CSC421/2516 Lecture 5: Convolutional Neural Networks & Image Classification

Jimmy Ba

Overview

So far in the course, we've seen two types of layers:

- fully connected layers
- embedding layers (i.e. lookup tables)

Different layers could be stacked together to build powerful models.

Let's add another layer type: convolution layers

Conv layers are very useful building blocks for computer vision applications.

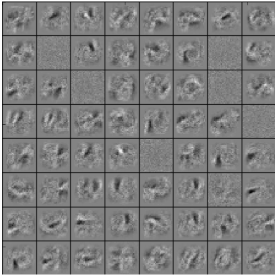
Overview

What makes vision hard?

- Vison needs to be robust to a lot of transformations or distortions:
  - change in pose/viewpoint
  - change in illumination
  - deformation
  - occlusion (some objects are hidden behind others)
- Many object categories can vary wildly in appearance (e.g. chairs)
- Geoff Hinton: "Imaging a medical database in which the age of the patient sometimes hops to the input dimension which normally codes for weight!"

Overview

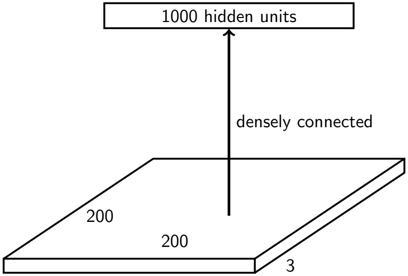
Recall we looked at some hidden layer features for classifying handwritten digits:



This isn't going to scale to full-sized images.

Overview

Suppose we want to train a network that takes a 200 × 200 RGB image as input.

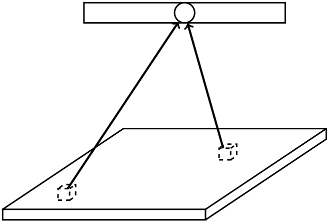


What is the problem with having this as the first layer?

- Too many parameters! Input size = 200 × 200 × 3 = 120K. Parameters = 120K × 1000 = 120 million.
- What happens if the object in the image shifts a little?

Overview

In the fully connected layer, each feature (hidden unit) looks at the entire image. Since the image is a BIG thing, we end up with lots of parameters.



But, do we really expect to learn a useful feature at the first layer which depends on pixels that are spatially far away ?

The far away pixels will probably belong to completely different objects (or object sub-parts). Very little correlation.

We want the incoming weights to focus on local patterns of the input image.

Overview

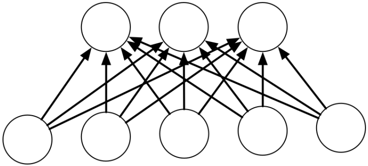
The same sorts of features that are useful in analyzing one part of the image will probably be useful for analyzing other parts as well.

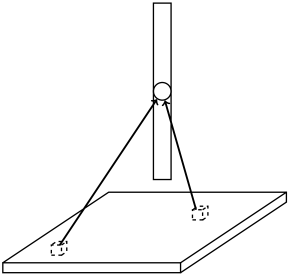
E.g., edges, corners, contours, object parts

We want a neural net architecture that lets us learn a set of feature detectors shared at all image locations.

Convolution Layers

Fully connected layers:

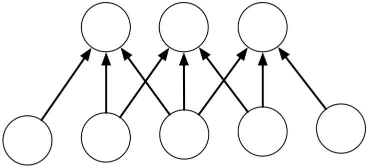


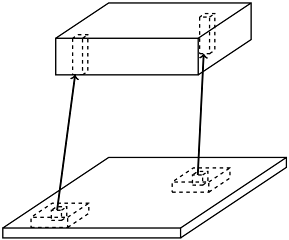


Each hidden unit looks at the entire image.

Convolution Layers

Locally connected layers:

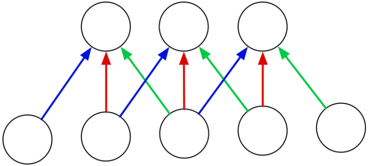


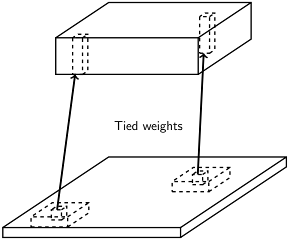


Each column of hidden units looks at a small region of the image.

Convolution Layers

Convolution layers:

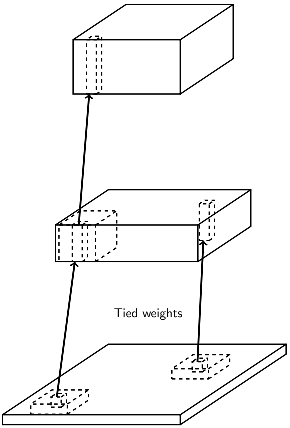




Each column of hidden units looks at a small region of the image, and the weights are shared between all image locations.

