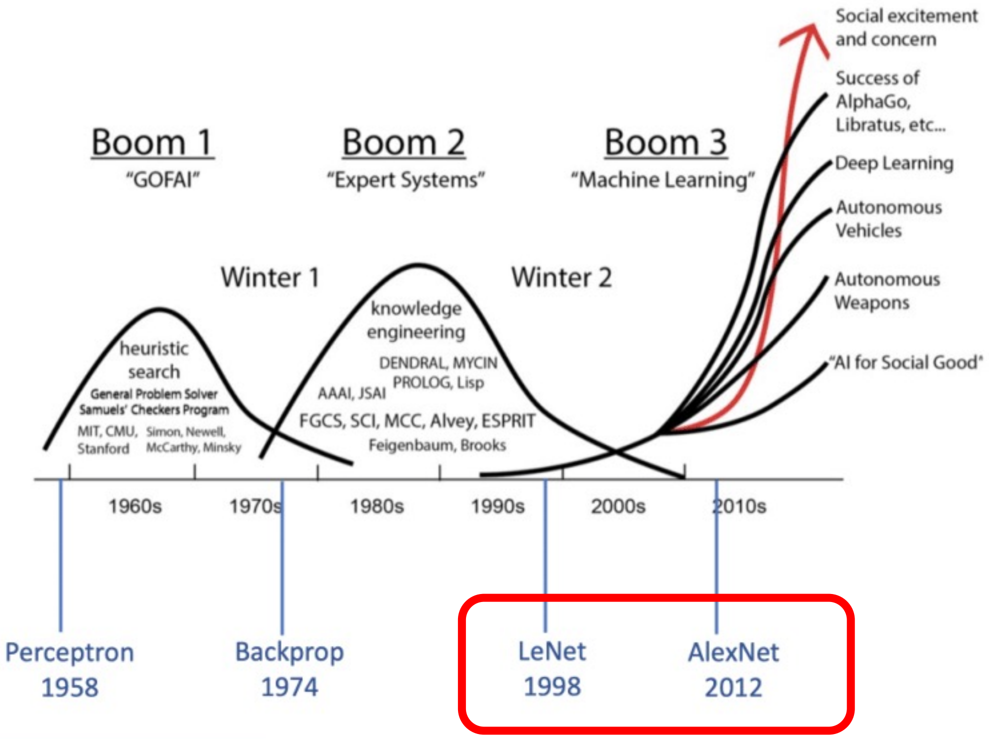
LMP 1210H: Basic Principles of Machine Learning in Biomedical Research

Bo Wang
AI Lead Scientist, PMCC, UHN
CIFAR AI Chair, Vector Institute
Assistant Professor, University of Toronto

A brief history



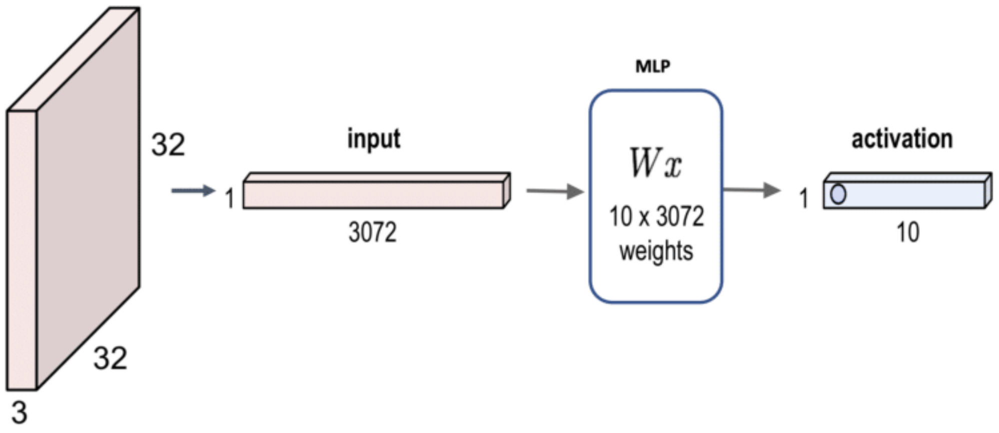
How do we teach computers vision?

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: "Imaging 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?

