Lecture 17
Convolutional Neural Networks

30 March 2016

Taylor B. Arnold
Yale Statistics
STAT 365/665

Yale

Notes:

- Problem set 6 is online and due **next** Friday, April 8th
- Problem sets 7,8, and 9 will be due on the remaining Fridays
- No class on April 11th or 13th

Convolutional Neural Networks

Convolution layers are a slightly more exotic variant on the dense linear layers we have been using so far. They simultaneously address several issues that are commonly seen in computer vision applications:

Convolutional Neural Networks

Convolution layers are a slightly more exotic variant on the dense linear layers we have been using so far. They simultaneously address several issues that are commonly seen in computer vision applications:

- We want to utilize the known geometry of the data (color channels and locality)

Convolutional Neural Networks

Convolution layers are a slightly more exotic variant on the dense linear layers we have been using so far. They simultaneously address several issues that are commonly seen in computer vision applications:

- We want to utilize the known geometry of the data (color channels and locality)
- Larger images require an enormous number of weights for even modestly sized hidden layers. Using 96x96 pixel images and a single hidden layer with 1024 nodes, requires just under ten million individual weights!

Convolutional Neural Networks

Convolution layers are a slightly more exotic variant on the dense linear layers we have been using so far. They simultaneously address several issues that are commonly seen in computer vision applications:

- We want to utilize the known geometry of the data (color channels and locality)
- Larger images require an enormous number of weights for even modestly sized hidden layers. Using 96x96 pixel images and a single hidden layer with 1024 nodes, requires just under ten million individual weights!
- Many applications require detecting or recognizing items that may be placed differently in the image field; it will be useful to have a method that is insensitive to small shifts in the individual pixels.

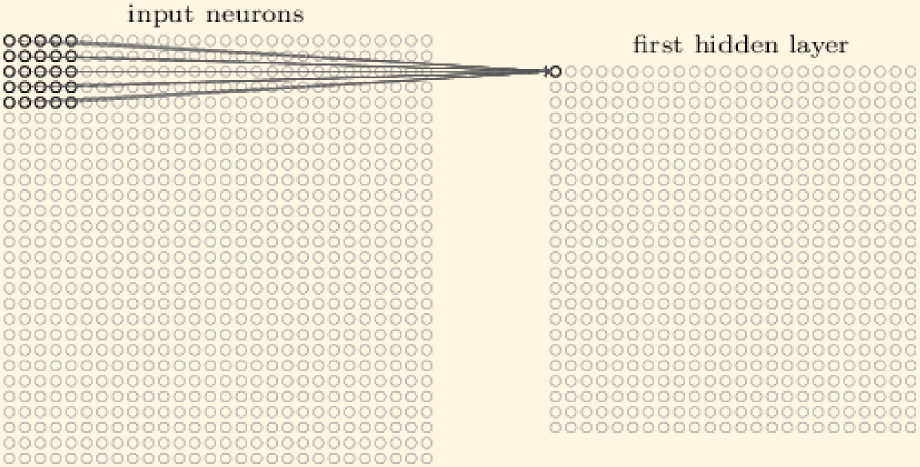
I'll restrict today's discussion to 2D-convolutions, as they are generally used in image processing, but note that they can also be applied to 1D, 3D and higher dimensions (we'll use 1D in the text analysis applications).

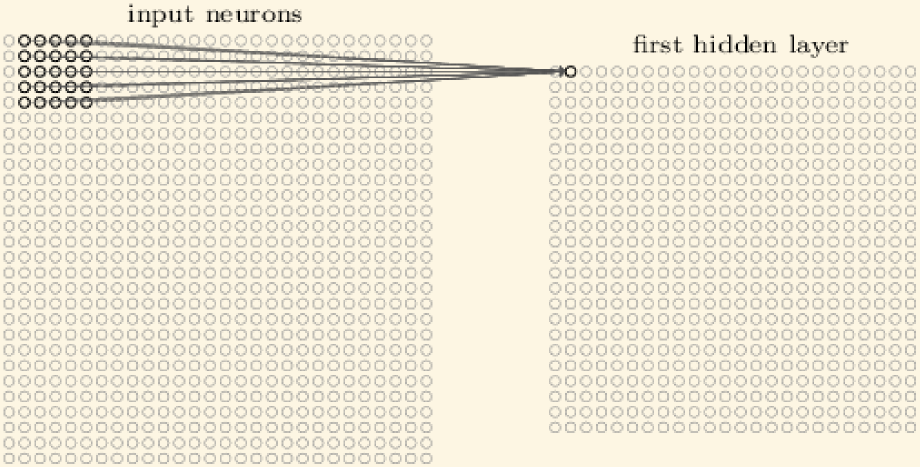
Convolutional Neural Networks, cont.

