Deep Object Recognition

Frank Dellaert
CS 6476 Computer Vision at Georgia Tech

Several Slides by Dhruv Bathra, James Hays, Kaiming He, and others
Dataset: ImageNet 2012

Deng et al. “Imagenet: a large scale hierarchical image database” CVPR 2009
Examples of hammer:
AlexNet

- CNN by Alex Krizhevsky, Ilya Sutskever and Geoffrey Hinton
- Competed in the ImageNet Large Scale Visual Recognition Challenge on September 30, 2012. Achieved a top-5 error of 15.3%, beating SOTA by 10%.
- Seen by many as the start of the DL revolution in CV.
- That claim is contested by Jürgen Schmidhuber, whose postdoc Dan Ciresan published a similar result in IJCAI 2011 (but on easier datasets).
- Both owe a debt to Fukushima, who invented CNNs in 1980, and Yann LeCun, who applied backprop to CNNs in 89.
Architecture for Classification

category prediction

LINEAR

FULLY CONNECTED

FULLY CONNECTED

MAX POOLING

CONV

CONV

CONV

MAX POOLING

LOCAL CONTRAST NORM

CONV

MAX POOLING

LOCAL CONTRAST NORM

CONV

input
Architecture for Classification

Total nr. params: 60M

4M  LINEAR  4M
16M FULLY CONNECTED  16M
37M FULLY CONNECTED  37M

Max Pooling
442K CONV  74M
1.3M CONV  224M
884K CONV  149M

Max Pooling
307K LOCAL CONTRAST NORM  223M

Max Pooling
35K LOCAL CONTRAST NORM  105M

Total nr. flops: 832M

Krizhevsky et al. “ImageNet Classification with deep CNNs” NIPS 2012
Optimization

SGD with momentum:
- Learning rate = 0.01
- Momentum = 0.9

Improving generalization by:
- Weight sharing (convolution)
- Input distortions
- Dropout = 0.5
- Weight decay = 0.0005
Results: ILSVRC 2012

**TASK 1 - CLASSIFICATION**

- CNN
- SIFT+FV
- SVM1
- SVM2
- NCM

**TASK 2 - DETECTION**

- CNN
- DPM-SVM1
- DPM-SVM2

Krizhevsky et al. “ImageNet Classification with deep CNNs” NIPS 2012
Beyond AlexNet
These are the “VGG” networks.
<table>
<thead>
<tr>
<th>ConvNet config. (Table 1)</th>
<th>smallest image side</th>
<th>top-1 val. error (%)</th>
<th>top-5 val. error (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>train (S)</td>
<td>test (Q)</td>
<td></td>
</tr>
<tr>
<td>B</td>
<td>256</td>
<td>224,256,288</td>
<td>28.2</td>
</tr>
<tr>
<td>C</td>
<td>256</td>
<td>224,256,288</td>
<td>27.7</td>
</tr>
<tr>
<td></td>
<td>384</td>
<td>352,384,416</td>
<td>27.8</td>
</tr>
<tr>
<td></td>
<td>[256; 512]</td>
<td>256,384,512</td>
<td>26.3</td>
</tr>
<tr>
<td>D</td>
<td>256</td>
<td>224,256,288</td>
<td>26.6</td>
</tr>
<tr>
<td></td>
<td>384</td>
<td>352,384,416</td>
<td>26.5</td>
</tr>
<tr>
<td></td>
<td>[256; 512]</td>
<td>256,384,512</td>
<td><strong>24.8</strong></td>
</tr>
<tr>
<td>E</td>
<td>256</td>
<td>224,256,288</td>
<td>26.9</td>
</tr>
<tr>
<td></td>
<td>384</td>
<td>352,384,416</td>
<td>26.7</td>
</tr>
<tr>
<td></td>
<td>[256; 512]</td>
<td>256,384,512</td>
<td><strong>24.8</strong></td>
</tr>
</tbody>
</table>
This is the “Inception” architecture or “GoogLeNet”