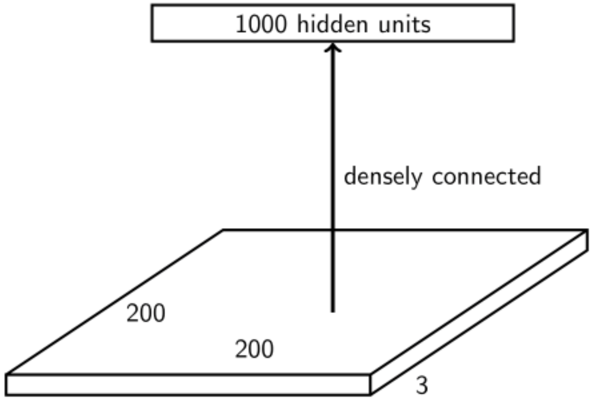
What will you do before this lecture?



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

How do we teach computers vision?

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

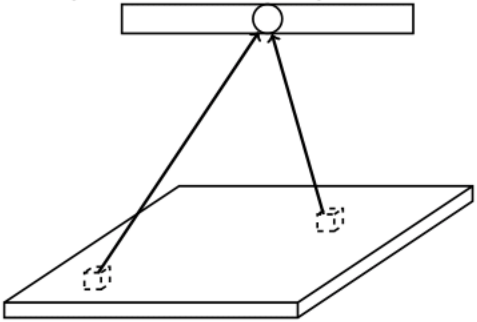


What is the problem with having this as the first layer?

- Too many parameters! Input size $= 200 \times 200 \times 3 = 120\text{K}$. 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.

A brief review: 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.

A brief review: 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.

A brief review: 2-D Convolution

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

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

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

Convolution

Some properties of convolution:

- Commutativity

$$a * b = b * a$$

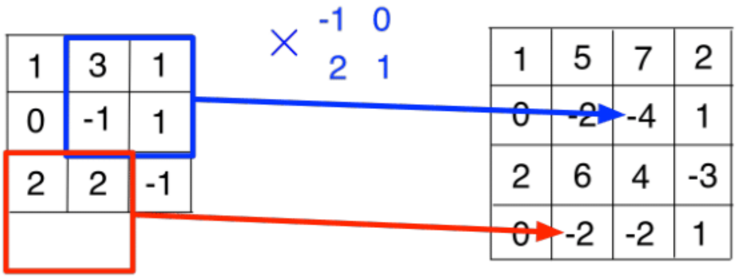
- Linearity

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

A brief review: 2-D Convolution

Flip-and-Filter

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



Apply convolutions on images

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

What does this convolution kernel do?

$*$

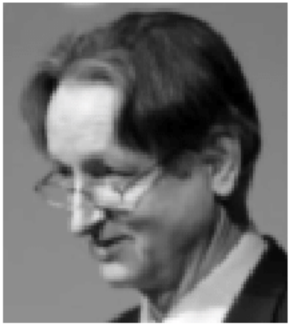
0	1	0
1	4	1
0	1	0

Apply convolutions on images

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

What does this convolution kernel do?

$$* \begin{array}{|c|c|c|} \hline 0 & 1 & 0 \\ \hline 1 & 4 & 1 \\ \hline 0 & 1 & 0 \\ \hline \end{array}$$



Answer: Blur

Note: We call the resulted image as an "activation map" by the kernel.

Apply convolutions on images

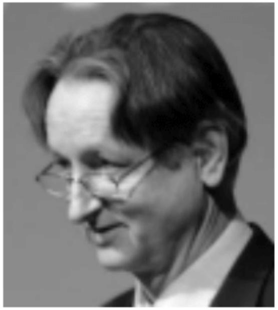
What does this convolution kernel do?

$*$

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

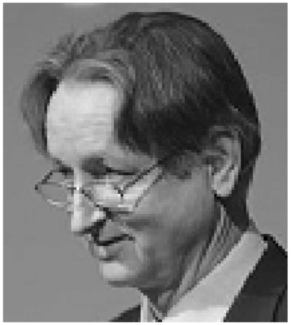
Apply convolutions on images

What does this convolution kernel do?



$*$

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



Answer: Sharpen

Note: We call the resulted image as an "activation map" by the kernel.

Apply convolutions on images

What does this convolution kernel do?

