DEEP LEARNING READING GROUP

MARCH 24, 2016

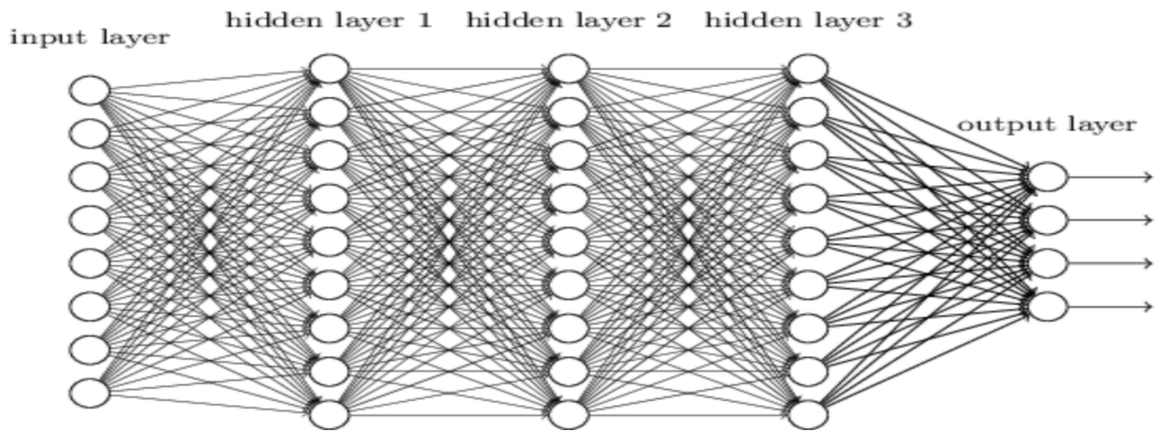
DATA SCIENCE PRACTICE

ISABEL RESTREPO, PAUL STEY

CONVOLUTIONAL NEURAL NETWORKS

Material and images from http://cs231n.github.io/convolutional-networks/

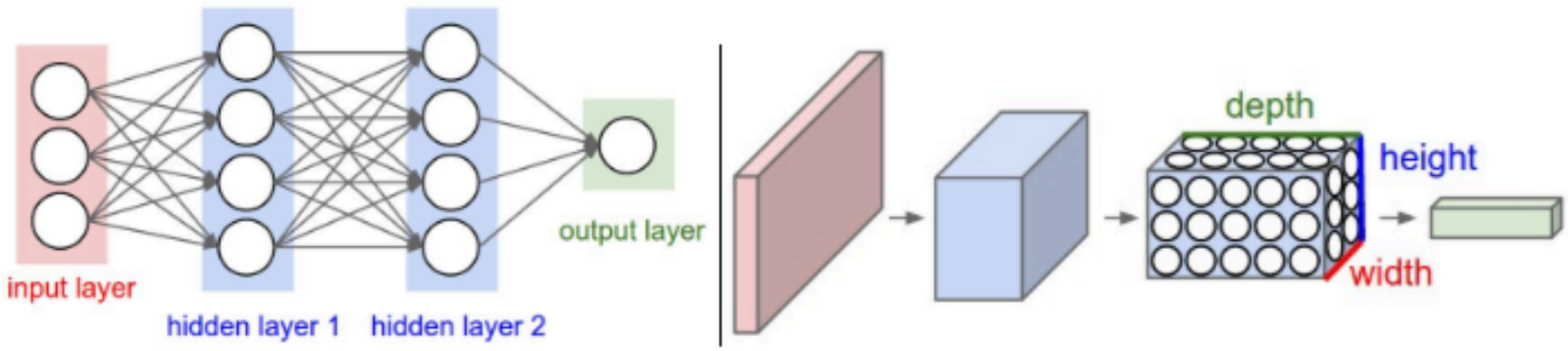
WHY DO WE NEED ConvNets?



- Regular Neural Nets don't scale well to images. For instance,
  - For images of size 32x32x3 - a single fully-connected neuron in a first hidden layer would have 32*32*3 = 3072 weights.
  - For example, an image of size 200x200x3, leads to = 200*200*3 = 120,000 weights.
- Full connectivity is wasteful and the huge number of parameters would quickly lead to overfitting.

WHAT ARE ConvNets?

- Very similar to neural-networks discussed thus far. Dot product, followed by non-linearity, and loss function at the end.
- Explicit assumption that input are images.
- Layers have neurons arranged in 3 dimensions (width, height, depth) to form an **activation volume**

