Object Recognition in Images and Video: CNNs

http://www.micc.unifi.it/bagdanov/obrec

Prof. Andrew D. Bagdanov

Dipartimento di Ingegneria dell’Informazione
Università degli Studi di Firenze
andrew.bagdanov AT unifi.it

27 April 2017
Outline

1. Overview
2. Deformable Part Models
3. Setting the Stage
4. CNNs: AlexNet
5. CNNs: Very Deep Networks
6. CNNs: Even Deeper Networks
7. CNNs: Fast-RCNN
8. Deeper, Bigger, and Better
9. Discussion
Overview
Overview

- In this lesson I will first do a quick overview of the Deformable Part Models detector.
- Then, I will present four works that radically changed the face of object recognition.
- These new models are based on a specific type of Neural Network.
- Specifically, they are based on Convolutional Neural Networks (CNNs).
- I will begin the discussion of CNNs with some motivating examples that demonstrate the basic building blocks of CNNs.
- We will see how this model (sort of) arises naturally from the Bag-of-Words model.
- And we will see how it has radically changed the landscape of object recognition.
Deformable Part Models
Deformable Part Models

[OTHER PRESENTATION]
Setting the Stage
Setting the Stage

- After ten years of incremental improvements of the Bag-of-Words model, we found ourselves reaching a sort of asymptote.
- It was unclear what would be the Next Big Thing in object recognition.
- The state-of-the-art in 2012 was: Fisher Vectors + Multiple Cues + Late Fusion (summing scores).
- Meanwhile, since the 1990s there was a competing paradigm for object recognition based on Neural Networks.
- This method for object recognition had been developed in the 1990s for character recognition.
- However, it never really broke through onto the object recognition scene.
- This is due to a number of factors that are important to understand.
Let’s look at a simple **Neural Network** architecture known as the Multilayer Perceptron (MLP):
Setting the Stage

- The MLP equation (one hidden layer):

  \[ \hat{y}(x) = \sigma(w_2^T \sigma(w_1^T z + b_1) + b_2) \]

- Except for the activation function \( \sigma \), this is a linear system.

- Common activation functions (elementwise):
  - \( \sigma(x) = \tanh(x) \)
  - \( \sigma(x) = (1 + e^{-x})^{-1} \)
  - \( \sigma(x) = \frac{\exp(x)}{\sum_i e^{x_i}} \)

![Graphs of common activation functions]
How do you train a model?

**Decide on a loss function:**

\[
L(y, \hat{y}(x)) = \frac{1}{C} \sum_i y_i \log(\hat{y}_i)
\]

And perform **gradient descent w.r.t. all model parameters:**

\[
\theta_{n+1} = \theta_n - \varepsilon \nabla_{\theta} L(y, \hat{y}(x))
\]

\[
\theta_{n+1} = \theta_n - \varepsilon \sum_{i=1}^{N} \frac{1}{N} \nabla_{\theta} L(y, \hat{y}(x_i))
\]

Where \( \varepsilon \) is the **learning rate**.

The standard algorithm for this is known as **backpropagation** and it is very clever and efficient.
Problems with this approach:

- **Model size**: many, many parameters for even small-sized images. This leads to memory and efficiency problems.
- **Overfitting**: many parameters (and limited training data) mean that it is easy to overfit the model to your training set.
- **Undergeneralization**: overfitting means that a trained model is unlikely to generalize to new data.
- **Vanishing gradients**: a known problem with backpropagation (due to application of the chain rule) leads to very small gradient values near the beginning of the network.
- **Saturating units**: traditional activation functions can lead to saturated units (outputs near 1 or 0 (or -1)), which have near-zero derivatives.

