

CSC2515 Lecture 6: Convolutional Networks

David Duvenaud

Based on Materials from Roger Grosse, University of Toronto

Announcements

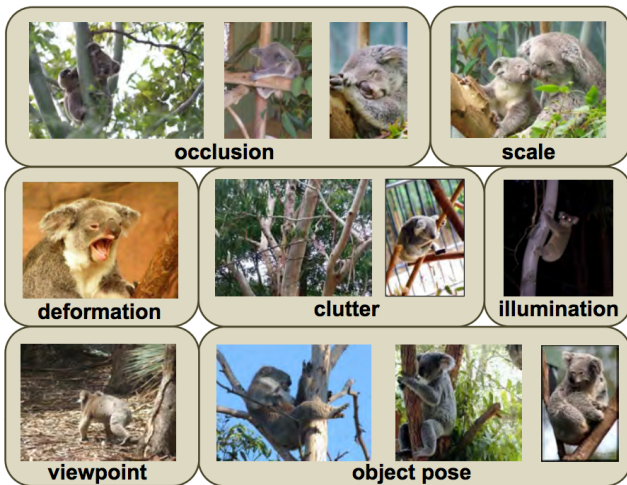
- Midterm grades this weekend
- Drop date March 1st
- HW3 released this weekend

Neural Nets for Visual Object Recognition

- People are very good at recognizing shapes
 - ▶ Intrinsically difficult, computers are bad at it
- Why is it difficult?

Why is it a Problem?

- Difficult scene conditions



[From: Grauman & Leibe]

Why is it a Problem?

- Huge within-class variations. Recognition is mainly about modeling variation.



[Pic from: S. Lazebnik]

Why is it a Problem?

- Tons of classes



[Biederman]

Neural Nets for Object Recognition

- People are very good at recognizing object
 - ▶ Intrinsically difficult, computers are bad at it
- Some reasons why it is difficult:
 - ▶ **Segmentation**: Real scenes are cluttered
 - ▶ **Invariances**: We are very good at ignoring all sorts of variations that do not affect class
 - ▶ **Deformations**: Natural object classes allow variations (faces, letters, chairs)
 - ▶ A huge amount of computation is required

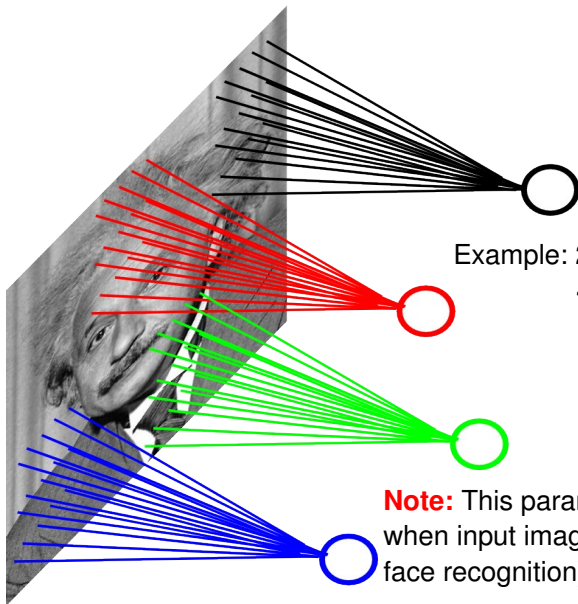
How to Deal with Large Input Spaces

- How can we apply neural nets to images?
- Images can have millions of pixels, i.e., \mathbf{x} is very high dimensional
- How many parameters do I have?

How to Deal with Large Input Spaces

- How can we apply neural nets to images?
- Images can have millions of pixels, i.e., \mathbf{x} is very high dimensional
- How many parameters do I have?
- Prohibitive to have fully-connected layers
- What can we do?
- We can use a **locally connected layer**

Locally Connected Layer



Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters

Note: This parameterization is good when input image is registered (e.g., face recognition). ³⁴

When Will this Work?

When Will this Work?

- This is good when the input is (roughly) registered



General Images

- The object can be anywhere



[Slide: Y. Zhu]

General Images

- The object can be anywhere



[Slide: Y. Zhu]

General Images

- The object can be anywhere



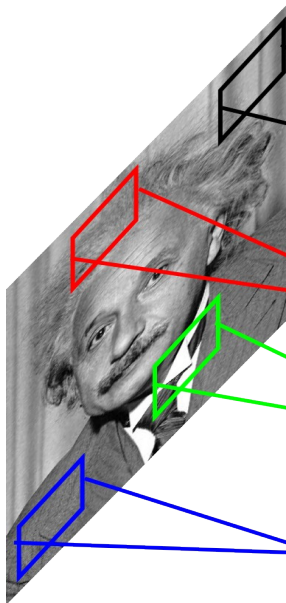
[Slide: Y. Zhu]

The Invariance Problem

- Our perceptual systems are very good at dealing with **invariances**
 - ▶ translation, rotation, scaling
 - ▶ deformation, contrast, lighting
- We are so good at this that it's hard to appreciate how difficult it is
 - ▶ It's one of the main difficulties in making computers perceive
 - ▶ We still don't have generally accepted solutions

Locally Connected Layer

STATIONARITY? Statistics is similar at different locations

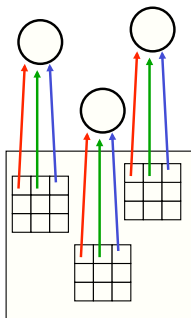


Example: 200x200 image
40K hidden units
Filter size: 10x10
4M parameters

Note: This parameterization is good when input image is registered (e.g., face recognition).