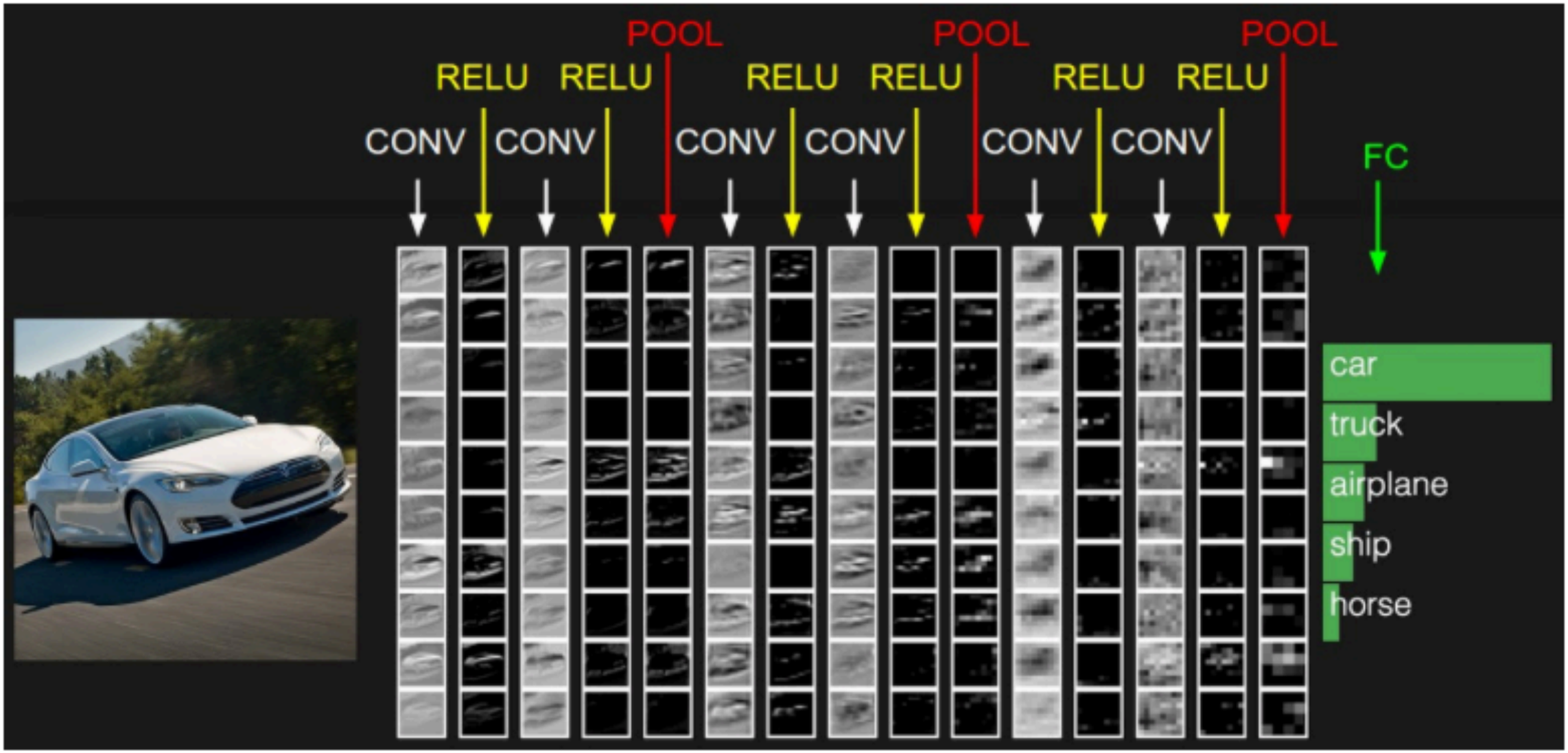


Left: A regular 3-layer Neural Network. Right: A ConvNet arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of neuron activations. In this example, the red input layer holds the image, so its width and height would be the dimensions of the image, and the depth would be 3 (Red, Green, Blue channels).

TYPES OF LAYERS USED TO BUILD ConvNets

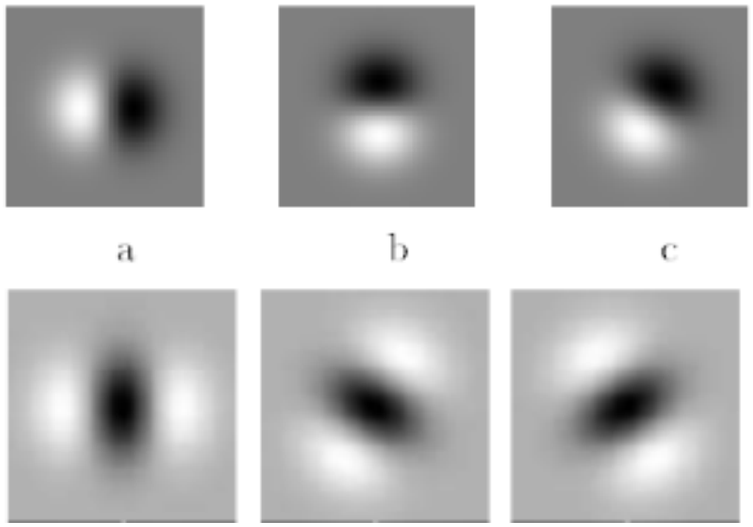
- Convolutional Layer
  - Input: 3-d volume.
  - Output: 3-d volume.
  - Convolve ‘filters’ with small regions in the image
  - Output depth, depends on the number of filters
- Pooling Layer
  - Downsampling along spatial dimensions (width, height)
- Fully-Connected Layer (what we’ve see so far)
  - Compute class score. Dimensions are transformed to 1x1x NUM_CLASSES