Going Deeply Convolutional

Convolution layers can be stacked:



Convolution

We've already been vectorizing our computations by expressing them in terms of matrix and vector operations.

Now we'll introduce a new high-level operation, convolution. Here the motivation isn't computational efficiency — we'll see more efficient ways to do the computations later. Rather, the motivation is to get some understanding of what convolution layers can do.

Convolution

We've already been vectorizing our computations by expressing them in terms of matrix and vector operations.

Now we'll introduce a new high-level operation, convolution. Here the motivation isn't computational efficiency — we'll see more efficient ways to do the computations later. Rather, the motivation is to get some understanding of what convolution layers can do.

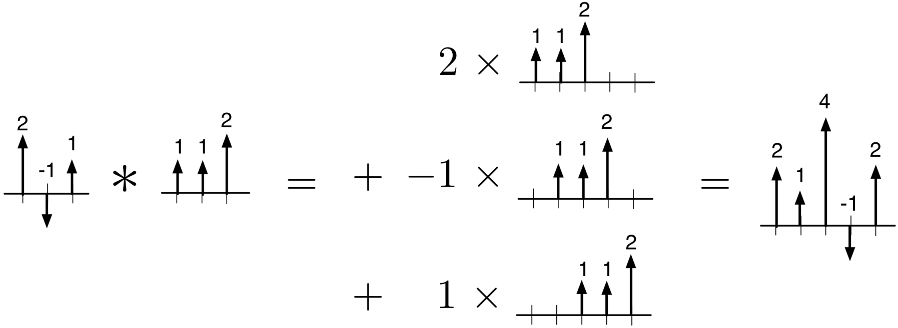
Let's look at the 1-D case first. If a and b are two arrays,

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

Note: indexing conventions are inconsistent. We'll explain them in each case.

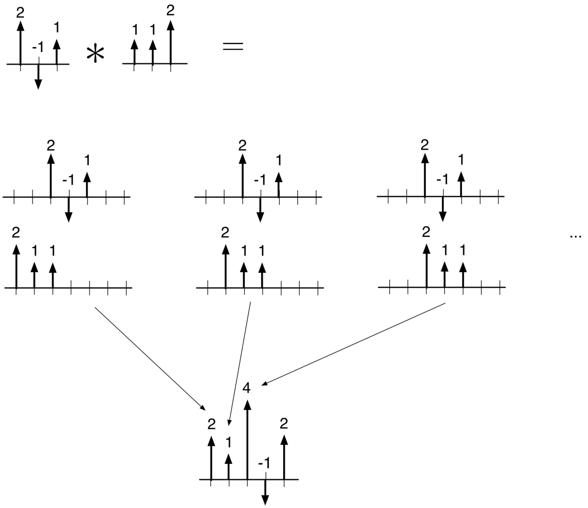
Convolution

Method 1: translate-and-scale



Convolution

Method 2: flip-and-filter



Convolution

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!)

Convolution

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

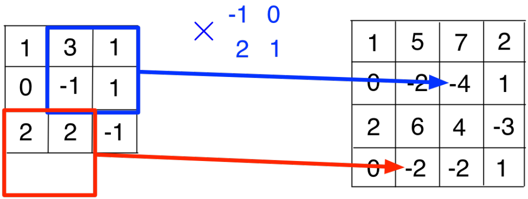
Method 1: Translate-and-Scale

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

2-D Convolution

Method 2: Flip-and-Filter

$$\begin{array}{|c|c|c|}\hline 1 & 3 & 1 \\ \hline 0 & -1 & 1 \\ \hline 2 & 2 & -1 \\ \hline \end{array} \ast \begin{array}{|c|c|}\hline 1 & 2 \\ \hline 0 & -1 \\ \hline \end{array}$$



2-D Convolution

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

What does this convolution kernel 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 convolution kernel do?

$*$

0	1	0
1	4	1
0	1	0

2-D Convolution

What does this convolution kernel do?

$*$

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

2-D Convolution

What does this convolution kernel do?

$*$

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

2-D Convolution

What does this convolution kernel do?

$*$

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

2-D Convolution

What does this convolution kernel do?

$*$

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

2-D Convolution

What does this convolution kernel do?

$$*$$

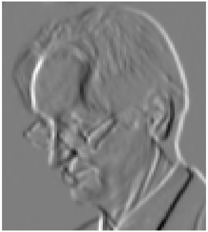
1	0	-1
2	0	-2
1	0	-1

2-D Convolution

What does this convolution kernel do?