The replicated feature approach

The red connections all have the same weight.

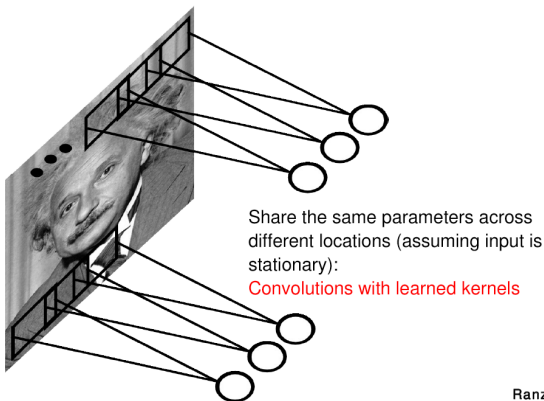


5

- Adopt approach apparently used in monkey visual systems
- Use many different copies of the same feature detector.
 - ▶ Copies have slightly different positions.
 - ▶ Could also replicate across scale and orientation.
 - ▶ Tricky and expensive
 - ▶ Replication **reduces the number of free parameters** to be learned.
- Use several **different feature types**, each with its own replicated pool of detectors.
 - ▶ Allows each patch of image to be represented in several ways.

Convolutional Neural Net

- Idea: statistics are similar at different locations (Lecun 1998)
- Connect each hidden unit to a small input patch and share the weight across space
- This is called a **convolution layer** and the network is a **convolutional network**



36

Ranzato 

- Convolution layers are named after the **convolution** operation.
- If a and b are two arrays,

$$(a * b)_t = \sum_{\tau} a_{\tau} b_{t-\tau}.$$

Convolution

Method 1: translate-and-scale

The diagram illustrates the 'translate-and-scale' method for 1D convolution. It shows the convolution of two 3x3 kernels.

Kernel 1 (Left): Values are 2, -1, 1. The -1 is represented by a downward arrow.

Kernel 2 (Middle): Values are 1, 1, 2.

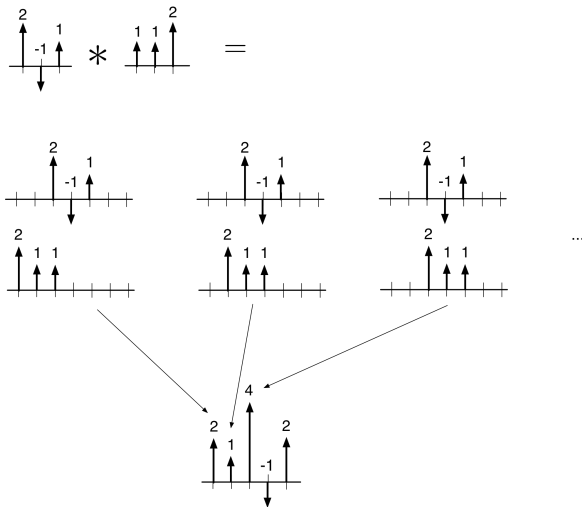
Calculation:

$$\begin{aligned} &= + 2 \times \begin{array}{|c|c|c|c|c|} \hline \uparrow 1 & \uparrow 1 & \uparrow 2 & & \\ \hline \end{array} \\ &\quad + -1 \times \begin{array}{|c|c|c|c|c|} \hline & \uparrow 1 & \uparrow 1 & \uparrow 2 & \\ \hline \end{array} \\ &\quad + 1 \times \begin{array}{|c|c|c|c|c|} \hline & & \uparrow 1 & \uparrow 1 & \uparrow 2 \\ \hline \end{array} \end{aligned}$$

Result (Right): The final output values are 2, 1, 4, -1, 2. The -1 is represented by a downward arrow.

Convolution

Method 2: flip-and-filter



Convolution can also be viewed as matrix multiplication:

$$(2, -1, 1) * (1, 1, 2) = \begin{pmatrix} 1 \\ 1 & 1 \\ 2 & 1 & 1 \\ & 2 & 1 \\ & & 2 \end{pmatrix} \begin{pmatrix} 2 \\ -1 \\ 1 \end{pmatrix}$$

Aside: This is how convolution is typically implemented. (More efficient than the fast Fourier transform (FFT) for modern conv nets on GPUs!)

Some properties of convolution:

- Commutativity

$$a * b = b * a$$

- Linearity

$$a * (\lambda_1 b + \lambda_2 c) = \lambda_1 a * b + \lambda_2 a * c$$

2-D Convolution

2-D convolution is defined analogously to 1-D convolution.

If A and B are two 2-D arrays, then:

$$(A * B)_{ij} = \sum_s \sum_t A_{st} B_{i-s, j-t}.$$

2-D Convolution

Method 1: Translate-and-Scale

$$\begin{bmatrix} 1 & 3 & 1 \\ 0 & -1 & 1 \\ 2 & 2 & -1 \end{bmatrix} * \begin{bmatrix} 1 & 2 \\ 0 & -1 \end{bmatrix} = 1 \times \begin{bmatrix} 1 & 3 & 1 & \\ 0 & -1 & 1 & \\ 2 & 2 & -1 & \\ & & & \end{bmatrix} + 2 \times \begin{bmatrix} & 1 & 3 & 1 \\ & 0 & -1 & 1 \\ & 2 & 2 & -1 \\ & & & \end{bmatrix} = \begin{bmatrix} 1 & 5 & 7 & 2 \\ 0 & -2 & -4 & 1 \\ 2 & 6 & 4 & -3 \\ 0 & -2 & -2 & 1 \end{bmatrix} + -1 \times \begin{bmatrix} & & & \\ & 1 & 3 & 1 \\ & 0 & -1 & 1 \\ & 2 & 2 & -1 \end{bmatrix}$$

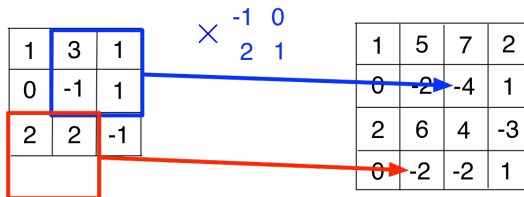
2-D Convolution

Method 2: Flip-and-Filter

1	3	1
0	-1	1
2	2	-1

 $*$

1	2
0	-1



2-D Convolution

The thing we convolve by is called a **kernel**, or **filter**.