ARCHITECTURE

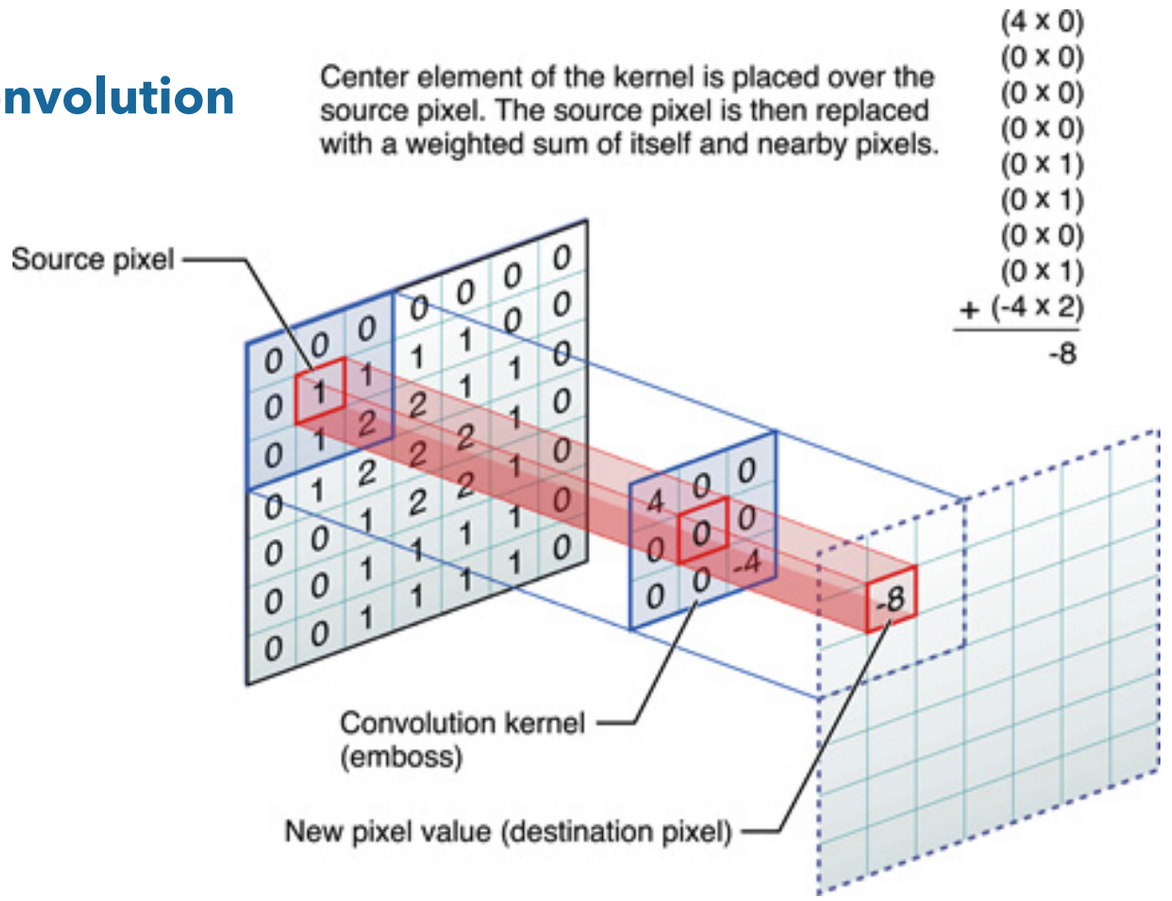


- CONV/FC layers perform transformations that are a function of not only the activations in the input volume, but also of the parameters (the weights and biases of the neurons).
- RELU/POOL layers will implement a fixed function. (E,g, RELU = max (0, x)
- RELU doesn’t change the dimensions of the input

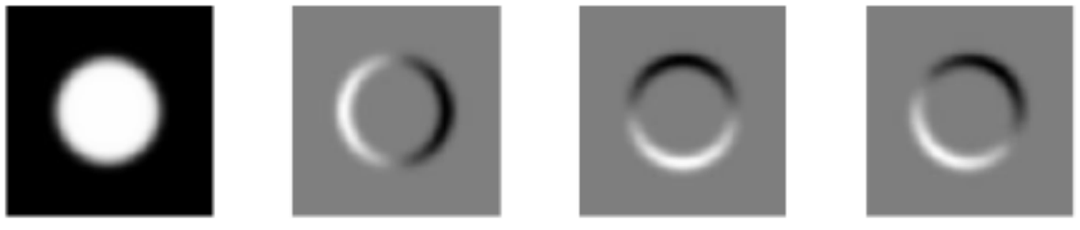
1. Filter Design. E.g., Steerable filters - Gaussian and it's derivatives