$*$

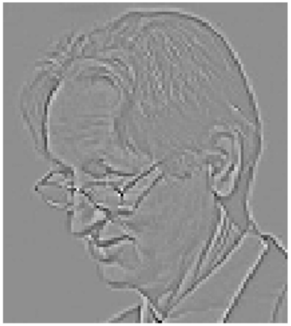
0	-1	0
-1	4	-1
0	-1	0

Apply convolutions on images

What does this convolution kernel do?

$*$

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



Answer: Edge Detection

Note: We call the resulted image as an "activation map" by the kernel.

Apply convolutions on images

What does this convolution kernel do?

$$*$$

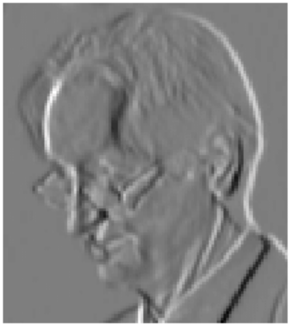
1	0	-1
2	0	-2
1	0	-1

Apply convolutions on images

What does this convolution kernel do?

$*$

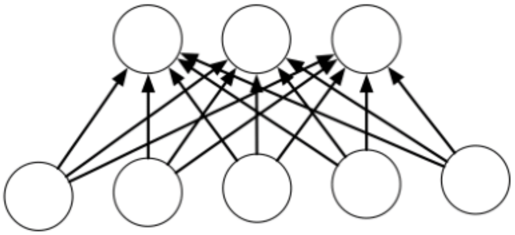
1	0	-1
2	0	-2
1	0	-1

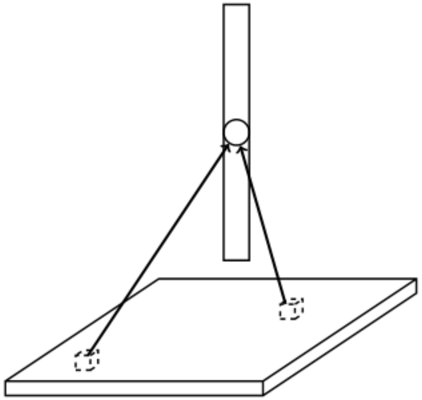


Answer: "Stronger" Edge Detection
Note: We call the resulted image as an "activation map" by the kernel.

A new layer: Convolution Layers

Fully connected layers:

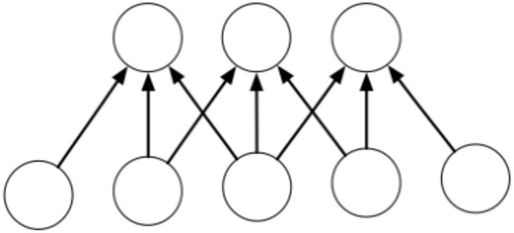


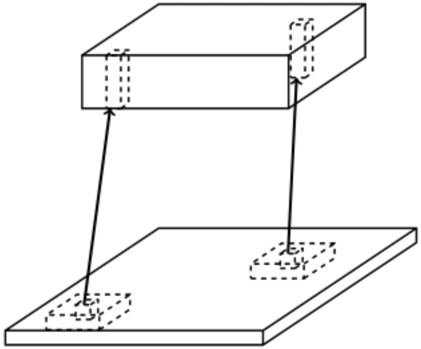


Each hidden unit looks at the entire image.

A new layer: Convolution Layers

Locally connected layers:

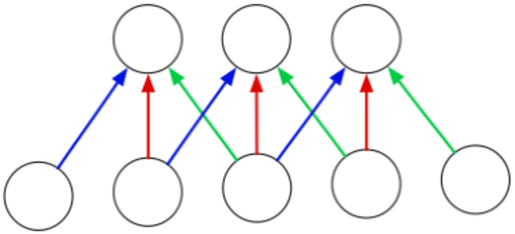


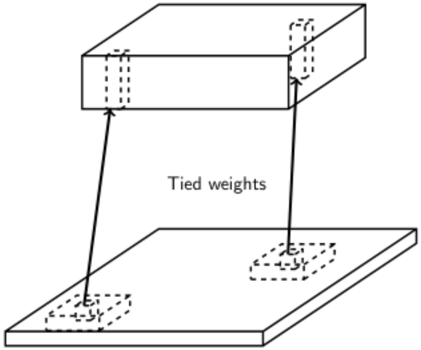


Each column of hidden units looks at a small region of the image.