*The architecture blocks are called “Inception” modules and the collection of them into a particular net is “GoogLeNet”
(a) Inception module, naïve version

(b) Inception module with dimensionality reduction
1x1 Convolutions

- Linearly reduce a set of \( n \) features to a set of \( m \) features. Example: \( 192 \rightarrow 32 \)
- I.e., matrix multiplication with \( m \times n \) matrix, at each location (32x192 in example: 32 “bases”)
- Typically followed by ReLU
<table>
<thead>
<tr>
<th>type</th>
<th>patch size/stride</th>
<th>output size</th>
<th>depth</th>
<th>#1×1</th>
<th>#3×3 reduce</th>
<th>#3×3 reduce</th>
<th>#5×5</th>
<th>#5×5 proj</th>
<th>params</th>
<th>ops</th>
</tr>
</thead>
<tbody>
<tr>
<td>convolution</td>
<td>7×7/2</td>
<td>112×112×64</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2.7K</td>
<td>34M</td>
</tr>
<tr>
<td>max pool</td>
<td>3×3/2</td>
<td>56×56×64</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>112K</td>
<td>360M</td>
</tr>
<tr>
<td>convolution</td>
<td>3×3/1</td>
<td>56×56×192</td>
<td>2</td>
<td>64</td>
<td>192</td>
<td></td>
<td></td>
<td></td>
<td>159K</td>
<td>128M</td>
</tr>
<tr>
<td>max pool</td>
<td>3×3/2</td>
<td>28×28×192</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>380K</td>
<td>304M</td>
</tr>
<tr>
<td>inception (3a)</td>
<td>28×28×256</td>
<td>2</td>
<td>64</td>
<td>96</td>
<td>128</td>
<td>16</td>
<td>32</td>
<td>32</td>
<td>364K</td>
<td>73M</td>
</tr>
<tr>
<td>inception (3b)</td>
<td>28×28×480</td>
<td>2</td>
<td>128</td>
<td>128</td>
<td>192</td>
<td>32</td>
<td>96</td>
<td>64</td>
<td>437K</td>
<td>88M</td>
</tr>
<tr>
<td>max pool</td>
<td>3×3/2</td>
<td>14×14×480</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>463K</td>
<td>100M</td>
</tr>
<tr>
<td>inception (4a)</td>
<td>14×14×512</td>
<td>2</td>
<td>192</td>
<td>96</td>
<td>208</td>
<td>16</td>
<td>48</td>
<td>64</td>
<td>580K</td>
<td>119M</td>
</tr>
<tr>
<td>inception (4b)</td>
<td>14×14×512</td>
<td>2</td>
<td>160</td>
<td>112</td>
<td>224</td>
<td>24</td>
<td>64</td>
<td>64</td>
<td>840K</td>
<td>170M</td>
</tr>
<tr>
<td>inception (4c)</td>
<td>14×14×512</td>
<td>2</td>
<td>128</td>
<td>128</td>
<td>256</td>
<td>24</td>
<td>64</td>
<td>64</td>
<td>1072K</td>
<td>54M</td>
</tr>
<tr>
<td>inception (4d)</td>
<td>14×14×528</td>
<td>2</td>
<td>112</td>
<td>144</td>
<td>288</td>
<td>32</td>
<td>64</td>
<td>64</td>
<td>1388K</td>
<td>71M</td>
</tr>
<tr>
<td>inception (4e)</td>
<td>14×14×832</td>
<td>2</td>
<td>256</td>
<td>160</td>
<td>320</td>
<td>32</td>
<td>128</td>
<td>128</td>
<td>1000K</td>
<td>1M</td>
</tr>
<tr>
<td>max pool</td>
<td>3×3/2</td>
<td>7×7×832</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>inception (5a)</td>
<td>7×7×832</td>
<td>2</td>
<td>256</td>
<td>160</td>
<td>320</td>
<td>32</td>
<td>128</td>
<td>128</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>inception (5b)</td>
<td>7×7×1024</td>
<td>2</td>
<td>384</td>
<td>192</td>
<td>384</td>
<td>48</td>
<td>128</td>
<td>128</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>avg pool</td>
<td>7×7/1</td>
<td>1×1×1024</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000K</td>
<td>1M</td>
</tr>
<tr>
<td>dropout (40%)</td>
<td>1×1×1024</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>linear</td>
<td>1×1×1000</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000K</td>
<td>1M</td>
</tr>
<tr>
<td>softmax</td>
<td>1×1×1000</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