$*$

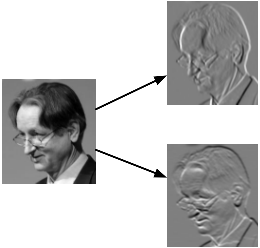
1	0	-1
2	0	-2
1	0	-1



Convolutional networks

Let's finally turn to convolutional networks. These have two kinds of layers: detection layers (or convolution layers), and pooling layers.

The convolution layer has a set of filters. Its output is a set of feature maps, each one obtained by convolving the image with a filter.

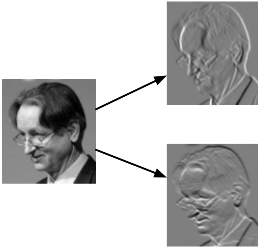


convolution

Convolutional networks

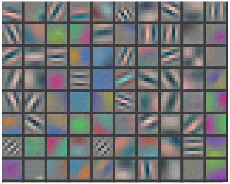
Let's finally turn to convolutional networks. These have two kinds of layers: detection layers (or convolution layers), and pooling layers.

The convolution layer has a set of filters. Its output is a set of feature maps, each one obtained by convolving the image with a filter.



convolution

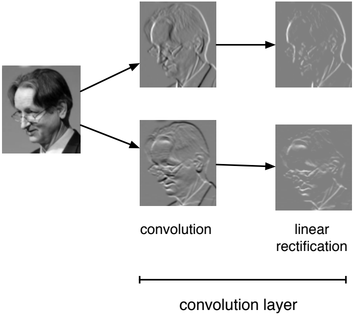
Example first-layer filters



(Zeiler and Fergus, 2013, Visualizing and understanding convolutional networks)

Convolutional networks

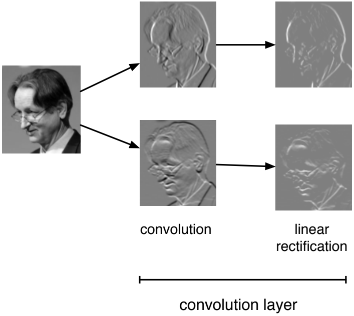
It's common to apply a linear rectification nonlinearity: $y_i = \max(z_i, 0)$



Why might we do this?

Convolutional networks

It's common to apply a linear rectification nonlinearity: $y_i = \max(z_i, 0)$

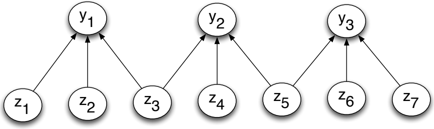


Why might we do this?

- Convolution is a linear operation. Therefore, we need a nonlinearity, otherwise 2 convolution layers would be no more powerful than 1.
- Two edges in opposite directions shouldn't cancel

Pooling layers

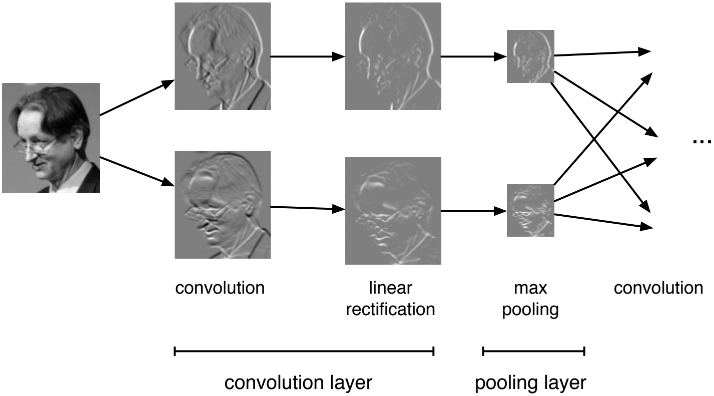
The other type of layer in a pooling layer. These layers reduce the size of the representation and build in invariance to small transformations.



Most commonly, we use max-pooling, which computes the maximum value of the units in a pooling group:

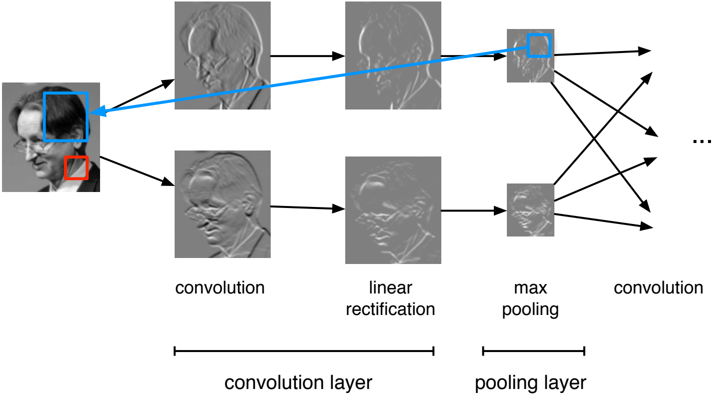
$$y_i = \max_{j \text{ in pooling group}} z_j$$

