine-tuning)</td>
<td>47.9%</td>
</tr>
</tbody>
</table>

Improved to **50.5%** in our upcoming ECCV’14 work: Simultaneous Localization and Detection. Hariharan et al.
Take away

- Dramatically better PASCAL mAP
- R-CNN outperforms other CNN-based methods on ImageNet detection
- Detection speed is manageable (~11s / image)
- Scales very well with number of categories (30ms for 20 → 200 classes!)
- R-CNN is simple and completely open source
Get the code and models!
bit.ly/rcnn-cvpr14
Supplementary slides follow
Ablation: skip fine-tuning

Step 1: pre-train
- Auxiliary task: ILSVRC 2012 classification (1.2 million images)

Step 2: fine-tune
- Target task: PASCAL VOC detection (~25k object labels)

Step 3: train SVMs
- PASCAL VOC object proposals
- CNN features
- ~2k windows / image
- Training labels
- Per-class SVM

Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

Reducing Overfitting

Our neural network architecture has 60 million parameters. Although the 1000 classes of ILSVRC make each training example impose 10 bits of constraint on the mapping from image to label, this turns out to be insufficient to learn so many parameters without considerable overfitting. Below, we describe the two primary ways in which we combat overfitting.

4.1 Data Augmentation

The easiest and most common method to reduce overfitting on image data is to artificially enlarge the dataset using label-preserving transformations (e.g., [25, 4, 5]). We employ two distinct forms of data augmentation, both of which allow transformed images to be produced from the original images with very little computation, so the transformed images do not need to be stored on disk.

In our implementation, the transformed images are generated in Python code on the CPU while the GPU is training on the previous batch of images. So these data augmentation schemes are, in effect, computationally free.

The first form of data augmentation consists of generating image translations and horizontal reflections. We do this by extracting random 224×224 patches (and their horizontal reflections) from the 256×256 images and training our network on these extracted patches. This increases the size of our training set by a factor of 2048, though the resulting training examples are, of course, highly interdependent. Without this scheme, our network suffers from substantial overfitting, which would have forced us to use much smaller networks. At test time, the network makes a prediction by extracting five 224×224 patches (the four corner patches and the center patch) as well as their horizontal reflections (hence ten patches in all), and averaging the predictions made by the network's softmax layer on the ten patches.

The second form of data augmentation consists of altering the intensities of the RGB channels in training images. Specifically, we perform PCA on the set of RGB pixel values throughout the ImageNet training set. To each training image, we add multiples of the found principal components.
### Pre-trained CNN + SVMs (no FT)

<table>
<thead>
<tr>
<th>Method</th>
<th>VOC 2007</th>
<th>VOC 2010</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regionlets (Wang et al. 2013)</td>
<td>41.7%</td>
<td>39.7%</td>
</tr>
<tr>
<td>SegDPM (Fidler et al. 2013)</td>
<td></td>
<td>40.4%</td>
</tr>
<tr>
<td>R-CNN pool&lt;sub&gt;5&lt;/sub&gt;</td>
<td>44.2%</td>
<td></td>
</tr>
<tr>
<td>R-CNN fc&lt;sub&gt;6&lt;/sub&gt;</td>
<td></td>
<td>46.2%</td>
</tr>
<tr>
<td>R-CNN fc&lt;sub&gt;7&lt;/sub&gt;</td>
<td>44.7%</td>
<td></td>
</tr>
</tbody>
</table>

**Metric:** mean average precision (higher is better)
Compare with fine-tuned R-CNN

<table>
<thead>
<tr>
<th>Method</th>
<th>VOC 2007</th>
<th>VOC 2010</th>
</tr>
</thead>
<tbody>
<tr>
<td>Regionlets (Wang et al. 2013)</td>
<td>41.7%</td>
<td>39.7%</td>
</tr>
<tr>
<td>SegDPM (Fidler et al. 2013)</td>
<td></td>
<td>40.4%</td>
</tr>
<tr>
<td><strong>R-CNN pool\textsubscript{5}</strong></td>
<td>44.2%</td>
<td></td>
</tr>
<tr>
<td><strong>R-CNN fc\textsubscript{6}</strong></td>
<td>46.2%</td>
<td></td>
</tr>
<tr>
<td><strong>R-CNN fc\textsubscript{7}</strong></td>
<td>44.7%</td>
<td></td>
</tr>
<tr>
<td><strong>R-CNN FT pool\textsubscript{5}</strong></td>
<td>47.3%</td>
<td></td>
</tr>
<tr>
<td><strong>R-CNN FT fc\textsubscript{6}</strong></td>
<td>53.1%</td>
<td></td>
</tr>
<tr>
<td><strong>R-CNN FT fc\textsubscript{7}</strong></td>
<td>54.2%</td>
<td>50.2%</td>
</tr>
</tbody>
</table>

Metric: mean average precision (higher is better)
3.3. Detection error analysis

Comparison to recent feature learning methods. Internally to their private DPM baselines—both use non-domain-specific non-linear classifiers on top of them. The second method, DPM HSC [5] uses only HOG features, our mAP is more than 20 percentage points to 54.2%. The boost from fine-tuning is that the pool replaces HOG with `|2, which suggests that the pool`, which suggests

We now look at results from our CNN after having fine-tuned its parameters on VOC 2007 trainval. The improvement is strikingly few feature learning methods have been tried on PAS-18. These methods achieve mAPs of...