These problems (and others) led the community to largely ignore the potential of these models for decades.
Nonetheless, there were staunch supporters of this paradigm.
Then, one day in 2010 (or 2011), the following conversation took place...
Setting the Stage

Then, at the ImageNet competition workshop at ECCV 2012 (right here in Florence!):

```
Task 1

<table>
<thead>
<tr>
<th>Team name</th>
<th>Filename</th>
<th>Error (5 guesses)</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SuperVision</td>
<td>test-preds-131-137-145-135-145f.txt</td>
<td>0.16422</td>
<td>Using only supplied training data</td>
</tr>
<tr>
<td>ISI</td>
<td>pred_FVs_wLACs_weighted.txt</td>
<td>0.26172</td>
<td>Weighted sum of scores from each classifier with SIFT+FV, LBP+FV, GIST+FV, and CSIFT+FV, respectively.</td>
</tr>
</tbody>
</table>
```
CNNs: AlexNet
AlexNet: Introduction

- We will now take a look at the International Large Scale Visual Recognition Competition (ILSVRC) submission that **changed everything**:


- In this paper the authors defined a convolutional network architecture that became the New Standard.
- This architecture systematically addresses **most** of the problems with training large network architectures.
- It is a **Convolutional Neural Network (CNN)** that is universally called AlexNet.
- It is also a **Deep Network** because it has many hidden layers.
- Hence the term **Deep Learning**.
AlexNet: The Architecture

- Let’s look first at the overall architecture and then analyze in detail how each component addresses specific problems.
- It is also helpful to examine how data flows through the network.
AlexNet: Sharing Weights

- The early layers of the network are convolutional.
- This means that the weights are shared across locations of the image.
- The input of size $w \times h \times d$ is transformed into an output of size $w \times h \times d'$.
- The outputs are called feature maps and they are derived by convolving the image with a 3D tensor of size $u \times v \times d'$.
- So, the number of parameters is “merely” $u \times v \times d' + d'$.
- The output feature maps can be very large however.
Like in the Bag-of-Words model, we can pool local features.

In AlexNet, they authors use $3 \times 3$ pooling regions with a stride of 2 pixels.

This means that after some convolutional layers the feature map size is reduced by a factor of 2.

They use max pooling: in each feature map, keep the maximum value in each overlapping $3 \times 3$ pooling region.

This helps to contain the size of feature maps propagated through the network.

And it also helps to build higher-level representations of the image.

This is because, halving the image resolution is the same as doubling the size of subsequent convolutions.
AlexNet: Unit Saturation

- Another innovation in AlexNet is the use of the Rectified Linear Unit (ReLU) activation function.

\[ \sigma(x) = \max(0, x) \]

- This activation function does not saturate like sigmoids.
- The result is a 6x speedup in training.
AlexNet: Reducing Overfitting

- Even with convolutional weight sharing, AlexNet still has 60M parameters.
- To reduce overfitting, the authors use two extra (now standard) tricks:
  - **Data augmentation**: random translations and reflections of input images are generated, plus random variation in principal directions of RGB space.
  - **Dropout**: an advanced trick from the Neural Network community which randomly removes half of the inputs to select layers at training time.
The AlexNet paper is an excellent resource because it explains all of the tricks necessary to get a deep network to learn:

- **Local response normalization:** keep local variation in feature maps under control (section 3.3).
- **Momentum:** limits the “skateboard” effect when following valleys in the loss surface, equivalent to L1 (or L2) regularization of weights (section 5).
- **Mini-batch Stochastic Gradient Descent (SGD):** with 1.2M training samples, we cannot consider the entire dataset in one batch; instead, randomly sample mini-batches of 128 images (section 5).
- **Multiple GPUs:** AlexNet was too big to fit in a single GPU (in 2012), so feature maps are split over two GPUs (section 3.2).
- **Model averaging:** state-of-the-art results are obtained by training multiple CNNs and averaging outputs.
AlexNet: Results

- The proof is in the pudding:

<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>

- And in the **representations** the network learns:
AlexNet: BOW vs. DCNNs

- **Feature detection:**
  - In BOW, we use **handcrafted** features as input to **pooling** and finally **classification** layers.
  - In DCNNs, we **learn** convolutional features, which are then pooled, and then shoved into classification layers.