Convolutional networks



Convolutional networks

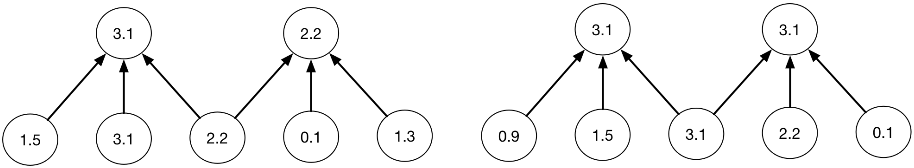
Because of pooling, higher-layer filters can cover a larger region of the input than equal-sized filters in the lower layers.



Equivariance and Invariance

We said the network's responses should be robust to translations of the input. But this can mean two different things.

- Convolution layers are equivariant: if you translate the inputs, the outputs are translated by the same amount.
- We'd like the network's predictions to be invariant: if you translate the inputs, the prediction should not change.
- Pooling layers provide invariance to small translations.



Convolution Layers

Each layer consists of several feature maps, or channels each of which is an array.

- If the input layer represents a grayscale image, it consists of one channel. If it represents a color image, it consists of three channels.

Each unit is connected to each unit within its receptive field in the previous layer. This includes *all* of the previous layer's feature maps.

Convolution Layers

For simplicity, focus on 1-D signals (e.g. audio waveforms). Suppose the convolution layer's input has J feature maps and its output has I feature maps. Let t index the locations. Suppose the convolution kernels have radius R, i.e. dimension $K = 2R + 1$.

Each unit in a convolution layer receives inputs from all the units in its receptive field in the previous layer:

$$y_{i,t} = \sum_{j=1}^{J} \sum_{\tau=-R}^{R} w_{i,j,\tau} x_{j,t+\tau}.$$

In terms of convolution,

$$\mathbf{y}_i = \sum_j \mathbf{x}_j * \mathrm{flip}(\mathbf{w}_{i,j}).$$

Backprop Updates (Optional)

How do we train a conv net? With backprop, of course!

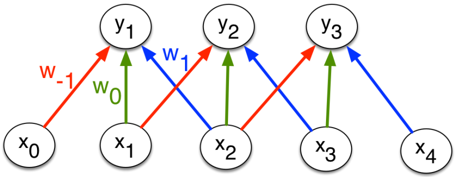
Recall what we need to do. Backprop is a message passing procedure, where each layer knows how to pass messages backwards through the computation graph. Let's determine the updates for convolution layers.

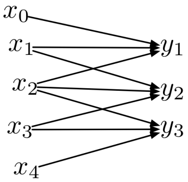
- We assume we are given the loss derivatives $\overline{y_{i,t}}$ with respect to the output units.
- We need to compute the cost derivatives with respect to the input units and with respect to the weights.

The only new feature is: how do we do backprop with tied weights?

Backprop Updates (Optional)

Consider the computation graph for the inputs:





Each input unit influences all the output units that have it within their receptive fields. Using the multivariate Chain Rule, we need to sum together the derivative terms for all these edges.

Backprop Updates (Optional)

Recall the formula for the convolution layer:

$$y_{i,t} = \sum_{j=1}^{J} \sum_{\tau=-R}^{R} w_{i,j,\tau} x_{j,t+\tau}.$$

We compute the derivatives, which requires summing over all the outputs units which have the input unit in their receptive field:

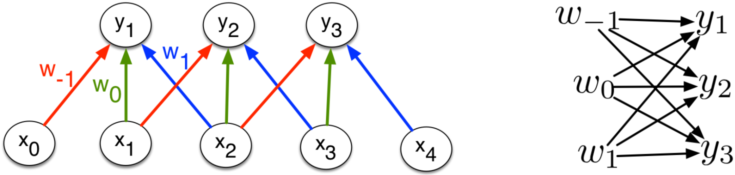
$$\begin{aligned} \overline{x_{j,t}} &= \sum_{\tau} \overline{y_{i,t-\tau}} \frac{\partial y_{i,t-\tau}}{\partial x_{j,t}} \\ &= \sum_{\tau} \overline{y_{i,t-\tau}} \, w_{i,j,\tau} \end{aligned}$$

Written in terms of convolution,

$$\overline{\mathbf{x}_j} = \overline{\mathbf{y}_i} * \mathbf{w}_{i,j}.$$

Backprop Updates (Optional)

Consider the computation graph for the weights:



Each of the weights affects all the output units for the corresponding input and output feature maps.