The resulting activations are rectified in three ways (full and half-waves), spatially pooled, unit...
3.3. Detection error analysis

The combination of HOG and sketch to-
False positive analysis

![Diagram showing false positive analysis](image_url)

- R-CNN fc6: animals
- R-CNN FT fc7: animals
- R-CNN FT fc7 BB: animals

After bounding-box regression
ImageNet LSVR Challenge

- Image classification *(not detection)*
- 1000 classes (vs. 20)
- 1.2 million training labels (vs. 25k)

bus *anywhere?*

[Deng et al. CVPR’09]
ILSVRC 2012 winner

“SuperVision” Convolutional Neural Network (CNN)

Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network’s input is 150,528-dimensional, and the number of neurons in the network’s remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

The second convolutional layer takes as input the (response-normalized and pooled) output of the first convolutional layer and filters it with $256$ kernels of size $5 	imes 5 	imes 48$.

The third, fourth, and fifth convolutional layers are connected to one another without any intervening pooling or normalization layers. The third convolutional layer has $384$ kernels of size $3 	imes 3 	imes 256$ connected to the (normalized, pooled) outputs of the second convolutional layer. The fourth convolutional layer has $384$ kernels of size $3 	imes 3 	imes 192$, and the fifth convolutional layer has $256$ kernels of size $3 	imes 3 	imes 192$. The fully-connected layers have $4096$ neurons each.

4 Reducing Overfitting

Our neural network architecture has 60 million parameters. Although the 1000 classes of ILSVRC make each training example impose 10 bits of constraint on the mapping from image to label, this turns out to be insufficient to learn so many parameters without considerable overfitting. Below, we describe the two primary ways in which we combat overfitting.

4.1 Data Augmentation

The easiest and most common method to reduce overfitting on image data is to artificially enlarge the dataset using label-preserving transformations (e.g., [25, 4, 5]). We employ two distinct forms of data augmentation, both of which allow transformed images to be produced from the original images with very little computation, so the transformed images do not need to be stored on disk. In our implementation, the transformed images are generated in Python code on the CPU while the GPU is training on the previous batch of images. So these data augmentation schemes are, in effect, computationally free.

The first form of data augmentation consists of generating image translations and horizontal reflections. We do this by extracting random $224 	imes 224$ patches (and their horizontal reflections) from the $256 	imes 256$ images and training our network on these extracted patches. This increases the size of our training set by a factor of 2048, though the resulting training examples are, of course, highly inter-dependent. Without this scheme, our network suffers from substantial overfitting, which would have forced us to use much smaller networks. At test time, the network makes a prediction by extracting five $224 	imes 224$ patches (the four corner patches and the center patch) as well as their horizontal reflections (hence ten patches in all), and averaging the predictions made by the network’s softmax layer on the ten patches.

The second form of data augmentation consists of altering the intensities of the RGB channels in training images. Specifically, we perform PCA on the set of RGB pixel values throughout the ImageNet training set. To each training image, we add multiples of the found principal components.

4 This is the reason why the input images in Figure 2 are $224 	imes 224$-dimensional.

## Impressive ImageNet results!

1000-way image classification

<table>
<thead>
<tr>
<th>Method</th>
<th>Top-5 error</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fisher Vectors (ISI)</td>
<td>26.2%</td>
</tr>
<tr>
<td>5 SuperVision CNNs</td>
<td>16.4%</td>
</tr>
<tr>
<td>7 SuperVision CNNs</td>
<td><strong>15.3%</strong></td>
</tr>
</tbody>
</table>

metric: classification error rate  (lower is better)

now: ~12%

But... does it generalize to other datasets and tasks?
[See also: DeCAF. Donahue et al., ICML 2014.]

Spirited debate at ECCV 2012
Bounding-box regression

Original:

Predicted:

\[
\begin{align*}
\Delta w \times w + w \\
\Delta h \times h + h \\
\end{align*}
\]

\[
\begin{align*}
\Delta x \times w + x, \\
\Delta y \times h + h \\
\end{align*}
\]
Comparison with DPM

R-CNN
- Cor: 65%
- Loc: 13%
- Sim: 18%
- Oth: 1%
- BG: 4%

DPM v5
- Cor: 25%
- Loc: 12%
- Sim: 43%
- Oth: 9%
- BG: 12%

animals
Localization errors dominate

**animals**
- Cor: 65%
- Loc: 13%
- Sim: 18%
- Oth: 1%
- BG: 4%

**dissimilar classes**
- Cor: 70%
- Loc: 15%
- Sim: 6%
- Oth: 3%
- BG: 6%

**similar classes**
- Cor: 52%
- Loc: 18%
- Sim: 8%
- Oth: 11%
- BG: 10%

**furniture**
- Cor: 70%
- Loc: 15%
- Sim: 6%
- Oth: 3%
- BG: 6%

Analysis software: D. Hoiem, Y. Chodpathumwan, and Q. Dai.