We can think of convolutional layers as being the same as a dense/linear layer, with two constraints applied to the weights and biases.

- Sparsity with local receptive field: the weight and biases for a given neuron are only non-zero over a small, local region of the input image.
- Shared weights: the non-zero weights and biases are the same across all of the neurons, up to a shifting over the image.

A picture should make this much more clear.


input neurons
first hidden layer


input neurons
first hidden layer

Filters

What we have just described is what we call a filter; a convolutional layer is made up of some predetermined number of filters. That is, we have multiple set of local weights that are applied over small chunks of the image. These allow us to capture different components that may all be useful for the classification task at hand.

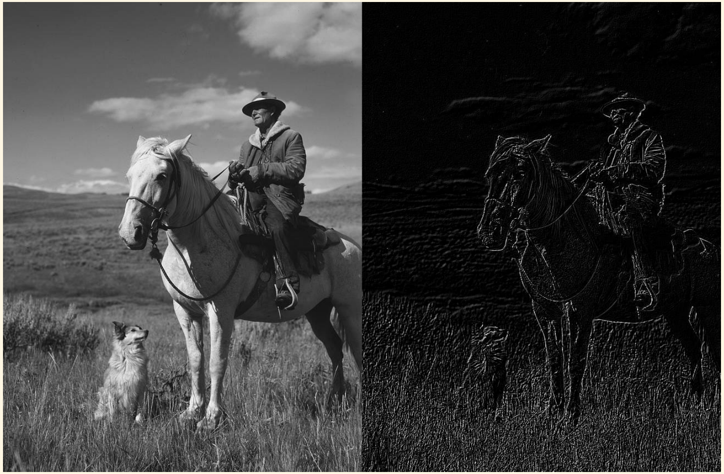
Filters: History

The idea of using a filter on images is not a new idea to neural networks; the difference here is the filters are adaptively learned from data.

For example, take the following fixed kernel weights:

$$\begin{pmatrix} 1 & 0 \\ 0 & -1 \end{pmatrix}$$

What happens when we apply this over an image?



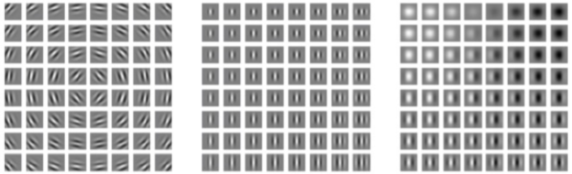
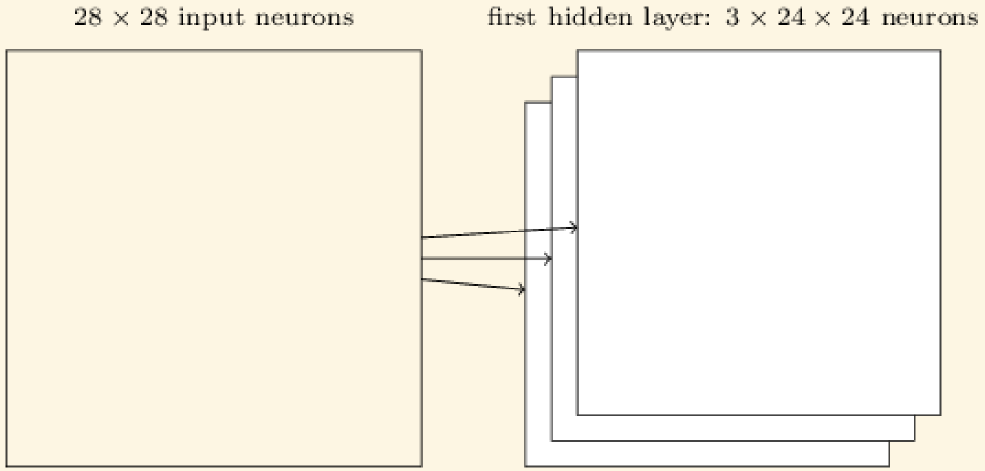