Backprop Updates (Optional)

Recall the formula for the convolution layer:

$$y_{i,t} = \sum_{j=1}^{J} \sum_{\tau=-R}^{R} w_{i,j,\tau} x_{j,t+\tau}.$$

We compute the derivatives, which requires summing over all spatial locations:

$$\begin{aligned} \overline{w_{i,j,\tau}} &= \sum_t \overline{y_{i,t}} \frac{\partial y_{i,t}}{\partial w_{i,j,\tau}} \\ &= \sum_t \overline{y_{i,t}} \, x_{j,t+\tau} \end{aligned}$$

After the break

After the break: **Apply CNN to Image Classification**

Object recognition

- Object recognition is the task of identifying which object category is present in an image.
- It's challenging because objects can differ widely in position, size, shape, appearance, etc., and we have to deal with occlusions, lighting changes, etc.
- Why we care about it
  - Direct applications to image search
  - Closely related to object detection, the task of locating all instances of an object in an image
    - E.g., a self-driving car detecting pedestrians or stop signs
- For the past 6 years, all of the best object recognizers have been various kinds of conv nets.

Datasets

- In order to train and evaluate a machine learning system, we need to collect a dataset. The design of the dataset can have major implications.
- Some questions to consider:
  - Which categories to include?
  - Where should the images come from?
  - How many images to collect?
  - How to normalize (preprocess) the images?

Image Classification

- Conv nets are just one of many possible approaches to image classification. However, they have been by far the most successful for the last 8 years.
- Biggest image classification "advances" of the last two decades
  - Datasets have gotten much larger (because of digital cameras and the Internet)
  - Computers got much faster
    - Graphics processing units (GPUs) turned out to be really good at training big neural nets; they're generally about 30 times faster than CPUs.
  - As a result, we could fit bigger and bigger neural nets.

MNIST Dataset

- 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.

ImageNet

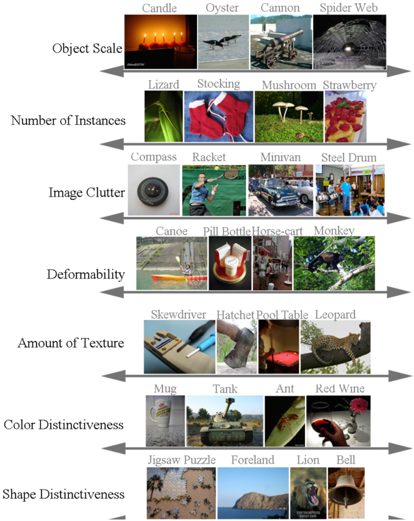
ImageNet is the modern object recognition benchmark dataset. It was introduced in 2009, and has led to amazing progress in object recognition since then.

ImageNet

- 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

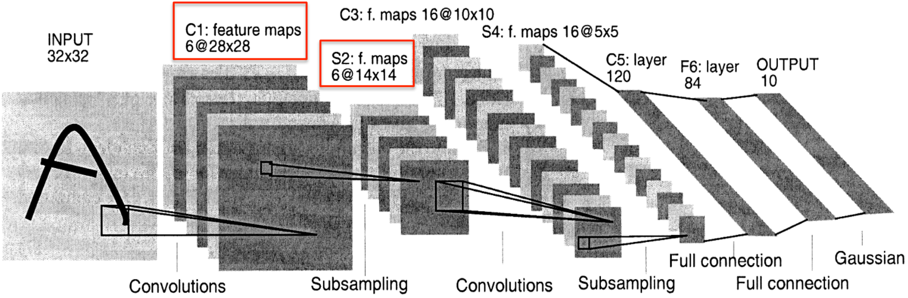
Size on disk:

MNIST
60 MB

ImageNet
50 GB

LeNet

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

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

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.

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

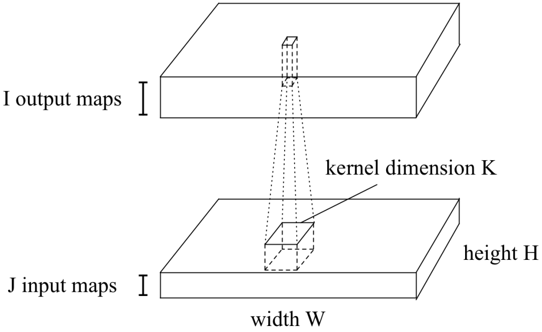
Size of a Conv Net

- 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).

