CS 360: Machine Learning

Sara Mathieson, Sorelle Friedler

Spring 2024

HAVERFORD COLLEGE

Admin

- **Lab 7** check in for Lab on Tues
  - Should be finished with the fully connected network
- **Project proposal** due April 8 (short)
- **Lab 8** (last lab) due Thurs April 18
- No Sorelle office hours today
- Sara office hours: **Mon 4-5pm in H110**

Outline for March 28

- Image data format and intro to Lab 7

- Layer-by-layer pretraining

- Convolutional neural networks (CNNs)

Outline for March 28

- Image data format and intro to Lab 7
- Layer-by-layer pretraining
- Convolutional neural networks (CNNs)

Lab 7 data pre-processing

- It is helpful to have our data be zero-centered, so we will subtract off the mean
- It is also helpful to have the features be on the same scale, so we will divide by the standard deviation
- We will compute the mean and std with respect to the *training data*, then apply the same transformation to all datasets

Lab 7 data pre-processing

- Input is now itself a multi-dimensional array
  - Also known as a **tensor**!

- For images, often the shape of each image will be (width, height, 3) for RGB channels

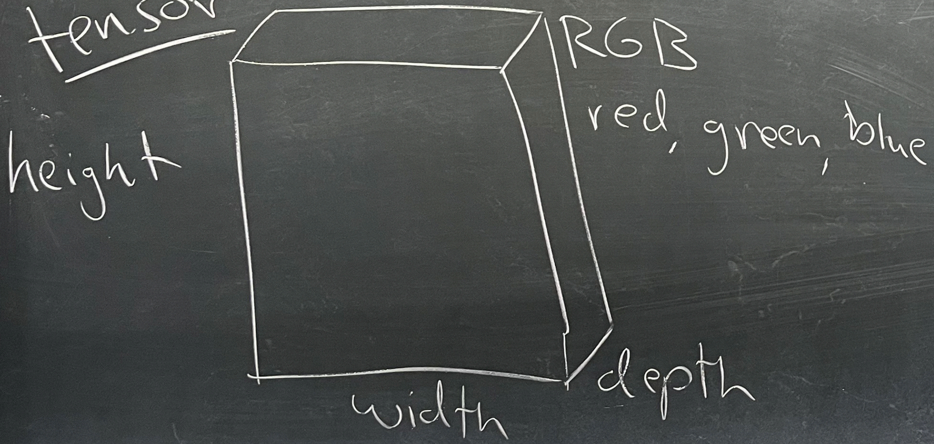
- Need to "**flatten**" or "unravel" for fully connected networks

$$x.\text{shape} = (32, 32, 3)$$

$$X.\text{shape} = (\underbrace{64}_{n}, \underbrace{32, 32, 3}_{p})$$

$$p = 32 \cdot 32 \cdot 3 = \boxed{3072}$$

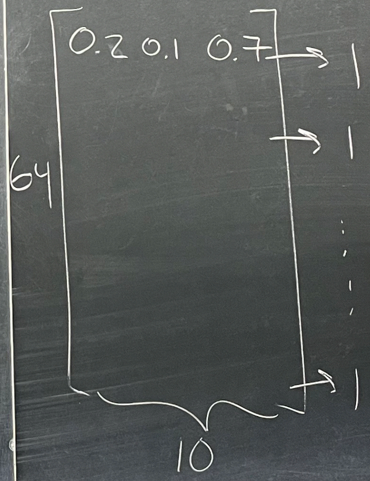
tensor



10 classes

output

(64, 10)



$$J(W^{(1)}, W^{(2)})$$

$$= \sum_{i=1}^{n} \| \vec{x}_i - \vec{x}_i^{\,*} \|^2$$

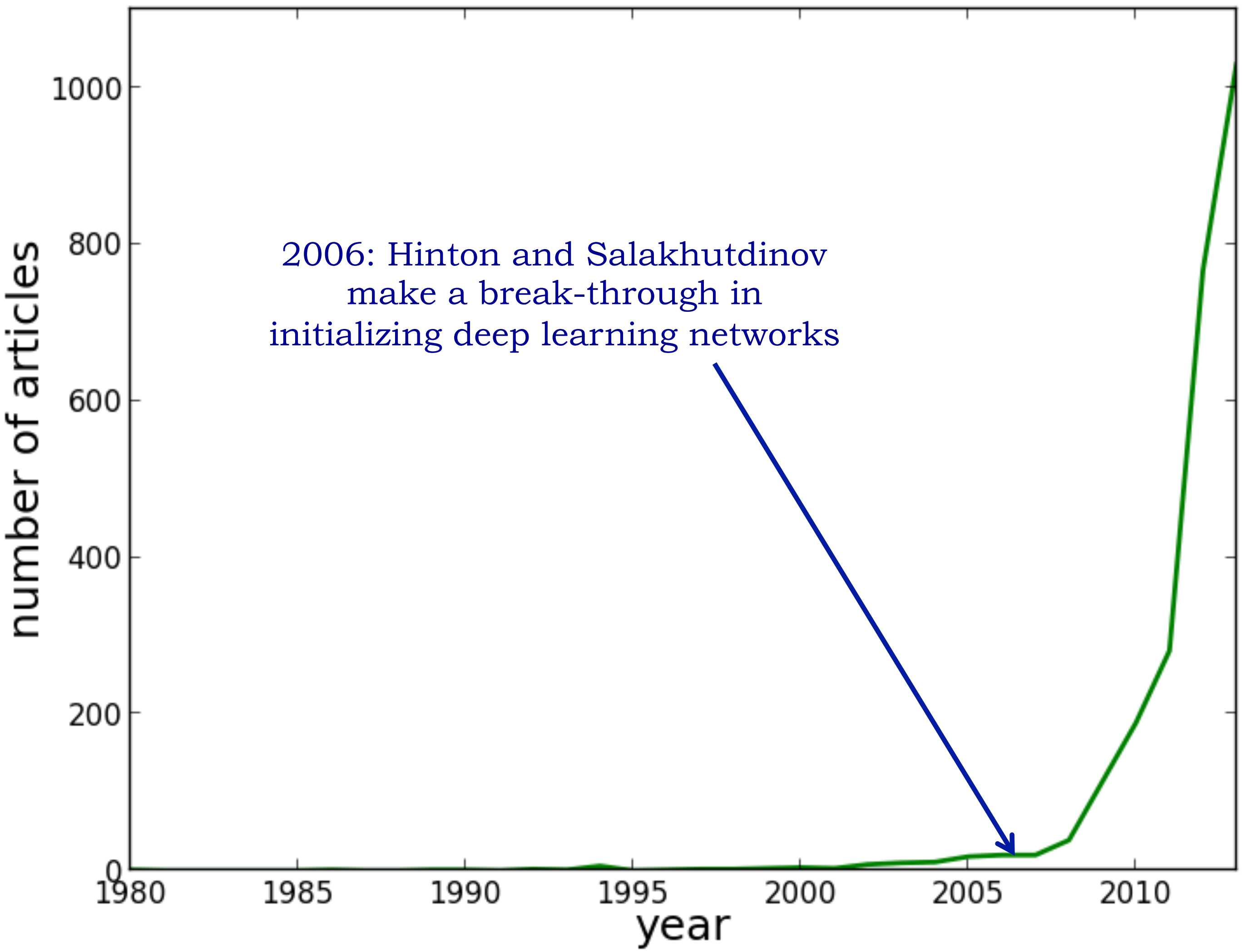
Orig

reconstructed

Outline for March 28

- Image data format and intro to Lab 7
- Layer-by-layer pretraining
- Convolutional neural networks (CNNs)

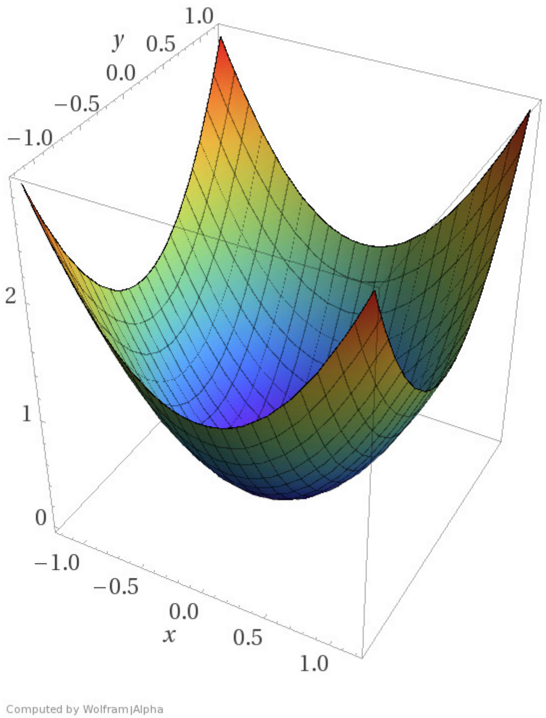
What was this breakthrough in deep learning?



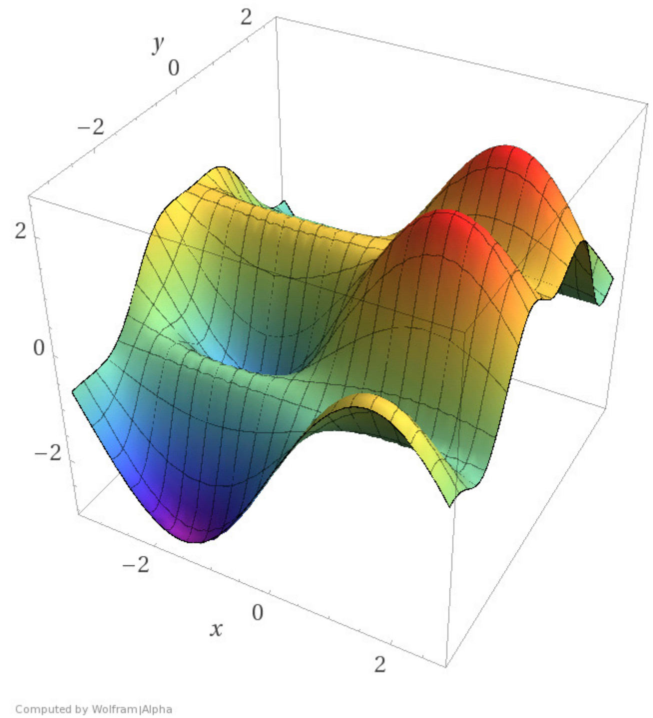
Images on next slides from
Hinton & Salakhutdinov (2006)

Fully connected networks have many parameters!

- As the number of parameters grows, a non-convex function often has more and more local minima
- Starting at a "good" point is crucial!



Convex



Non-convex

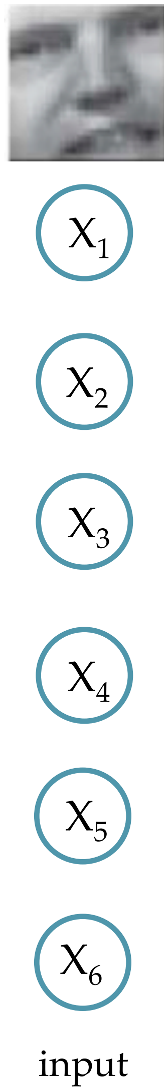
Image: O'Reilly Media

Unsupervised pre-training

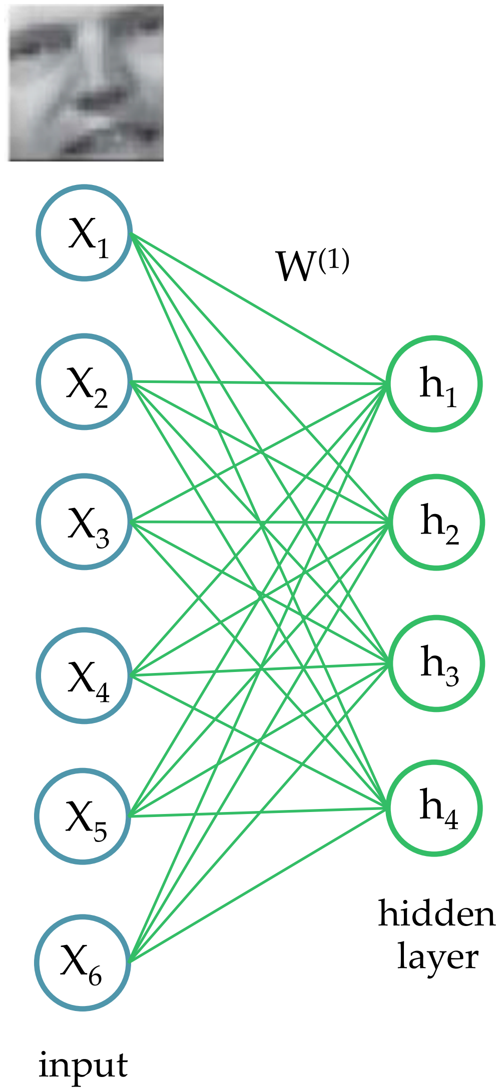
- Unsupervised pre-training uses latent structure in the data as a starting point for weight initialization
- After this process, the network is “fine-tuned”
- In practice this has been found to increase accuracy on specific tasks (which could be specified after feature learning)