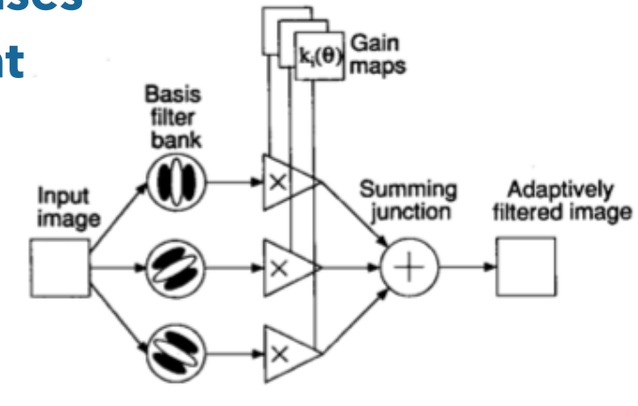
2. Convolution

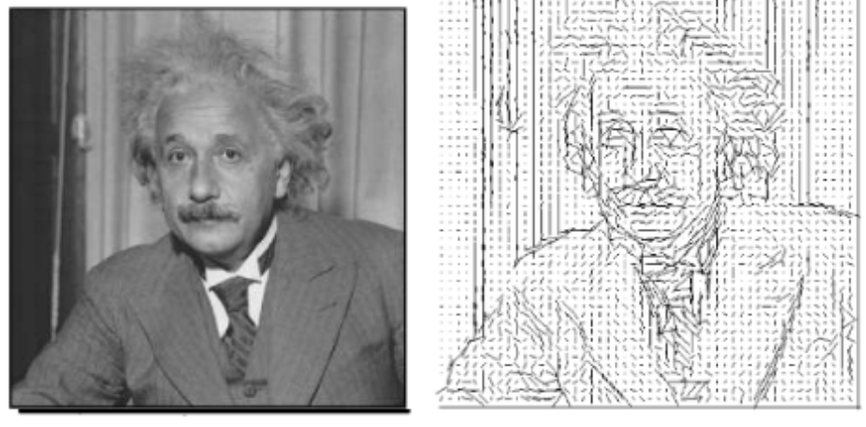


3. Individual responses (apply a, b, c)



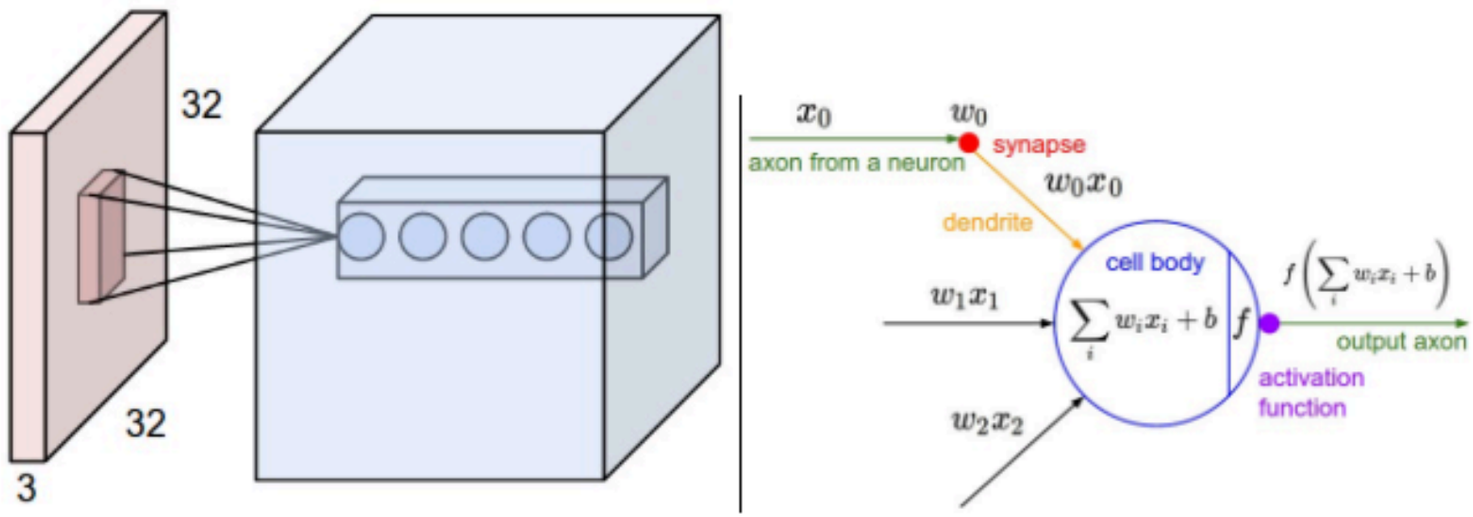
4. Compose responses (optional) - gradient orientation





CONNECTING EACH NEURON TO THE INPUT VOLUME

- **Receptive field (hyper-parameter):** Spatial extend of each neuron = filter size
- Connections are local in space (along width and height), but always full along the entire depth of the input volume.
  - Example: Suppose an input volume had size [16x16x20]. Then using an example receptive field size of 3x3, every neuron in the Conv Layer would now have a total of 3*3*20 = 180 connections to the input volume. Notice that, again, the connectivity is local in space (e.g. 3x3), but full along the input depth (20)

