COMS 4995 Lecture 5: Convolutional Neural Networks & Image Classification

Richard Zemel

Logistics

- Assignment 1 due Friday at 2pm
- Late policy: Except in the case of an official Student Medical Certificate, assignments will be accepted with a 10% penalty every 24 hours from the deadline, up to 5 days. After that no credit.
- Project: guidelines released; suggestions to follow
- Midterm
  - Will cover up thru CNNs (Monday's lecture)
  - Estimated time: 1 hour, but will have full period
  - Reviews: Thursday at 5, Monday at 4.

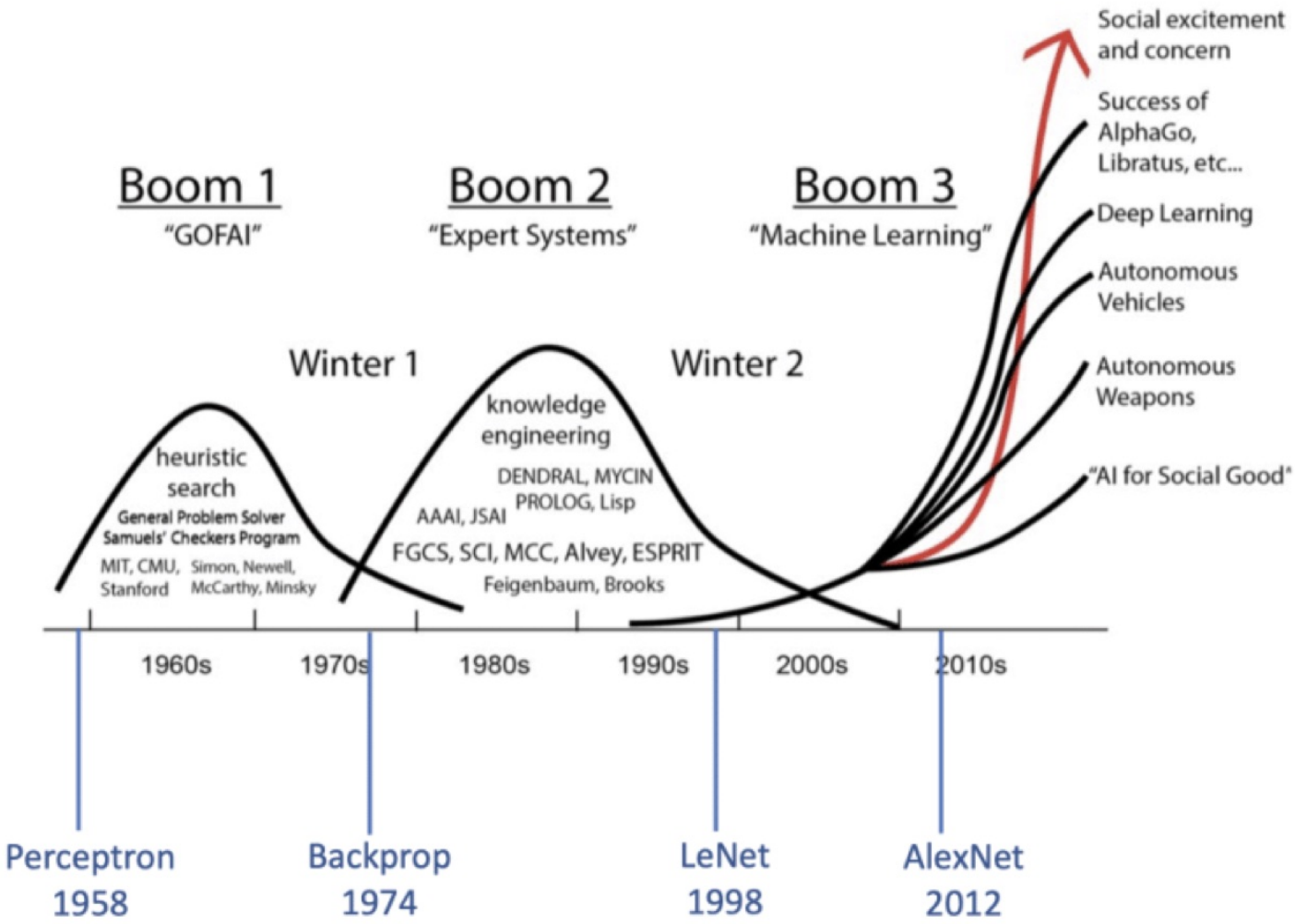
Overview

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.