- **Local feature pooling:**
  - In BOW we use **spatial pooling** to add structure to our final representation (Spatial Pyramids).
  - In DCNNs, we use **max pooling** to reduce feature map size and create higher-level features.

- **Global feature pooling:**
  - In BOW, we compute a **global** image representation via **pooling**.
  - In DCNNs, we compute a **global** image representation via **fully-connected** (sometimes called **dense**) layers.

- **Training:**
  - In BOW, we use **handcrafted representations**, followed by shallow classifier learning (e.g. an SVM).
  - In DCNNs, we perform **end-to-end** training of the **entire** architecture.
AlexNet: Reflections

- AlexNet took the object recognition world by storm.
- Many of the elements of the model are not really new.
- However, this was the first work to *convincingly* demonstrate how state-of-the-art object recognition systems can be trained end-to-end on real problems.
- This was made possible by a number of confluent development:
  - The availability of enormous amounts of annotate data (ImageNet, with 1.2M training images).
  - Modern GPUs, which make convolutions super fast.
  - Decades of persistent theoretical development (ReLUs, fast backprop, dropout, etc).
CNNs: Very Deep Networks
Very Deep Networks

- Very soon after the ILSVRC 2012 results, the community began experimenting with newer, deeper architectures for CNNs.
- We will look at an architecture that became (and still is) a standard one.


- In this paper the authors performed a thorough exploration of the architectural parameter space.
- They varied the hyperparameters (e.g. number of layers, size of convolutions, etc).
- And established a new baseline for CNN-based object recognition.
- These networks are known as VGG16 and VGG19 (VGG = Visual Geometry Group from Oxford).
VGG: The Setup

- Input to the networks is a fixed, $224 \times 224 \times 3$ image tensor.
- The mean RGB value is first subtracted from all training images to center the data.
- All convolutions are $3 \times 3 \times d$ or $1 \times 1 \times d$ in size ($d$ is an arbitrary number of feature maps) with a stride of 1.
- The idea is: if you need larger convolutions, just go deeper.
- Max pooling is done over non-overlapping $2 \times 2$ windows with a stride of 2 ($2x$ reduction in size).
- All hidden layer use a ReLU activation function, but do not do local response normalization.
The following configurations were considered:

<table>
<thead>
<tr>
<th>ConvNet Configuration</th>
<th>A</th>
<th>A-LRN</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>11 weight layers</td>
<td>11 weight layers</td>
<td>13 weight layers</td>
<td>16 weight layers</td>
<td>16 weight layers</td>
<td>19 weight layers</td>
</tr>
<tr>
<td>input (224 × 224 RGB image)</td>
<td>conv3-64</td>
<td>conv3-64</td>
<td>conv3-64</td>
<td>conv3-64</td>
<td>conv3-64</td>
<td>conv3-64</td>
</tr>
<tr>
<td>maxpool</td>
<td>conv3-128</td>
<td>conv3-128</td>
<td>conv3-128</td>
<td>conv3-128</td>
<td>conv3-128</td>
<td>conv3-128</td>
</tr>
<tr>
<td>maxpool</td>
<td>conv3-256</td>
<td>conv3-256</td>
<td>conv3-256</td>
<td>conv3-256</td>
<td>conv3-256</td>
<td>conv3-256</td>
</tr>
<tr>
<td>maxpool</td>
<td>conv3-512</td>
<td>conv3-512</td>
<td>conv3-512</td>
<td>conv3-512</td>
<td>conv3-512</td>
<td>conv3-512</td>
</tr>
<tr>
<td>maxpool</td>
<td>conv3-1000</td>
<td>soft-max</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
The training procedure is similar to AlexNet:
- The training is carried out by optimising the multinomial logistic regression objective using mini-batch gradient descent.
- The batch size is 256, with momentum of 0.9
- Training was regularised by weight decay (the L2 penalty multiplier set to $10^4$ and dropout on the first two fully-connected layers).
- The learning rate was initially set to $10^2$ and decreased by a factor of 10 when the validation set accuracy stopped improving.
- Learning was stopped after 370K iterations (74 epochs).