A new layer: Convolution Layers

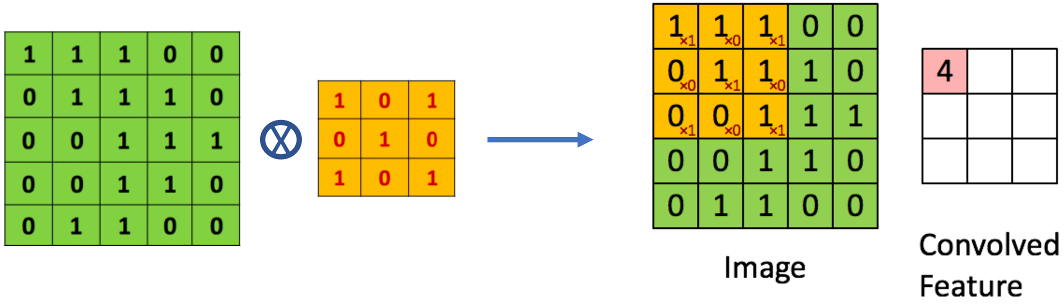
Convolution layers:





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

Example Time: A closer look at convolution



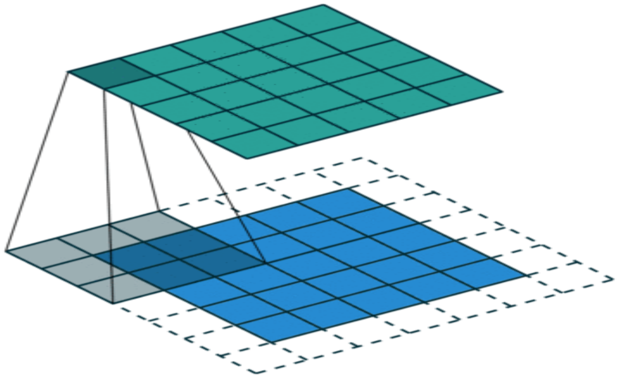
Input: 5 x 5

Kernel: 3 x 3

Output: 3x 3

3 = (5 – 3) + 1

Example Time: A closer look at convolution with padding



Input: 5 x 5

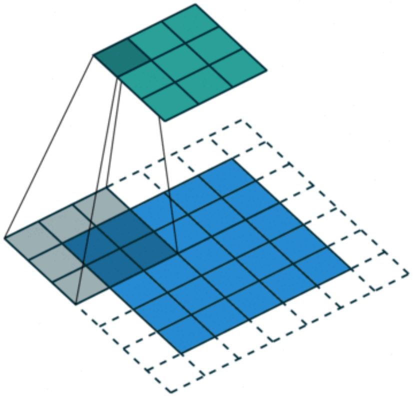
Kernel: 3 x 3

Padding: 1

Output: 5 x 5

5 = (5 – 3 + 2 * 1) + 1

Example Time: A closer look at convolution with stride



Input: 5 x 5

Kernel: 3 x 3

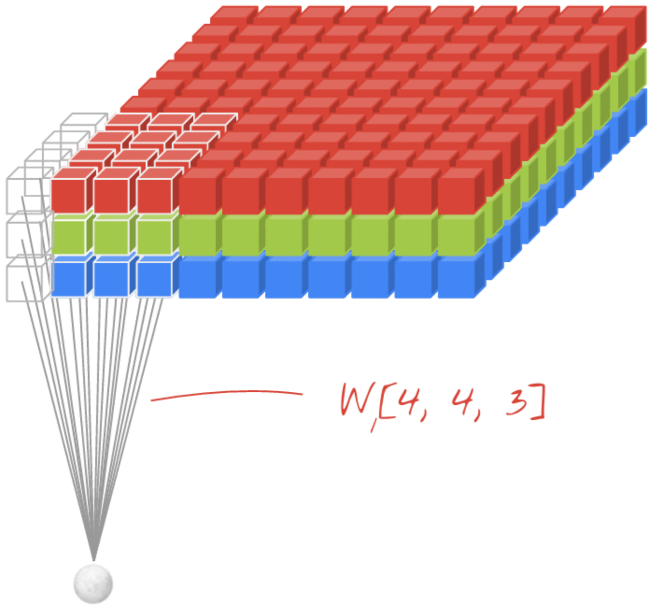
Padding: 1

Stride = 2

Output: 3 x 3

3 = (5 – 3 + 2 * 1)/2 + 1

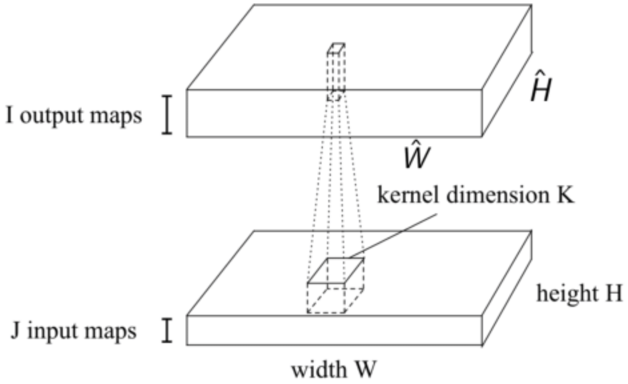
A closer look at convolution with high dimensional inputs



$W_1[4, 4, 3]$

Note: Kernel depth always equals to the input depth.