A brief history



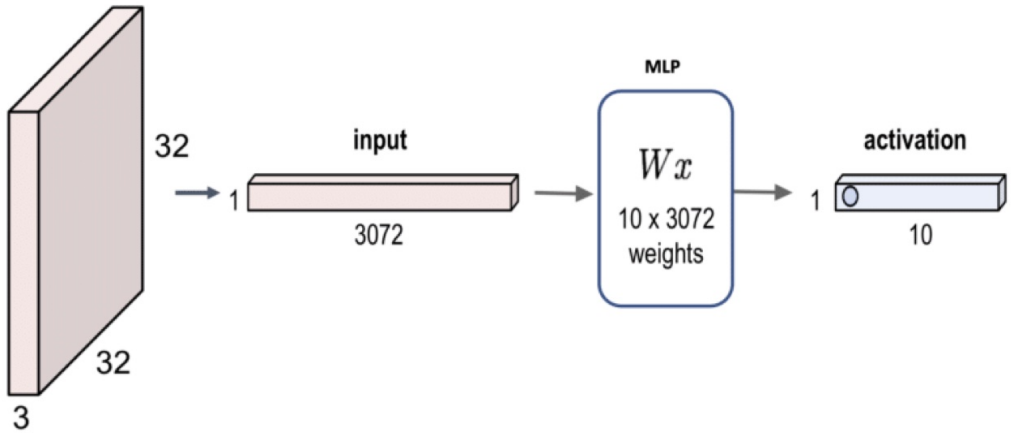
How do we teach computers vision?

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: "Imagine a medical database in which the age of the patient sometimes hops to the input dimension which normally codes for weight!"

How do we teach computers vision?

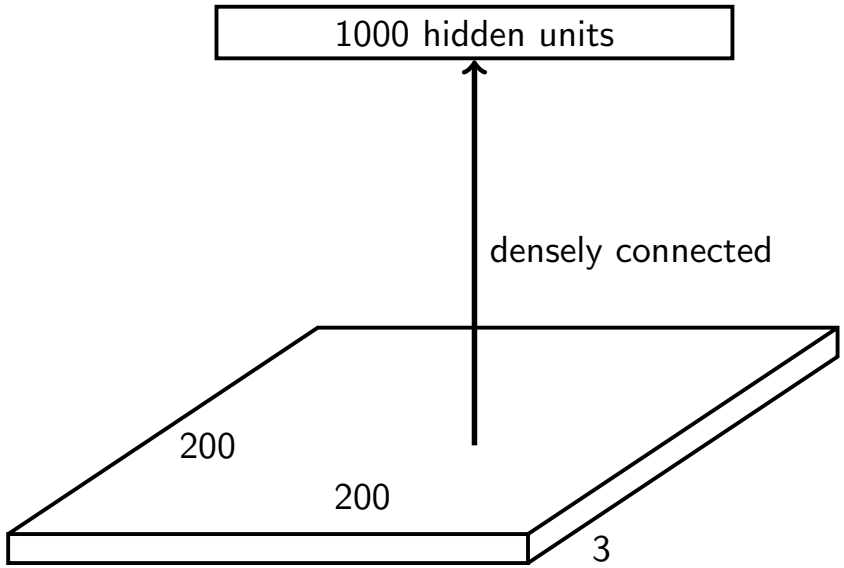
What will you do?



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.