Initialization:
- CNNs are extremely sensitive to initialization of the weights.
- For training VGG networks, the authors use a combination of random initialization and pre-training.
Note that training images are all scaled to $224 \times 224$ pixels before passing them through the network.

This is the same as AlexNet, and clearly can affect the image content by introducing artifacts (consider portrait images).

In VGG networks, images are isotropically scaled so that the smallest dimension has fixed size.

Then subimage of size $224 \times 224$ is randomly cropped from the scaled image.

The authors evaluated randomly scaling to between 256 and 384 pixels for the smallest dimension.
At testing time, there are five strategies for image scaling evaluated (and their combinations):

- **Dense**: the network is fully convolutionalized (I will explain this on the next slide), evaluated densely on the input image, and results are globally pooled.
- **Single-scale**: a single isotropic scale is used.
- **Multi-scale**: like at training time, images are scaled to three discrete isotropic scales.
- **Multi-crop**: multiple, random crops are taken from the fully-convolutional output for average pooling.
Below is a diagram of a typical ConvNet.

How can we make it independent of the image size?
VGG: Results

- Even when only considering a single input scale, the results are impressive.
- **Note:** deeper is better, LRN doesn’t help, *scale jittering* at test time does.

<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>A</td>
<td>256</td>
<td>256</td>
<td>29.6</td>
</tr>
<tr>
<td>A-LRN</td>
<td>256</td>
<td>256</td>
<td>29.7</td>
</tr>
<tr>
<td>B</td>
<td>256</td>
<td>256</td>
<td>28.7</td>
</tr>
<tr>
<td>C</td>
<td>256</td>
<td>256</td>
<td>28.1</td>
</tr>
<tr>
<td></td>
<td>384</td>
<td>384</td>
<td>28.1</td>
</tr>
<tr>
<td></td>
<td>[256;512]</td>
<td>384</td>
<td>27.3</td>
</tr>
<tr>
<td>D</td>
<td>256</td>
<td>256</td>
<td>27.0</td>
</tr>
<tr>
<td></td>
<td>384</td>
<td>384</td>
<td>26.8</td>
</tr>
<tr>
<td></td>
<td>[256;512]</td>
<td>384</td>
<td>25.6</td>
</tr>
<tr>
<td>E</td>
<td>256</td>
<td>256</td>
<td>27.3</td>
</tr>
<tr>
<td></td>
<td>384</td>
<td>384</td>
<td>26.9</td>
</tr>
<tr>
<td></td>
<td>[256;512]</td>
<td>384</td>
<td>25.5</td>
</tr>
</tbody>
</table>
VGG: Results

- Using **multiple scales** leads to even better performance:

<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>[256; 512]</td>
<td>256,384,512</td>
<td>26.3</td>
<td>8.2</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>[256; 512]</td>
<td>256,384,512</td>
<td><strong>24.8</strong></td>
<td>7.5</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>[256; 512]</td>
<td>256,384,512</td>
<td><strong>24.8</strong></td>
<td>7.5</td>
</tr>
</tbody>
</table>

- As does **fusing** multiple cropping strategies:

<table>
<thead>
<tr>
<th>ConvNet config. (Table 1)</th>
<th>Evaluation method</th>
<th>top-1 val. error (%)</th>
<th>top-5 val. error (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>D</td>
<td>dense</td>
<td>24.8</td>
<td>7.5</td>
</tr>
<tr>
<td></td>
<td>multi-crop</td>
<td>24.6</td>
<td>7.5</td>
</tr>
<tr>
<td></td>
<td>multi-crop &amp; dense</td>
<td><strong>24.4</strong></td>
<td>7.2</td>
</tr>
<tr>
<td>E</td>
<td>dense</td>
<td>24.8</td>
<td>7.5</td>
</tr>
<tr>
<td></td>
<td>multi-crop</td>
<td>24.6</td>
<td>7.4</td>
</tr>
<tr>
<td></td>
<td>multi-crop &amp; dense</td>
<td><strong>24.4</strong></td>
<td><strong>7.1</strong></td>
</tr>
</tbody>
</table>
Finally, model averaging over multi-scale, multi-crop models leads to state-of-the-art performance:

<table>
<thead>
<tr>
<th>Method</th>
<th>top-1 val. error (%)</th>
<th>top-5 val. error (%)</th>
<th>top-5 test error (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>VGG (2 nets, multi-crop &amp; dense eval.)</td>
<td>23.7</td>
<td>6.8</td>
<td>6.8</td>
</tr>
<tr>
<td>VGG (1 net, multi-crop &amp; dense eval.)</td>
<td>24.4</td>
<td>7.1</td>
<td>7.0</td>
</tr>
<tr>
<td>VGG (ILSVRC submission, 7 nets, dense eval.)</td>
<td>24.7</td>
<td>7.5</td>
<td>7.3</td>
</tr>
<tr>
<td>GoogLeNet (Szegedy et al., 2014) (1 net)</td>
<td>-</td>
<td>-</td>
<td>7.9</td>
</tr>
<tr>
<td>GoogLeNet (Szegedy et al., 2014) (7 nets)</td>
<td>-</td>
<td>-</td>
<td>6.7</td>
</tr>
<tr>
<td>MSRA (He et al., 2014) (11 nets)</td>
<td>-</td>
<td>-</td>
<td>8.1</td>
</tr>
<tr>
<td>MSRA (He et al., 2014) (1 net)</td>
<td>27.9</td>
<td>9.1</td>
<td>9.1</td>
</tr>
<tr>
<td>Clarifai (Russakovsky et al., 2014) (multiple nets)</td>
<td>-</td>
<td>-</td>
<td>11.7</td>
</tr>
<tr>
<td>Clarifai (Russakovsky et al., 2014) (1 net)</td>
<td>-</td>
<td>-</td>
<td>12.5</td>
</tr>
<tr>
<td>Zeiler &amp; Fergus (Zeiler &amp; Fergus, 2013) (6 nets)</td>
<td>36.0</td>
<td>14.7</td>
<td>14.8</td>
</tr>
<tr>
<td>Zeiler &amp; Fergus (Zeiler &amp; Fergus, 2013) (1 net)</td>
<td>37.5</td>
<td>16.0</td>
<td>16.1</td>
</tr>
<tr>
<td>OverFeat (Sermanet et al., 2014) (7 nets)</td>
<td>34.0</td>
<td>13.2</td>
<td>13.6</td>
</tr>
<tr>
<td>OverFeat (Sermanet et al., 2014) (1 net)</td>
<td>35.7</td>
<td>14.2</td>
<td>-</td>
</tr>
<tr>
<td>Krizhevsky et al. (Krizhevsky et al., 2012) (5 nets)</td>
<td>38.1</td>
<td>16.4</td>
<td>16.4</td>
</tr>
<tr>
<td>Krizhevsky et al. (Krizhevsky et al., 2012) (1 net)</td>
<td>40.7</td>
<td>18.2</td>
<td>-</td>
</tr>
</tbody>
</table>
VGG: Analysis

- The **VGG16** and **VGG19** networks are still in common use today (though used in novel ways).
- In this paper the authors significantly improved over the previous generation by going deeper.
- Again, most of the ideas are not new, but systematic exploration of the design space led to significant improvements.
- Note that the networks are deeper, but have a smaller memory footprint at training time due to carefully balancing the size of feature maps.
- **Dense** evaluation of the network at test time can also increase performance, leading to fully convolutional networks that are independent of input image size.
- The architecture is still a classical ConvNet.
CNNs: Even Deeper Networks
Of course, Google had to jump into the game.

We will now consider a different architecture for CNNs, one that will allow us to go even deeper:


This will use the idea of fully convolutional networks to both go deeper and to limit size of intermediate feature maps.
GoogLeNet: Observations

- **Trend**: bigger and deeper networks (and multiple models at that).