What does this filter do?



*

0	1	0
1	4	1
0	1	0

2-D Convolution

The thing we convolve by is called a **kernel**, or **filter**.

What does this filter do?



*

0	1	0
1	4	1
0	1	0



2-D Convolution

What does this filter do?



*

0	-1	0
-1	8	-1
0	-1	0

2-D Convolution

What does this filter do?



*

0	-1	0
-1	8	-1
0	-1	0



2-D Convolution

What does this filter do?



*

0	-1	0
-1	4	-1
0	-1	0

2-D Convolution

What does this filter do?



*

0	-1	0
-1	4	-1
0	-1	0



2-D Convolution

What does this filter do?



*

1	0	-1
2	0	-2
1	0	-1

2-D Convolution

What does this filter do?

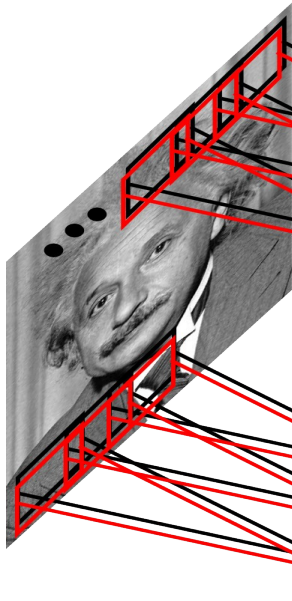


*

1	0	-1
2	0	-2
1	0	-1



Convolutional Layer



Learn **multiple filters**.

E.g.: 200x200 image
100 Filters
Filter size: 10x10
10K parameters

Convolutional Layer

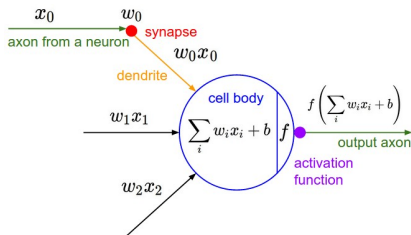
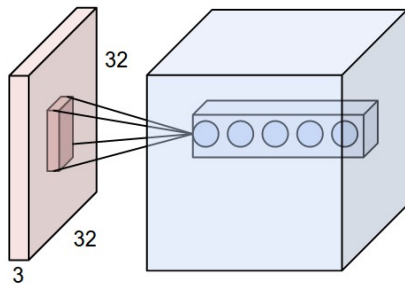


Figure: **Left:** CNN, **right:** Each neuron computes a linear and activation function

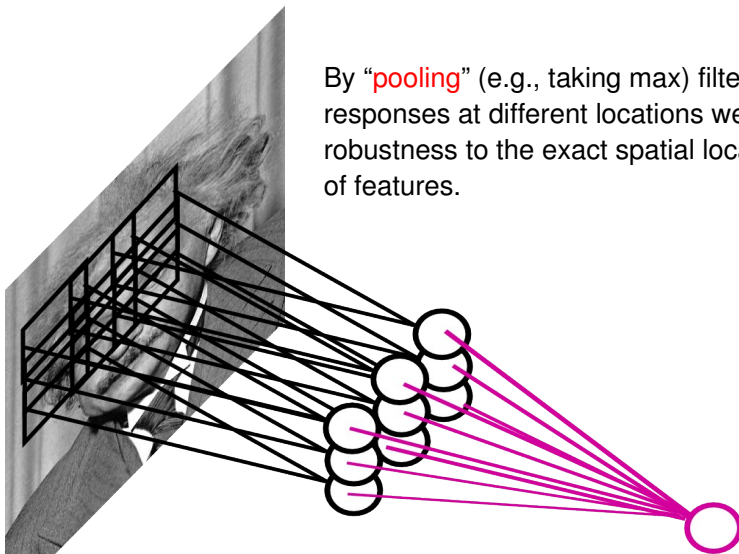
Hyperparameters of a convolutional layer:

- The number of filters (controls the **depth** of the output volume)
- The **stride**: how many units apart do we apply a filter spatially (this controls the spatial size of the output volume)
- The size $w \times h$ of the filters

[<http://cs231n.github.io/convolutional-networks/>]

Pooling Layer

By “pooling” (e.g., taking max) filter responses at different locations we gain robustness to the exact spatial location of features.



Pooling Options

- **Max Pooling**: return the maximal argument
- **Average Pooling**: return the average of the arguments
- Other types of pooling exist.