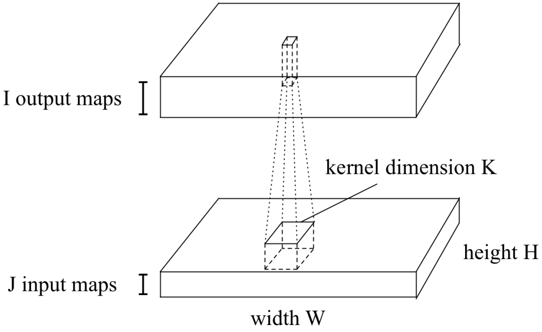
Size of a Conv Net

- 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

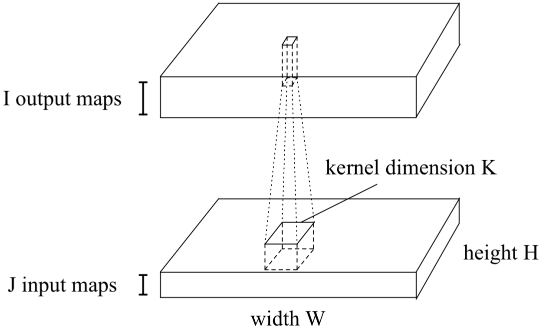


Size of a Conv Net



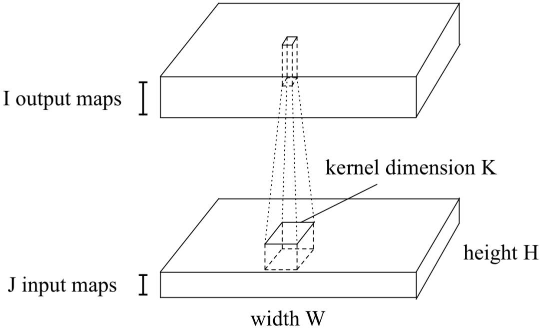
	fully connected layer	convolution layer
# output units		

Size of a Conv Net



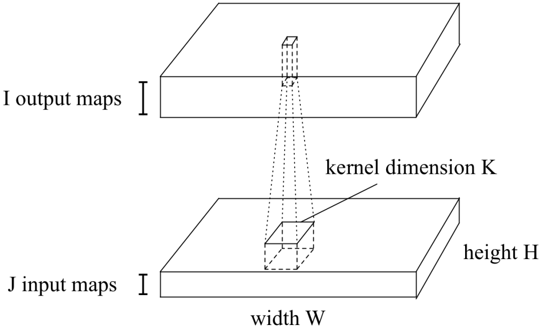
	fully connected layer	convolution layer
# output units	WHI	WHI

Size of a Conv Net



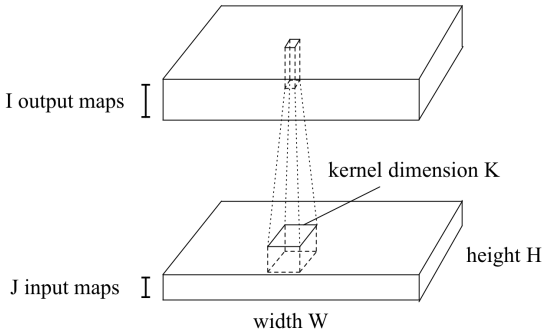
	fully connected layer	**convolution layer**
# output units	WHI	WHI
# weights		

Size of a Conv Net



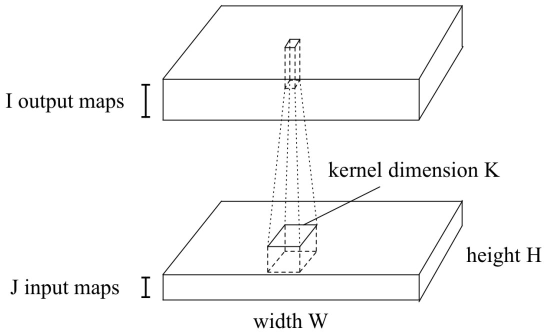
	fully connected layer	convolution layer
# output units	WHI	WHI
# weights	W^2H^2IJ	

Size of a Conv Net



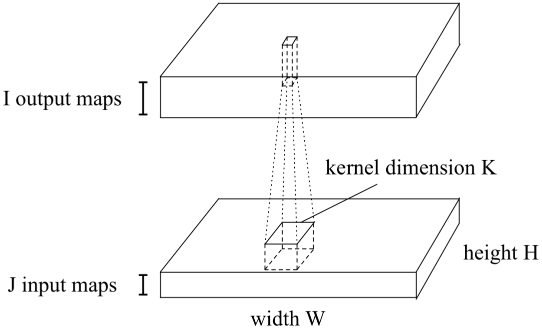
	fully connected layer	convolution layer
# output units	WHI	WHI
# weights	W^2H^2IJ	K^2IJ