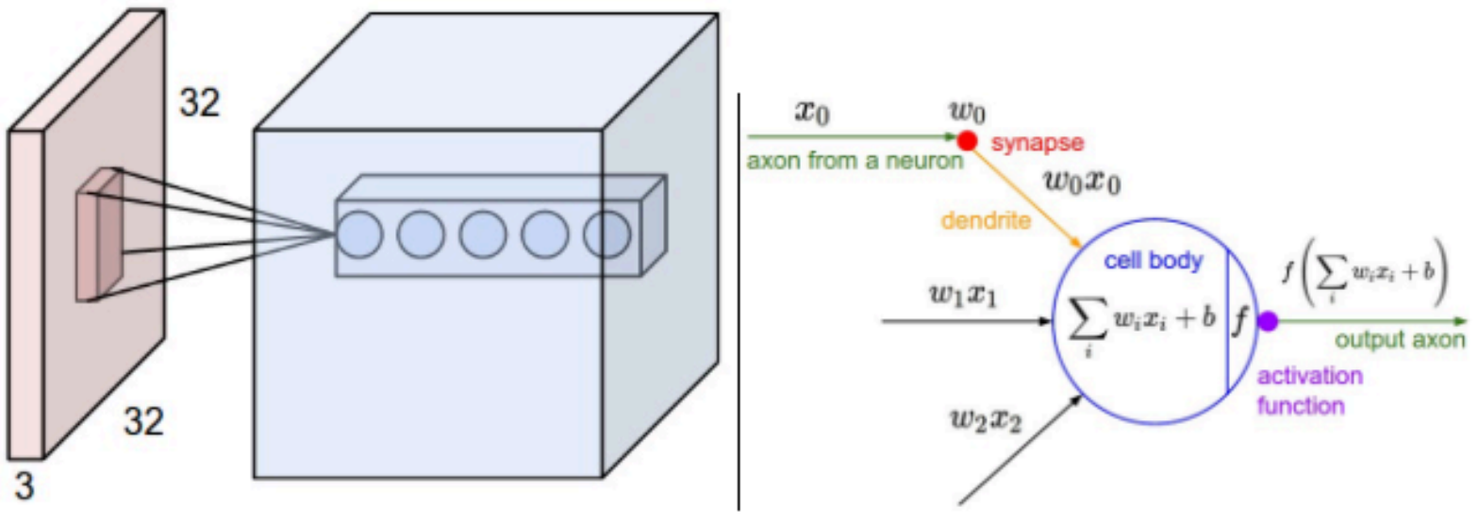


NUMBER OF NEURONS IN THE OUTPUT VOLUME

- **Depth (hyper-parameter)**: Number of filters. Number of patterns to look for in the input… e.g. think back to steerable filters. The set of neurons that are looking at the same region may be referred as **depth column** or **fiber**

- **Stride (hyper-parameter):** Number of pixels used used when sliding the filter. This controls overlap between neurons. If stride is 1, we move one pixel at a time (common but depends on width of filter)

- **Padding (hyper-parameter):** Number of zeros to add to the border of the volume. Convenient for convolution



Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	0	1	1	2	0	0
0	2	2	1	0	0	0
0	2	1	1	1	1	0
0	1	0	0	0	0	0
0	1	1	1	0	0	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	1	0	1	1	0	0
0	2	1	0	2	2	0
0	0	0	2	1	0	0
0	0	1	2	0	1	0
0	2	2	0	1	1	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	0	0	1	1	2	0
0	1	0	2	2	0	0
0	2	2	0	0	1	0
0	2	1	2	0	1	0
0	2	0	0	2	2	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

1	1	0
-1	0	1
0	1	-1

w0[:,:,1]

-1	1	-1
1	1	-1
0	0	1

w0[:,:,2]

1	1	1
1	1	1
1	0	1

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

0	0	-1
-1	1	1
1	0	0

w1[:,:,1]

0	-1	0
-1	-1	0
-1	0	0

w1[:,:,2]

1	0	-1
0	0	1
0	0	0

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

o[:,:,0]

4	9	5
13	9	4
7	7	9

o[:,:,1]

0	3	-5
1	-4	-1
-1	-1	-3

toggle movement

Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	0	1	1	2	0	0
0	2	2	1	0	0	0
0	2	1	1	1	1	0
0	1	0	0	0	0	0
0	1	1	1	0	0	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	1	0	1	1	0	0
0	2	1	0	2	2	0
0	0	0	2	1	0	0
0	0	1	2	0	1	0
0	2	2	0	1	1	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	0	0	1	1	2	0
0	1	0	2	2	0	0
0	2	2	0	0	1	0
0	2	1	2	0	1	0
0	2	0	0	2	2	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

1	1	0
-1	0	1
0	1	-1

w0[:,:,1]

-1	1	-1
1	1	-1
0	0	1

w0[:,:,2]

1	1	1
1	1	1
1	0	1

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

0	0	-1
-1	1	1
1	0	0

w1[:,:,1]

0	-1	0
-1	-1	0
-1	0	0

w1[:,:,2]

1	0	-1
0	0	1
0	0	0

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

o[:,:,0]

4	9	5
13	9	4
7	7	9

o[:,:,1]

0	3	-5
1	-4	-1
-1	-1	-3

toggle movement

Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	0	1	1	2	0	0
0	2	2	1	0	0	0
0	2	1	1	1	1	0
0	1	0	0	0	0	0
0	1	1	1	0	0	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	1	0	1	1	0	0
0	2	1	0	2	2	0
0	0	0	2	1	0	0
0	0	1	2	0	1	0
0	2	2	0	1	1	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	0	0	1	1	2	0
0	1	0	2	2	0	0
0	2	2	0	0	1	0
0	2	1	2	0	1	0
0	2	0	0	2	2	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

1	1	0
-1	0	1
0	1	-1

w0[:,:,1]

-1	1	-1
1	1	-1
0	0	1

w0[:,:,2]

1	1	1
1	1	1
1	0	1

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

0	0	-1
-1	1	1
1	0	0

w1[:,:,1]

0	-1	0
-1	-1	0
-1	0	0

w1[:,:,2]

1	0	-1
0	0	1
0	0	0

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

o[:,:,0]

4	9	5
13	9	4
7	7	9

o[:,:,1]

0	3	-5
1	-4	-1
-1	-1	-3

toggle movement

Input Volume (+pad 1) (7x7x3)

x[:,:,0]

0	0	0	0	0	0	0
0	0	1	1	2	0	0
0	2	2	1	0	0	0
0	2	1	1	1	1	0
0	1	0	0	0	0	0
0	1	1	1	0	0	0
0	0	0	0	0	0	0

x[:,:,1]