Autoencoders

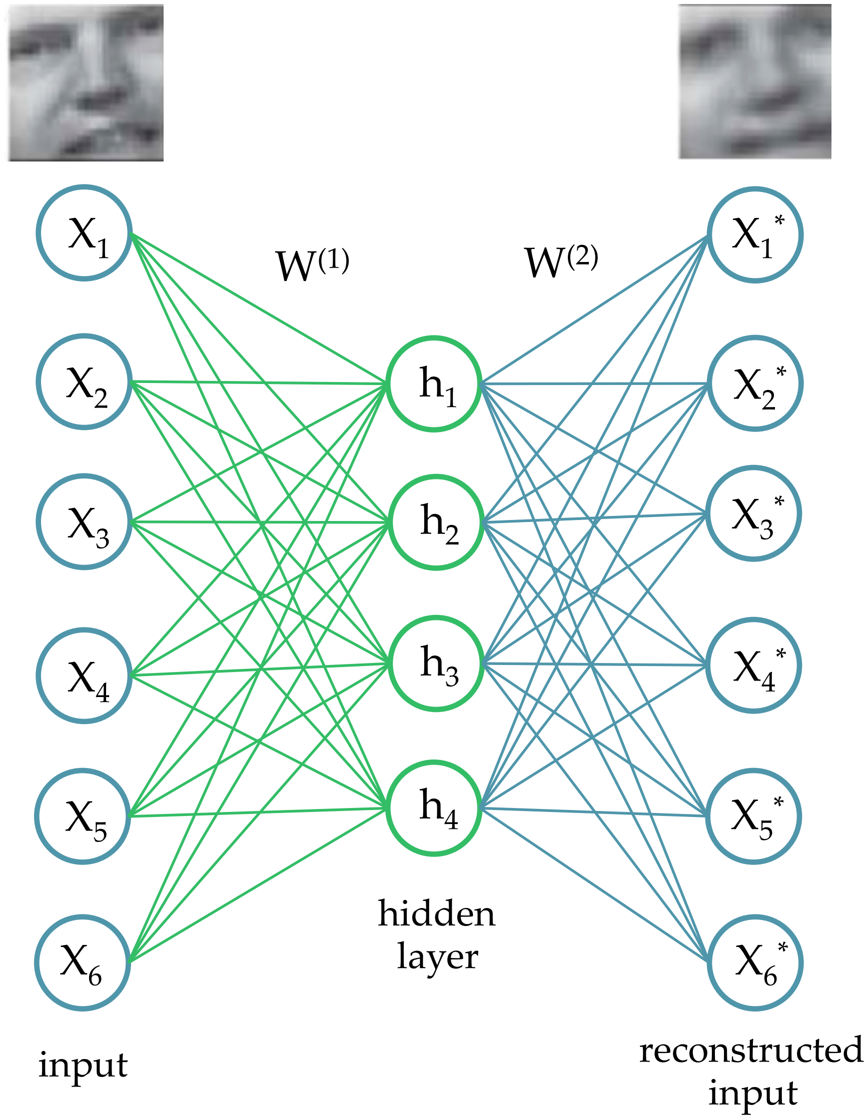
Detour to autoencoders



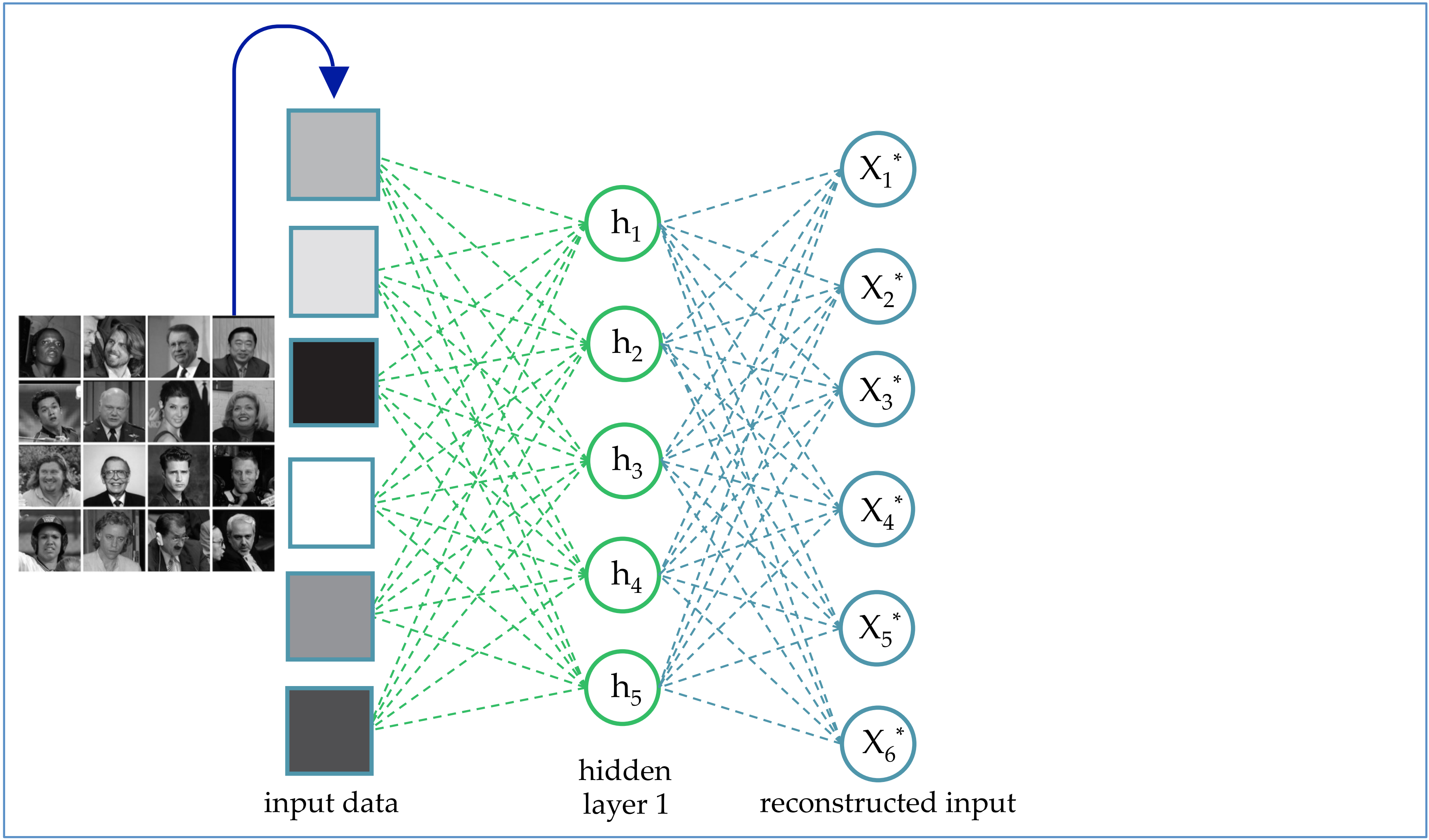
Detour to autoencoders



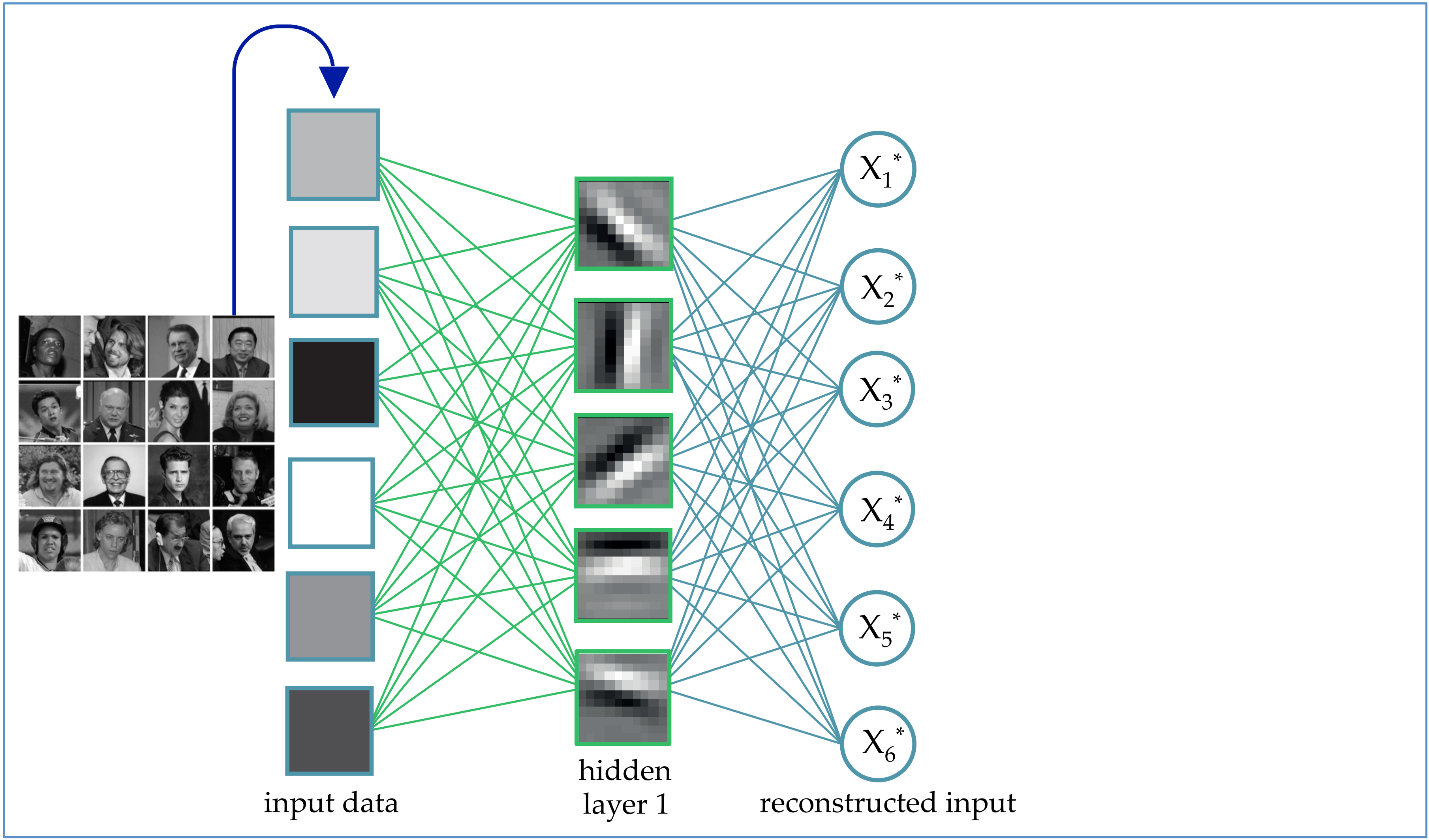
Detour to autoencoders



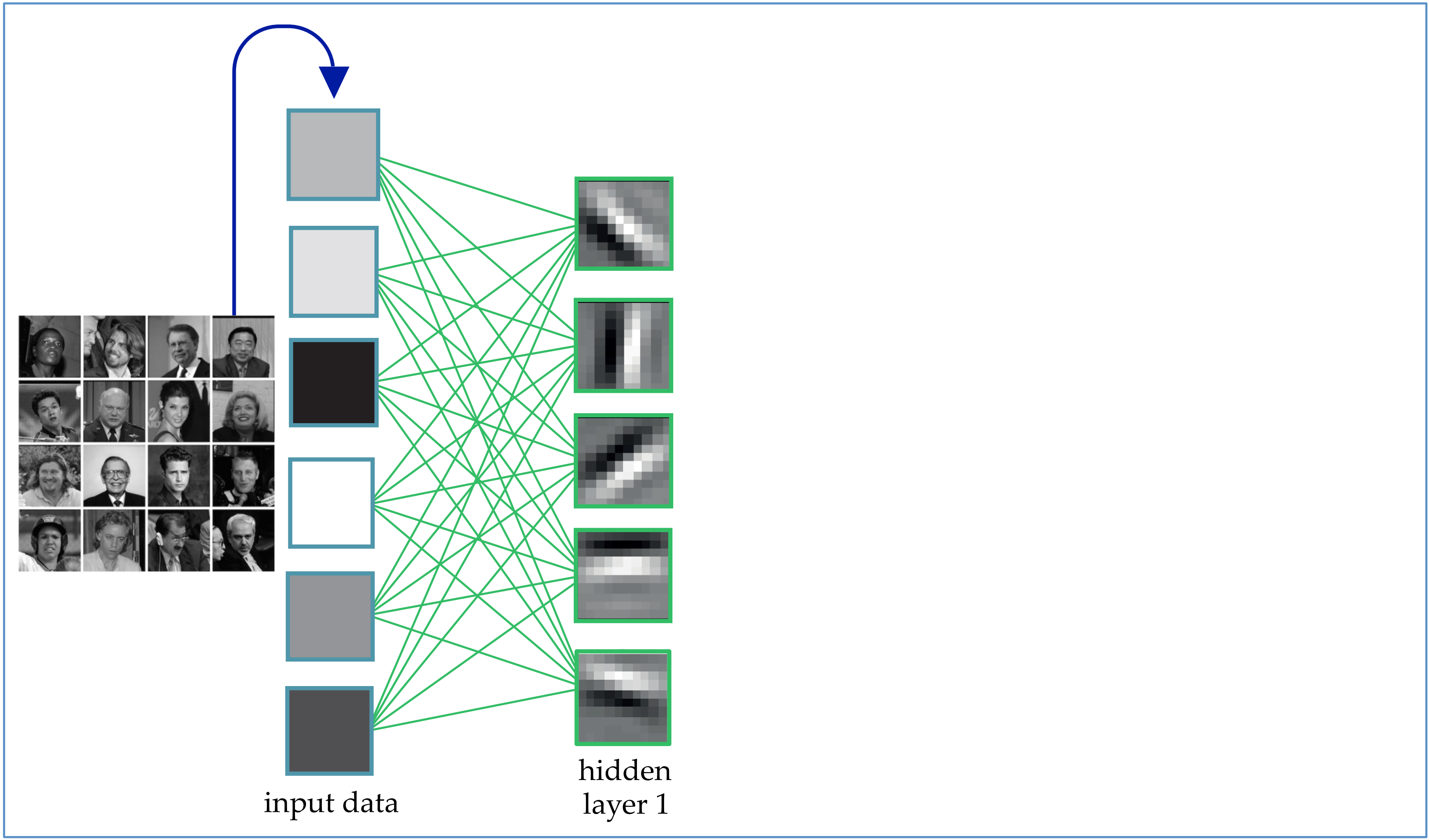
Use <u>unsupervised pre-training</u> to find a function from the input to itself



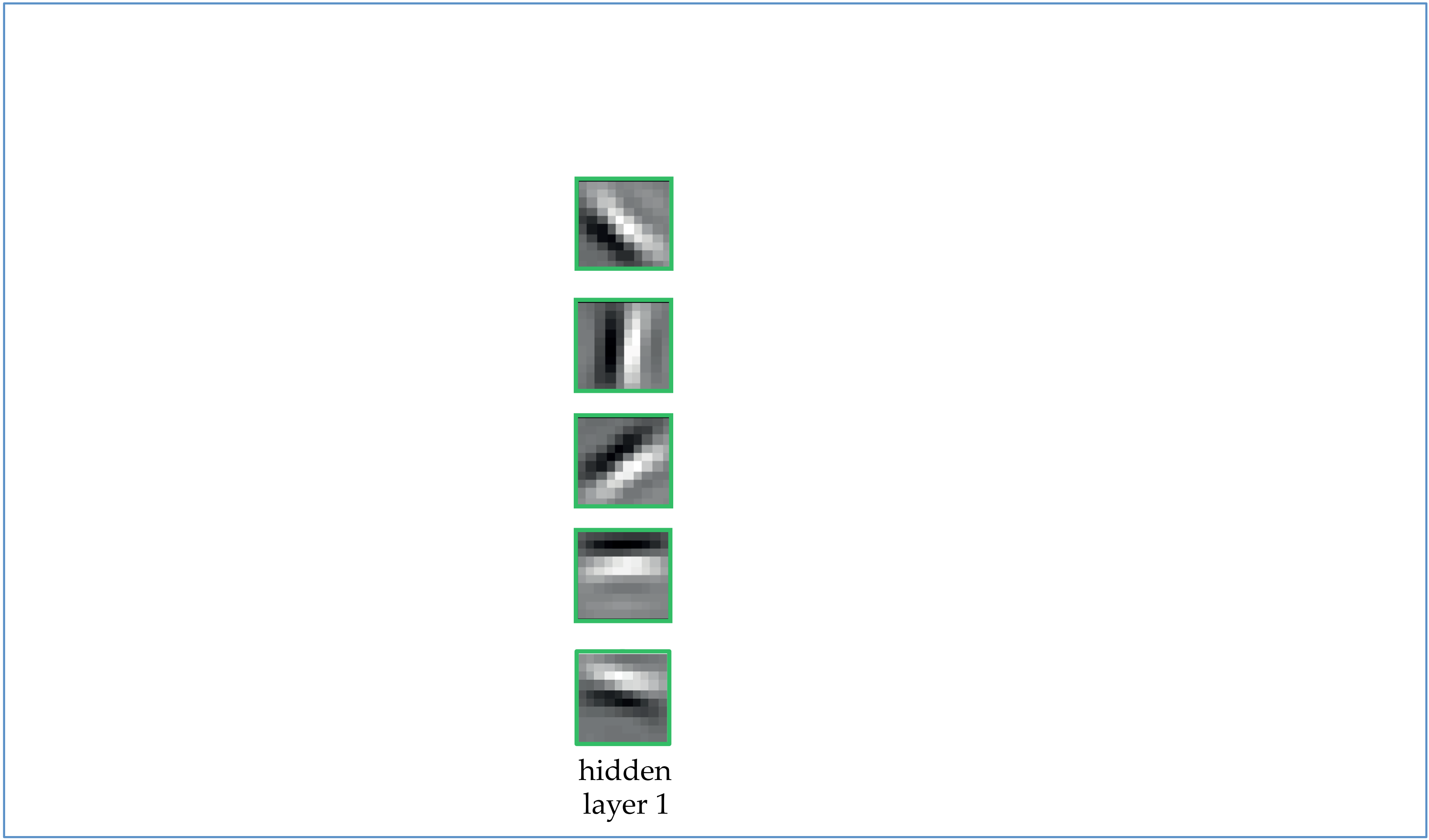
Hidden units can be interpreted as edges



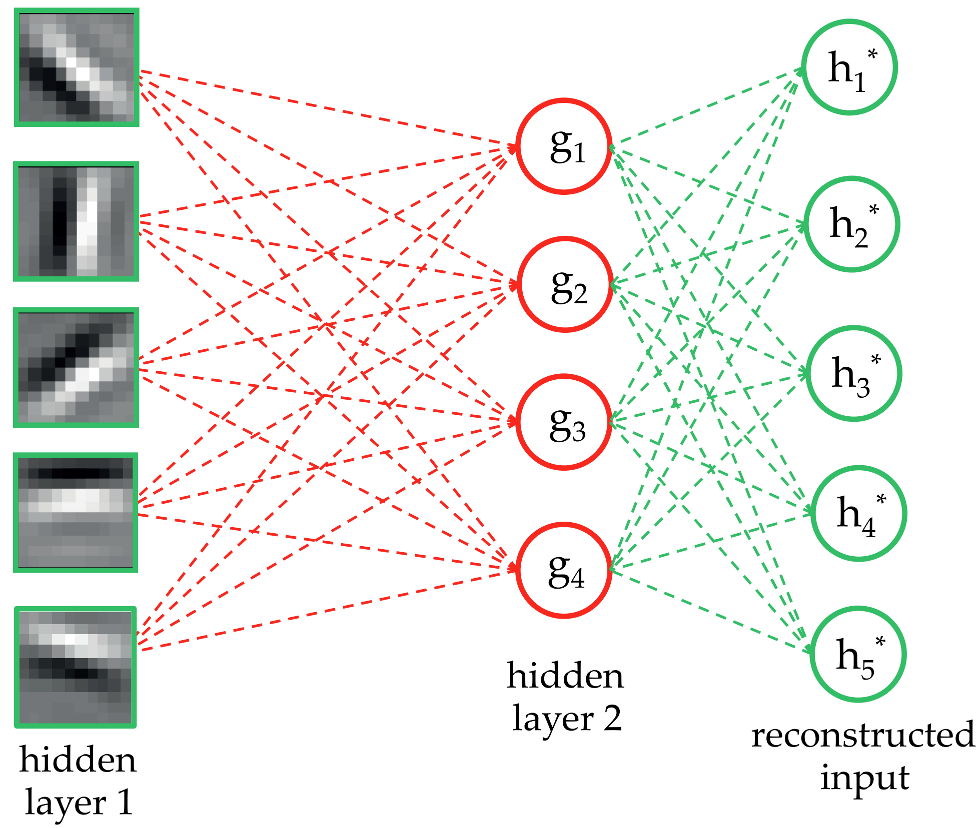
Now: throw away reconstruction and input



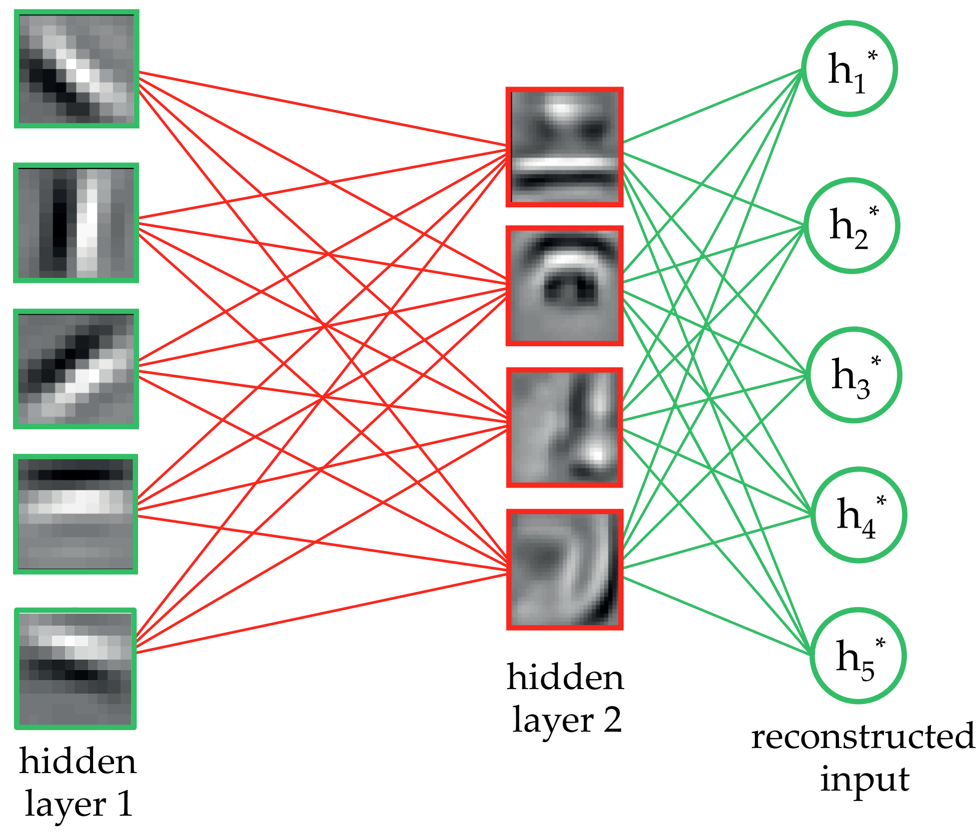
Now: throw away reconstruction and input



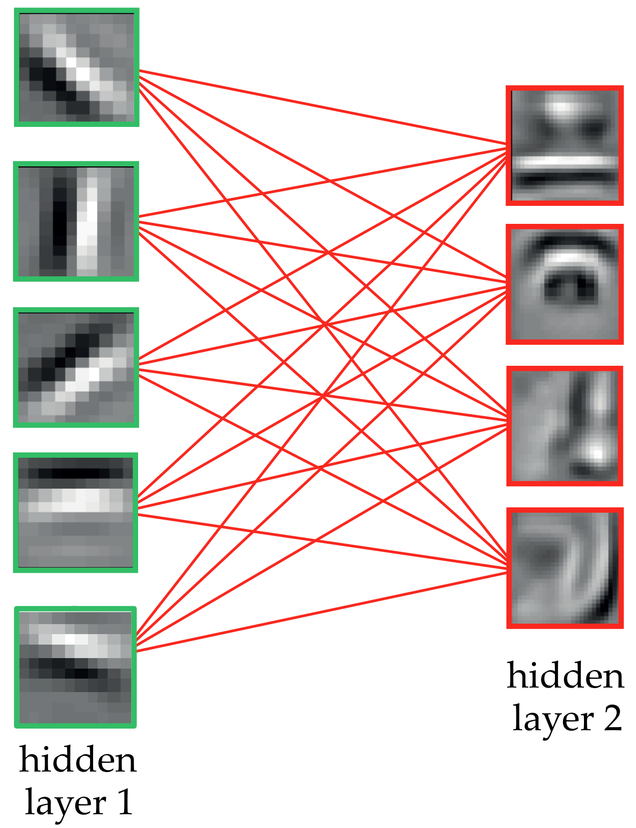
Then repeat the entire process for each layer



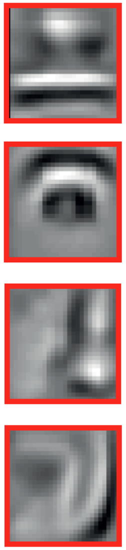
Then repeat the entire process for each layer