Summary : Convolution Layer



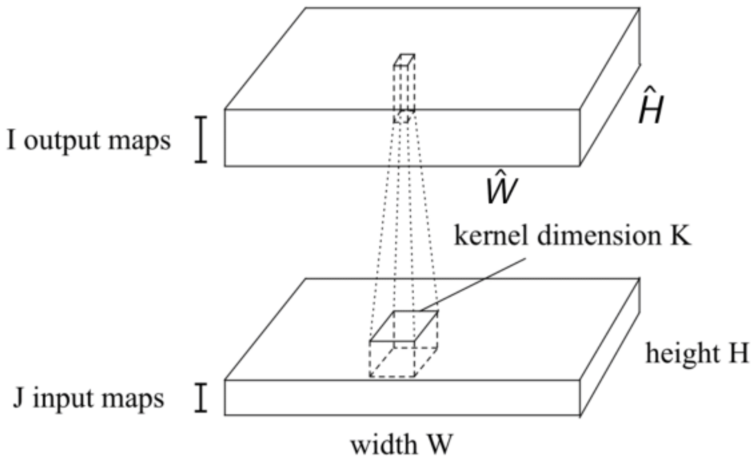
- Input: An array of size $W \times H \times J$
- Hyper-parameters:
  - Number of filters: M
  - Size of filters: K
  - the stride: S
  - Number of zero-padding: P

- Output: Feature maps of size $\hat{W} \times \hat{H} \times I$
  - $\hat{W} = (W - K + 2P)/S + 1$
  - $\hat{H} = (H - K + 2P)/S + 1$
  - $I = M$

Let's do some counting: Size of a Conv Net

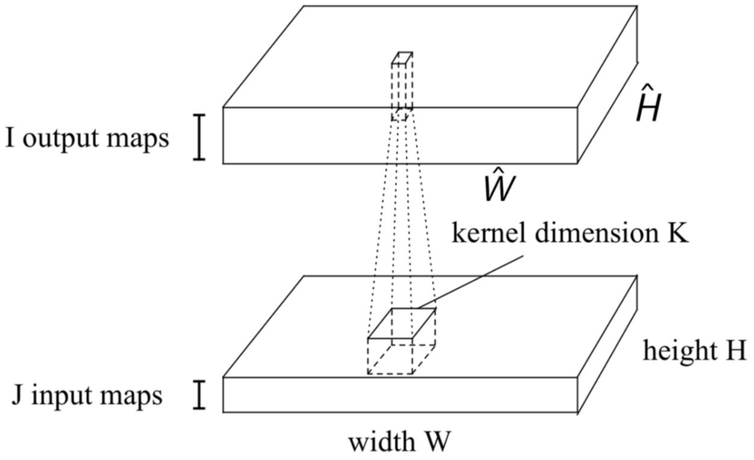
- Ways to measure the size of a network:
  - **Number of units.** This is important because the activations need to be stored in memory during training (i.e. backprop).
  - **Number of weights.** This is important because the weights need to be stored in memory, and because the number of parameters determines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