Pooling

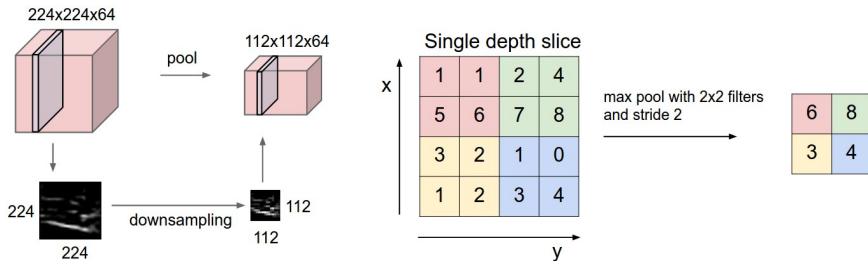


Figure: **Left:** Pooling, **right:** max pooling example

Hyperparameters of a pooling layer:

- The spatial extent F
- The stride

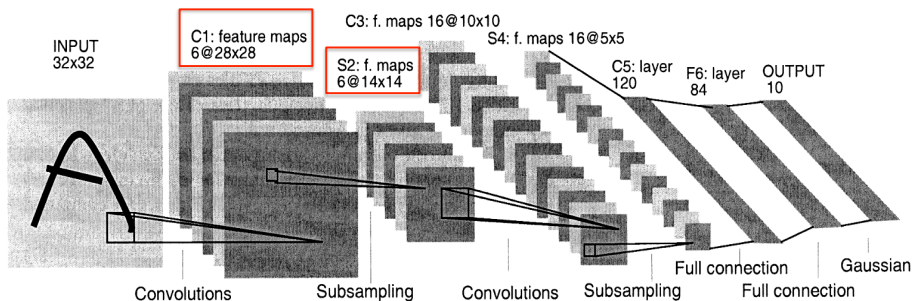
[<http://cs231n.github.io/convolutional-networks/>]

Backpropagation with Weight Constraints

- The backprop procedure from last lecture can be applied directly to conv nets.
- This is covered in csc2516.
- As a user, you don't need to worry about the details, since they're handled by automatic differentiation packages.

- MNIST dataset of handwritten digits
 - ▶ **Categories:** 10 digit classes
 - ▶ **Source:** Scans of handwritten zip codes from envelopes
 - ▶ **Size:** 60,000 training images and 10,000 test images, grayscale, of size 28×28
 - ▶ **Normalization:** centered within in the image, scaled to a consistent size
 - ▶ The assumption is that the digit recognizer would be part of a larger pipeline that segments and normalizes images.
- In 1998, Yann LeCun and colleagues built a conv net called [LeNet](#) which was able to classify digits with 98.9% test accuracy.
 - ▶ It was good enough to be used in a system for automatically reading numbers on checks.

Here's the LeNet architecture, which was applied to handwritten digit recognition on MNIST in 1998:



Size of a Conv Net

- Ways to measure the size of a network:
 - ▶ **Number of units.** This is important because the activations need to be stored in memory during training (i.e. backprop).

Size of a Conv Net

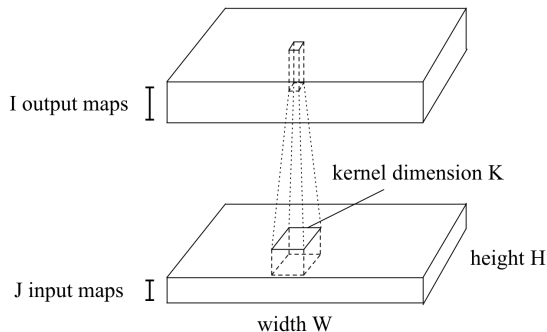
- Ways to measure the size of a network:
 - ▶ **Number of units.** This is important because the activations need to be stored in memory during training (i.e. backprop).
 - ▶ **Number of weights.** This is important because the weights need to be stored in memory, and because the number of parameters determines the amount of overfitting.

- Ways to measure the size of a network:
 - ▶ **Number of units.** This is important because the activations need to be stored in memory during training (i.e. backprop).
 - ▶ **Number of weights.** This is important because the weights need to be stored in memory, and because the number of parameters determines the amount of overfitting.
 - ▶ **Number of connections.** This is important because there are approximately 3 add-multiply operations per connection (1 for the forward pass, 2 for the backward pass).

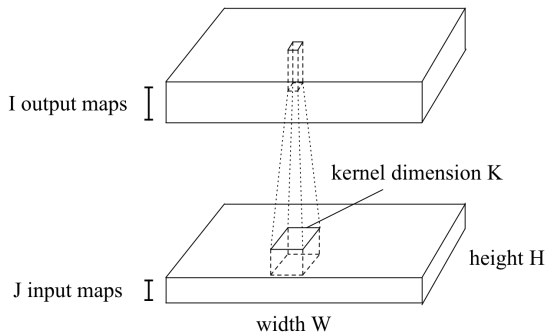
Size of a Conv Net

- Ways to measure the size of a network:
 - ▶ **Number of units.** This is important because the activations need to be stored in memory during training (i.e. backprop).
 - ▶ **Number of weights.** This is important because the weights need to be stored in memory, and because the number of parameters determines the amount of overfitting.
 - ▶ **Number of connections.** This is important because there are approximately 3 add-multiply operations per connection (1 for the forward pass, 2 for the backward pass).
- We saw that a fully connected layer with M input units and N output units has MN connections and MN weights.
- The story for conv nets is more complicated.

