THE EFFECT OF DATA

**Semantic Tasks:** Need to be evaluated (and trained) on data

LAST TIME

- Talked about "core semantic vision tasks"

Progress: Move from anecdata to data

The PASCAL Visual Object Classes Challenge 2007
**THE EFFECT OF DATA**

**Semantic Tasks:** Need to be evaluated (and trained) on data

- Huge effort to label a large number of images (using Amazon MTurk)
- Labels were from "Word-net" words
- It was thought that having a taxonomy of words would be helpful for visual semantics (turns out not).

**ImageNet**

The number of images for the largest task is shown for each task. Manual labels between the first task and ImageNet are collected in red.

**THE EFFECT OF DATA**

**Semantic Tasks:** Need to be evaluated (and trained) on data

- Huge effort to label a large number of images (using Amazon MTurk)
- Labels were from "Word-net" words
- It was thought that having a taxonomy of words would be helpful for visual semantics (turns out not).

**ImageNet**

1000 classes x 1300 images per class = 1.3 Million Images

Changed the nature of ML methods.
The effect of data

Top-5 error rate on ImageNet

ILSVRC 2010  ILSVRC 2011  ILSVRC 2012

NIPS 2012
ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky
University of Toronto

Ilya Sutskever
University of Toronto

Geoffrey E. Hinton
University of Toronto

All based on convolutional neural networks
THE EFFECT OF DATA

- Older ML Methods designed for small training sets
  - Used more complex optimization methods (than gradient descent): second order methods, etc.
  - Methods had better guarantees if you chose “simpler” classifiers
  - And in practice, gave you better results than neural networks
  - But were quadratic in training set size
- With training set of millions, quadratic-time optimization was not feasible.
- So people first moved to gradient descent, but with the same simple classifiers.
- Found that with additional computation power, if you train with small step size for many iterations (still better than quadratic), gradient descent gives you a reasonable answer.
- But then, since gradient descent was working, the question was why not try more complex classifiers?
- And Krizhevsky and others demonstrated: in this large training set / high training computation budget, deep neural networks are much better!

ARCHITECTURES

Broad Design Principles

- Think of a network that can “express” the operations that you think are needed to solve the problem
  - What kind of a “receptive” field should it have.
  - How non-linear does it need to be.
  - What should be the nature of the flow of information across the image.
- Make sure it’s a function you can actually learn.
  - Think of the flow of gradients.
- Try to make other architectures that you know can be successfully trained as a starting point.
- Dealing with Overfitting: One approach:
  - First find the biggest deepest network that will overfit the data
    (Given enough capacity, CNNs will often be able to just memorize the dataset)
  - Then scale it down so that it generalizes.

ARCHITECTURES

Let’s consider image classification.

- We will fix our input image to be a specific size.
  - Typically choose square images of size $S \times S$
  - Given image, resize proportionally so that smaller side (height or width) is $S$
  - Then take an $S$ crop along the other direction
  - (Sometime take multiple crops and average)
- The final output will be a $C$ dimensional vector for $C$ classes.
  - Train using soft-max cross entropy.
  - Classify using arg-max
- Often, you’ll hear about Top-K error.
  - How often is the true class in the top K highest values of predicted $C$ vector.

ARCHITECTURES

Let’s talk about VGG-16. Winner of Imagenet 2014.

Reference
Ken Chatfield, Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman, “Return of the Devil in the Details: Delving Deep into Convolutional Nets”

Karen Simonyan & Andrew Zisserman, “Very Deep Convolutional Networks for Large-Scale Image Recognition.”

Four kinds of layers:

- Convolutional
- Max-pooling
- Fully Connected
- Soft-max
ARCHITECTURES

- Convolutional Layers

  Take a spatial input, produce a spatial output.

  $B \times H \times W \times C \Rightarrow B \times H' \times W' \times C'$

  Can also combine with down-sampling.

  $g[b, y, x, c] = \sum_k \sum_y \sum_x f[b, y+s, x+s, k, c](k, k, c_1, c_2)$

  Here, $s$ is stride.

- PSET 5 asks you to implement ‘valid convolution’
- But often combined with padding (just like in regular convolution)

ARCHITECTURES

Max-Pooling Layer

$B \times H \times W \times C \Rightarrow B \times H' \times W' \times C$

$g[b, y, x, c] = \max_{k, k_y} f[b, y+s, x+s, k, c](k, k_y, c_1)$

For each channel, choose the maximum value in a spatial neighborhood.

- What will the gradients of this look like?
- Motivated by intuition from traditional object recognition (deformable part models). Allows for some ‘slack’ in exact spatial location.

ARCHITECTURES

VGG-16

- Input is a 224x224x3 Image

  - Block 1
    - 3x3 Conv (Pad 1): 3->64 + RELU (*pad 1 means on all sides, all conv layers have a “bias”)

Question: Input activation is $B \times H \times W \times C_1$, and I convolve it with a kernel of size $K \times K \times C_1 \times C_2$, what is the size of my output? Assume ‘valid’ convolution.