Let's do some counting: Size of a Conv Net



	fully connected layer	convolution layer
# output units	$\hat{W}\hat{H}I$	$\hat{W}\hat{H}I$
# weights	$W\hat{W}H\hat{H}IJ$	K^2IJ
# connections	$W\hat{W}H\hat{H}IJ$	$\hat{W}\hat{H}K^2IJ$

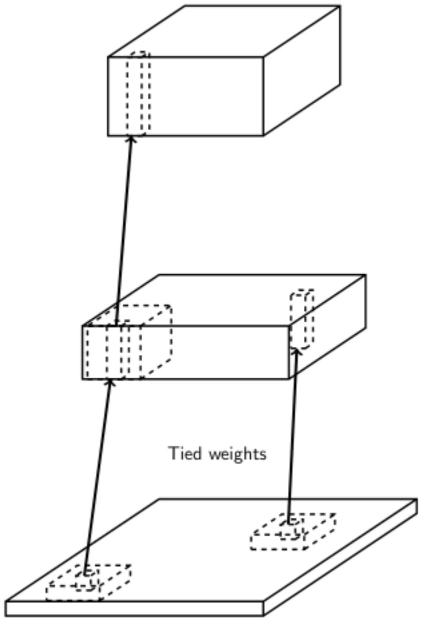
Let's do some counting: Size of a Conv Net (Including bias terms)



	fully connected layer	convolution layer
# output units	$\hat{W}\hat{H}I$	$\hat{W}\hat{H}I$
# weights	$W\hat{W}H\hat{H}IJ + \hat{W}\hat{H}I$	$K^2IJ + I$
# connections	$W\hat{W}H\hat{H}IJ + \hat{W}\hat{H}I$	$\hat{W}\hat{H}K^2IJ + \hat{W}\hat{H}I$

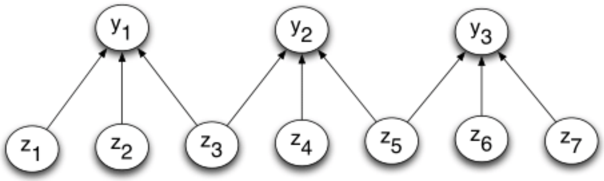
Going Deeply Convolutional

Convolution layers can be stacked:



Pooling layers

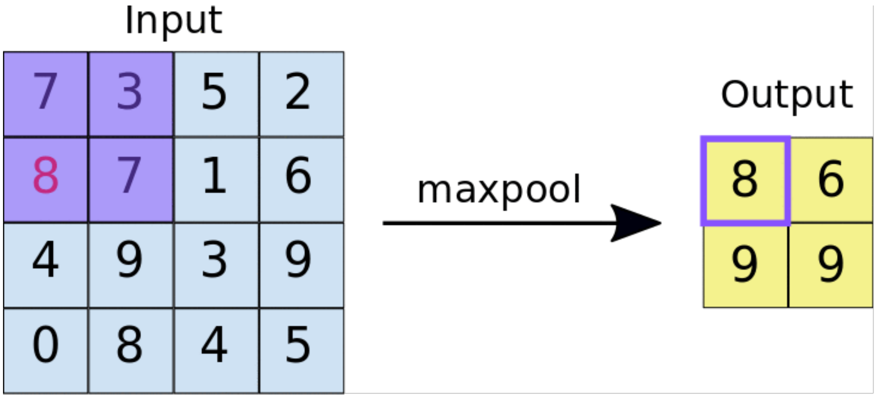
The other type of layer in a pooling layer. These layers reduce the size of the representation and build in invariance to small transformations.



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

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

Pooling Layer

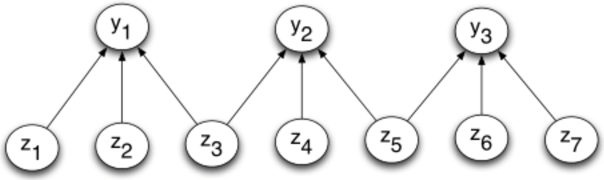


Input: 4 x 4
Kernel: 2 x 2
Stride: 2

Output: 2 x 2

2 = (4 – 2) / 2 + 1

Summary : Pooling Layer



- Input: An array of size $W \times H \times J$
- Hyper-parameters:
  - Size of filters: K
  - the stride: S