- **Problems**:
  - **Problem 1**: bigger networks need a lot more annotated training data, which is extremely expensive to collect.
  - **Problem 2**: hardware resources are finite, including memory and CPU cycles.
  - **CPU hog**: convolutional layers over many feature maps.

- **Ideas**:
  - Use multiple, multi-resolution convolutions at each layer to better capture local structure.
  - Use fully-convolutional layers (i.e. $1 \times 1 \times d$) convolutions to *reduce feature map dimensionality* before expensive convolutions.
Thus, the Inception Module was born:

- The naive module concatenates multiple feature map representations at each level.
- This includes expensive 5x5 and 3x3 convolutions.
- The full Inception Module applies 1x1 convolutions to reduce dimensionality by first convolving with 1x1 filters.
GoogLeNet: Putting it Together

- The GoogLeNet name is an homage to the first ConvNet proposed by Yann LeCun in 1989.

<table>
<thead>
<tr>
<th>type</th>
<th>patch size/</th>
<th>output size</th>
<th>depth</th>
<th>#1x1</th>
<th>#3x3 reduce</th>
<th>#5x5 reduce</th>
<th>pool proj</th>
<th>params</th>
<th>ops</th>
</tr>
</thead>
<tbody>
<tr>
<td>convolution</td>
<td>7x7/2</td>
<td>112x112x64</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2.7K</td>
<td>34M</td>
</tr>
<tr>
<td>max pool</td>
<td>3x3/2</td>
<td>56x56x64</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>convolution</td>
<td>3x3/1</td>
<td>56x56x192</td>
<td>2</td>
<td></td>
<td>64</td>
<td>192</td>
<td></td>
<td>112K</td>
<td>360M</td>
</tr>
<tr>
<td>max pool</td>
<td>3x3/2</td>
<td>28x28x192</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>inception (3a)</td>
<td>28x28x256</td>
<td>2</td>
<td>64</td>
<td>96</td>
<td>128</td>
<td>16</td>
<td>32</td>
<td>159K</td>
<td>128M</td>
</tr>
<tr>
<td>inception (3b)</td>
<td>28x28x480</td>
<td>2</td>
<td>128</td>
<td>128</td>
<td>192</td>
<td>32</td>
<td>96</td>
<td>380K</td>
<td>304M</td>
</tr>
<tr>
<td>max pool</td>
<td>3x3/2</td>
<td>14x14x480</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>inception (4a)</td>
<td>14x14x512</td>
<td>2</td>
<td>192</td>
<td>96</td>
<td>208</td>
<td>16</td>
<td>48</td>
<td>364K</td>
<td>73M</td>
</tr>
<tr>
<td>inception (4b)</td>
<td>14x14x512</td>
<td>2</td>
<td>160</td>
<td>112</td>
<td>224</td>
<td>24</td>
<td>64</td>
<td>437K</td>
<td>88M</td>
</tr>
<tr>
<td>inception (4c)</td>
<td>14x14x512</td>
<td>2</td>
<td>128</td>
<td>128</td>
<td>256</td>
<td>24</td>
<td>64</td>
<td>463K</td>
<td>100M</td>
</tr>
<tr>
<td>inception (4d)</td>
<td>14x14x528</td>
<td>2</td>
<td>112</td>
<td>144</td>
<td>288</td>
<td>32</td>
<td>64</td>
<td>580K</td>
<td>119M</td>
</tr>
<tr>
<td>inception (4e)</td>
<td>14x14x832</td>
<td>2</td>
<td>256</td>
<td>160</td>
<td>320</td>
<td>32</td>
<td>128</td>
<td>840K</td>
<td>170M</td>
</tr>
<tr>
<td>max pool</td>
<td>3x3/2</td>
<td>7x7x832</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>inception (5a)</td>
<td>7x7x832</td>
<td>2</td>
<td>256</td>
<td>160</td>
<td>320</td>
<td>32</td>
<td>128</td>
<td>1072K</td>
<td>54M</td>
</tr>
<tr>
<td>inception (5b)</td>
<td>7x7x1024</td>
<td>2</td>
<td>384</td>
<td>192</td>
<td>384</td>
<td>48</td>
<td>128</td>
<td>1388K</td>
<td>71M</td>
</tr>
<tr>
<td>avg pool</td>
<td>7x7/1</td>
<td>1x1x1024</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dropout (4%)</td>
<td>1x1x1024</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>linear</td>
<td>1x1x1000</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000K</td>
<td>1M</td>
</tr>
<tr>
<td>softmax</td>
<td>1x1x1000</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
GoogLeNet: A Tour