GoogLeNet: Only 6.8 million parameters. AlexNet ~60 million, VGG up to 138 million
Results

- ILSVRC 2014:

<table>
<thead>
<tr>
<th>Team</th>
<th>Year</th>
<th>Place</th>
<th>Error (top-5)</th>
<th>Uses external data</th>
</tr>
</thead>
<tbody>
<tr>
<td>SuperVision</td>
<td>2012</td>
<td>1st</td>
<td>16.4%</td>
<td>no</td>
</tr>
<tr>
<td>SuperVision</td>
<td>2012</td>
<td>1st</td>
<td>15.3%</td>
<td>Imagenet 22k</td>
</tr>
<tr>
<td>Clarifai</td>
<td>2013</td>
<td>1st</td>
<td>11.7%</td>
<td>no</td>
</tr>
<tr>
<td>Clarifai</td>
<td>2013</td>
<td>1st</td>
<td>11.2%</td>
<td>Imagenet 22k</td>
</tr>
<tr>
<td>MSRA</td>
<td>2014</td>
<td>3rd</td>
<td>7.35%</td>
<td>no</td>
</tr>
<tr>
<td>VGG</td>
<td>2014</td>
<td>2nd</td>
<td>7.32%</td>
<td>no</td>
</tr>
<tr>
<td>GoogLeNet</td>
<td>2014</td>
<td>1st</td>
<td>6.67%</td>
<td>no</td>
</tr>
</tbody>
</table>

Table 2: Classification performance.
Revolution of Depth

AlexNet, 8 layers
(ILSVRC 2012)

VGG, 19 layers
(ILSVRC 2014)

GoogleNet, 22 layers
(ILSVRC 2014)

Surely it would be ridiculous to go any deeper...

Introducing: ResNet

AlexNet, 8 layers (ILSVRC 2012)  VGG, 19 layers (ILSVRC 2014)  ResNet, 152 layers (ILSVRC 2015)

Revolution of Depth

ImageNet Classification top-5 error (%)

Revolution of Depth

Engines of visual recognition

HOG, DPM 34 layers
AlexNet (RCNN) 58 layers
VGG (RCNN) 66 layers
ResNet (Faster RCNN) 101 layers

PASCAL VOC 2007 Object Detection mAP (%)

86

*w/ other improvements & more data

Deep Residual Learning

• $F(x)$ is a *residual* mapping w.r.t. identity

$H(x) = F(x) + x$

• If identity were optimal, easy to set weights as 0

• If optimal mapping is closer to identity, easier to find small fluctuations

Deep ResNets can be trained without difficulties

Deeper ResNets have **lower training error**, and also lower test error

ResNets @ ILSVRC & COCO 2015 Competitions

- 1st places in all five main tracks
  - ImageNet Classification: “Ultra-deep” 152-layer nets
  - ImageNet Detection: 16% better than 2nd
  - ImageNet Localization: 27% better than 2nd
  - COCO Detection: 11% better than 2nd
  - COCO Segmentation: 12% better than 2nd
- 57K citations (in 6 years)

*improvements are relative numbers
Object Detection Architectures

Image Classification (what?)