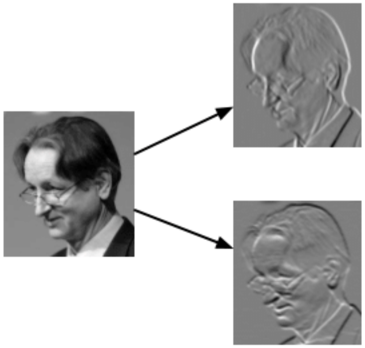
- Output: Feature maps of size $\hat{W} \times \hat{H} \times I$
  - $\hat{W} = (W - K)/S + 1$
  - $\hat{H} = (H - K)/S + 1$
  - $I = J$

Common Setting: K = 2, S = 2

Convolutional networks

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

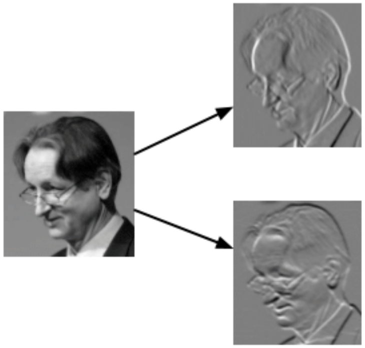
convolution layer has a set of filters. Its output is a set of feature maps, each one obtained by convolving the image with a filter.



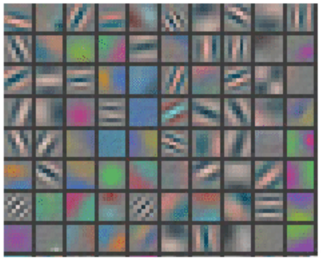
Convolutional networks

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.



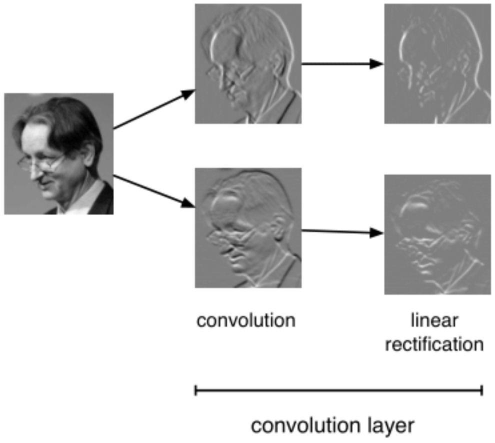
Example first-layer filters



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

Convolutional networks

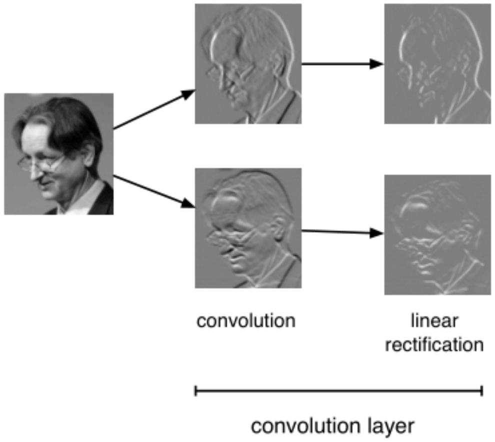
It's common to apply a linear rectification nonlinearity: $y_i = \max(z_i, 0)$



Why might we do this?

Convolutional networks

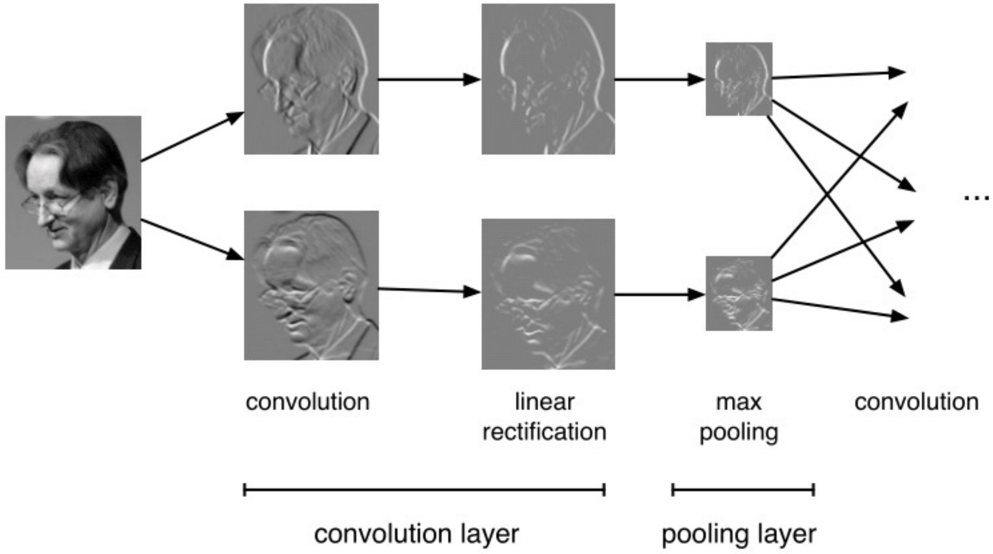
It's common to apply a linear rectification nonlinearity: $y_i = \max(z_i, 0)$