$B \times (H - K + 1) \times (W - K + 1) \times C_2$

Question: What if I do this with a stride of 2?

Downsample above by 2. Think of what happens when sizes are even or odd.

$B \times \left(\frac{(H - K)}{2}\right) + 1 \times \left(\frac{(W - K)}{2}\right) + 1 \times C_2$

In general, you want to pad such that $H - K$ and $W - K$ are even, so that you keep the right and bottom edge of your images.
**VGG-16**

Input is a 224x224x3 Image
- Block 1
  - 3x3 Conv (Pad 1): 3→64 + RELU
  - 3x3 Conv (Pad 1): 64→64 + RELU
  - 2x2 Max-Pool (Pad 0, Stride 2): 64→64
Input to Block 2 is 112x112x64 (called pool1)
- Block 2
  - 3x3 Conv (Pad 1): 64→128 + RELU
  - 3x3 Conv (Pad 1): 128→128 + RELU
  - 2x2 Max-Pool (Pad 0, Stride 2): 128→128
Input to Block 3 is 56x56x128 (called pool2)
- Block 3
  - 3x3 Conv (Pad 1): 128→256 + RELU
  - 3x3 Conv (Pad 1): 256→256 + RELU
  - 3x3 Conv (Pad 1): 256→256 + RELU
  - 2x2 Max-Pool (Pad 0, Stride 2): 256→256
Input to Block 4 is 28x28x256 (called pool3)
- Block 4
  - 3x3 Conv (Pad 1): 256→512 + RELU
  - 3x3 Conv (Pad 1): 512→512 + RELU
  - 3x3 Conv (Pad 1): 512→512 + RELU
  - 2x2 Max-Pool (Pad 0, Stride 2): 512→512
Input to Block 5 is 14x14x512 (called pool4)
- Block 5
  - 3x3 Conv (Pad 1): 512→512 + RELU
  - 3x3 Conv (Pad 1): 512→512 + RELU
  - 3x3 Conv (Pad 1): 512→512 + RELU
  - 2x2 Max-Pool (Pad 0, Stride 2): 512→512
Output of Block 5 is 7x7x512 (called pool5)

This is the final output that is trained with a softmax + cross entropy.
- Lots of layers: 138 Million Parameters
- Compared to previous architectures, used really small conv filters.
  - This has now become standard.
  - Two 3x3 layers is “better” than a single 5x5 layer.
    - More non-linear
    - Fewer independent weights
- Train this with backprop!
- Back in the day, would take a week or more.

**TRAINING IN PRACTICE**

- Remember: Gradient Descent is Fragile

**The Effect of Parameterization**

\[ f(x; \theta) = \theta x \]
\[ f'(x; \theta') = 2\theta' x \]

- Two representations of the same hypothesis space.
- Let’s initialize \( \theta' = \theta / 2 \). Say \( \theta = 10, \theta' = 5 \).
- Compute loss with respect to the same example.
  - Say \( \nabla \theta L = \nabla \theta' = 1, x = 1 \)
  - What are \( \nabla \theta \) and \( \nabla \theta' \)?
    - \( \nabla \theta = 1, \nabla \theta' = 2. \)
- Update with learning rate = 1
- Updated \( \theta = 9, \theta = 3 \)
- \( f(x) = 9x, f'(x) = 6x \)

**Initialization**

- Because we’re using a first order method, it is important to make sure that the activations of all layers are the same magnitude.
- Because we have RELUs \( y = \max(0, x) \), it is important to make sure roughly half the expectations are positive.
- Normalize your inputs to be 0-mean and unit variance.
  - Compute dataset mean and standard deviation, subtract and divide from all inputs.
  - For images, you usually compute the mean over all pixels—so single normalization for all pixels in the input.
    - But different for different color channels.
  - Sometimes, you will want ‘per-location’ normalization.
  - Other times, you will normalize each input by its own pixel mean and standard deviation.
Initialization

- Initialize all biases to 0. Why? Don’t want to shift the mean.
- Now initialize weights randomly so that variance of outputs = variance of inputs = 1.
  - Use approximation \( \text{var}(wx) = \text{var}(w)\text{var}(x) \) for scalar \( w \) and \( x \).
- Say you have a fully connected layer: \( y = W^T x \)
  - \( x \) is \( N \)-dimensional. We assume its 0-mean unit-variance coming in.
  - \( W \) is \( N \times M \) dimensional.
  - We will initialize \( W \) with 0-mean and variance \( \sigma^2 \).
  - What should be the value of \( \sigma^2 \)? Take 5 mins.
    - \( \sigma^2 = 1/N \).
- Now what about a convolution layer with kernel size \( K \times K \times C_1 \times C_2 \)?
  - Initialize kernel with 0-mean and variance \( \sigma^2 \). What should \( \sigma^2 \) be?
    - \( \sigma^2 = 1/(K^2 C_1) \)

Solution: Use truncated distributions that are forced to have 0 probability outside a range.
- Uniform distribution, “truncated-normal”
- Figure out what parameters of this distribution should be to have equivalent variance.