Figure 9.13: Gabor functions with a variety of parameter settings. White indicates large positive weight, black indicates large negative weight, and the background gray corresponds to zero weight. *Left)* Gabor functions with different values of the parameters that control the coordinate system: x_0, y_0, and τ. Each gabor function in this grid is assigned a value of x_0 and y_0 proportional to its position in its grid, and τ is chosen so that each Gabor is sensitive to the direction radiating out from the center of the grid. For the other two plots, x_0, y_0, and τ are fixed to zero. *Center)* Gabor functions with different Gaussian scale parameters $beta_x$ and β_y. Gabor functions are arranged in increasing width (decreasing β_x) as we move left to right through the grid, and increasing height (decreasing β_y) as we move top to bottom. For the other two plots, the β values are fixed to $1.5 \times$ the image width. *Right)* Gabor functions with different sinusoid parameters f and ϕ. As we move top to bottom, f increases, and as we move left to right, ϕ increases. For the other two plots, ϕ is fixed to 0 and f is fixed to $5\times$ the image width.

A convolutional layer with 3 filters:



Zero padding

You may have noticed that the grid of pixels of the output image will have fewer rows and columns than the input image. In some cases this is okay, but it is often advantageous to preserve the size of the grid (there are several reasons that we want to grid size to be divisible by 2^n for some reasonably large n). To do so, we can add extra rows and columns of zeros (zero padding) to the input before applying the

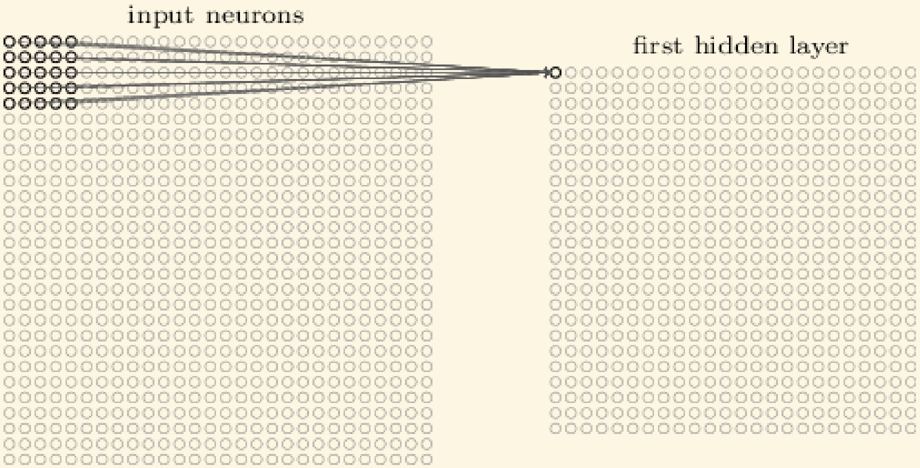
Kernel and stride

The architecture of a (2D) convolution layer is primarily determined by four numbers:

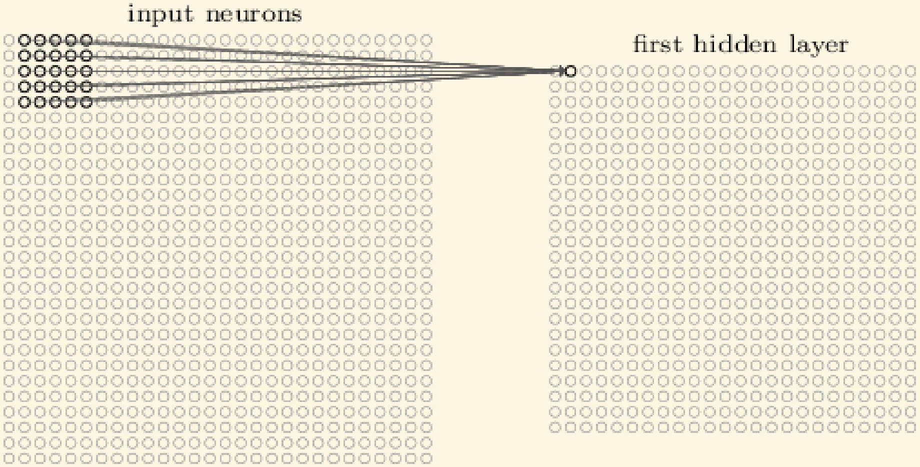
1. the number of filters, F
2. the height of the kernel, k_h
3. the width of the kernel, k_w
4. the stride, k_s

The stride tells us how far the local set of weights is shifted before being applied. We'll mostly use a stride of 1, but just note that other values are possible.