Size of a Conv Net



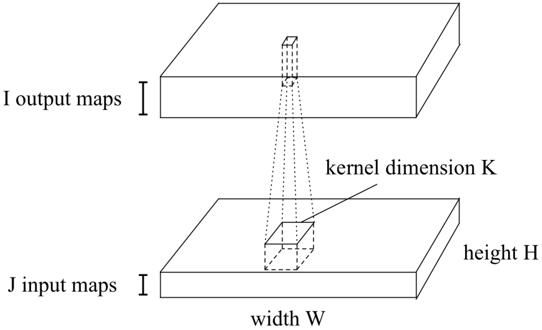
	fully connected layer	convolution layer
# output units	WHI	WHI
# weights	W^2H^2IJ	K^2IJ
# connections		

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?

Size of a Conv Net

- 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!

Size of a Conv Net

	LeNet (1989)	**LeNet (1998)**	**AlexNet (2012)**
classification task	digits	digits	objects

Size of a Conv Net

	LeNet (1989)	LeNet (1998)	AlexNet (2012)
classification task	digits	digits	objects
categories	10	10	1,000

Size of a Conv Net

	LeNet (1989)	LeNet (1998)	AlexNet (2012)
classification task	digits	digits	objects
categories	10	10	1,000
image size	16×16	28×28	$256 \times 256 \times 3$

Size of a Conv Net

	LeNet (1989)	LeNet (1998)	AlexNet (2012)
classification task	digits	digits	objects
categories	10	10	1,000
image size	16×16	28×28	$256 \times 256 \times 3$
training examples	7,291	60,000	1.2 million

Size of a Conv Net

	LeNet (1989)	**LeNet (1998)**	**AlexNet (2012)**
classification task	digits	digits	objects
categories	10	10	1,000
image size	16×16	28×28	$256 \times 256 \times 3$
training examples	7,291	60,000	1.2 million
units	1,256	8,084	658,000

Size of a Conv Net

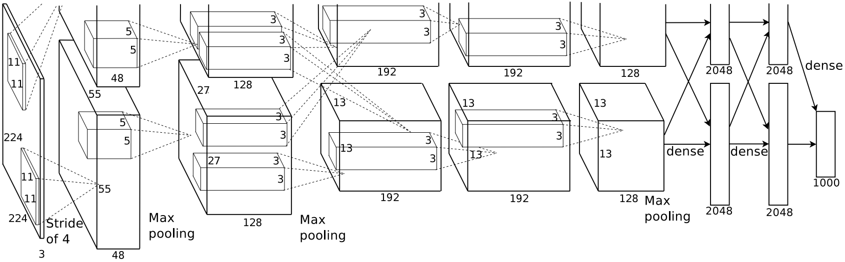
	LeNet (1989)	**LeNet (1998)**	**AlexNet (2012)**
classification task	digits	digits	objects
categories	10	10	1,000
image size	16×16	28×28	$256 \times 256 \times 3$
training examples	7,291	60,000	1.2 million
units	1,256	8,084	658,000
parameters	9,760	60,000	60 million

Size of a Conv Net

	LeNet (1989)	LeNet (1998)	AlexNet (2012)
classification task	digits	digits	objects
categories	10	10	1,000
image size	16×16	28×28	$256 \times 256 \times 3$
training examples	7,291	60,000	1.2 million
units	1,256	8,084	658,000
parameters	9,760	60,000	60 million
connections	65,000	344,000	652 million

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)

- They used lots of tricks we've covered in this course (ReLU units, weight decay, data augmentation, SGD with momentum, dropout)
- AlexNet's stunning performance on the ILSVRC is what set off the deep learning boom of the last 6 years.

GoogLeNet

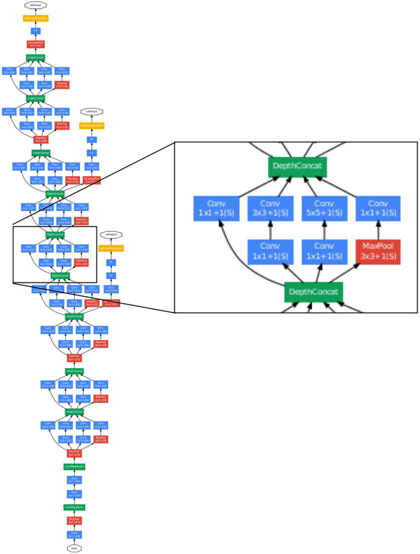
GoogLeNet, 2014.

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



GoogLeNet

- 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)
  - GoogLeNet 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!)

Classification

ImageNet results over the years. Note that errors are top-5 errors (the network gets to make 5 guesses).

