**Slide 7-9: Convolutional Neural Networks – First without the brain stuff**

**Slide 7-10: Convolution Layer**

* We first start off with an image 32 x 32 x3 CIFAR-10
* CNN operate over volumes
* All of these layers in between are going to take volumes of activations and they are going to produce volumes of activations
* Our intermediates will not just be vectors as they are with Neural Networks but they will have these spatial dimensions of width, height, and depth that we’ll maintain throughout the computation
* Here the depth is not the depth of a network
* The convolutional Layer is the core building block of a CNN

**Slide 7-11**

* The way a convolutional layer works is as follows

1. We receive some kind of input volume that goes into the layer. We have spatially small filters in the CNN
2. Suppose we have only one filter

**Slide 7-12**

* Since the input image has three channels, the depth of three remains the same
* What we do now is we take this filter and convolve it over this input volume
* Convolve: we slide the filter spatially though all spatial locations of this input volume and we are going to be computing dot products along the way.

**Slide 7- 13**

* We are going to learn these filters, the w’s
* As we are sliding the filter through the volume we are computing wTx + b
  + Where x is a small piece of the input volume
* At every single position in the input volume we are computing 5x5x3= 75 dot products + 1 for bias

**Slide 7-14**

* As we slide the filter spatially we end up carving out an activation map of activations of the responses of that filter at every spatial position
* Therefor sliding the 5x5 filter over the 32x32 input volume will give us a 28x28 activation map of how much this filter likes any spatial position in the input volume
* The 28 comes from the fact that in a 32x32 image we only have 28x28 distinct positions

**Slide 7-15**

* We’ll end up with an entire set of filters which will produce their own activation maps
* Suppose we have six filters (the # of filters is a hyper parameter)
* Sliding each of these six filters through the input volume (which is known as the convolution operation) produces a 28x28x6 activation map
* This 6-deep activation map will be fed to the next layer

**Slide 7-16 thru 7-18**

* ConvNet is a sequence of Convolutional Layers interspersed with activation functions

**Slide 7-19**

* Well end up with a feature hierarchy
* Suppose we have this Convolutional Network with all of these convolutional layers along the way
* When we look at the filters of the very first layer we find that through back prop the will tune themselves to be certain features
  + All of the filters will be looking for these features in the original image when we convolve through the input volume
* The second convolutional layer will be dot products over the outputs of the first conv layer and we will see that we may end up with filters that get excited about certain features
* This happened through each successive layer of the CNN

**Slide 7-20**

* We end up with an image which is very similar to what Hubel & Weisel may have imagined

**Slide 7-21**

* Lets consider the image of a piece of a car
* On the first convolutional layer we have trained 32 filters of 5x5 spatially
* These are example activation maps from what they look like in practice
* White corresponds to high activations and black to low activations
* All the activation maps produced by these filters will then be stacked up and fed to the next convolutional layer

**Slide 7-22**

* Layout of a convolutional network
* 3 core building blocks
  + Convolutional layer
  + RELU
  + Pooling
  + Fully Connected Layer
* Here we are visualizing that every column is a volume of representation along the way of a convolutional network
* Every row is an activation map
* There are 10 filters
* Through the network piece by piece we are creating higher levels of abstraction

**Slide 7-23 thru 7-28**

* Sliding 3x3 filter on a 7x7 image produces 5x5 output

**Slide 7-29**

* We can try it with a stride of 2
* Stride refers to how much we shift the filter at a time (hyper parameter)

**Slide 7-31**

* We get a 3x3 output with stride 2

**Slide 7-33**

* Stride 3 we’ll say it can’t fit because its not evenly divisible

**Slide 7-34**

* Output size = (N-F) /stride + 1
* Formula tells us exactly how many can fit with a specific stride

**Slide 7-35**

* It is very common to pad the input sometimes with zeros
* Padding will be a hyper parameter to the convolutional layer

**Slide 7-36**

* If we do this then our output size is the same as our input size
  + i.e. if we have in input image of 7x7 and we pad 1 pixel row and column then the output size is 7x7
* Padding make the flow of data much easier so we will therefore be using it a lot

**Slide 7-37**

* However if we have a filter of size FxF then the number of rows and columns we zero pad will be determined by
  + (F-1)/2

**Slide 7-38**

* If we don’t pad then we will see that the size of the image will shrink over time
* We end up with a very quick decrease in spatial size
* If we have hundreds of layers we don’t want to shrink the image.