0	0	0	0	0	0	0
0	1	0	1	1	0	0
0	2	1	0	2	2	0
0	0	0	2	1	0	0
0	0	1	2	0	1	0
0	2	2	0	1	1	0
0	0	0	0	0	0	0

x[:,:,2]

0	0	0	0	0	0	0
0	0	0	1	1	2	0
0	1	0	2	2	0	0
0	2	2	0	0	1	0
0	2	1	2	0	1	0
0	2	0	0	2	2	0
0	0	0	0	0	0	0

Filter W0 (3x3x3)

w0[:,:,0]

1	1	0
-1	0	1
0	1	-1

w0[:,:,1]

-1	1	-1
1	1	-1
0	0	1

w0[:,:,2]

1	1	1
1	1	1
1	0	1

Bias b0 (1x1x1)

b0[:,:,0]

1

Filter W1 (3x3x3)

w1[:,:,0]

0	0	-1
-1	1	1
1	0	0

w1[:,:,1]

0	-1	0
-1	-1	0
-1	0	0

w1[:,:,2]

1	0	-1
0	0	1
0	0	0

Bias b1 (1x1x1)

b1[:,:,0]

0

Output Volume (3x3x2)

o[:,:,0]

4	9	5
13	9	4
7	7	9

o[:,:,1]

0	3	-5
1	-4	-1
-1	-1	-3

toggle movement

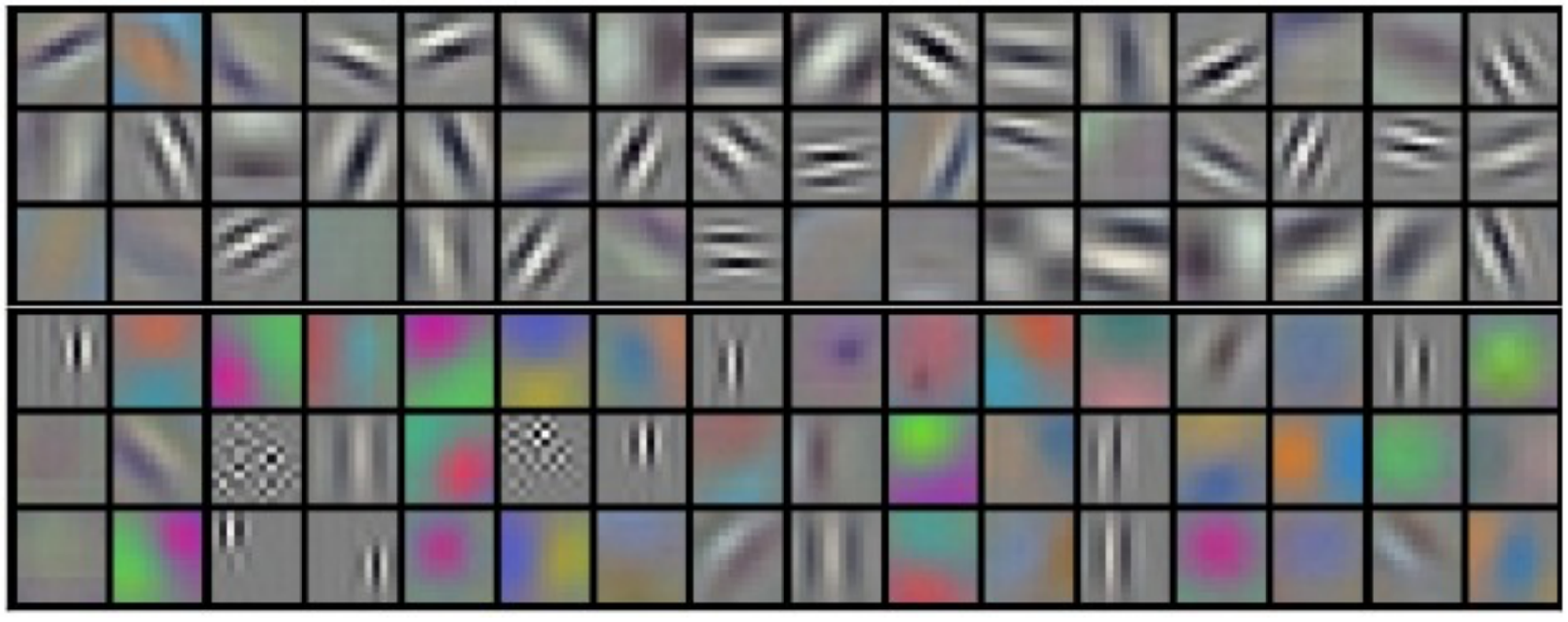
MUTUAL CONSTRAINS OF HYPER-PARAMETERS

- If stride is one, S=1, it's common to set padding P=(F-1)/2 to ensure that input and output volumes have the same size
- The number of neurons that fit is (W - F + 2P)/S +1. Must set hyper-parameters so there number of neurons is an integer!
- **Real-world example:** (Winner of ImageNet 2012) Images [227x227x3], F=11, S=4, P=0, K=96 .
  - Number of neurons (first layer)
    - (227 - 11)/4 + 1 = 55 (per width)
    - 55*55* 96 = 290,400 (width* height*depth)
    - Each neuron has 11*11*3 weights and 1 bias
    - 290400 * 364 = 105,705,600 parameters in the first layer!