Then repeat the entire process for each layer

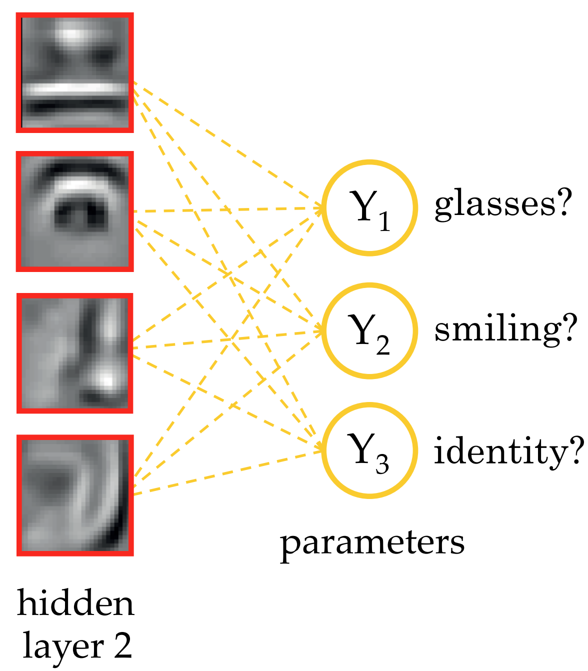


Then repeat the entire process for each layer

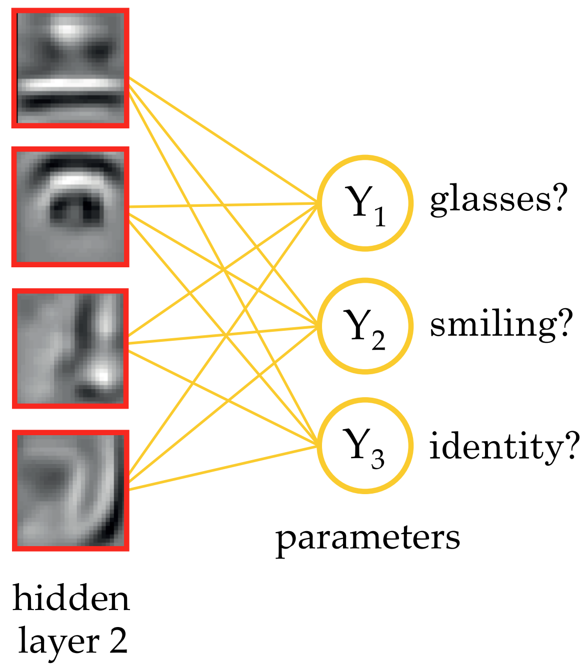


hidden
layer 2

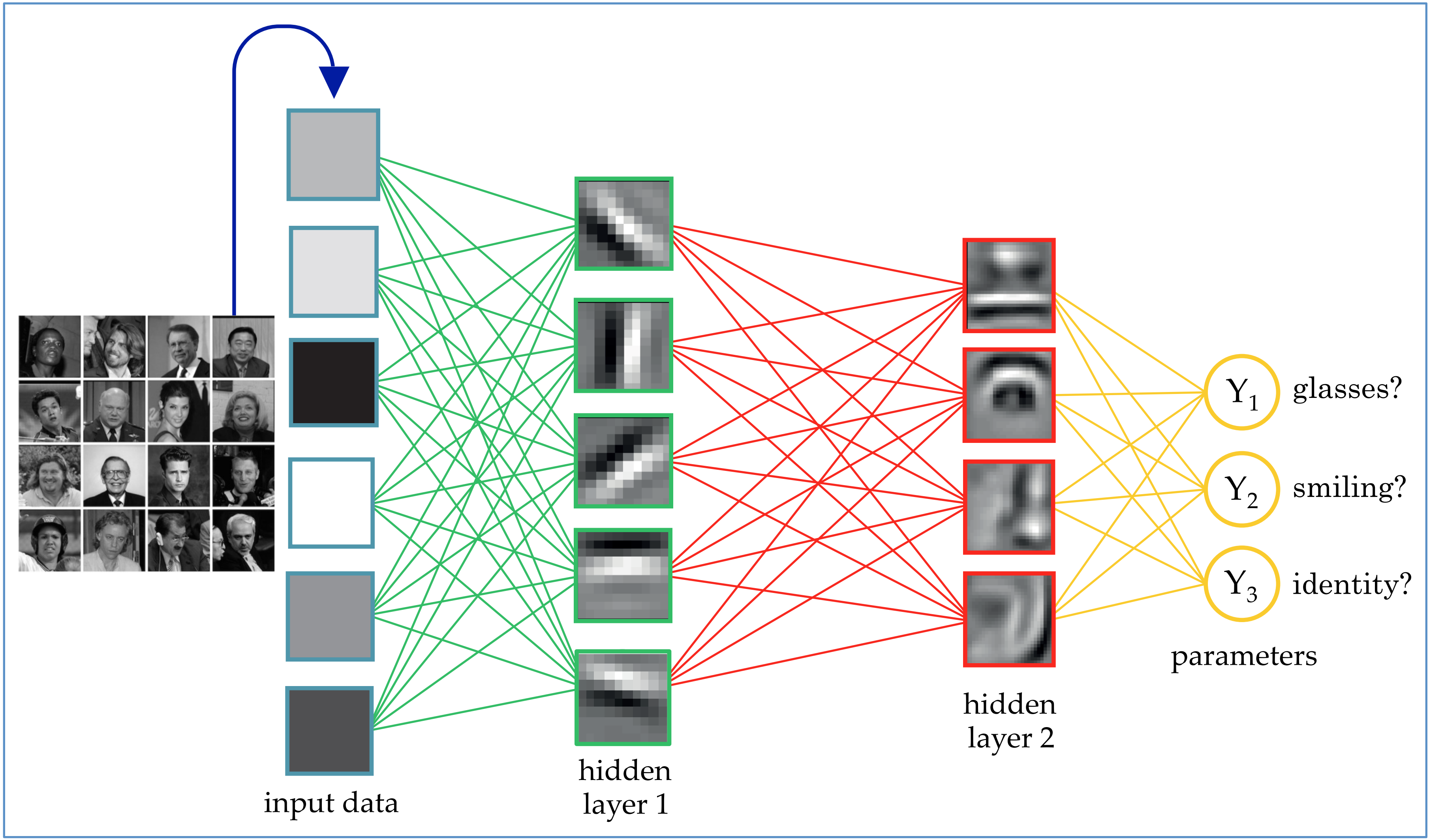
In the last layer, use the outputs (supervised)



In the last layer, use the outputs (supervised)



Finally, “fine-tune” the entire network!



Outline for March 28

- Image data format and intro to Lab 7
- Layer-by-layer pretraining
- Convolutional neural networks (CNNs)

Motivation for moving away from FC architectures

- For a 32x32x3 image (very small!) we have p=3072 features in the input layer
- For a 200x200x3 image, we would have p=120,000! *doesn't scale*
- FC networks do not explicitly account for the structure of an image and the correlations/ relationships between nearby pixels

Idea: 3D volumes of neurons

- Do not "flatten" image, keep it as a volume with *height*, *width*, and *depth*
- For **CIFAR-10**, we would have:
  - Height=32, Width=32, Depth=3
- Each layer is also a 3 dimensional volume
- The final output layer is 1x1xC, where C is the number of classes (10 for CIFAR-10)

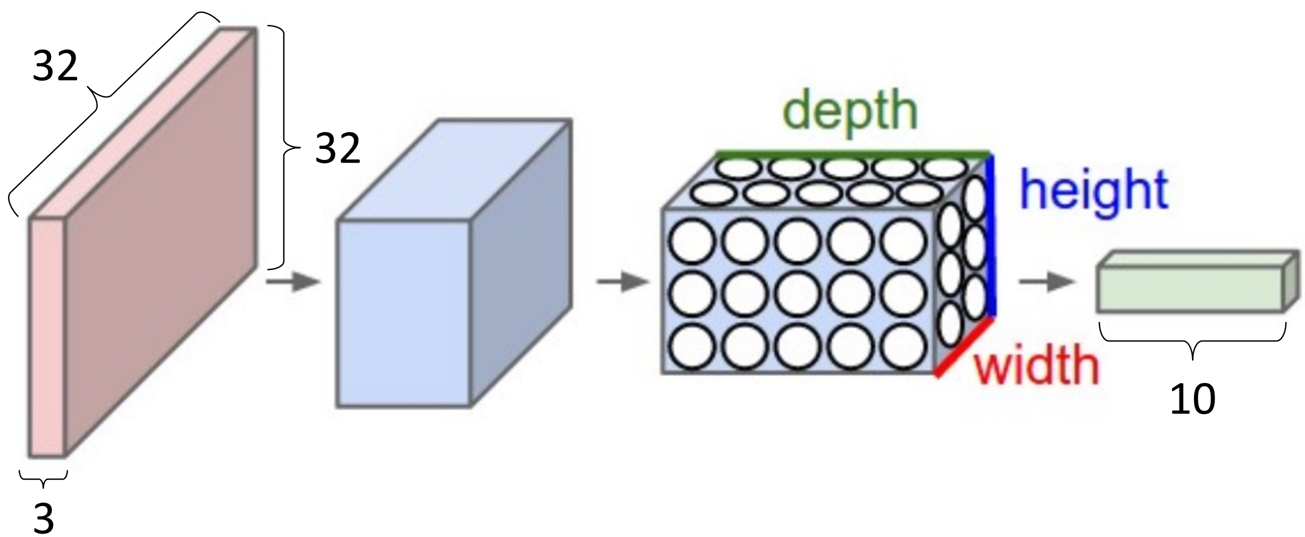


Image: modified from Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

Layers of a Convolutional Neural Network (CNN)

- **INPUT**: raw pixels of a color image, i.e. 32x32x3

More info: http://cs231n.github.io/convolutional-networks/

Layers of a Convolutional Neural Network (CNN)

- **INPUT**: raw pixels of a color image, i.e. 32x32x3

- **CONV**: compute information about a local region of the image using a filter. Example: 12 filters would product a volume of 32x32x12

More info: http://cs231n.github.io/convolutional-networks/

Layers of a Convolutional Neural Network (CNN)

- **INPUT**: raw pixels of a color image, i.e. 32x32x3

- **CONV**: compute information about a local region of the image using a filter. Example: 12 filters would product a volume of 32x32x12

- **RELU**: apply max(0,x), same volume 32x32x12

More info: http://cs231n.github.io/convolutional-networks/

Layers of a Convolutional Neural Network (CNN)

- **INPUT**: raw pixels of a color image, i.e. 32x32x3
- **CONV**: compute information about a local region of the image using a filter. Example: 12 filters would product a volume of 32x32x12
- **RELU**: apply max(0,x), same volume 32x32x12
- **POOL**: downsample, i.e. with result 16x16x12

More info: http://cs231n.github.io/convolutional-networks/

Layers of a Convolutional Neural Network (CNN)

- **INPUT**: raw pixels of a color image, i.e. 32x32x3
- **CONV**: compute information about a local region of the image using a filter. Example: 12 filters would product a volume of 32x32x12
- **RELU**: apply max(0,x), same volume 32x32x12
- **POOL**: downsample, i.e. with result 16x16x12
- **FC** (fully-connected): produce probabilities for each class, i.e. volume 1x1x10

More info: http://cs231n.github.io/convolutional-networks/

Example CNN architecture

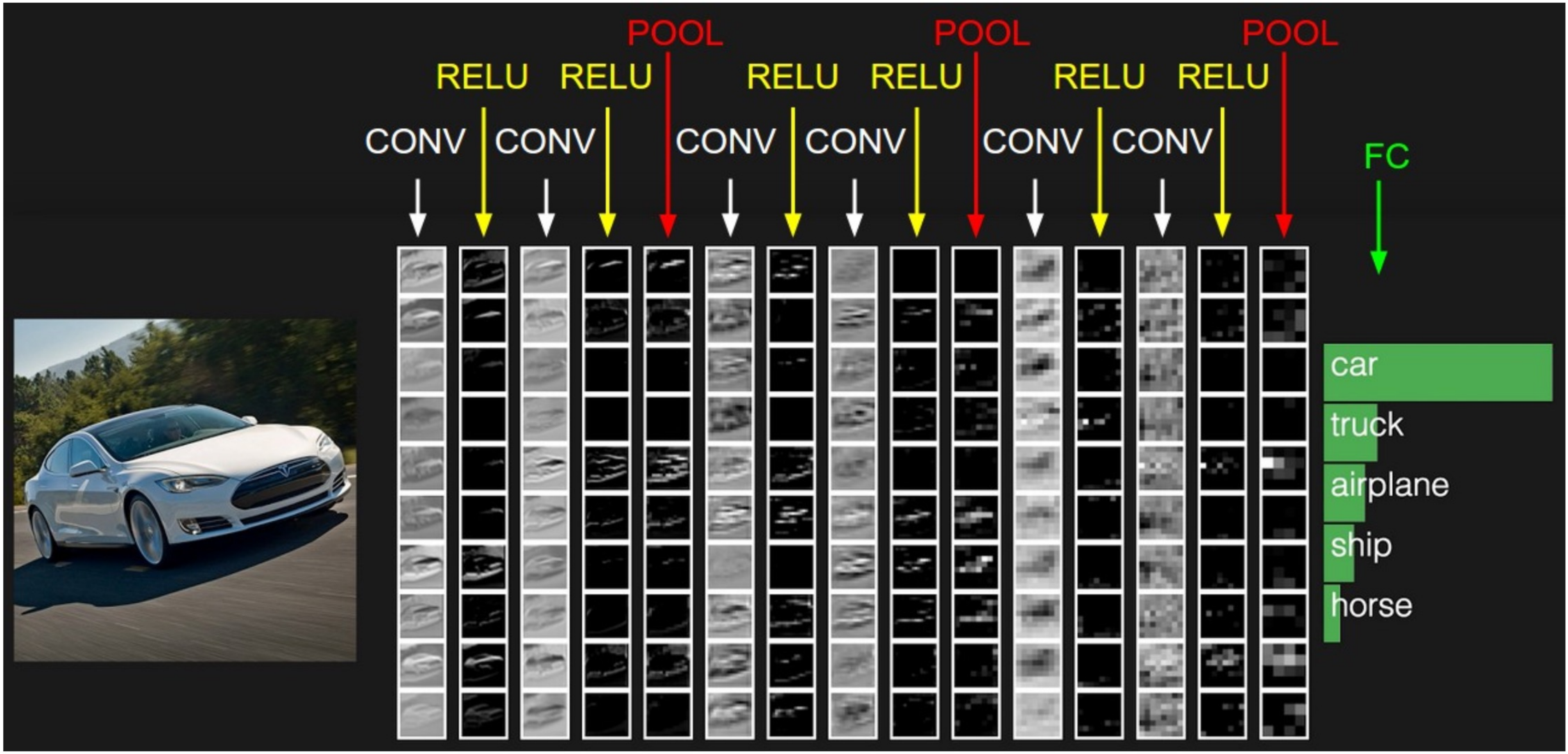
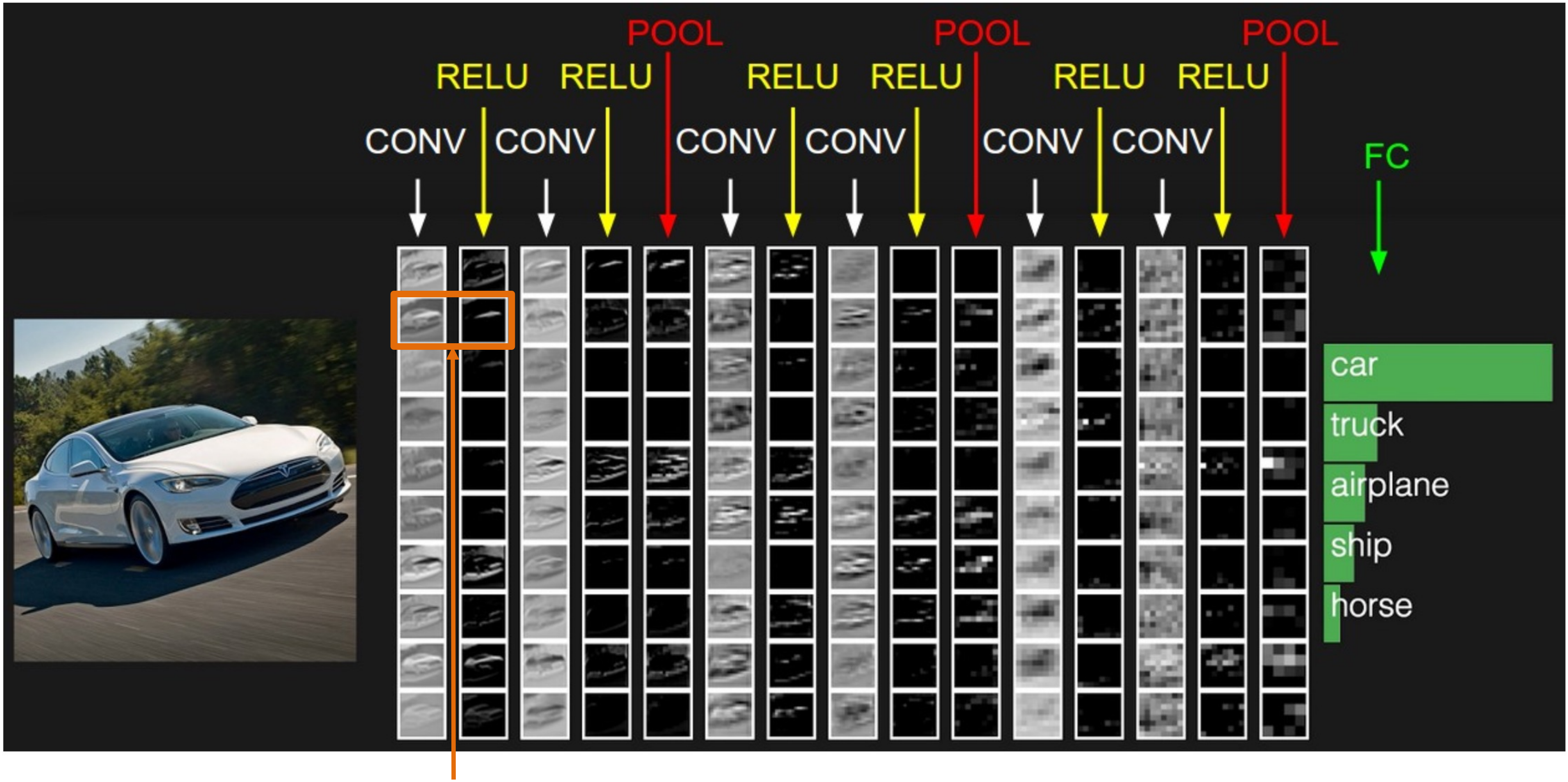


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

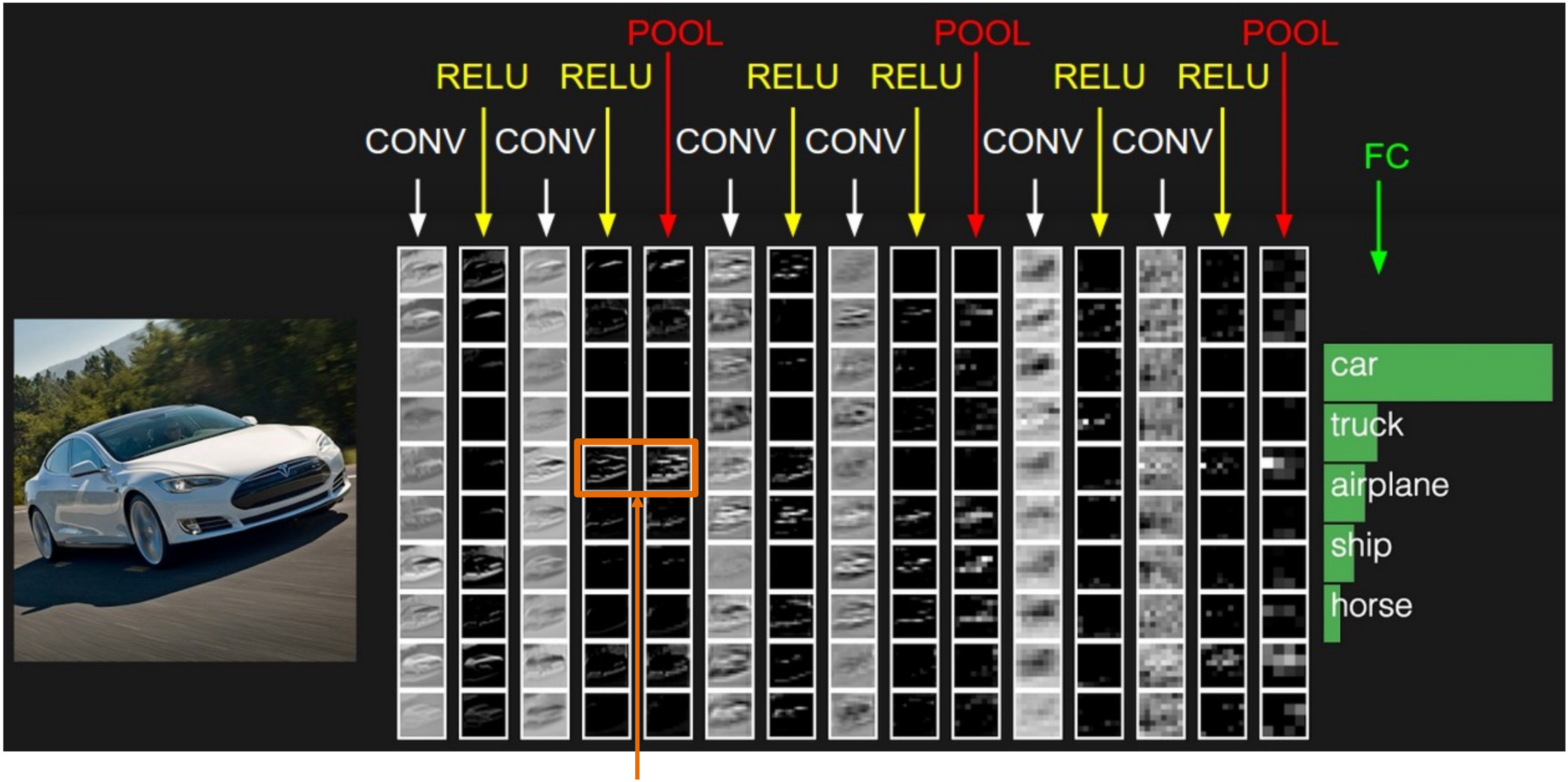
Example CNN architecture



ReLU zeros out less relevant information, highlighting an interesting feature (i.e. hood of car here)

Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

Example CNN architecture



POOL reduces the size of the volume
but keeps relevant features

Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

Visualization of an entire network

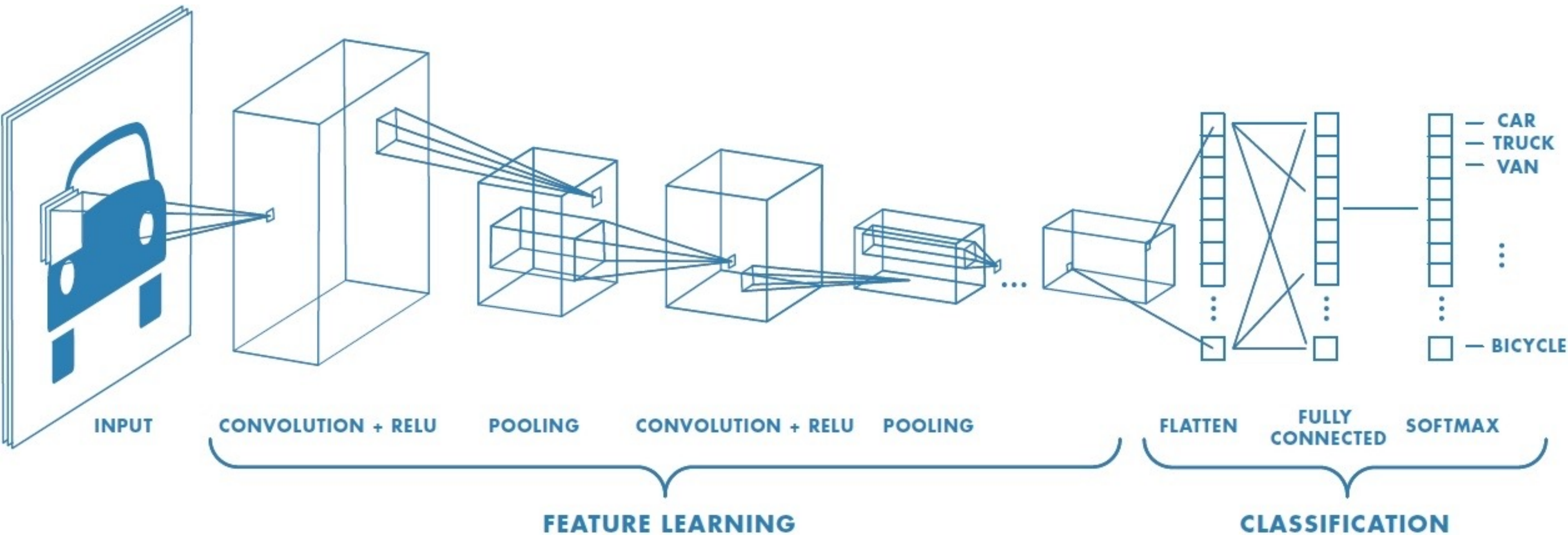


Image from MathWorks: https://www.mathworks.com/solutions/deep-learning/convolutional-neural-network.htr

Idea: local “receptive field”

- A convolutional filter (matrix) can pick up on local features in the original image through an element-wise dot-product

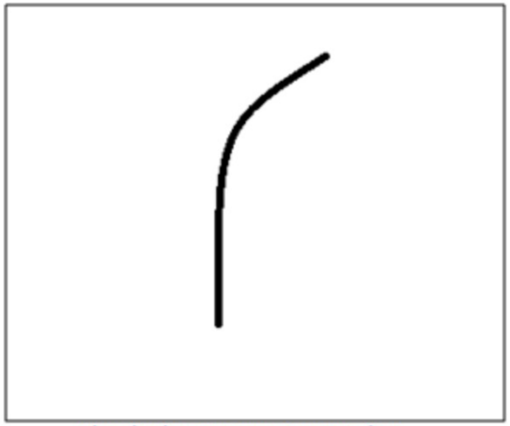
- Note an important *asymmetry*: we will look at a small “patch” of the image relative to its height and width, but we will **look all the way through the depth**!

Intuition: as learning progresses, filters become specialized for certain types of features

Example:
"Curve" filter

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter



Visualization of a curve detector filter

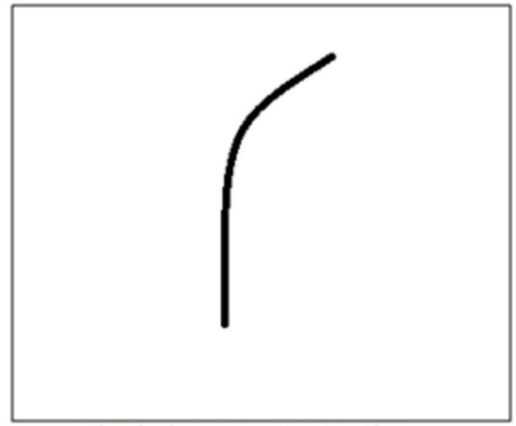
Images by: Adit Deshpande

Intuition: as learning progresses, filters become specialized for certain types of features

Example:
“Curve” filter

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

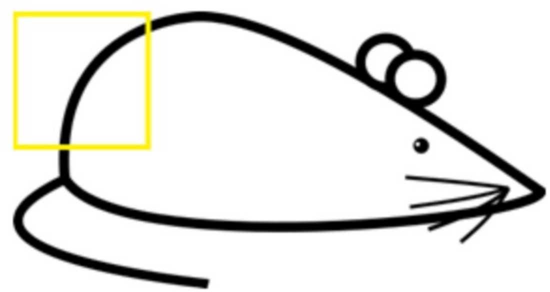
Pixel representation of filter



Visualization of a curve detector filter

Say we apply this filter to an image

Original image



Visualization of the filter on the image

Images by: Adit Deshpande

Output of convolutions will “light up” if filter “matches” receptive field, but not otherwise

Visualization of the receptive field

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

Pixel representation of the receptive field

*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Output to next layer:
6600

Multiplication and Summation = (50*30)+(50*30)+(50*30)+(20*30)+(50*30) = 6600 (A large number!)

Images by: Adit Deshpande

Output of convolutions will “light up” if filter “matches” receptive field, but not otherwise

Visualization of the receptive field

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

Pixel representation of the receptive field

$*$

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Output to next layer:
6600

Multiplication and Summation = (50*30)+(50*30)+(50*30)+(20*30)+(50*30) = 6600 (A large number!)

Visualization of the filter on the image

0	0	0	0	0	0	0
0	40	0	0	0	0	0
40	0	40	0	0	0	0
40	20	0	0	0	0	0
0	50	0	0	0	0	0
0	0	50	0	0	0	0
25	25	0	50	0	0	0

Pixel representation of receptive field

$*$

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Output to next layer:
0

Multiplication and Summation = 0

Images by: Adit Deshpande

Examples of learned filters

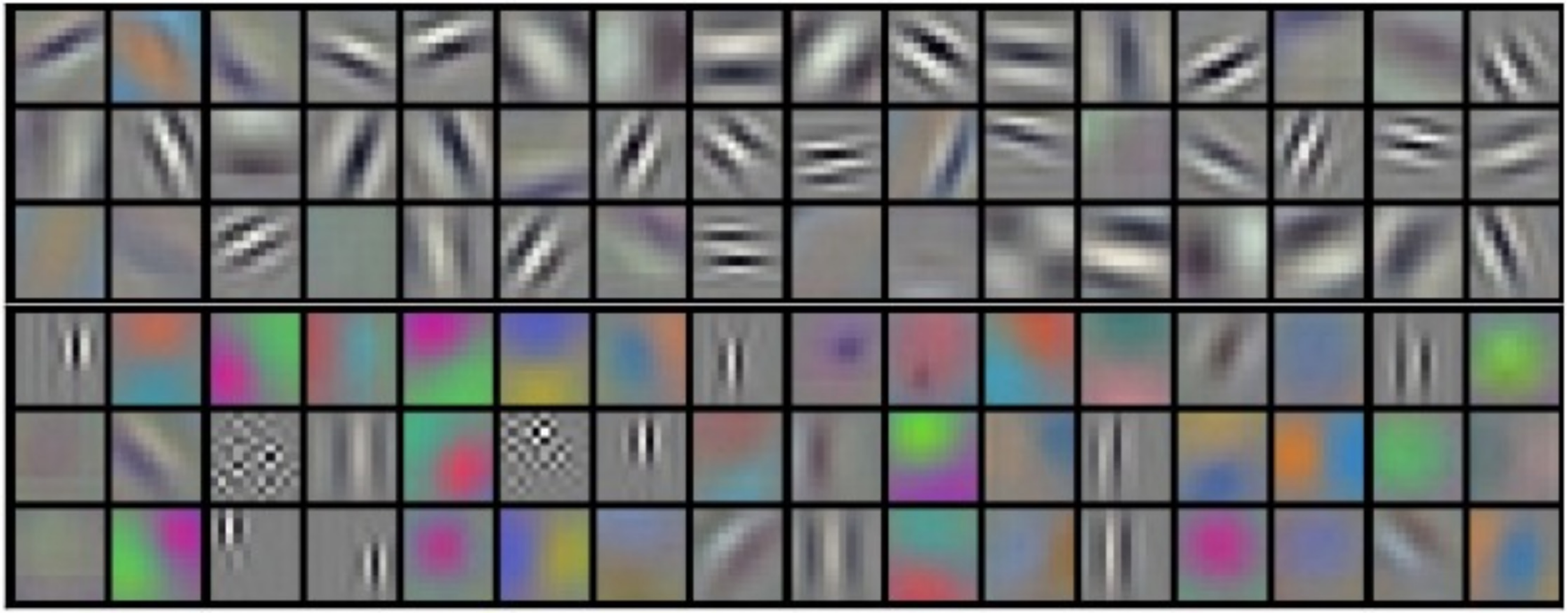


Image: Krizhevsky et al. (2012) https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

Math behind convolutions

(actually cross-correlations!)

https://en.wikipedia.org/wiki/Cross-correlation

CNN

image

2	1	0
0	1	0
0	0	1

filter (learned)

w_{11} w_{12}

1	0	0
0	1	0
0	0	1

$\boxed{2+1+1=4}$

element-wise dot product

0	0	0
1	1	1
0	0	0

$\Rightarrow \boxed{1}$

horizontal line filter

Cross-correlations — 1D

f (image)

i =		1	2	3	4	5	6	
	0	3	-1	2	0	1	8	0

zero pad — input — zero pad

g(filter)

-1	0	1
1	0	2

$$\boxed{\underbrace{f\star g}_{\text{function}}\underbrace{(i)}_{\text{arg}} = \sum_{m=-M}^{M} f(i+m)\,g(m)}$$

-1	0	1	2	3	4	5	6	7	8
0	0							0	0

M — input — M

$M=2$

-2	-1	0	1	2
5	1	0	2	-3

filter

1x5

odd

\# neurons

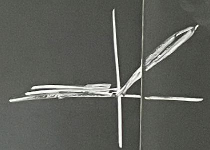
$\boxed{M=1}$

$\boxed{i=2}$

$$f \star g(2) = \overbrace{f(2-1)g(-1)}^{m=-1} + \overbrace{f(2)\,g(0)}^{m=0} + \overbrace{f(2+1)g(1)}^{m=1}$$

$$= 3 \cdot 1 + (-1) \cdot 0 + 2 \cdot 2$$

$$= \boxed{7}$$



$i =$ 1	2	3	4	5	6
-2	7	-1	4	16	1

$\underbrace{f \star g}_{i=1,2,3,\dots,6} =$ output

RELU (activation) ← on own!

0	7	0	4	16	1
max		max		max	

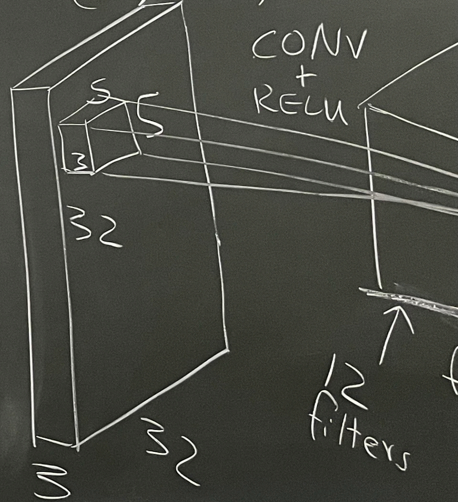
(max) POOL

7	4	16

output after layer

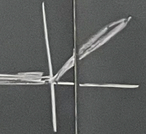
ONE LAYER

(32, 32, 3)



$$\ldots(2)\underbrace{g(0)}_{m=0} + \underbrace{f(z+1)g(1)}_{m=1}$$

$\ldots + 2 \cdot 2$



utput

un!

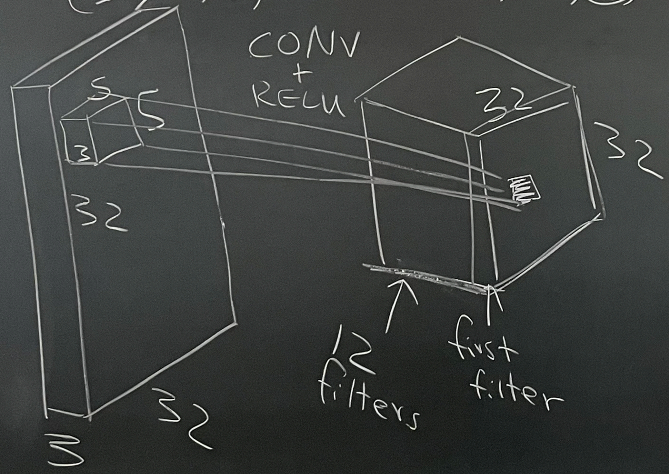
ayer

ONE LAYER

(32,32,3)

12 filters in first CONV layer

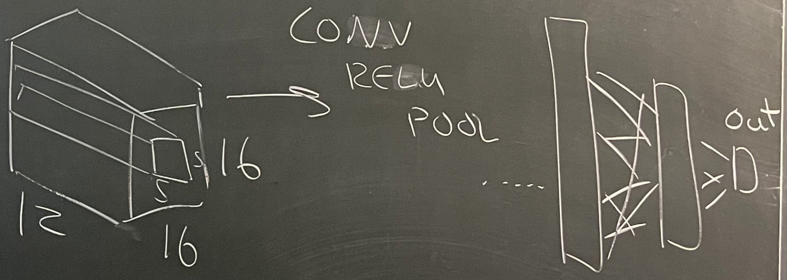
(32,32,12)

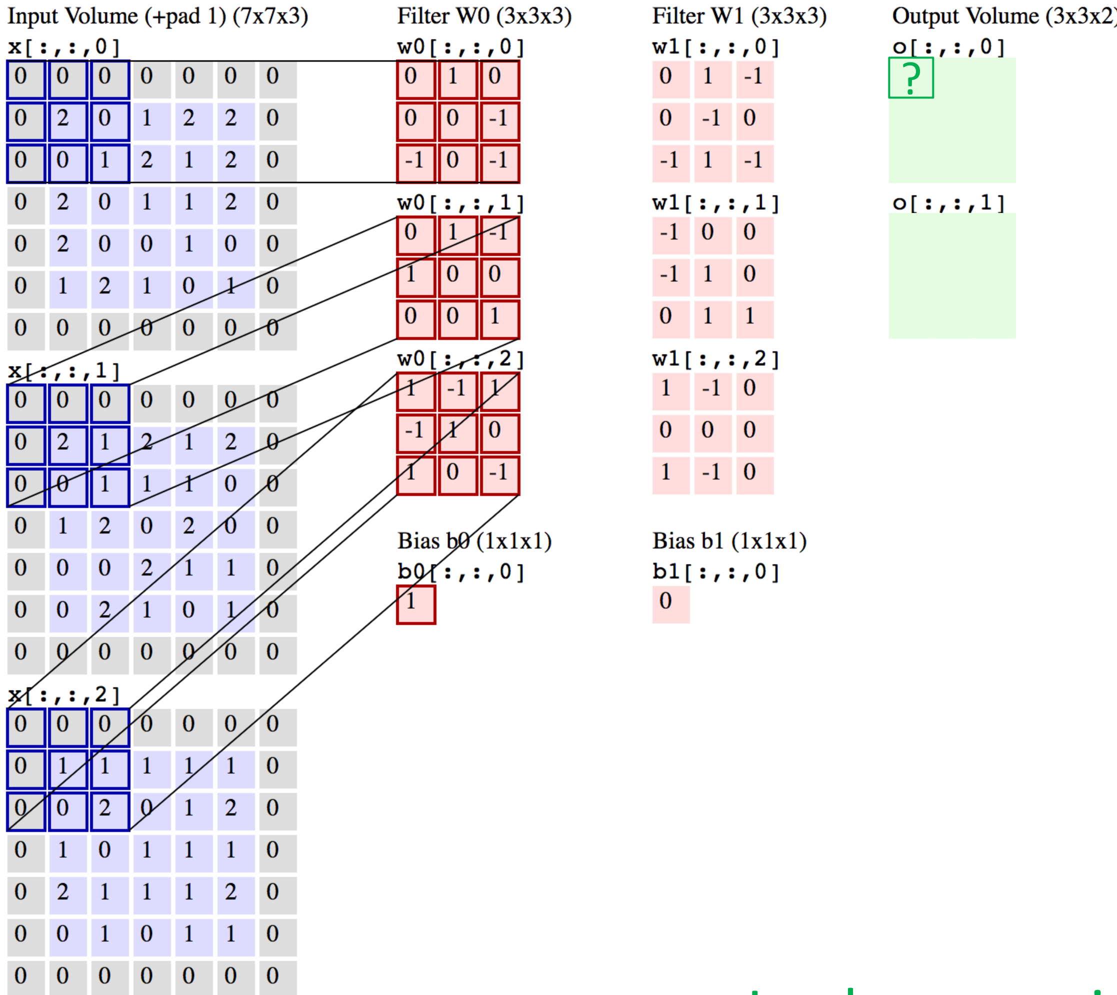


hyperparameters

POOL (16,16,12) → (8,8,24)