Size of a Conv Net

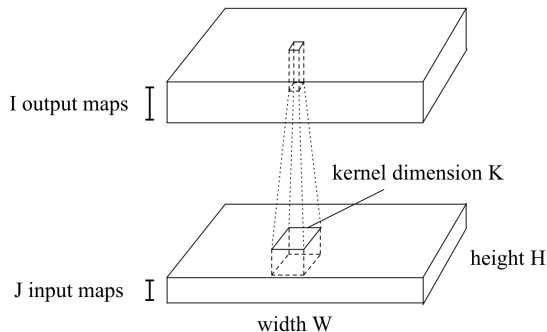


Size of a Conv Net



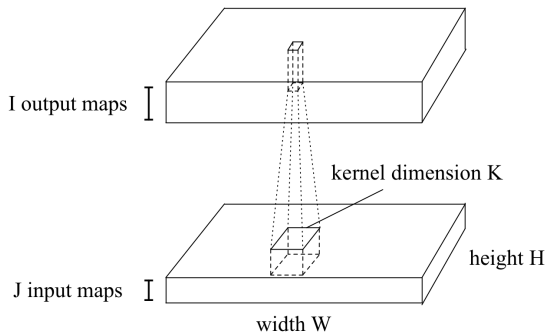
output units **fully connected layer** **convolution layer**

Size of a Conv Net



	fully connected layer	convolution layer
# output units	WHI	WHI

Size of a Conv Net



output units
weights

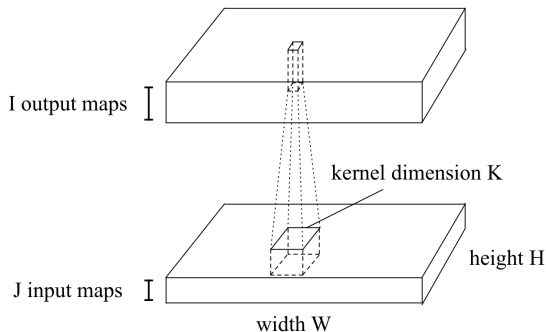
fully connected layer

WHI

convolution layer

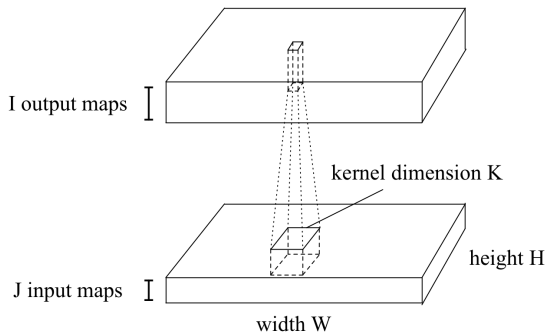
WHI

Size of a Conv Net



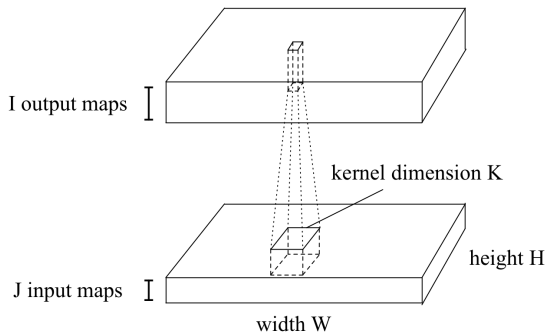
	fully connected layer	convolution layer
# output units	WHI	WHI
# weights	W^2H^2IJ	

Size of a Conv Net



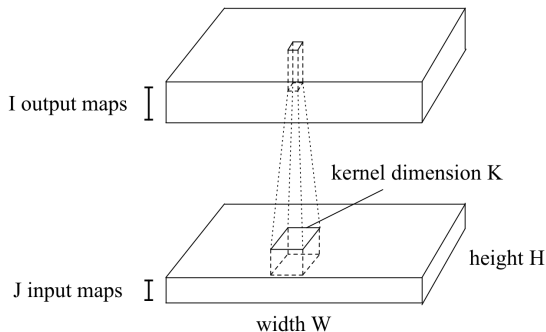
	fully connected layer	convolution layer
# output units	WHI	WHI
# weights	W^2H^2IJ	K^2IJ

Size of a Conv Net



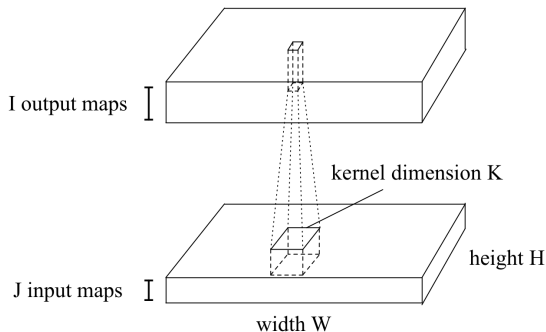
	fully connected layer	convolution layer
# output units	WHI	WHI
# weights	W^2H^2IJ	K^2IJ
# connections		

Size of a Conv Net



	fully connected layer	convolution layer
# output units	WHI	WHI
# weights	W^2H^2IJ	K^2IJ
# connections	W^2H^2IJ	

Size of a Conv Net



	fully connected layer	convolution layer
# output units	WHI	WHI
# weights	W^2H^2IJ	K^2IJ
# connections	W^2H^2IJ	WHK^2IJ

Size of a Conv Net

Sizes of layers in LeNet:

Layer	Type	# units	# connections	# weights
C1	convolution	4704	117,600	150
S2	pooling	1176	4704	0
C3	convolution	1600	240,000	2400
S4	pooling	400	1600	0
F5	fully connected	120	48,000	48,000
F6	fully connected	84	10,080	10,080
output	fully connected	10	840	840

Conclusions?

- Rules of thumb:
 - ▶ Most of the units and connections are in the convolution layers.
 - ▶ Most of the weights are in the fully connected layers.
- If you try to make layers larger, you'll run up against various resource limitations (i.e. computation time, memory)
 - ▶ Conv nets have gotten a LOT larger since 1998!