**Slide 7-44 : Parameter Summary**

* Using the four hyper parameters K,F,S,and P we can computer the activation output
* Common settings:
  + K is usually chosen as powers of 2 for computational reasons
  + A 1x1 convolutional filter does make sense

**Slide 7-45**

* Suppose we have an input volume of 56x56x64.
* With a 1x1 convolution we are in fact doing a 64 dimensional dot product along one “fiber” of the input volume
* We are doing a dot product be we are not merging anything spatially

**Slide 7-49: Brain/Neuron View of CONV Layer**

* Let say we are sliding through this input image and at this particular position this filter is computing the dot product

**Slide 7-50**

* This is analogous to what we have seen before where these neurons computed wTx+b on their inputs
* We can interpret the output of the filter at this position as just a neuron that is fixed in space that happens to be looking at a small region in the input image and is computing wTx+b
* The neurons connections are just in this particular part of the image only
* Local connectivity only
* We would say that this neurons receptive field is 5x5

**Slide 7-51**

* Also, as we slide the filter through with these weights, we use the same weighs throughout
* For each activation map we envision a 28x28 grid and these neurons are only looking at their own 5x5 patch in the input volume but all of them share parameters because its one filter sliding and computing the outputs
* So all the neurons have the same weights “w”

**Slide 7-52**

* Of course we have several different filters
* We end up having a 3D volume of neurons arranged in a 3D spatial layout and all of them are looking at the input image in a local pattern and sharing parameters across space
* But across depth they are all different neurons
* Every filter will have different weights but the weights will be the same for all neurons in the same filter

**Slide 7-53**

* We have covered the convolutional layers and the RELU layers
* What the Pooling layer do is reduce the spatial size of the image

**Slide 7-54: Pooling**

* They take your input volume and squish it down spatially
* This is the down sample operation
* This happens on every single activation map independently

**Slide 7-55: Max Pooling**

* Mathematically the most common form of down sampling is Max Pooling
* Take the max of each 2x2 piece
* Another method is to take the average of each 2x2 piece instead of the max but this does not work as well.

**Slide 7-56:**

* Instead of 4 parameters as with the convolutional layer, here we are only taking 2
  + Filter size F
  + Stride S

**Slide 7-57**

* Not that many possibilities
* Output depth is preserved D2 = D1

**Slide 7-58: Fully Connected Layer**

* Each pooling layer downsamples
  + 32 to 16
  + 16 to 8
  + 8 to 4
* By the end at the Fully Connected Layer we have a volume that is 4x4x10

**Slide 7-60: Case Studies for Convolutional Networks**

* LeNet-5 1998
  + 32x32 image
  + 6 kernels 5x5
  + Subsampling = max pooling
  + CONV-POOL-CONV-POOL-CONV-FC

**Slide 7-61 AlexNet 2012**

* 227x277x3 images
* 2 separate streams since at the time of development GPUs were not that good and Alex needed to use two GPUs

**Slide 7-68**

* Normalization is not used anymore since there are no improvements in using it

**Slide 7-70: ZFNet 2013**

* Built on AlexNet but for CONV1 originally there was an 11x11 stride 4 convolution but they found that it was too drastic and that too much and producing artifacts in the first layer
* They instead recommended 7x7 filters of stride 2
* CONV3,4,5 they used larger filters
  + Increasing these gives better performance

**Slide 7-71: VGGNet 2014**

* They found that there were way too many parameters in how you change the filters
* They therefore committed to
  + CONV 3x3 stride 1 pad 1
  + MAX POOL 2x2 stride 2

Slide