24 filters in the second CONV layer





In-class exercise

Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

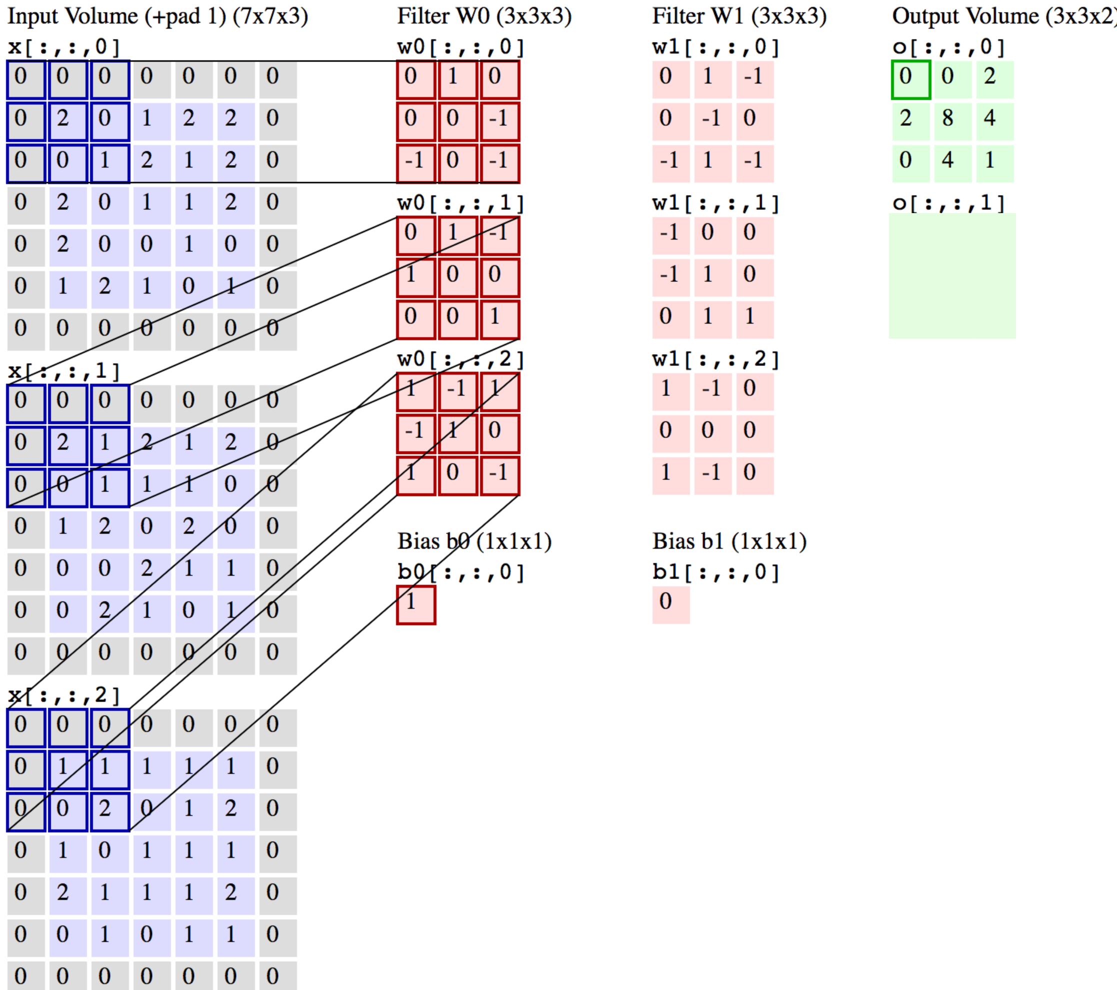


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

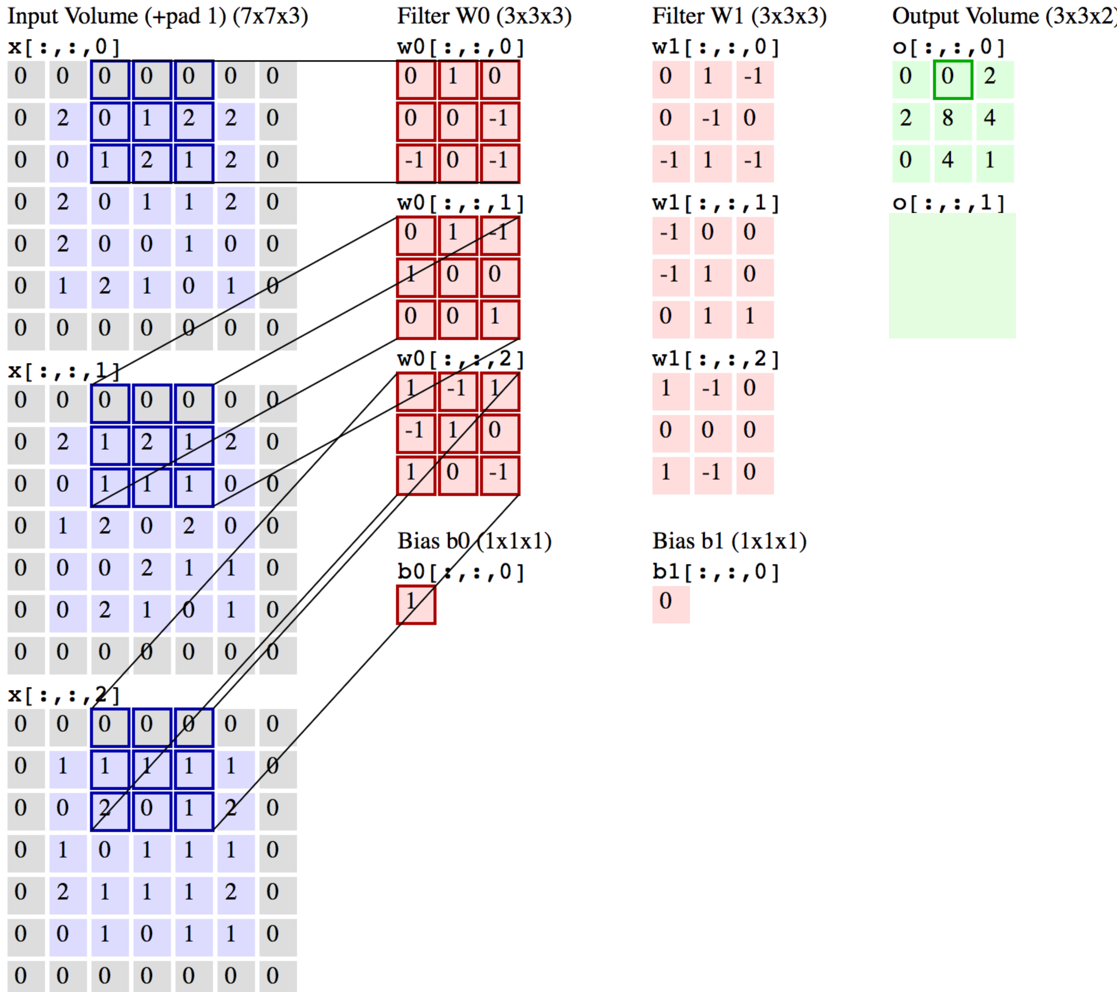


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

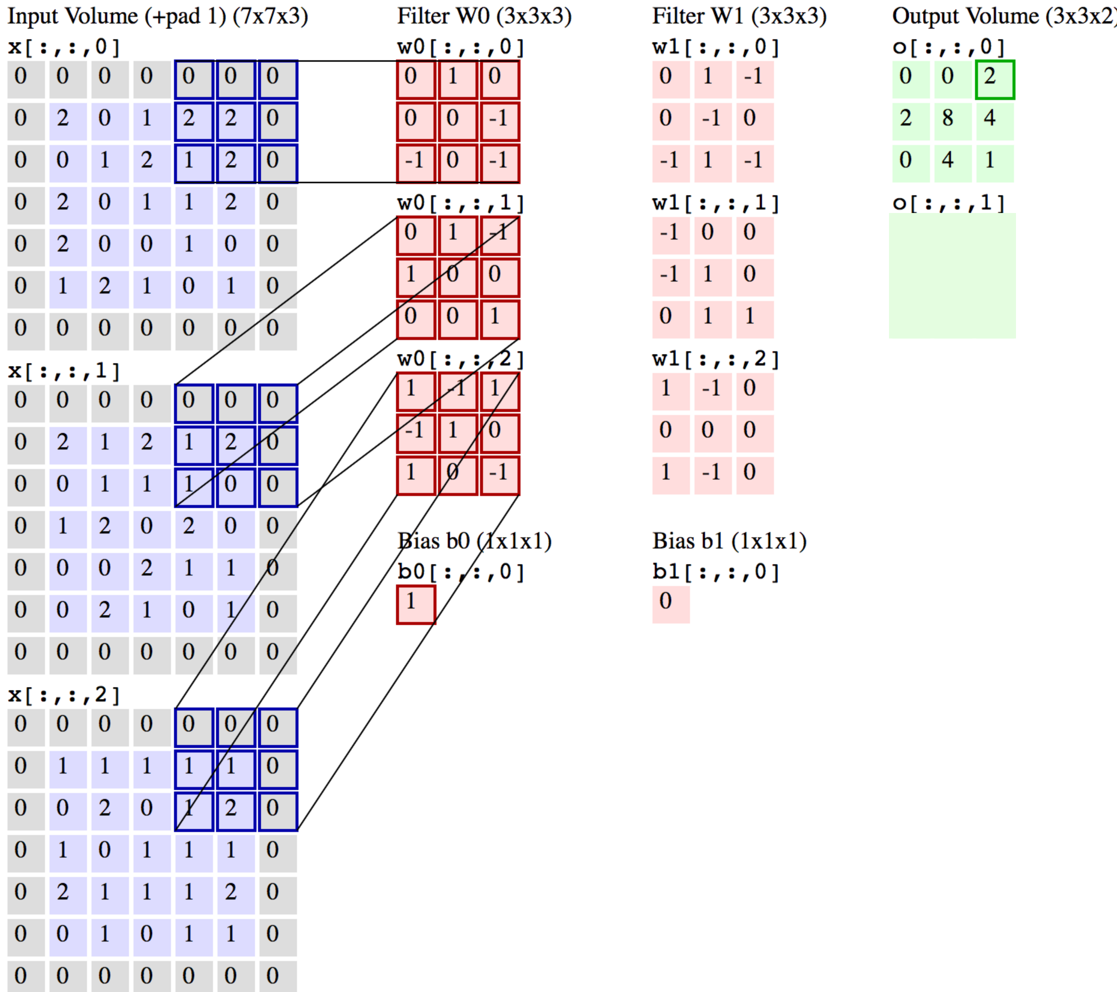


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

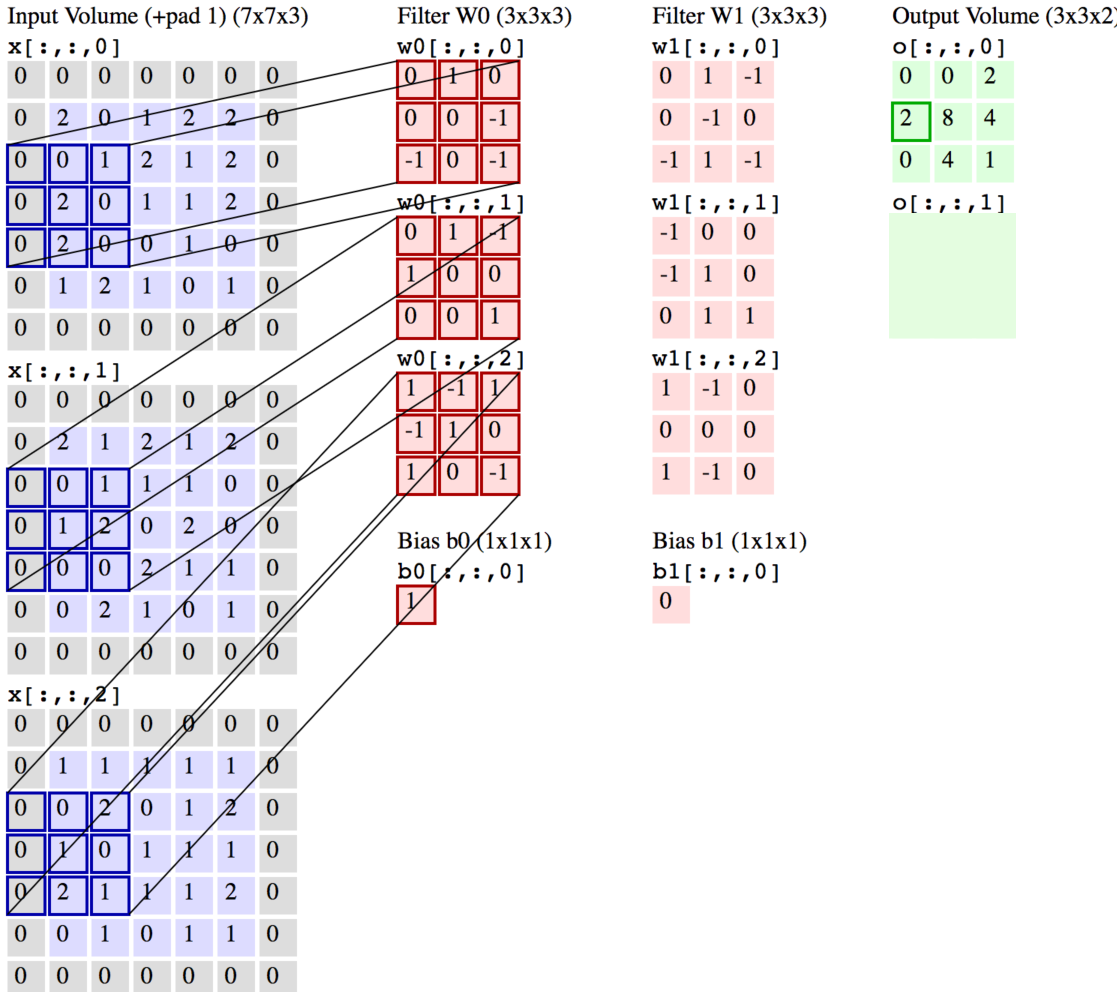


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

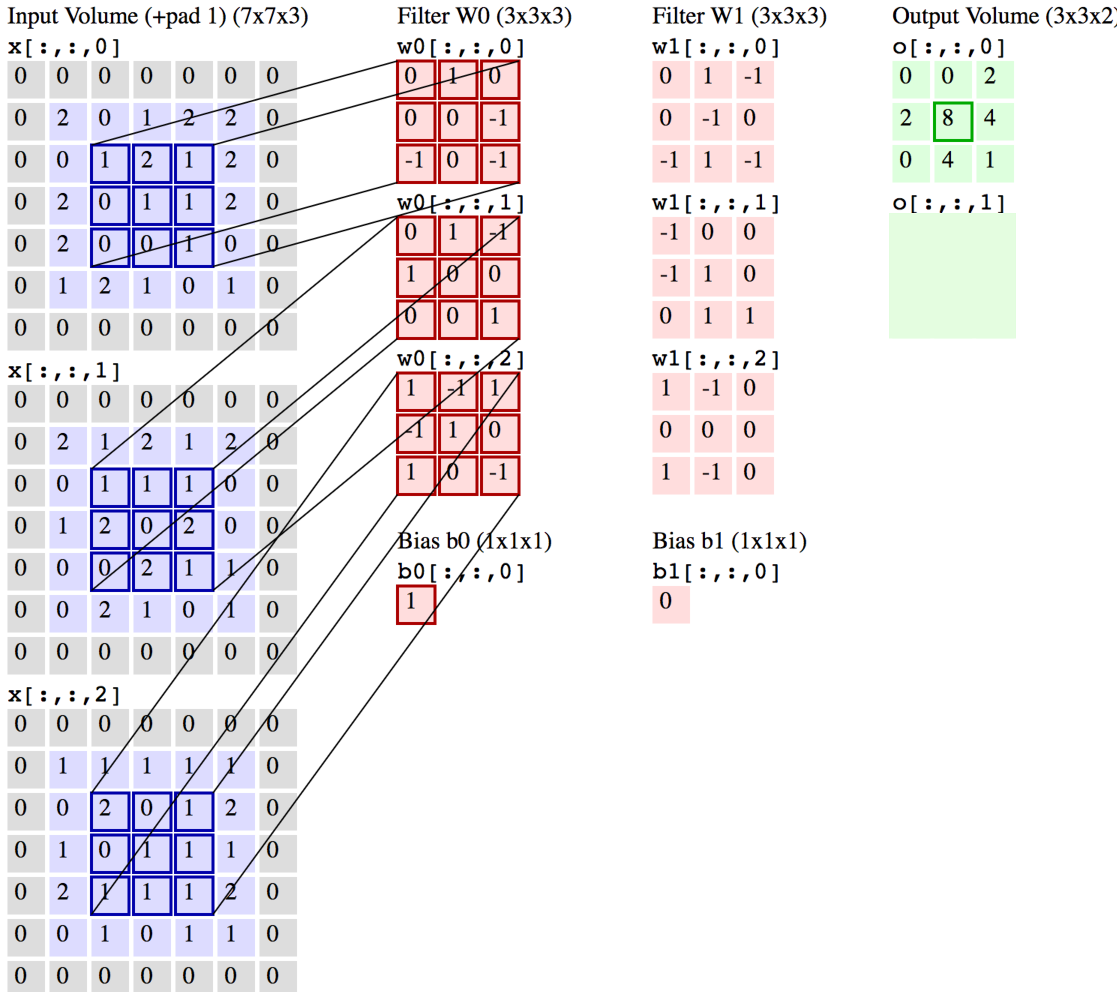


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

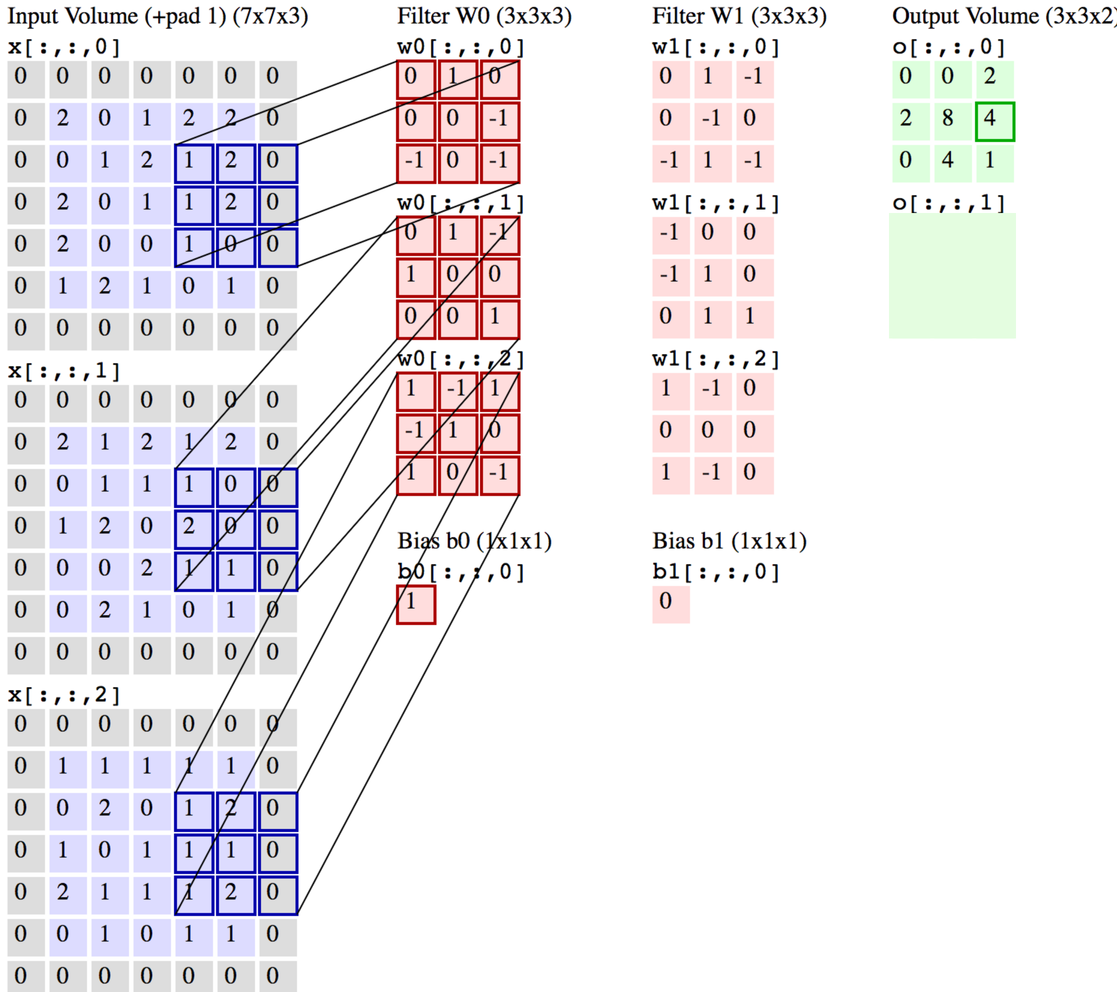


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

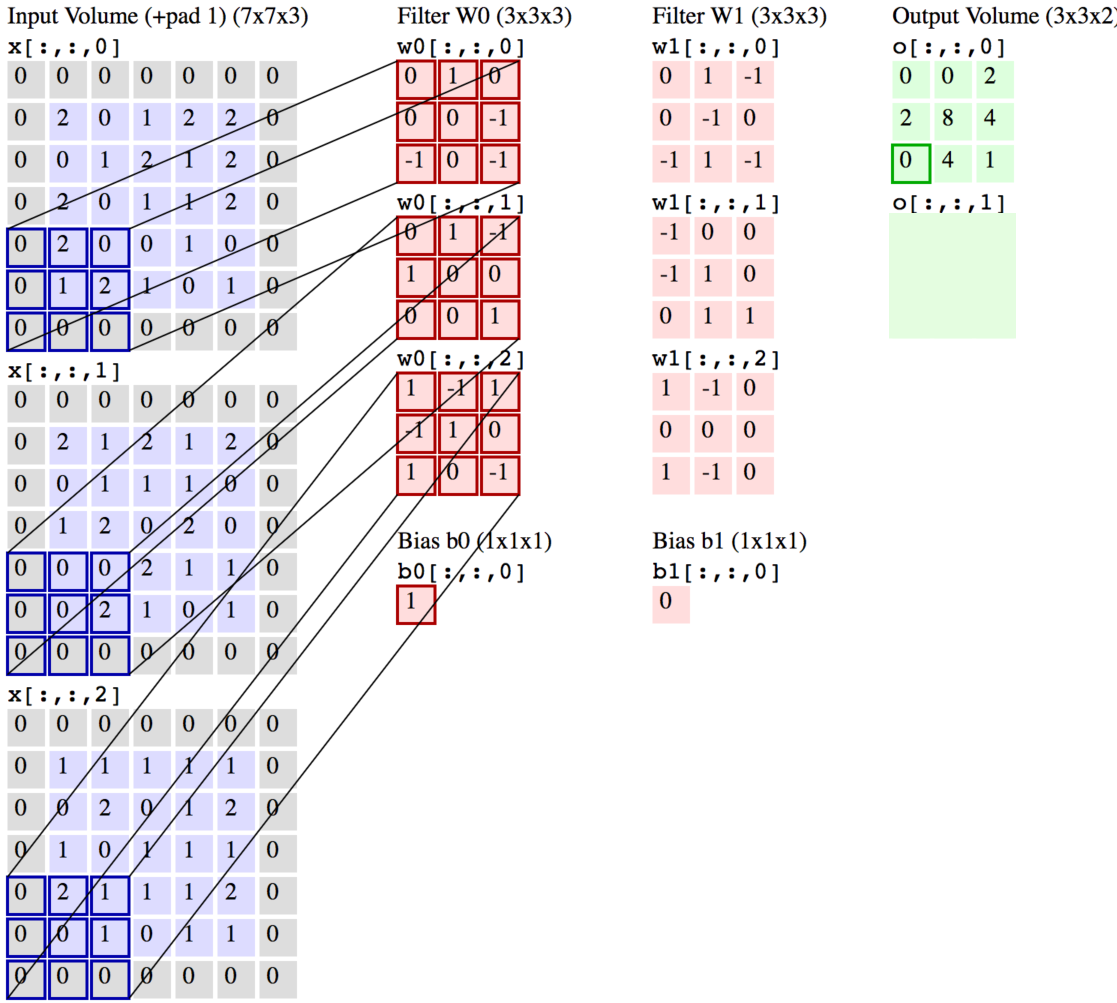


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