ImageNet

ImageNet is the modern object recognition benchmark dataset. It was introduced in 2009, and has led to amazing progress in object recognition since then.

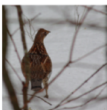
ILSVRC



flamingo



cock



ruffed grouse



quail



partridge

...



Egyptian cat



Persian cat



Siamese cat

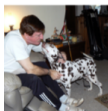


tabby



lynx

...



dalmatian



keeshond



miniature schnauzer



standard schnauzer



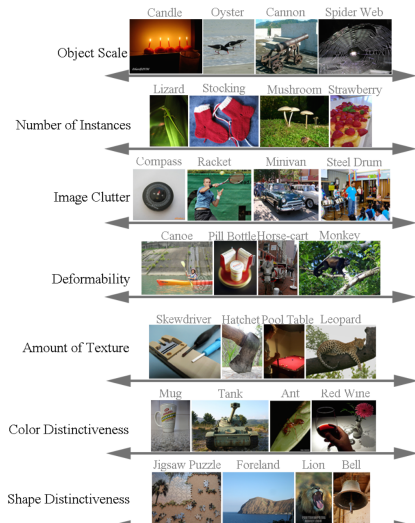
giant schnauzer

...

- Used for the ImageNet Large Scale Visual Recognition Challenge (ILSVRC), an annual benchmark competition for object recognition algorithms
- Design decisions
 - ▶ **Categories:** Taken from a lexical database called WordNet
 - ▶ WordNet consists of “synsets”, or sets of synonymous words
 - ▶ They tried to use as many of these as possible; almost 22,000 as of 2010
 - ▶ Of these, they chose the 1000 most common for the ILSVRC
 - ▶ The categories are really specific, e.g. hundreds of kinds of dogs
 - ▶ **Size:** 1.2 million full-sized images for the ILSVRC
 - ▶ **Source:** Results from image search engines, hand-labeled by Mechanical Turkers
 - ▶ Labeling such specific categories was challenging; annotators had to be given the WordNet hierarchy, Wikipedia, etc.
 - ▶ **Normalization:** none, although the contestants are free to do preprocessing

ImageNet

Images and object categories vary on a lot of dimensions



Russakovsky et al.

ImageNet

Size on disk:

MNIST
60 MB

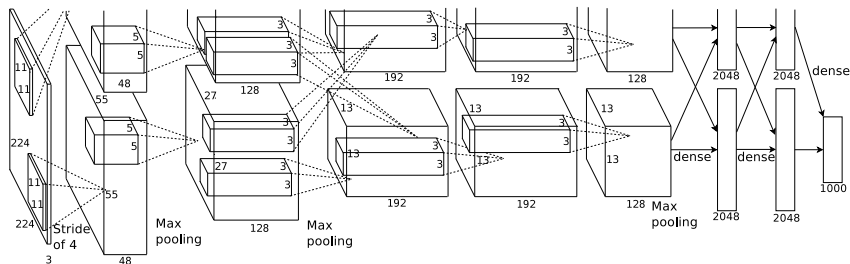


ImageNet
50 GB



AlexNet

- AlexNet, 2012. 8 weight layers. 16.4% top-5 error (i.e. the network gets 5 tries to guess the right category).



(Krizhevsky et al., 2012)

- The two processing pathways correspond to 2 GPUs. (At the time, the network couldn't fit on one GPU.)
- AlexNet's stunning performance on the ILSVRC is what set off the deep learning boom of the last 6 years.

Inception

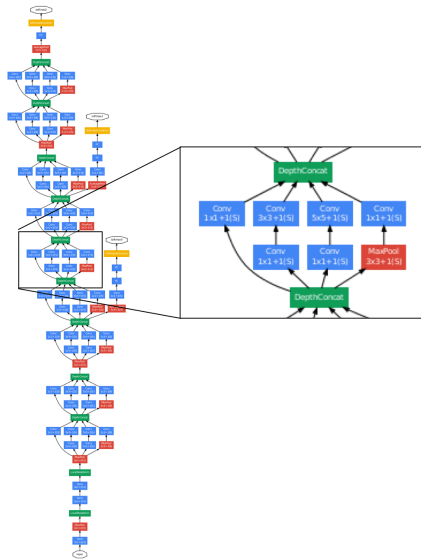
Inception, 2014. (“We need to go deeper!”)

22 weight layers

Fully convolutional (no fully connected layers)

Convolutions are broken down into a bunch of smaller convolutions

6.6% test error on ImageNet

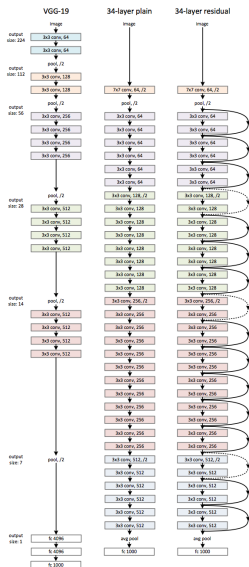
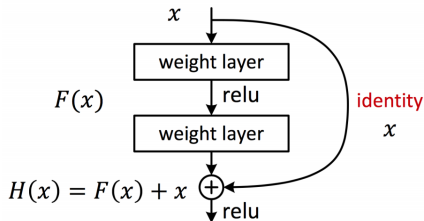


(Szegedy et al., 2014)

- They were really aggressive about cutting the number of parameters.
 - ▶ Motivation: train the network on a large cluster, run it on a cell phone
 - ▶ Memory at test time is the big constraint.
 - ▶ Having lots of units is OK, since the activations only need to be stored at training time (for backpropagation).
 - ▶ Parameters need to be stored both at training and test time, so these are the memory bottleneck.
 - ▶ How they did it
 - ▶ No fully connected layers (remember, these have most of the weights)
 - ▶ Break down convolutions into multiple smaller convolutions (since this requires fewer parameters total)
 - ▶ Inception has “only” 2 million parameters, compared with 60 million for AlexNet
 - ▶ This turned out to improve generalization as well. (Overfitting can still be a problem, even with over a million images!)