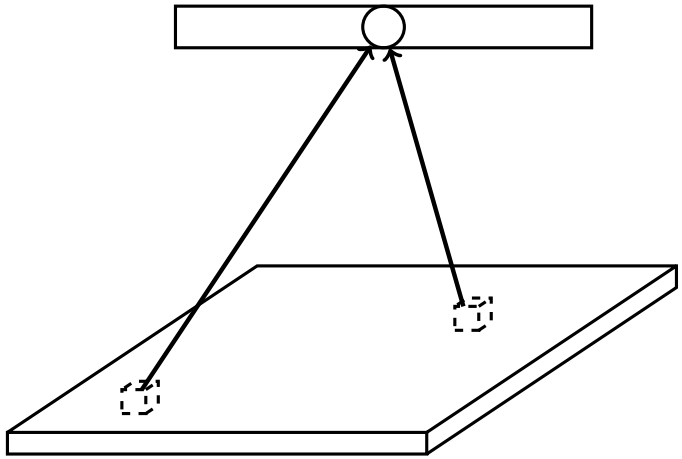


What is the problem with having this as the first layer?

- Too many parameters! Input size $= 200 \times 200 \times 3 = 120$K. Parameters $= 120\text{K} \times 1000 = 120$ million.
- What happens if the object in the image shifts a little?

How do we teach computers vision?

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.

How do we teach computers vision?

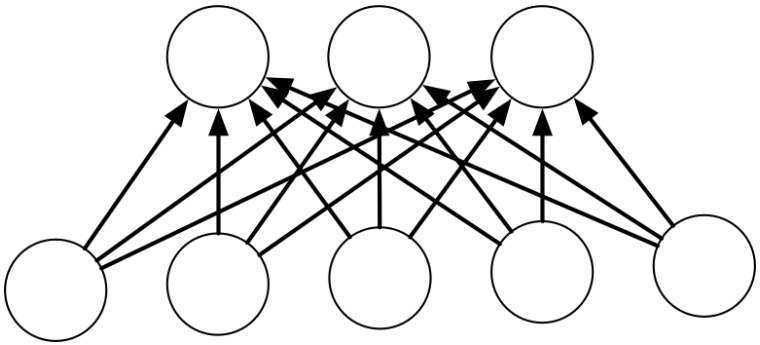
The same sorts of features that are useful in analyzing one part of the image will probably be useful for analyzing other parts as well.

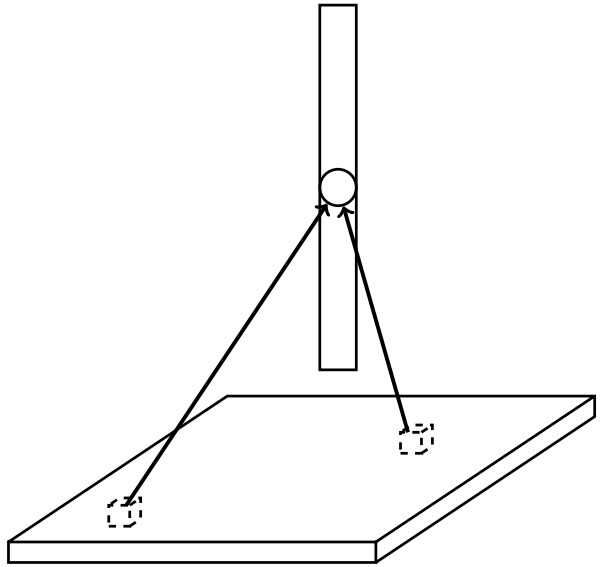
E.g., edges, corners, contours, object parts

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:

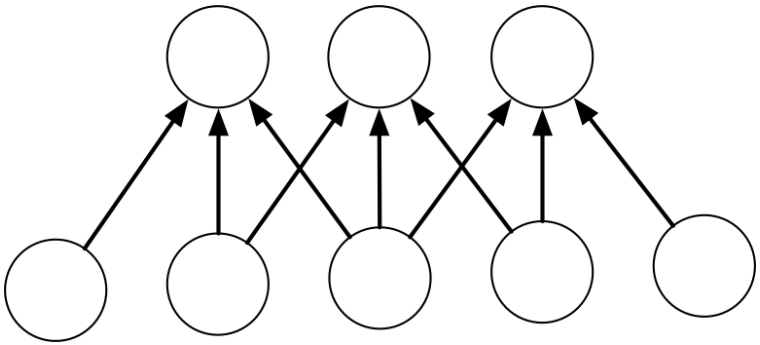


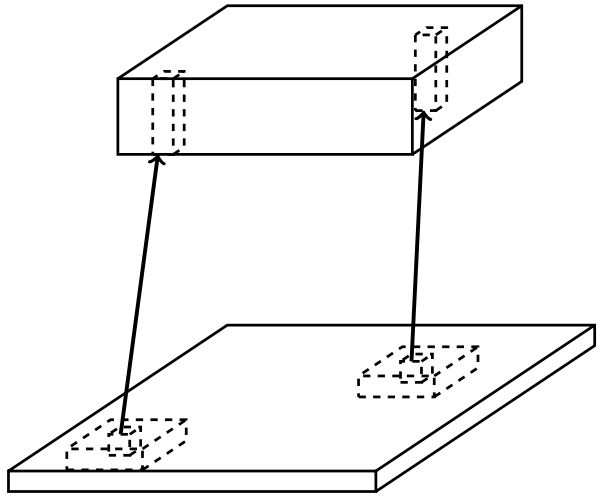


Each hidden unit looks at the entire image.

Convolution Layers

Locally connected layers:

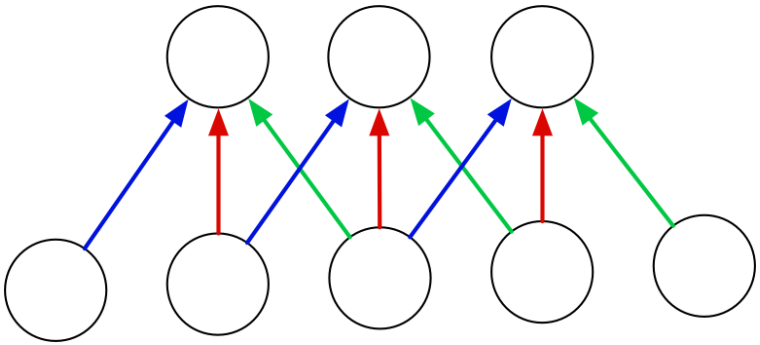


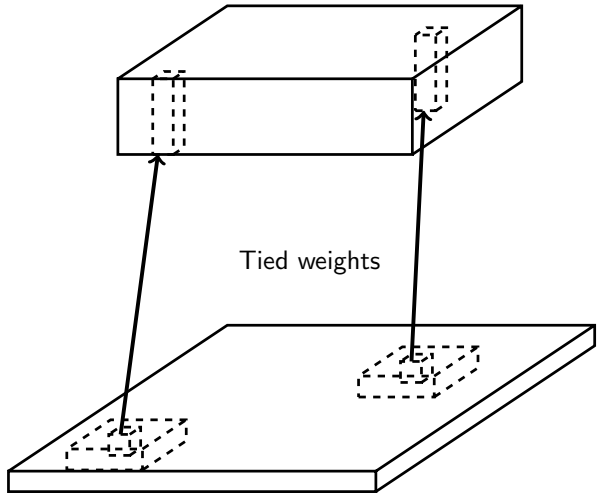


Each column of hidden units looks at a small region of the image.

Convolution Layers

Convolution layers:

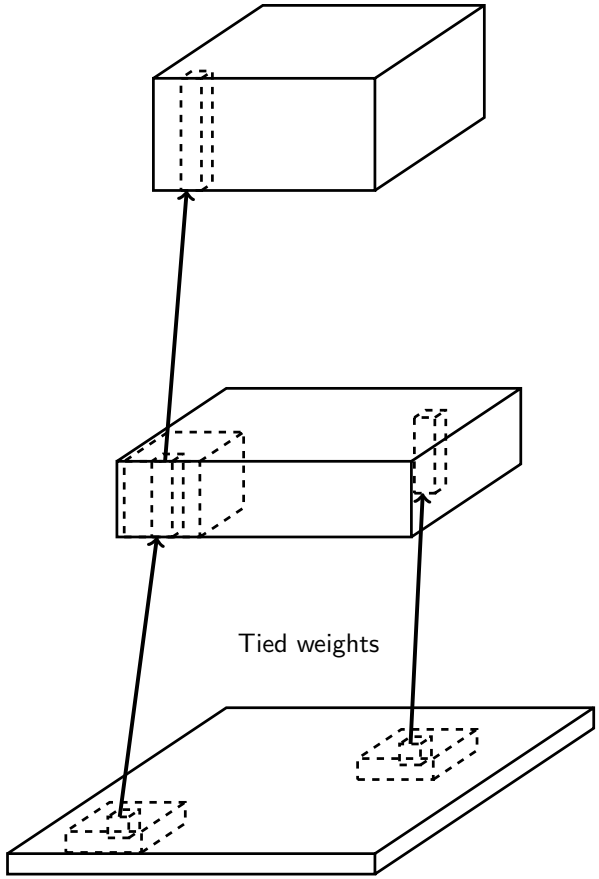




Each column of hidden units looks at a small region of the image, and the weights are shared between all image locations.

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.

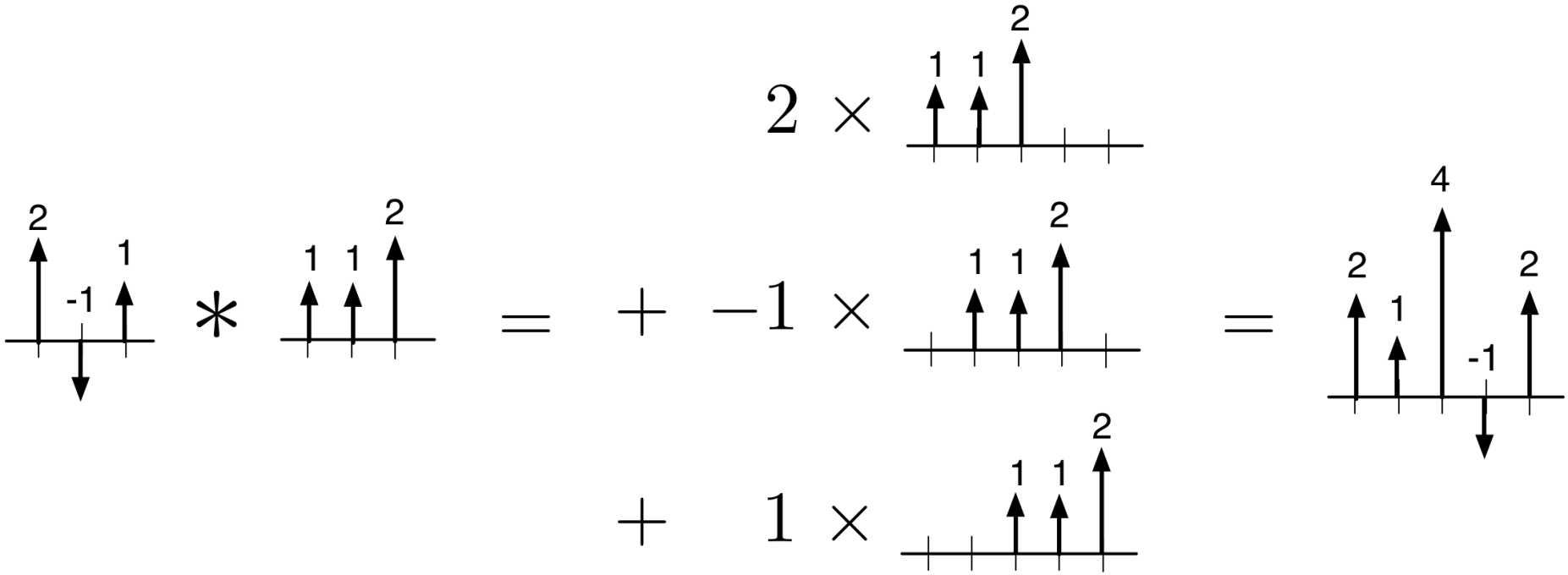
Let's look at the 1-D case first. If a and b are two arrays,