- Let’s take a detailed look at the monster of a figure 3.
- First the stem.
- Then a cascade of Inception Modules.
- Then the auxiliary loss layers.
- The final output layers.
GoogLeNet: Training

- The training procedure of GoogLeNet is a complete mess.
- This is typical of the type of training that happens leading up to a competition.
- They experimented with many configurations, keeping some, discarding others.
- In any case, they use SGD on CPUs (they’re Google, they have CPUs at their disposal).
- Final results are based on a combination of 7 trained GoogLeNet variants.
- At test time they pass 144 224 × 224 RGB images through the network and average the outputs.
GoogLeNet: Results

- The progress in two years was **significant**: 

<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>

- And the effect of all of the tricks were also **significant**: 

<table>
<thead>
<tr>
<th>Number of models</th>
<th>Number of Crops</th>
<th>Cost</th>
<th>Top-5 error</th>
<th>compared to base</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>10.07%</td>
<td>base</td>
</tr>
<tr>
<td>1</td>
<td>10</td>
<td>10</td>
<td>9.15%</td>
<td>-0.92%</td>
</tr>
<tr>
<td>1</td>
<td>144</td>
<td>144</td>
<td>7.89%</td>
<td>-2.18%</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>7</td>
<td>8.09%</td>
<td>-1.98%</td>
</tr>
<tr>
<td>7</td>
<td>10</td>
<td>70</td>
<td>7.62%</td>
<td>-2.45%</td>
</tr>
<tr>
<td>7</td>
<td>144</td>
<td>1008</td>
<td>6.67%</td>
<td>-3.45%</td>
</tr>
</tbody>
</table>
GoogLeNet: Reflections

- The GoogLeNet architecture has 12x fewer parameters than AlexNet.
- And it makes less than 1/3 of the errors.
- This architecture demonstrates that CNNs can be tamed in size complexity.
- In the long run, this is important for deployment on limited hardware.
- GoogLeNet also popularized the fully convolutional layer architecture, which has been used (for example) for object segmentation.
CNNs: Fast-RCNN
Fast-RCNN: The Idea

- We will now look at a network designed for object detection rather than just recognition.


- This is a detection technique that actually uses a method from the pre-CNN revolution (Selective Search):

  - **Main idea:** propose likely object locations in a class-independent way, using only image content; classify each proposal.
Fast-RCNN: The Idea

- **Problem**: how to do this efficiently?
- **[SWITCH PRESENTATION]**
Deeper, Bigger, and Better
Deeper, Bigger, and Better

Diagram showing the trade-off between Top-1 accuracy and operations (G-Ops) for various convolutional neural networks (CNNs), including:

- Inception-v3
- ResNet-50
- ResNet-101
- ResNet-34
- Inception-v4
- ResNet-152
- VGG-16
- VGG-19
- GoogLeNet
- ENet
- BN-NIN
- BN-AlexNet
- AlexNet
Deeper, Bigger, and Better

- We have come a long way in five short years.
- However, though we have left behind (for the most part) the era of handcrafted features, we have entered the era of handcrafted architectures.
- It can be hard to make sense out of the confusing array of CNN architectures out there.
- It is even harder to optimally train these networks.
Discussion
Discussion

Discuss