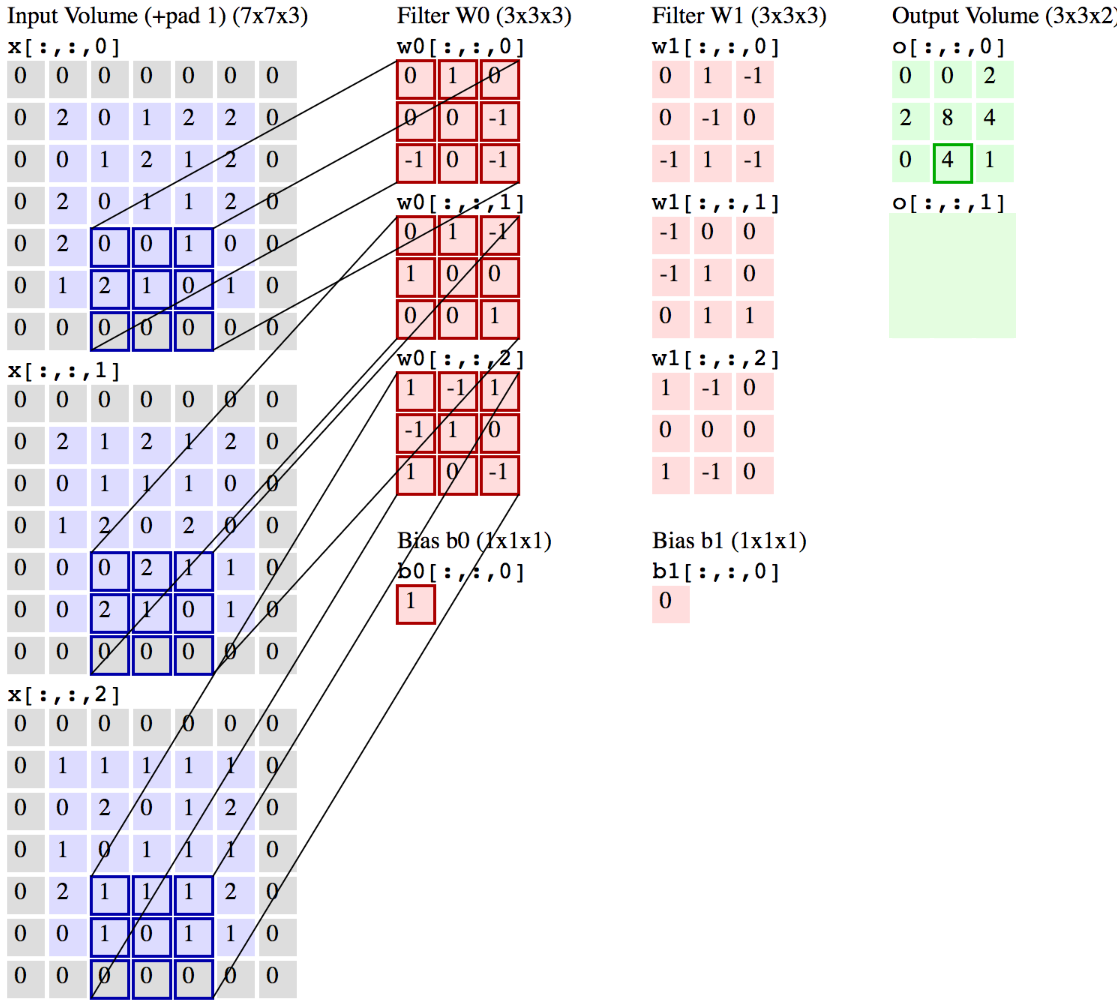


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

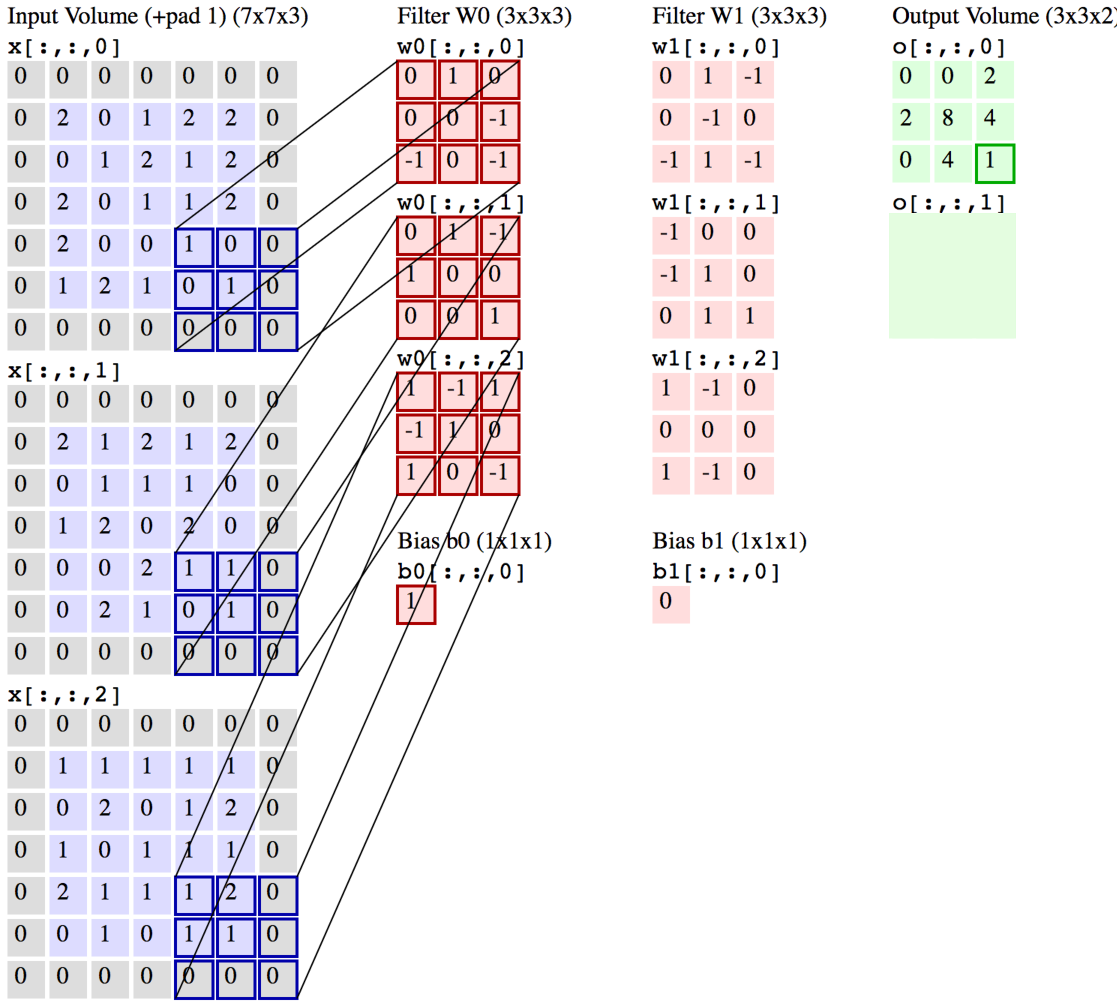


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

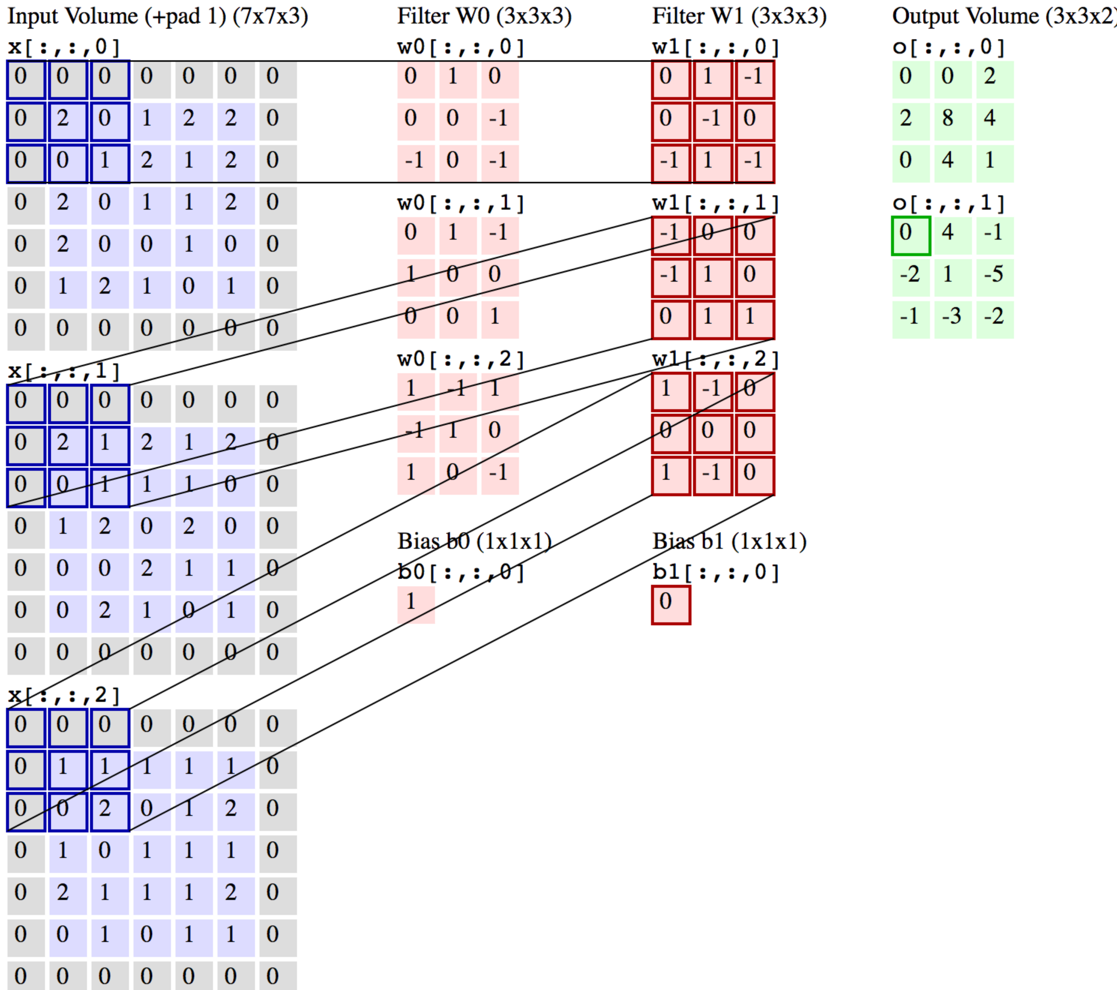


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

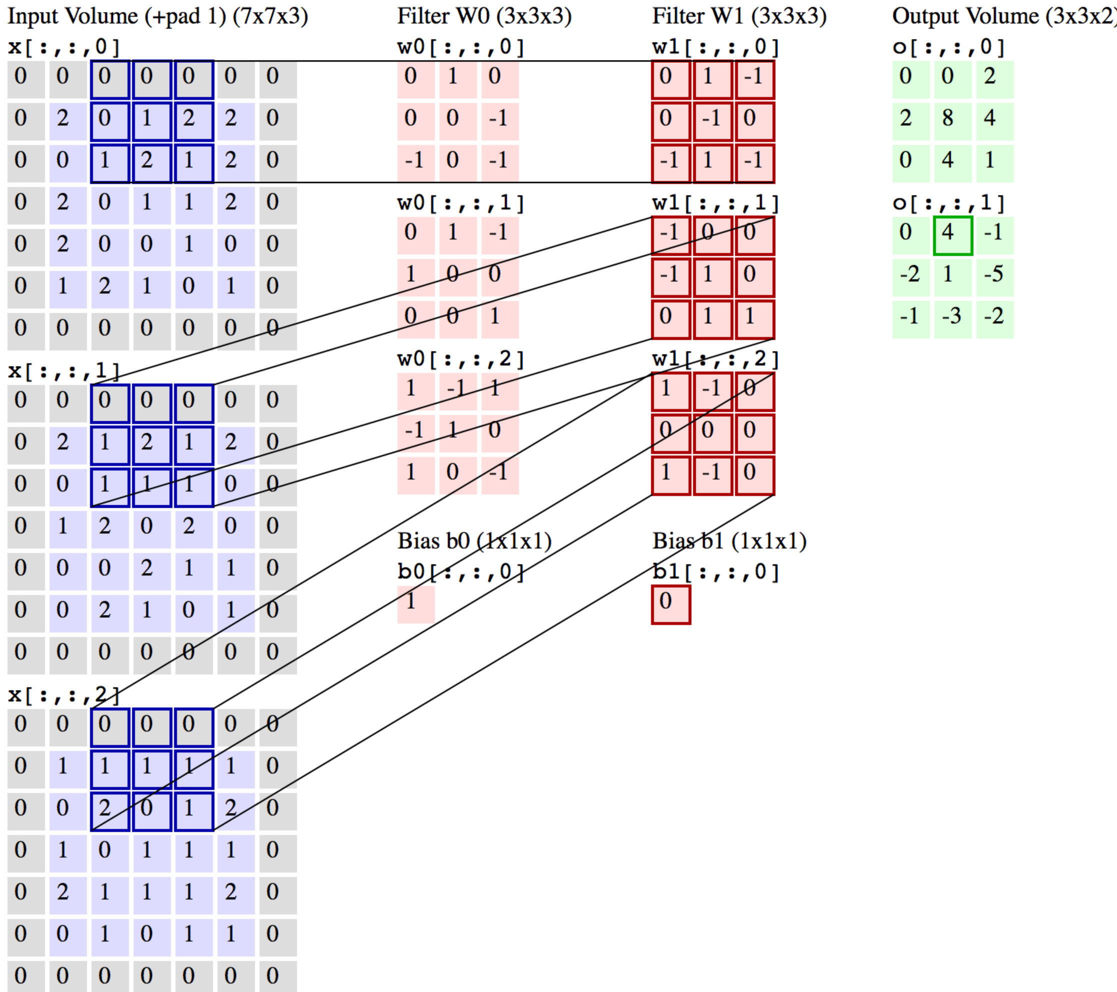


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

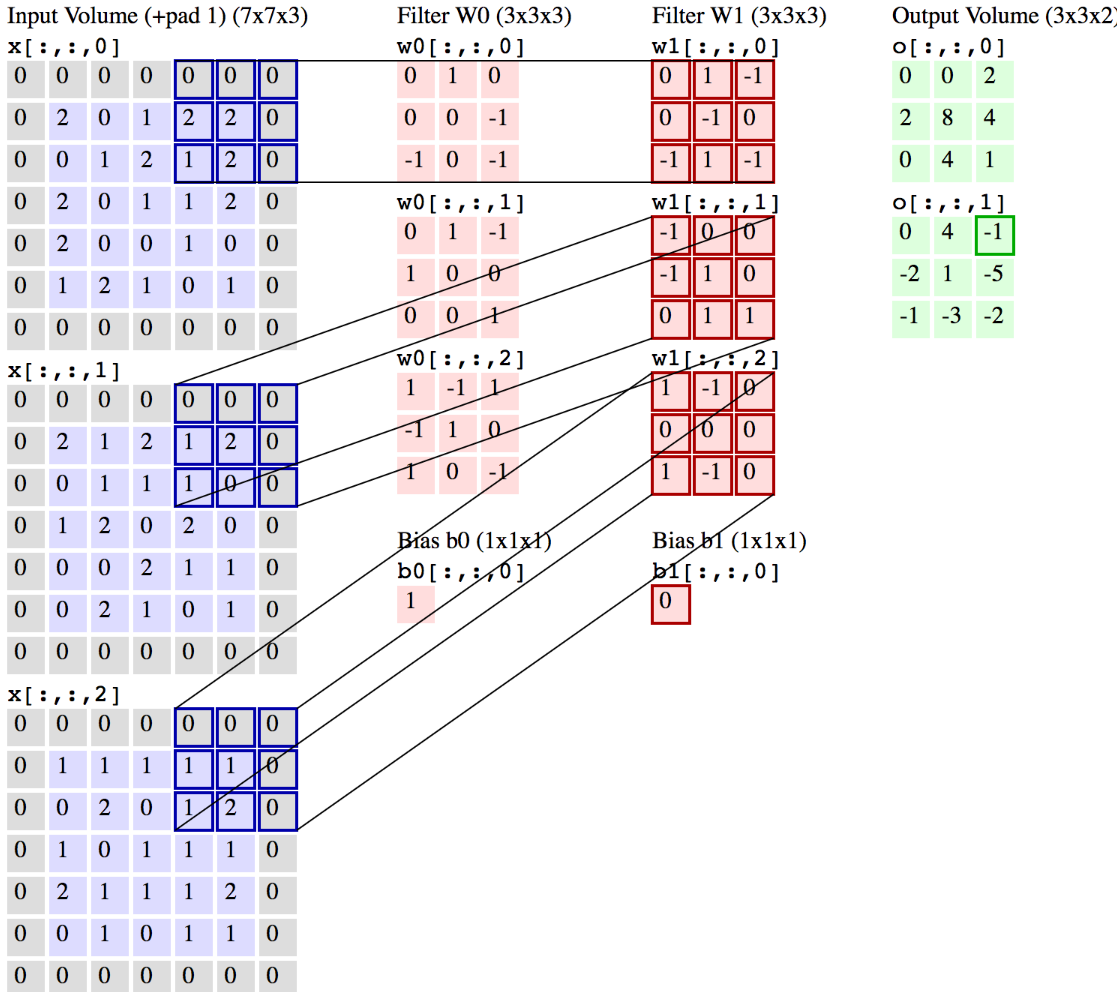


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

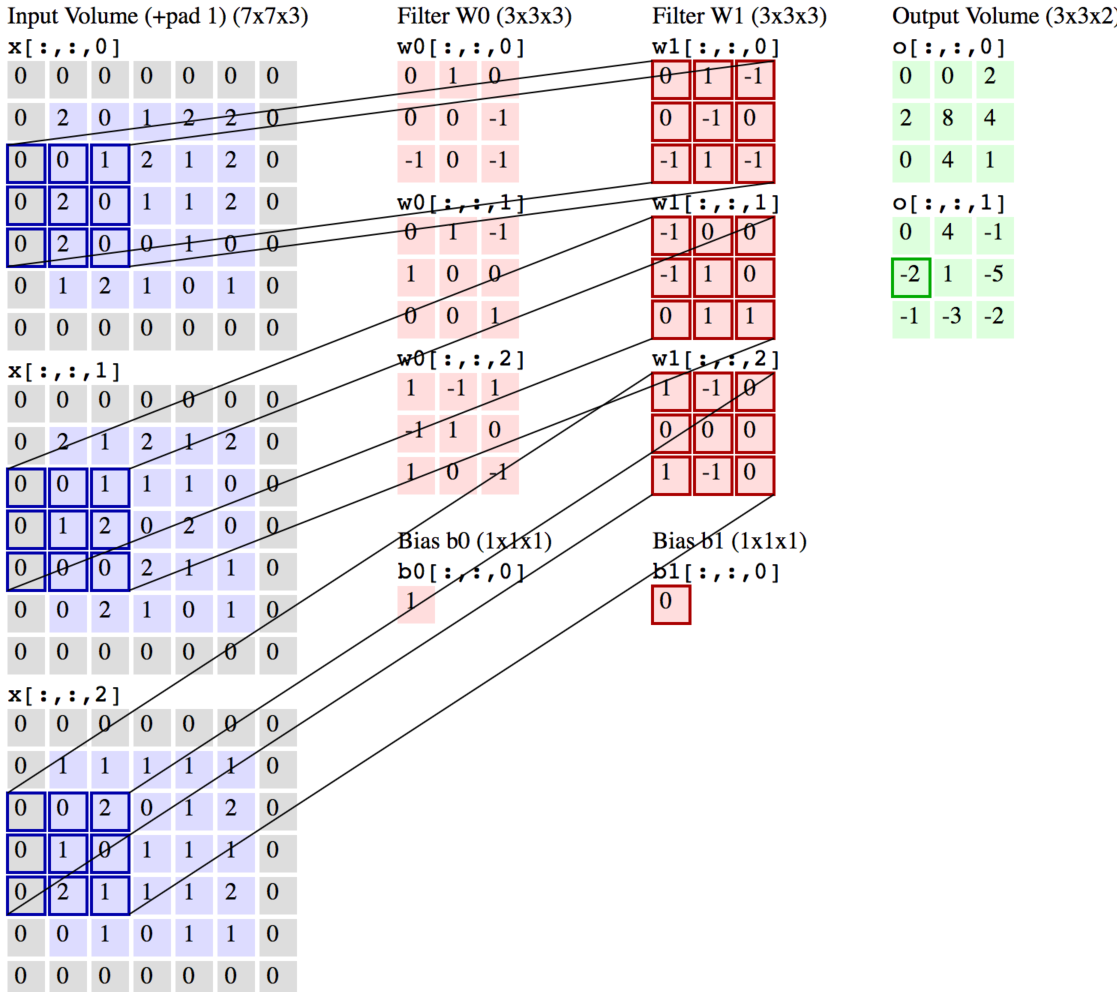


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

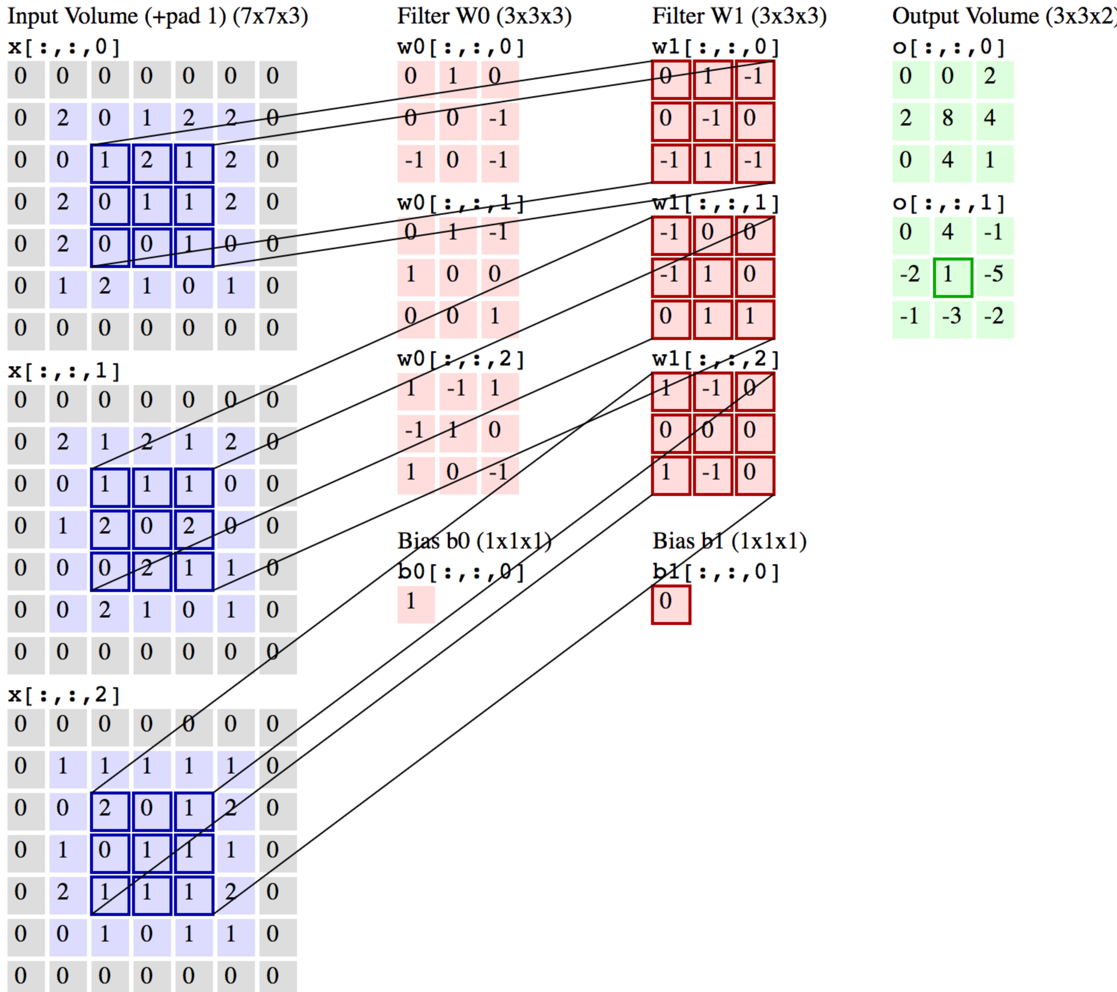


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

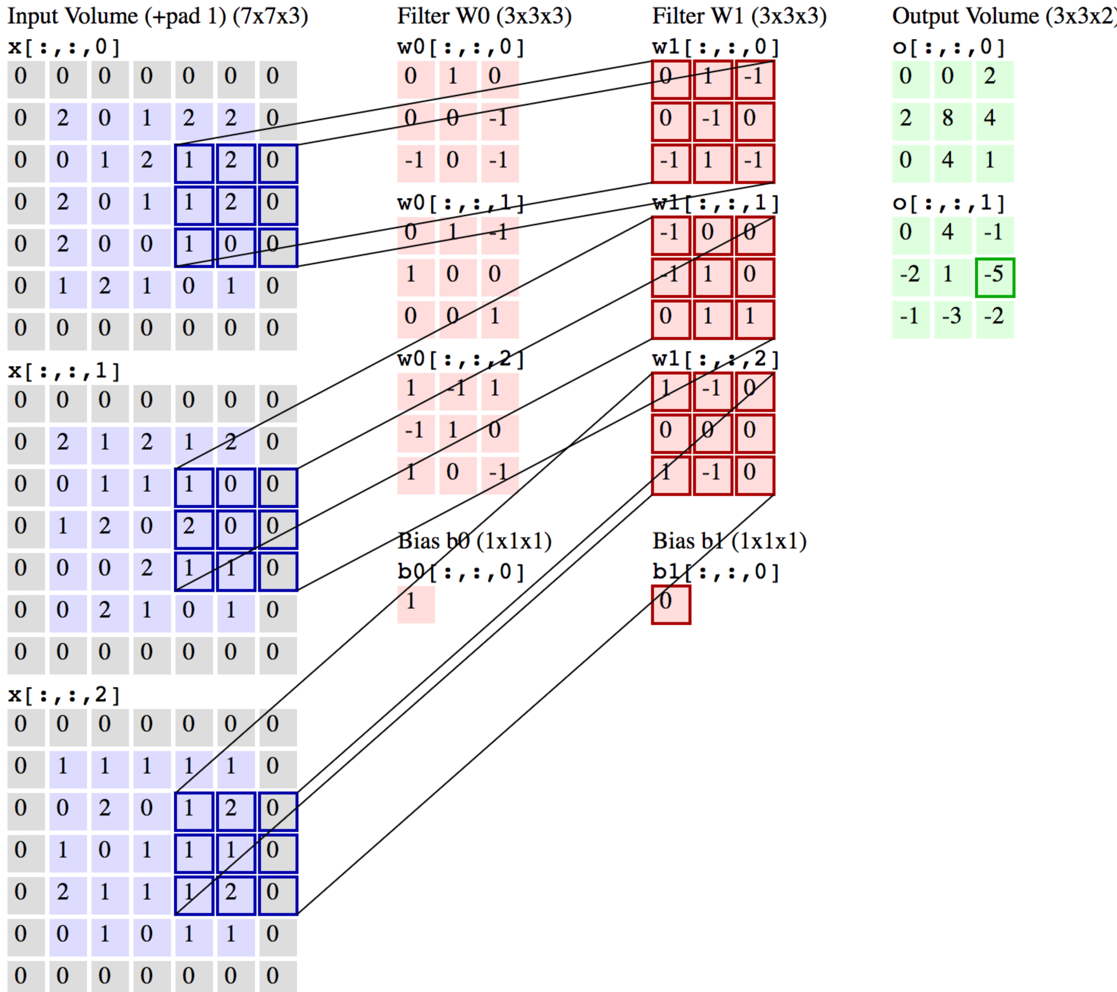


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