A 5x5 kernel:



A stride of 1:



A nice demo of applying convolution over a grid using alternative strides and zero padding:

`http://cs231n.github.io/assets/conv-demo/index.html`

Tensors

A convolution can also be described in purely algebraic terms. We are defining a map from a three dimensional space to a three dimensional space:

$$F^1 \times W^1 \times H^1 \rightarrow F^2 \times W^2 \times H^2$$

Where F is the number of filters, W the width of the image, and H the height of the image. You can now see why tensors are considered a generalization of a matrix operations, and why they are so important in learning neural network models.

Tensors

A convolution can also be described in purely algebraic terms. We are defining a map from a three dimensional space to a three dimensional space:

$$F^1 \times W^1 \times H^1 \to F^2 \times W^2 \times H^2$$

Where F is the number of filters, W the width of the image, and H the height of the image. You can now see why tensors are considered a generalization of a matrix operations, and why they are so important in learning neural network models.

The value F^1 is equal to one for the MNIST dataset, but for CIFAR-10 we would set it to 3 to deal with the 3 color channels.

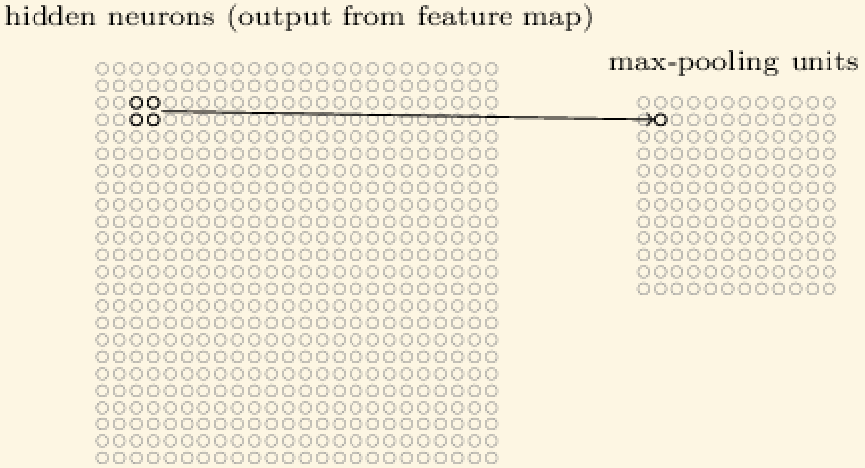
Pooling layers

Convolutional neural networks have another type of layer that can also be described by applying a function locally to small section of the image. These layers are called pooling layers; they differ from convolution layers because:

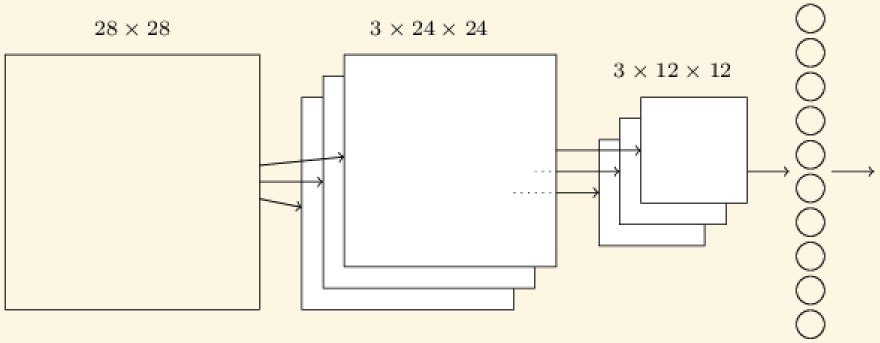
- they have no learned weights
- may not be linear functions of their inputs
- are applied separately to each kernel
- the stride is usually equal to the size of the kernel; that is, the regions are non-overlapping

The most common type of pooling (by far) is called max-pooling, with a 2x2 kernel and stride of 2. This halves the extent of each dimension (reduces the number of data points by a factor of 4 for 2D images) and can greatly decrease the learning time and improve over-fitting of the data.

Max pooling using a 2x2 filter and a stride of 2:



An example of convolution followed by max pooling:



Let's now apply convolutional networks to the MNIST dataset using the keras library.

A demo of applying convolutional neural networks to the MNIST dataset:

`http://cs.stanford.edu/people/karpathy/convnetjs/demo/mnist.html`