Object Detection (what + where?)
Object Detection: Early Work

Szegedy et al. “DNN for object detection” NIPS 2013
Object Detection: R-CNN

Region-based CNN pipeline

Proposals by “Selective search” algorithm (2013)
Two “heads”:
• classifier
• BB regressor


Object Detection: R-CNN

• R-CNN
Object Detection: Fast R-CNN

- Fast R-CNN

pre-computed Regions-of-Interest (RoIs) → CNN

shared conv layers

feature features

RoI pooling

End-to-End training

Girshick. Fast R-CNN. ICCV 2015
Object Detection: Fast R-CNN

- Fast R-CNN
Object Detection: Faster R-CNN

• Introduces “Region Proposal Networks” (RPNs)
• Solely based on CNN: use for classification and regions
• Each step is end-to-end

Region Proposal Nets in Faster R-CNN

- In paper: $k=9$ (3 scales, 3 aspect ratios)
- Sibling objectness ($2k$) and BB regression($4k$) outputs

Object Detection

ImageNet data → backbone structure → pre-train classification network → features → detection network → fine-tune detection data

- AlexNet
- VGG-16
- GoogleNet
- ResNet-101

“plug-in” features → independently developed detectors

- R-CNN
- Fast R-CNN
- Faster R-CNN
- MultiBox
- SSD
- ...

Slide by Kaiming He,
Faster R-CNN w Resnet

• Simply “Faster R-CNN + ResNet”

<table>
<thead>
<tr>
<th>CNN</th>
<th>mAP@.5</th>
<th>mAP@.5:.95</th>
</tr>
</thead>
<tbody>
<tr>
<td>VGG-16</td>
<td>41.5</td>
<td>21.5</td>
</tr>
<tr>
<td>ResNet-101</td>
<td>48.4</td>
<td>27.2</td>
</tr>
</tbody>
</table>

COCO detection results
ResNet-101 has 28% relative gain vs VGG-16

Faster R-CNN Efficiency

- Expensive “Selective Search” is gone
Object Detection

• RPN learns proposals by extremely deep nets
  • Uses only 300 proposals (no hand-designed proposals)

• Add components:
  • Iterative localization
  • Context modeling
  • Multi-scale testing

• All are based on CNN features; all are end-to-end

• All benefit more from deeper features – cumulative gains!

ResNet’s object detection result on COCO

Results on real video. Models trained on MS COCO (80 categories).
(frame-by-frame; no temporal processing)

disk video is available online: https://youtu.be/WZmSMkK9VuA

More Visual Recognition Tasks

ResNet-based methods lead on these benchmarks (incomplete list):

- ImageNet classification, detection, localization
- MS COCO detection, segmentation
- PASCAL VOC detection, segmentation
- Depth estimation [Laina et al 2016]
- Segment proposal [Pinheiro et al 2016]
- ...

Slide by Kaiming He,
• 7x7 grid of cells
• Each grid cell predicts
  – 2 bounding boxes (x, y, w, h, confidence) = 10 reals
  – Probabilities over 20 classes
• Final output: 7x7x30 tensor (30 = 5+5+20)
• “the fastest extant object detector” at CVPR 2016
Yolo 1,2,3...

- YOLO: CVPR 2016
- YOLO9000 (YOLOv2) = CVPR 2017
- YOLOv3: Arxiv 2018

Quotes from “YOLOv3: An Incremental Improvement”:
Sometimes you just kinda phone it in for a year, you know? … Spent a lot of time on Twitter. Played around with GANs a little.
Detectron includes implementations of the following:

- **Mask R-CNN** — *Marr Prize at ICCV 2017*
- **RetinaNet** — *Best Student Paper Award at ICCV 2017*
- **Faster R-CNN**
- **RPN**
- **Fast R-CNN**
- **R-FCN**

using the following backbone network architectures:

- **ResNeXt\{50,101,152\}**
- **ResNet\{50,101,152\}**
- **Feature Pyramid Networks** (with ResNet/ResNeXt)
- **VGG16**