Year	**Model**	**Top-5 error**
2010	Hand-designed descriptors + SVM	28.2%
2011	Compressed Fisher Vectors + SVM	25.8%
2012	AlexNet	16.4%
2013	a variant of AlexNet	11.7%
2014	GoogLeNet	6.6%
2015	deep residual nets	4.5%

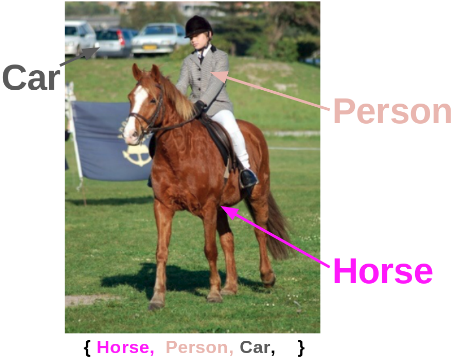
We'll cover deep residual nets later in the course, since they require an idea we haven't covered yet.

Human-performance is around 5.1%.

They stopped running the object recognition competition because the performance is already so good.

Beyond Classification

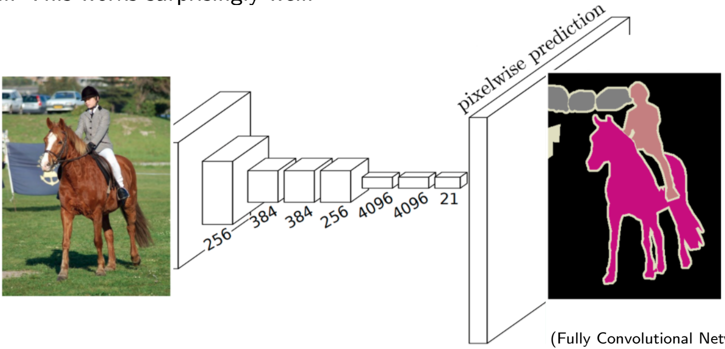
- The classification nets map the entire input image to a pre-defined class categories.
- But there are more than just class labels in an image.
  - where is the foreground object? how many? what is in the background?



(PASCAL VOC 2012)

Semantic Segmentation

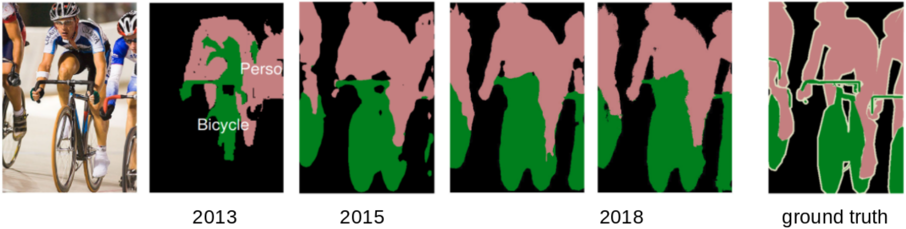
- Semantic segmentation, a natural extention of classification, focuses on making dense classification of class labels for every pixel.
- It is an important step towards complete scene understanding in compter vision.
  - Semantic segmentation is a stepping stone for many of the high-level vision tasks, such as object detection, Visual Question Answering (VQA).
- A naive approach is to adapt the existing object classification conv nets for each pixel. This works surprisingly well.



(Fully Convolutional Networks, 2015)

Semantic Segmentation

- After the success of CNN classifiers, segmentation models quickly moved away from hand-craft features and pipelines but instead use CNN as the main structure.
- Pre-trained ImageNet classification network serves as a building block for all the state-of-the-art CNN-based segmentation models.



from left to wright (Li, et. al., (CSI), CVPR, 2013; Long, et. al., (FCN), CVPR 2015; Chen et. al., (DeepLab), PAMI 2018)

Supervised Pre-training and Transfer Learning

- In practice, we will rarely train an image classifier from scratch.
    - It is unlikely we will have millions of cleanly labeled images for our specific datasets.
- If the dataset is a computer vision task, it is common to fine-tune a pre-trained conv net on ImageNet or OpenImage.
- Just like semantic segmentation tasks, we will fix most of the weights in the pre-trained network. Only the weights in the last layer will be randomly initialized and learnt on the current dataset/task.

Supervised Pre-training and Transfer Learning

- When to fine-tune?
  - How many training examples we have in the new dataset/task?
    - Fewer new examples: more weights from the pre-trained networks are fixed.
  - How similar is the new dataset to our pre-training dataset? Microspy images v.s. natural images:
    - more fine-tuning is needed for dissimilar datasets.
  - Learning rate for the fine-tuning stage is often much lower than the learning rate used for training from scratch.