150 Layers!

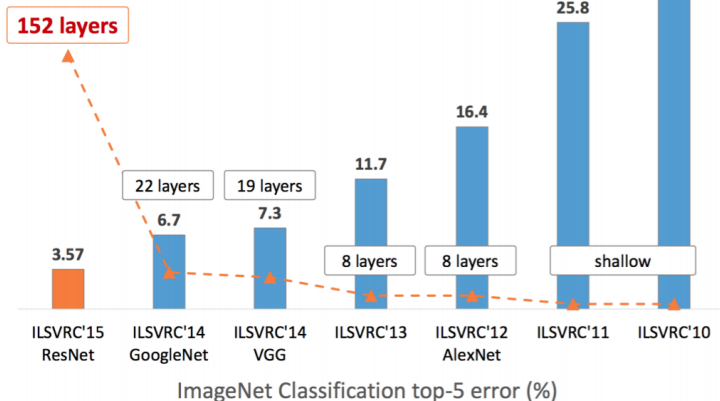
- Networks are now at 150 layers
- They use a skip connections with special form
- In fact, they don't fit on this screen
- Amazing performance!
- A lot of “mistakes” are due to wrong ground-truth



[He, K., Zhang, X., Ren, S. and Sun, J., 2015. Deep Residual Learning for Image Recognition. arXiv:1512.03385, 2016]

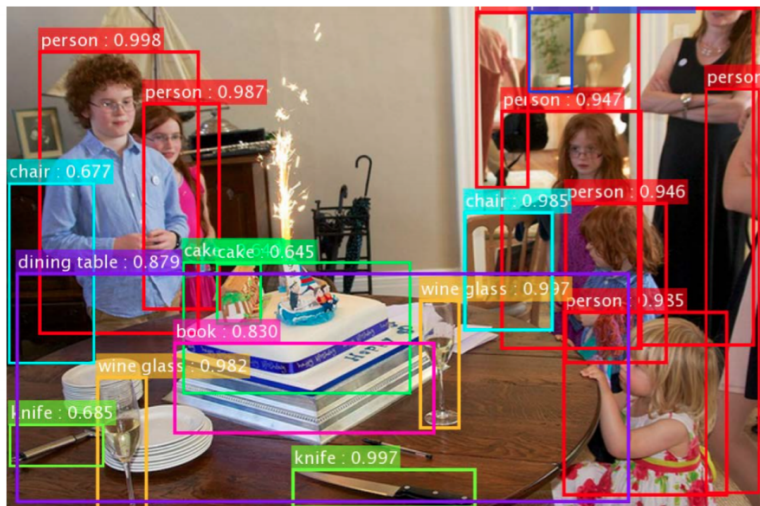
Results: Object Classification

Revolution of Depth



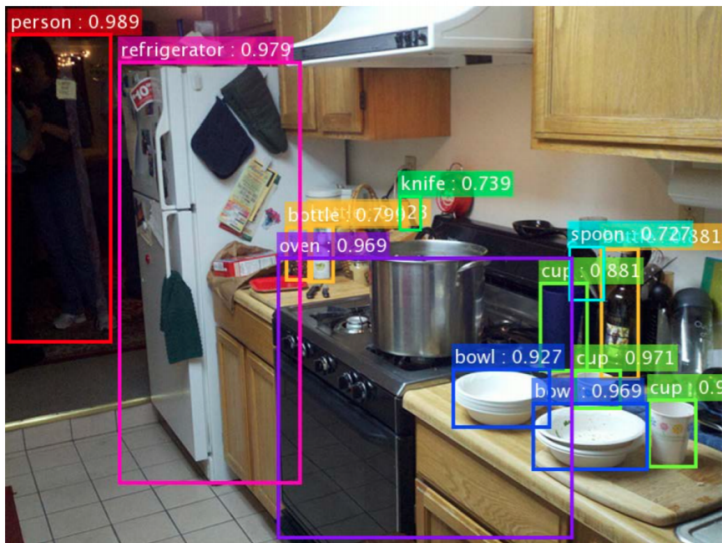
Slide: R. Liao, Paper: [He, K., Zhang, X., Ren, S. and Sun, J., 2015. Deep Residual Learning for Image Recognition. arXiv:1512.03385, 2016]

Results: Object Detection



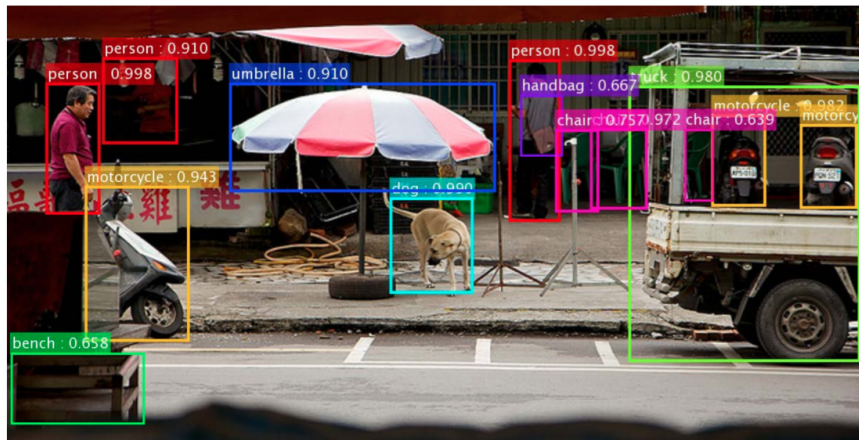
Slide: R. Liao, Paper: [He, K., Zhang, X., Ren, S. and Sun, J., 2015. Deep Residual Learning for Image Recognition. arXiv:1512.03385, 2016]

Results: Object Detection



Slide: R. Liao, Paper: [He, K., Zhang, X., Ren, S. and Sun, J., 2015. Deep Residual Learning for Image Recognition.