Why might we do this?

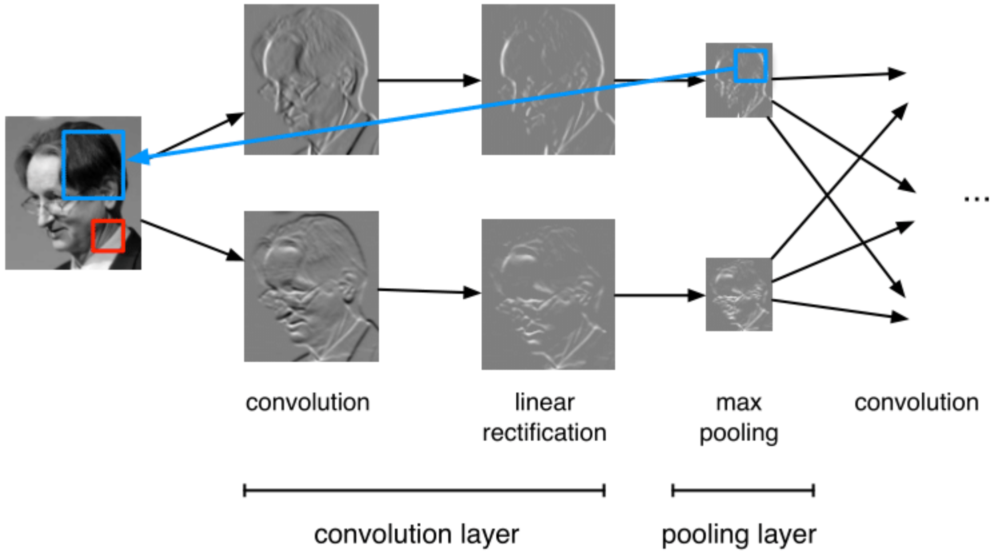
- Convolution is a linear operation. Therefore, we need a nonlinearity, otherwise 2 convolution layers would be no more powerful than 1.
- Two edges in opposite directions shouldn't cancel

Convolutional networks



Convolutional networks

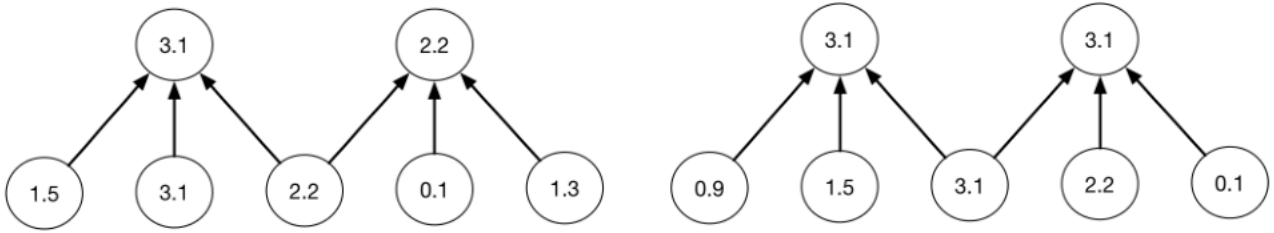
Because of pooling, higher-layer filters can cover a larger region of the input than equal-sized filters in the lower layers.



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.


3.1
2.2
1.5
3.1
2.2
0.1
1.3
3.1
3.1
0.9
1.5
3.1
2.2
0.1