PARAMETER SHARING

If one feature (filter/kernel/depth slice) is useful at pixel (x1, y1), then it is also useful at pixel (x2,y2)

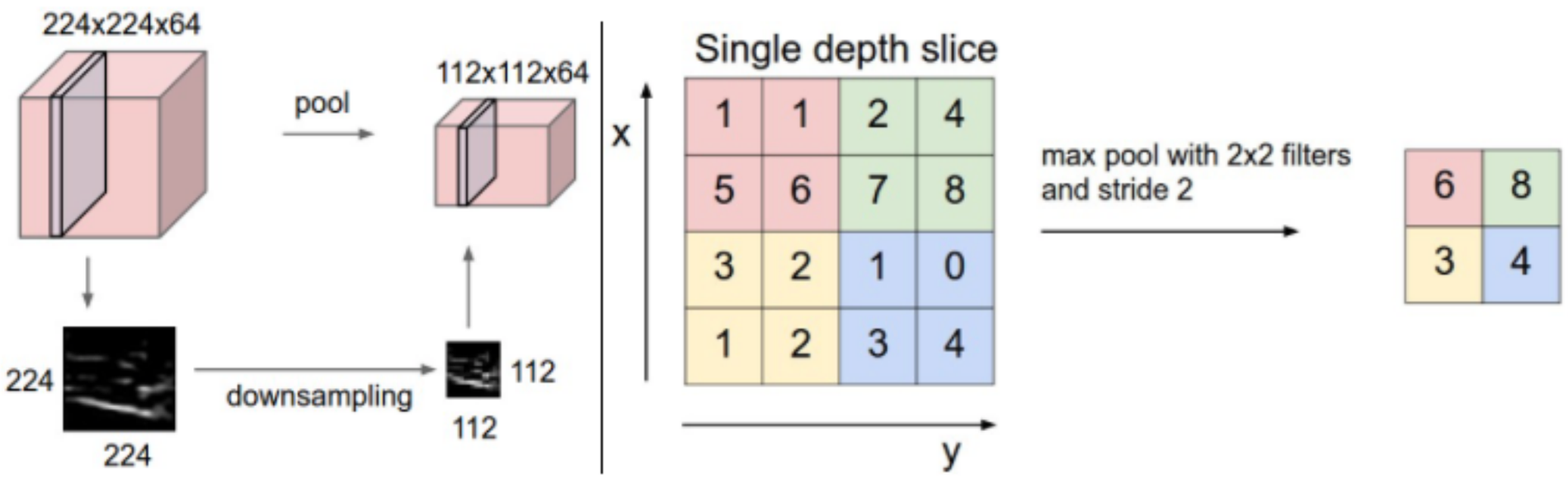
- All neurons in the same depth slice have the same weights and bias.
- Example before 96*11*11*3 = 34,848 unique weights + 96 biases = 34,944 parameters
- Then forward pass can be seen as the convolution of that slice neuron's weights with the input volume.
- Weights = filter/kernel that is convolved with the input



Example filters learned by Krizhevsky et al. Each of the 96 filters shown here is of size [11x11x3],

POOLING LAYER

- Inserted in-between Convolutional layers
- Goal: To reduce spatial size, amount of parameters, computation requirements and overfitting
- Operates independently on every depth slice and resizes spatially using MAX (could also be average, L2)
- Introduces no parameters, since it is a fixed operation
- Hyper-parameters : F (spatial size) and S (stride)



FUTURE – HOPE OF GETTING RID OF POOLING

- Striving for Simplicity: Newer works propose discarding pooling. Suggest using larger strides
- Variational auto encoders
- Generative Adversarial Networks

NORMALIZATION LAYER

- Have shown no real advantages

HYPER-PARAMETERS AND LAYER SIZING

- Input layer should be divisible by 2 many times
- Conv Layer use small filters (3x3), (5x5) (notice odd) and stride S=1 (reduce sizing headaches - downsampling done by pooling)
- Use padding that doesn't alter dimension
- Pool layers (2x2) F=2 and S = 2. Discards 75% of activations
- Compromise based on memory constraints (GPU). May have to sacrifice at the 1st layer.

POPULAR NETWORKS

- LeNet: LeCun 90's - read zip code (first successful ConvNet)
- AlexNet: Alex Krizhevsky, Ilya Sutskever and Geoff Hinton. Popularize ConNets for computer vision. ImageNet. Deeper, bigger than LeNet
- ZF Net: Matthew Zeiler and Rob Fergus. ILSVRC 2013 winner. Improved by expanding the size of the middle convolutional layers and making the stride and filter size on the first layer smaller.
- GoogLeNet: Szegedy et al. from Google. ILSVRC 2014 winner. Reduced the number of parameters in the network form 60M to 4M (inception module)
- ResNet. Residual Network. Kaiming He et al. ILSVRC 2015 winner. It features special skip connections and a heavy use of batch normalization. State of the art as of 2016. Torch Blog Post with Facebook's Github repo