Results: Object Detection

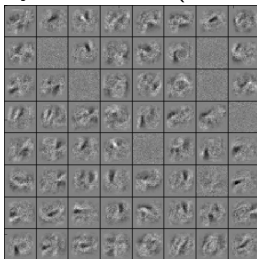


Slide: R. Liao, Paper: [He, K., Zhang, X., Ren, S. and Sun, J., 2015. Deep Residual Learning for Image Recognition. arXiv:1512.03385, 2016]

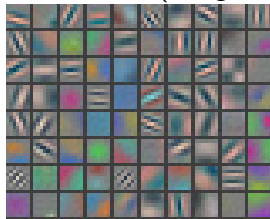
What Do Networks Learn?

- Recall: we can understand what first-layer features are doing by visualizing the weight matrices.

Fully connected (MNIST)



Convolutional (ImageNet)



- Higher-level weight matrices are hard to interpret.
- The better the input matches these weights, the more the feature activates.
 - Obvious generalization: visualize higher-level features by seeing what inputs activate them.

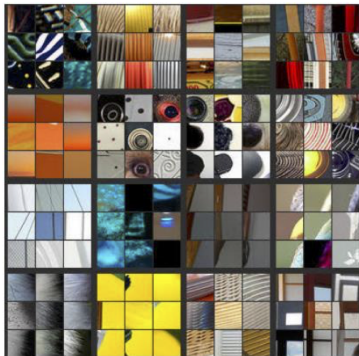
What Do Networks Learn?

- One way to formalize: pick the images and locations in the training set which activate a unit most strongly.
- Here's the visualization for layer 1:



What Do Networks Learn?

- Layer 3:



What Do Networks Learn?

- Layer 4:



What Do Networks Learn?

- Layer 5:



What Do Networks Learn?

- Higher layers seem to pick up more abstract, high-level information.
- Problems?

What Do Networks Learn?

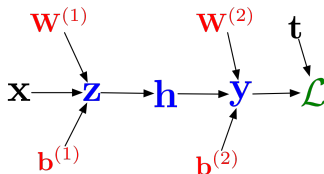
- Higher layers seem to pick up more abstract, high-level information.
- Problems?
 - ▶ Can't tell what the unit is actually responding to in the image.
 - ▶ We may read too much into the results, e.g. a unit may detect red, and the images that maximize its activation will all be stop signs.

What Do Networks Learn?

- Higher layers seem to pick up more abstract, high-level information.
- Problems?
 - ▶ Can't tell what the unit is actually responding to in the image.
 - ▶ We may read too much into the results, e.g. a unit may detect red, and the images that maximize its activation will all be stop signs.
- Can use input gradients to diagnose what the unit is responding to.
 - ▶ Optimize an image from scratch to increase a unit's activation

Optimizing the Image

- Recall the computation graph:



- From this graph, you could compute $\partial\mathcal{L}/\partial\mathbf{x}$, but we never made use of this.

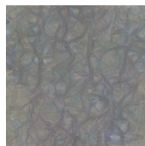
Optimizing the Image

- Can do gradient ascent on an image to maximize the activation of a given neuron.
- Requires a few tricks to make this work; see <https://distill.pub/2017/feature-visualization/>

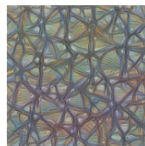
Starting from random noise, we optimize an image to activate a particular neuron (layer mixed4a, unit 11).



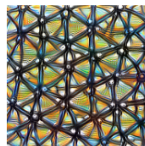
Step 0



Step 4



Step 48



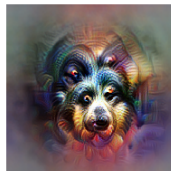
Step 2048

Optimizing the Image

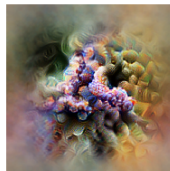
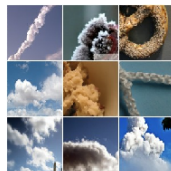
Dataset Examples show us what neurons respond to in practice



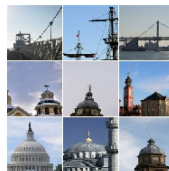
Baseball—or stripes?
mixed4a, Unit 6



Animal faces—or snouts?
mixed4a, Unit 240



Clouds—or fluffiness?
mixed4a, Unit 453



Buildings—or sky?
mixed4a, Unit 492

Optimizing the Image

- Higher layers in the network often learn higher-level, more interpretable representations



Edges (layer conv2d0)

Textures (layer mixed3a)

Patterns (layer mixed4a)

<https://distill.pub/2017/feature-visualization/>

Optimizing the Image

- Higher layers in the network often learn higher-level, more interpretable representations



Parts (layers mixed4b & mixed4c)

Objects (layers mixed4d & mixed4e)

<https://distill.pub/2017/feature-visualization/>