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

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

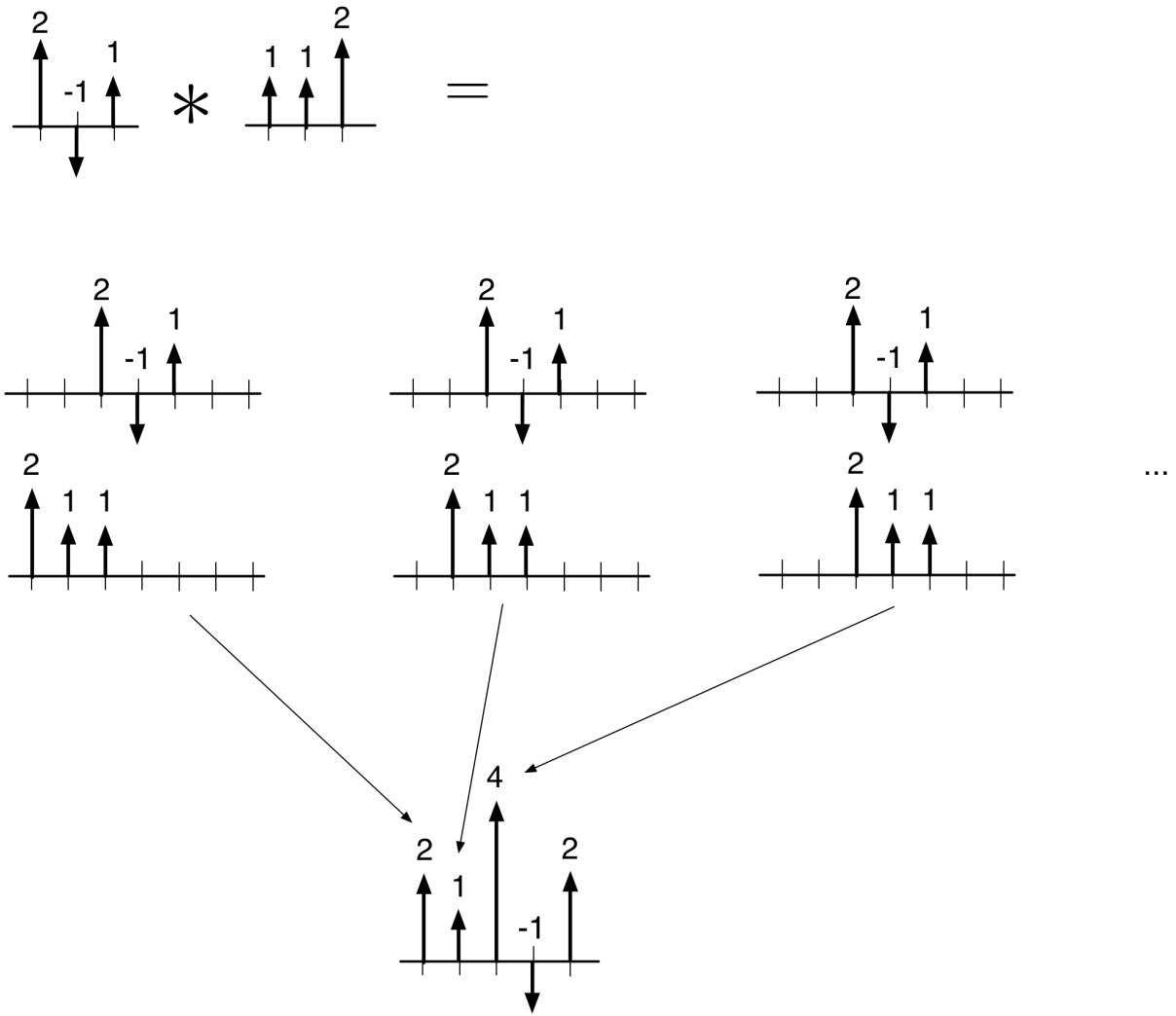
Convolution

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

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

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

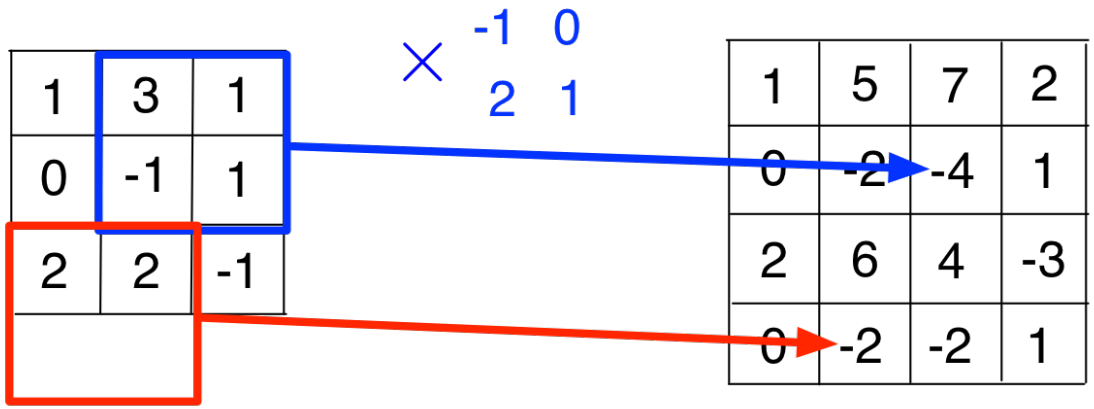
2-D Convolution

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 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

Answer: Blur

Note: We call the resulting image an "activation map" by the kernel

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