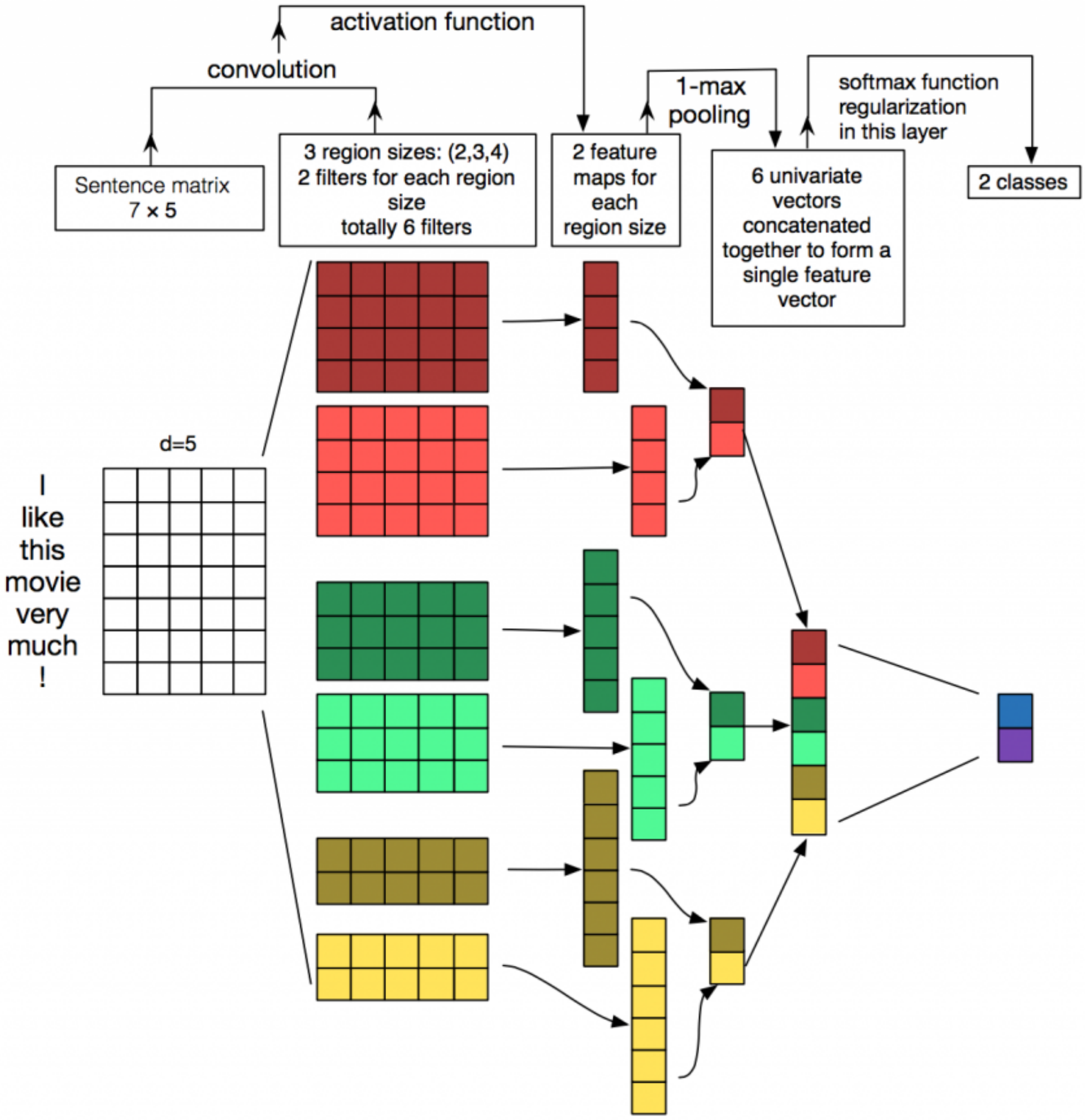
RESOURCES

- 9 Top CNN papers
- Web Demo
- Benchmarks

CONV NETS FOR NLP

Material from here

- Instead of image, inputs are sentences represented as matrix
- Each row in the matrix is a vector (low-dimensional) representation of a word using word2vec or Glove or a one-hot vector that index the word into a vocabulary. For a 10 word sentence using a 100-dimensional embedding, the input is a 10 x 100 matrix. That's the NLP image
- Filters slide over full rows (width = 100 in example above), height typically 2-5


activation function
convolution
1-max pooling
softmax function regularization in this layer
Sentence matrix 7 × 5
3 region sizes: (2,3,4) 2 filters for each region size totally 6 filters
2 feature maps for each region size
6 univariate vectors concatenated together to form a single feature vector
2 classes
d=5
I
like
this
movie
very
much
!

DO THEY MAKE SENSE TO USE FOR NLP?

- CNN success in computer vision are attributed to Local Invariance and Compositionally. But this arguments are not true for sentences to the same extent.
  - My green house
  - I own a house that is green
  - Position of words may change - but meaning may not. But if we have an eye on top of another instead of to the side, we may be talking about a different creature.
- **ALL models are wrong, but some are useful.** So is Bag of Words
- CNNs are fast and efficient representations (compared to n-grams)
- With a large vocabulary, computing anything more than 3-grams can quickly become expensive. Convolutional Filters learn good representations automatically, without representing the whole vocabulary. Can have filters of size larger than 5. Many of the learned filters in the first layer are capturing features quite similar (but not limited) to n-grams, but represent them in a more compact way.

USES

- Sentiment Analysis
- Topic Categorization
  - E.g, hashtags for Facebook posts, sentiment of consumer reviews, twitter posts
- Some current works also are looking at applying CNNs directly to characters. (Speech tagging). Work well on large data sets - not so well an smaller one. Require deep networks.

RESOURCES

- Example with TensorFlow