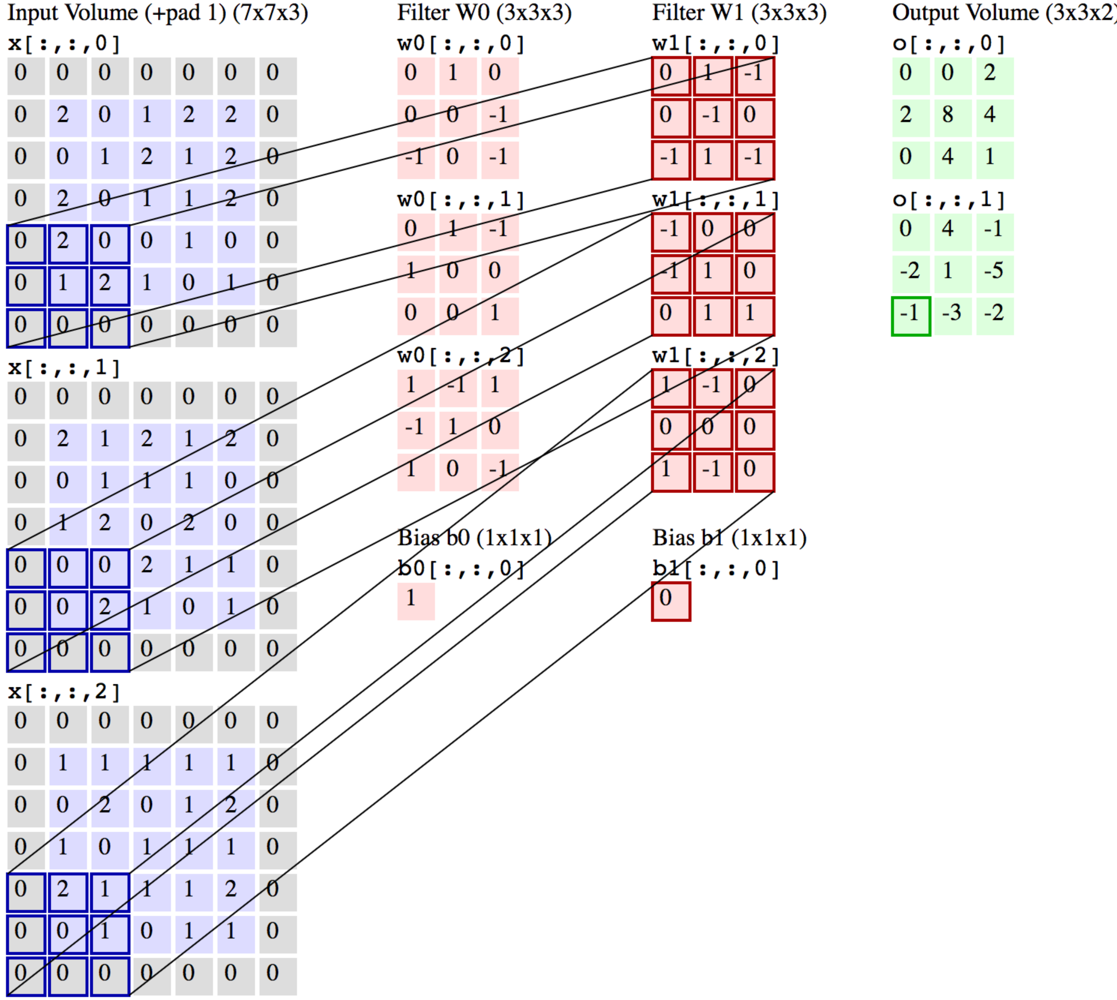


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

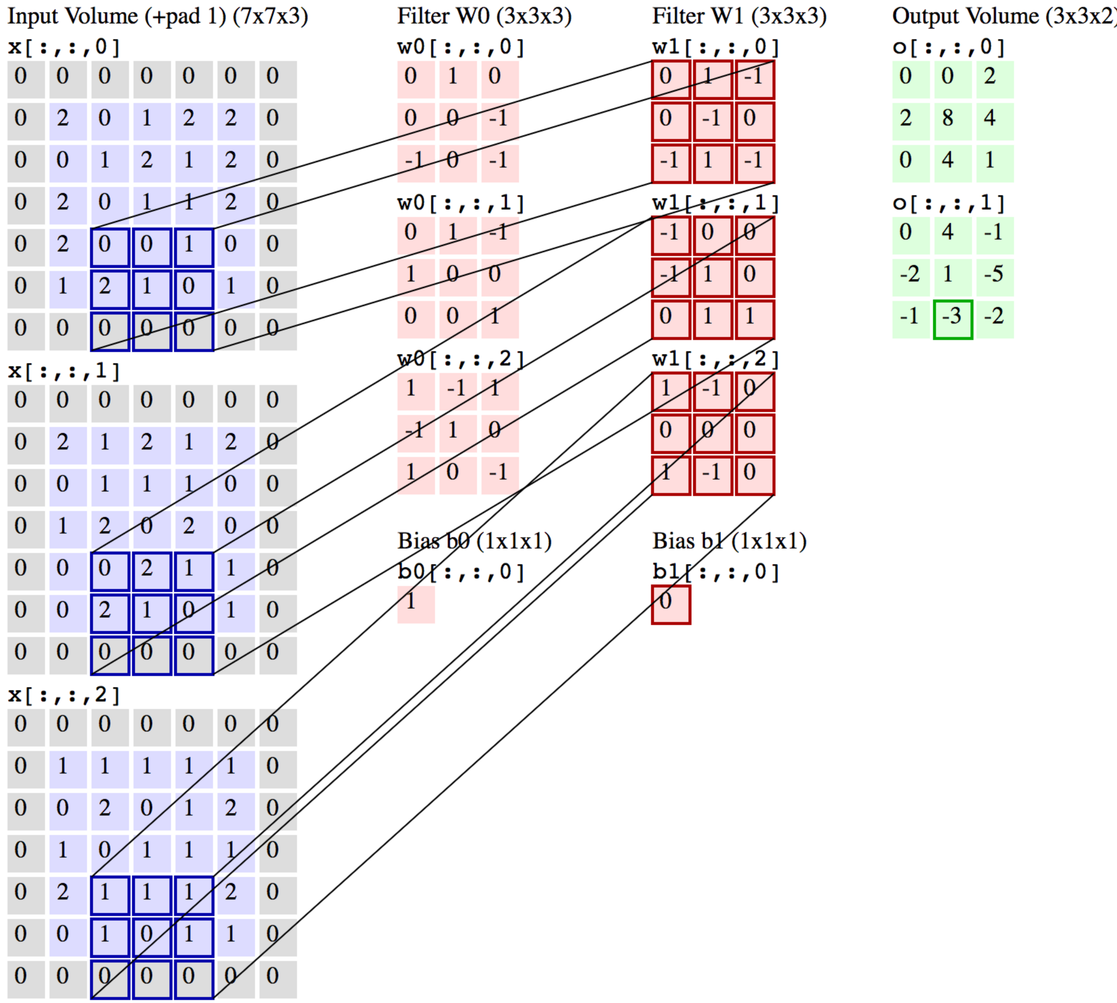


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

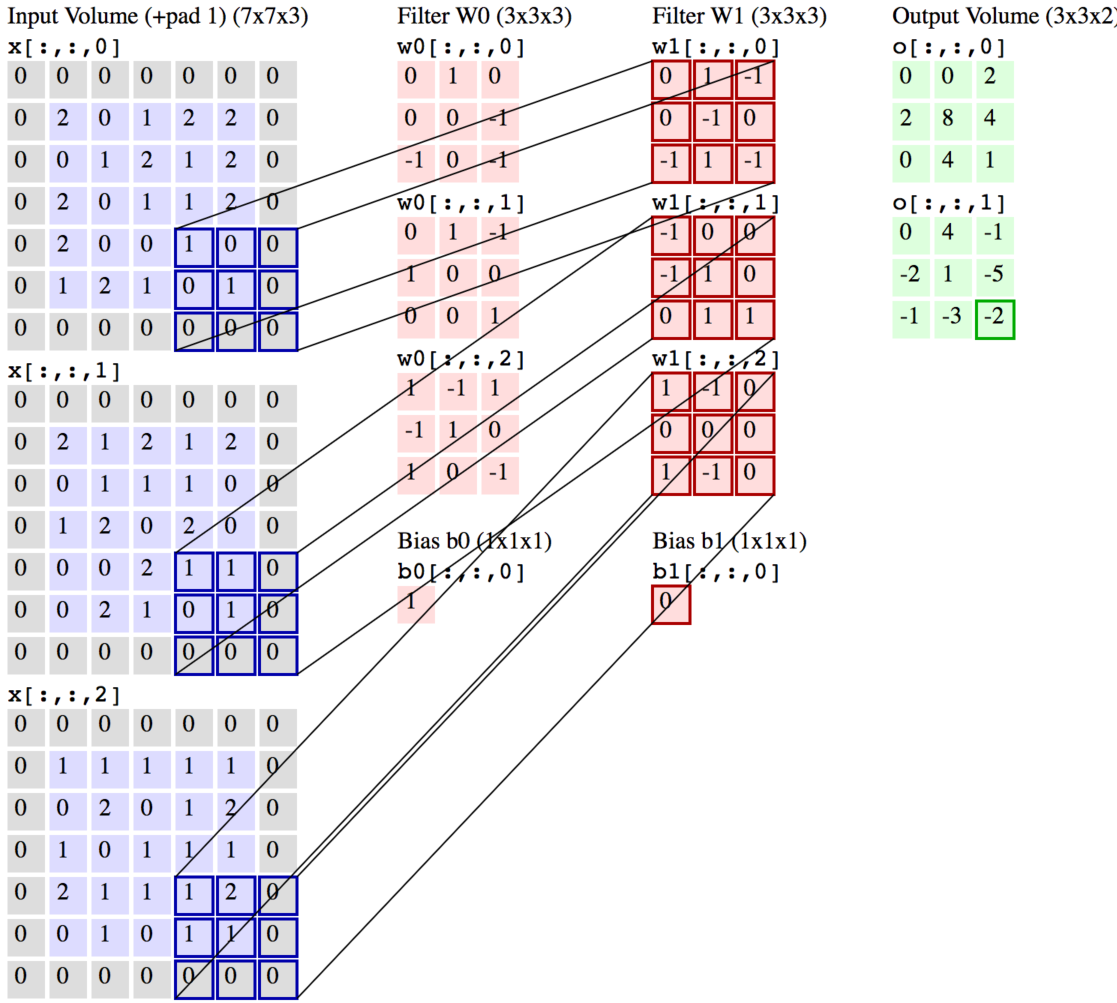
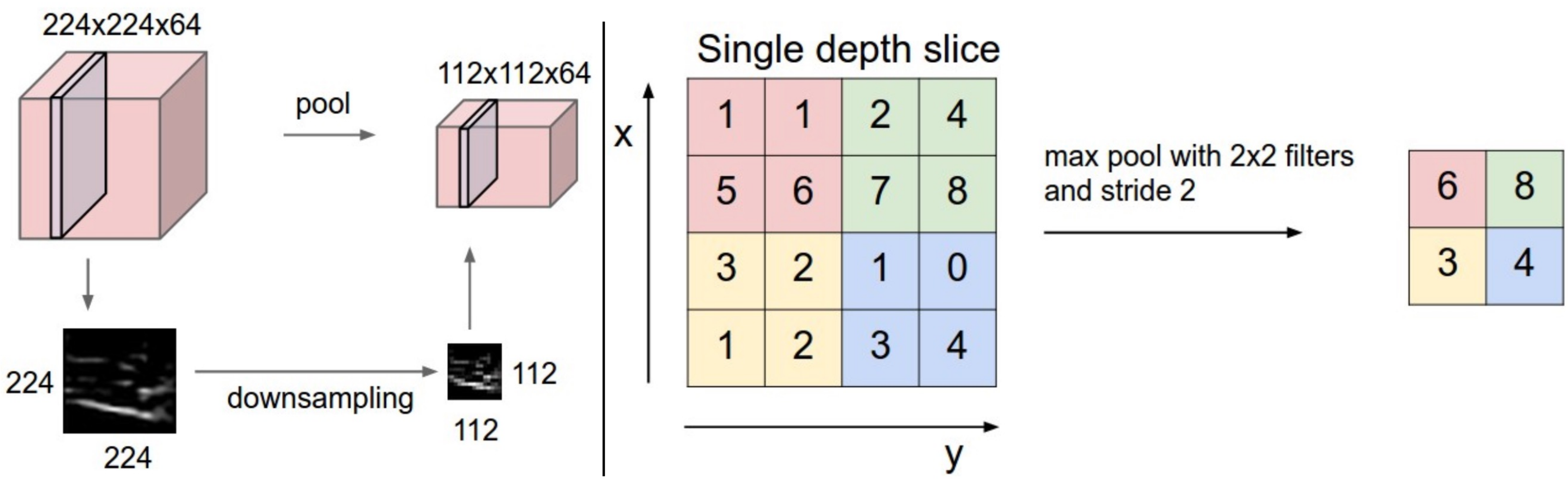


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

Pooling



Pooling layer downsamples the volume spatially, independently in each depth slice of the input volume. **Left:** In this example, the input volume of size [224x224x64] is pooled with filter size 2, stride 2 into output volume of size [112x112x64]. Notice that the volume depth is preserved. **Right:** The most common downsampling operation is max, giving rise to **max pooling**, here shown with a stride of 2. That is, each max is taken over 4 numbers (little 2x2 square).

Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

Handout 18

(a) Which steps require parameter learning? (out of CONV, RELU, POOL, FLATTEN, FC)

(b) First layer params

(c) Second layer params

(d) Third layer params

(e) Total # params

Handout 18

(a) Which steps require parameter learning? (out of CONV, RELU, POOL, FLATTEN, FC)

CONV, FC

(b) First layer params

(c) Second layer params

(d) Third layer params

(e) Total # params

Handout 18

(a) Which steps require parameter learning? (out of CONV, RELU, POOL, FLATTEN, FC)

CONV, FC

(b) First layer params $5*5*3*20 + 20 = 1520$

(c) Second layer params $3*3*20*10 + 10 = 1810$

(d) Third layer params $8*8*10*10 + 10 = 6410$

(e) Total # params 9740

Handout 18

(a) Which steps require parameter learning? (out of CONV, RELU, POOL, FLATTEN, FC)

CONV, FC

(b) First layer params 5*5*3*20 + 20 = 1520

(c) Second layer params 3*3*20*10 + 10 = 1810

(d) Third layer params 8*8*10*10 + 10 = 6410

(e) Total # params 9740

If we had a FC with p_1=100 and p_2=50, we would have 312,860 params to learn (check this after class). CNN is much better!

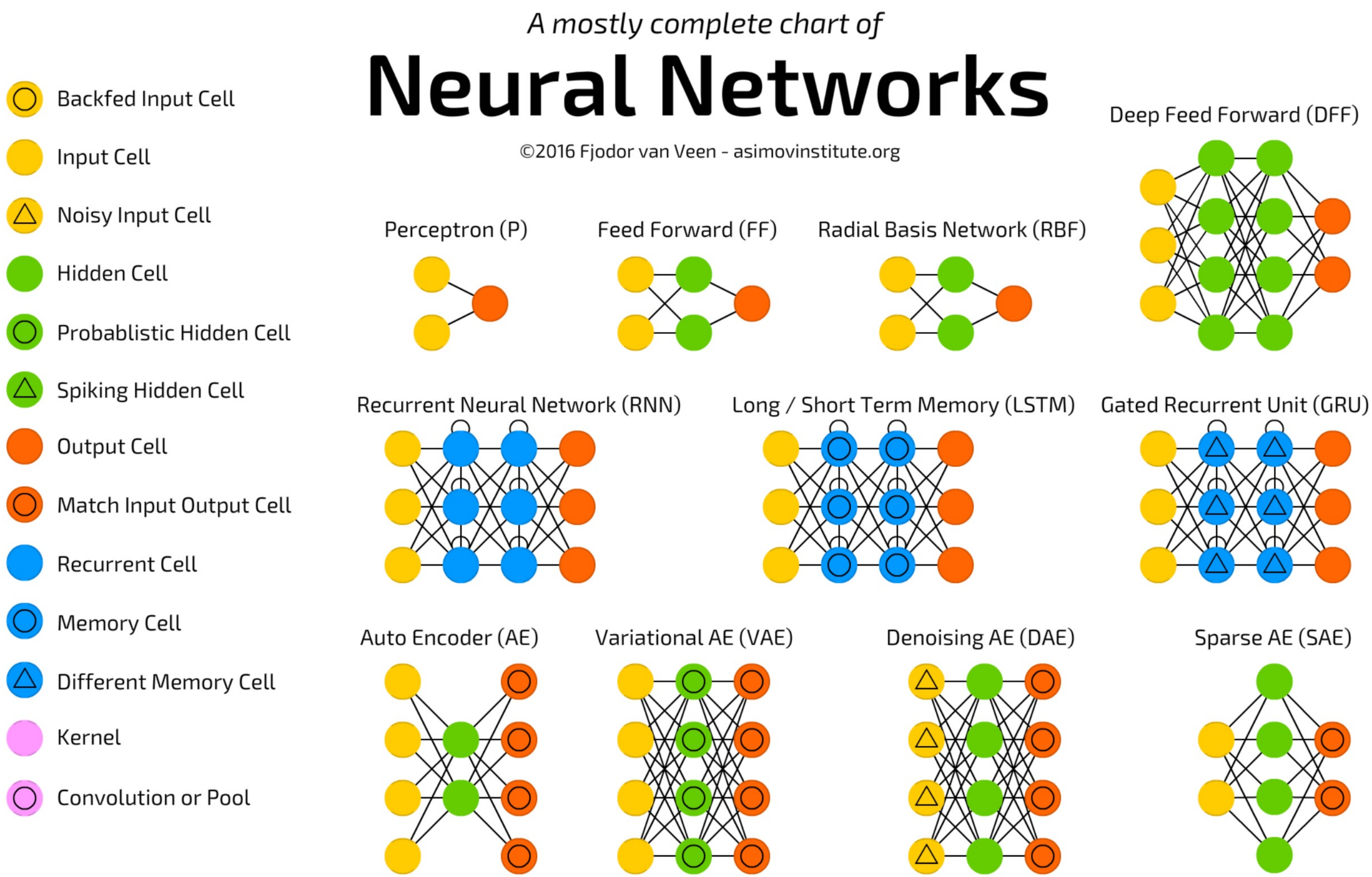


Image from Neural Network Zoo: http://www.asimovinstitute.org/neural-network-zoo/

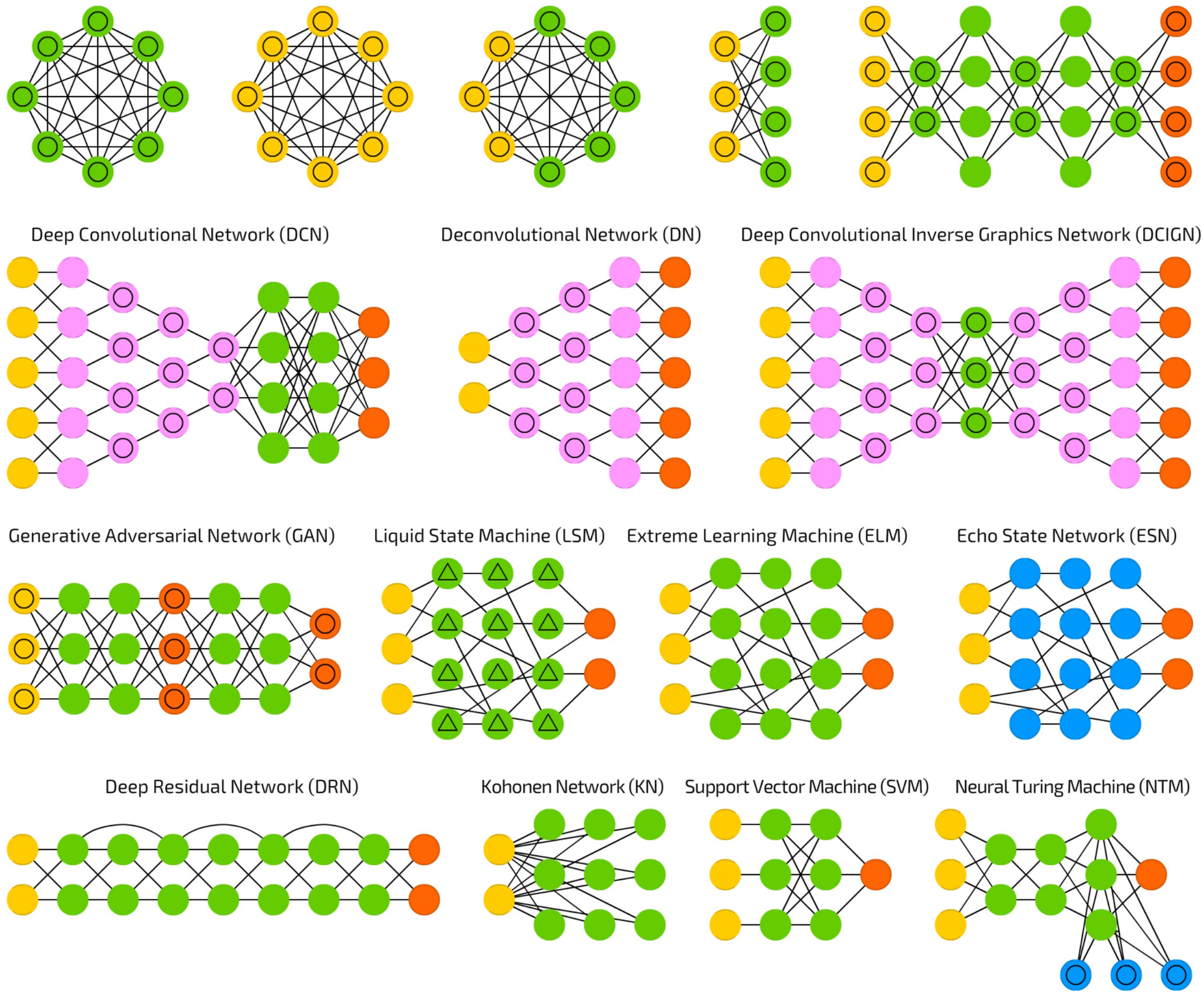


Image from Neural Network Zoo: http://www.asimovinstitute.org/neural-network-zoo/

Training concerns

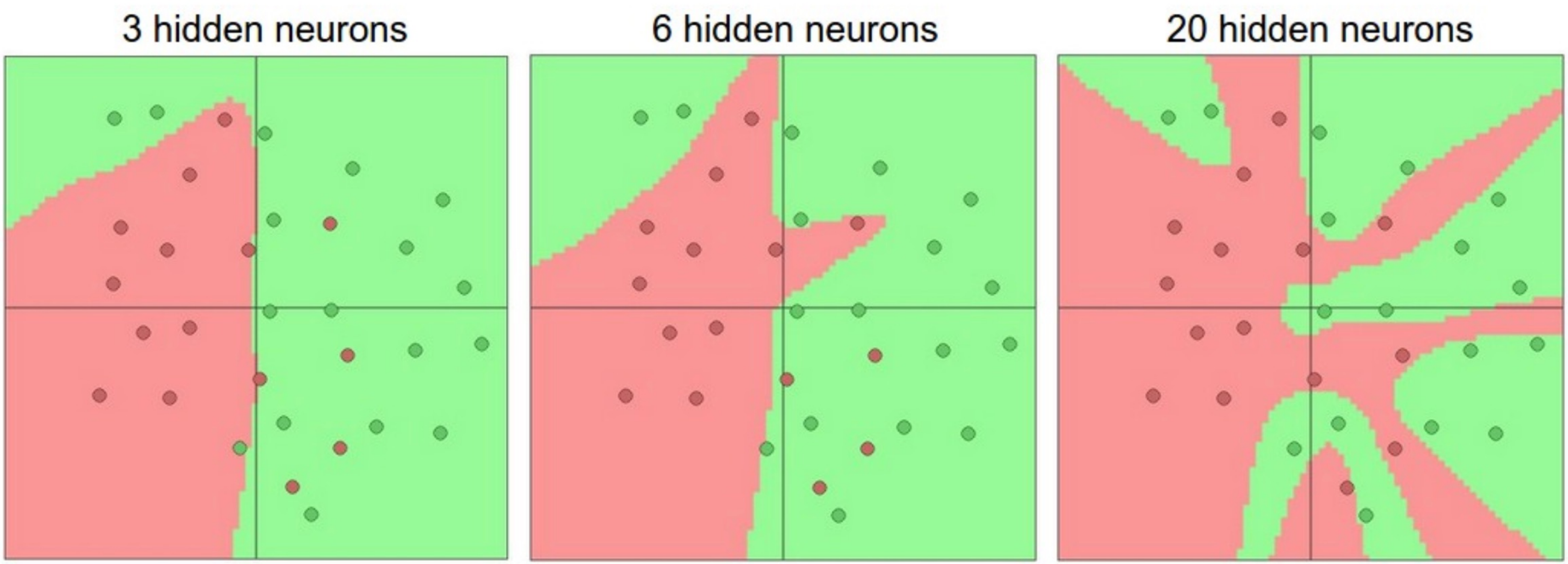
Weight initialization

- We still have to initialize the pre-training

- All 0's initialization is bad! Causes nodes to compute the same outputs, so then the weights go through the same updates during gradient descent

- Need asymmetry! => usually use small random values

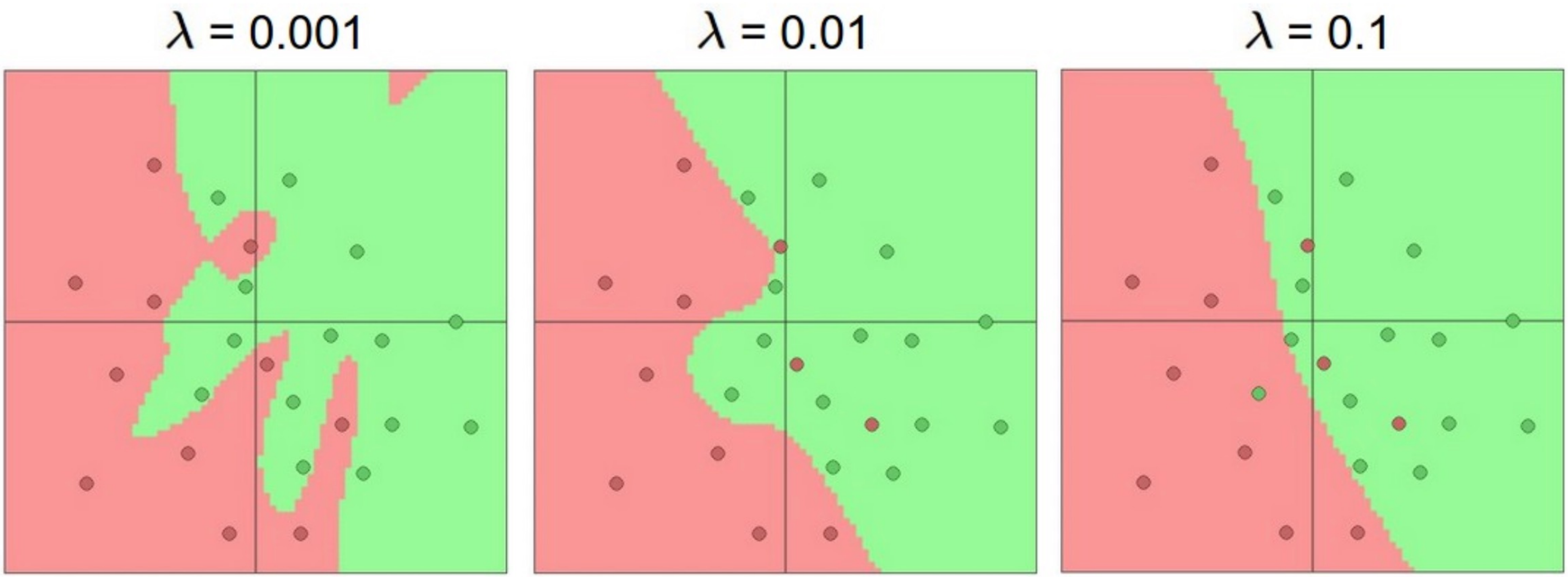
More hidden units can contribute to overfitting



Larger Neural Networks can represent more complicated functions. The data are shown as circles colored by their class, and the decision regions by a trained neural network are shown underneath. You can play with these examples in this ConvNetsJS demo.

Image from: http://cs231n.github.io/neural-networks-1/

However! It is always better to use a more expressive network and regularize in other ways

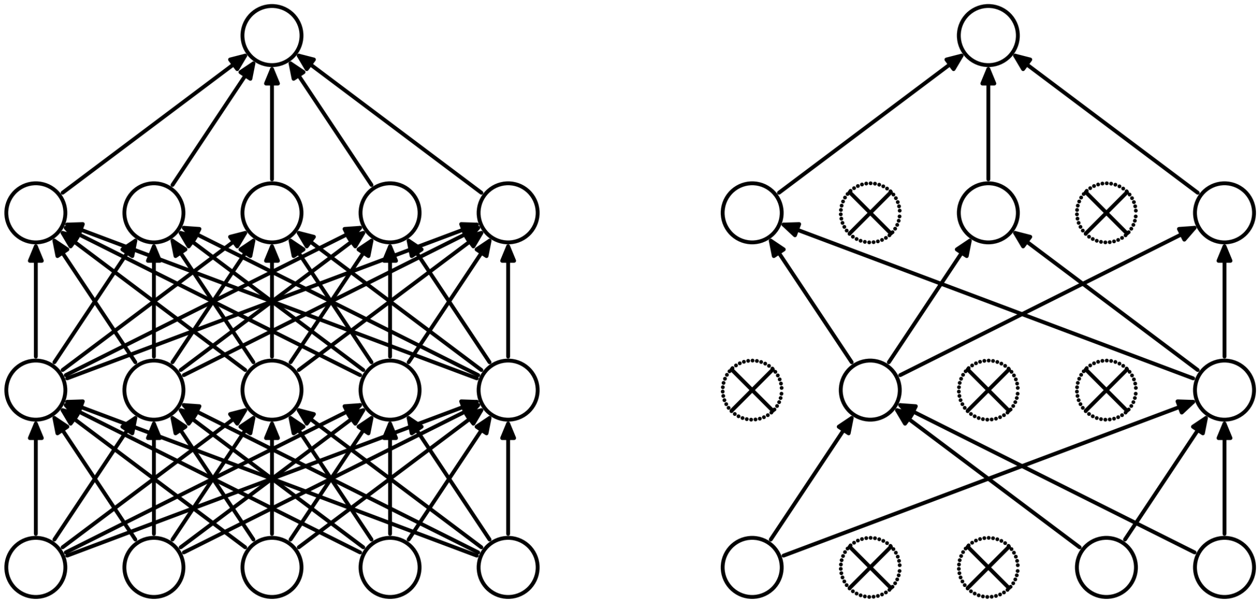


The effects of regularization strength: Each neural network above has 20 hidden neurons, but changing the regularization strength makes its final decision regions smoother with a higher regularization. You can play with these examples in this ConvNetsJS demo.

Image from: http://cs231n.github.io/neural-networks-1/

One regularization approach: dropout

- Idea: keep a neuron active with some probability p, otherwise, do not send its output forward to the next layer



(a) Standard Neural Net (b) After applying dropout.

Image and more information: “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”
http://www.cs.toronto.edu/~rsalakhu/papers/srivastava14a.pdf