Answer: Sharpen

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

Answer: Edge Detection

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?

$*$

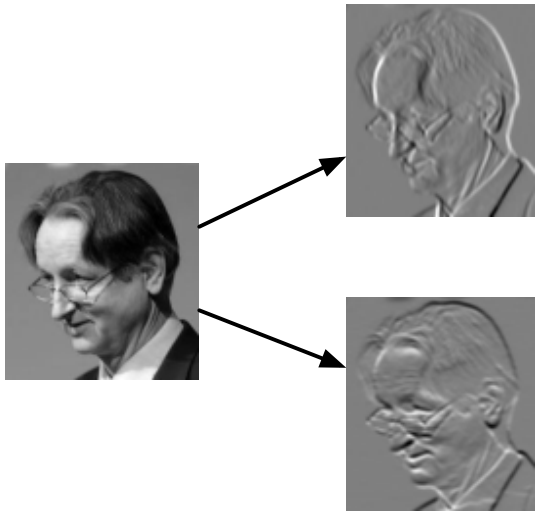
1	0	-1
2	0	-2
1	0	-1

Answer: Stronger Edge Detection

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.

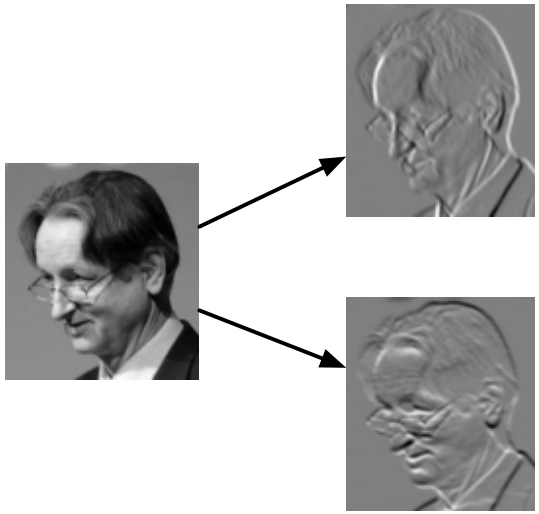


convolution

Convolutional networks

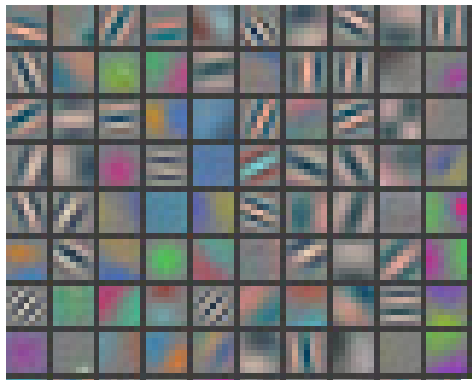
Let's finally turn to convolutional networks. These have two kinds of layers: detection layers (or convolution layers), and pooling layers.

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)