CS 360: Machine Learning

Prof. Sara Mathieson

Fall 2019

HAVERFORD COLLEGE

Admin

- Office hours **Friday 3-4:30pm** (in lab)
- Project proposal was due last night
  - repos and feedback coming soon
- **Lab 8** due Sunday Nov 17
  - check in during lab today (Part 1 complete)
- **Midterm 2**: Nov 21 (in-class)
  - pick up a study guide
  - take home due Tues Nov 26
- Nov 28-29: Thanksgiving break!

Project Lab Notebook

- As you as you receive your git repo, start creating a “lab notebook” in your README
- This should say who was working, what date, how long, and briefly what you did

Sara: 03-07-18 (2hrs)

- now averaging the Markov chain, fixed all the results
- combined ancestral 1000 genomes still running (need to start similar for SGDP)
- started new runs with filtering to only have selected alleles in the “selected pop” and only have ancestral alleles in the “reference panel”

Project Deliverables

- Main deliverable: presentation

  - Group of 1: 5 min
  - Group of 2: 9 min
  - Group of 3: 12 min

- On git:
  - Lab Notebook
  - Project Code
  - Presentation Slides

Mid-semester feedback

- Handouts earlier (more time to go over, more check-ins with groups)
- TA hours are not always convenient
- Opportunity to revise mistakes

Mid-semester feedback

- Make sure others are not confused during worksheets
- Ask and answer more on Piazza
- More participation and questions in class
- Start labs earlier, go to office hours more
- Do extensions and readings

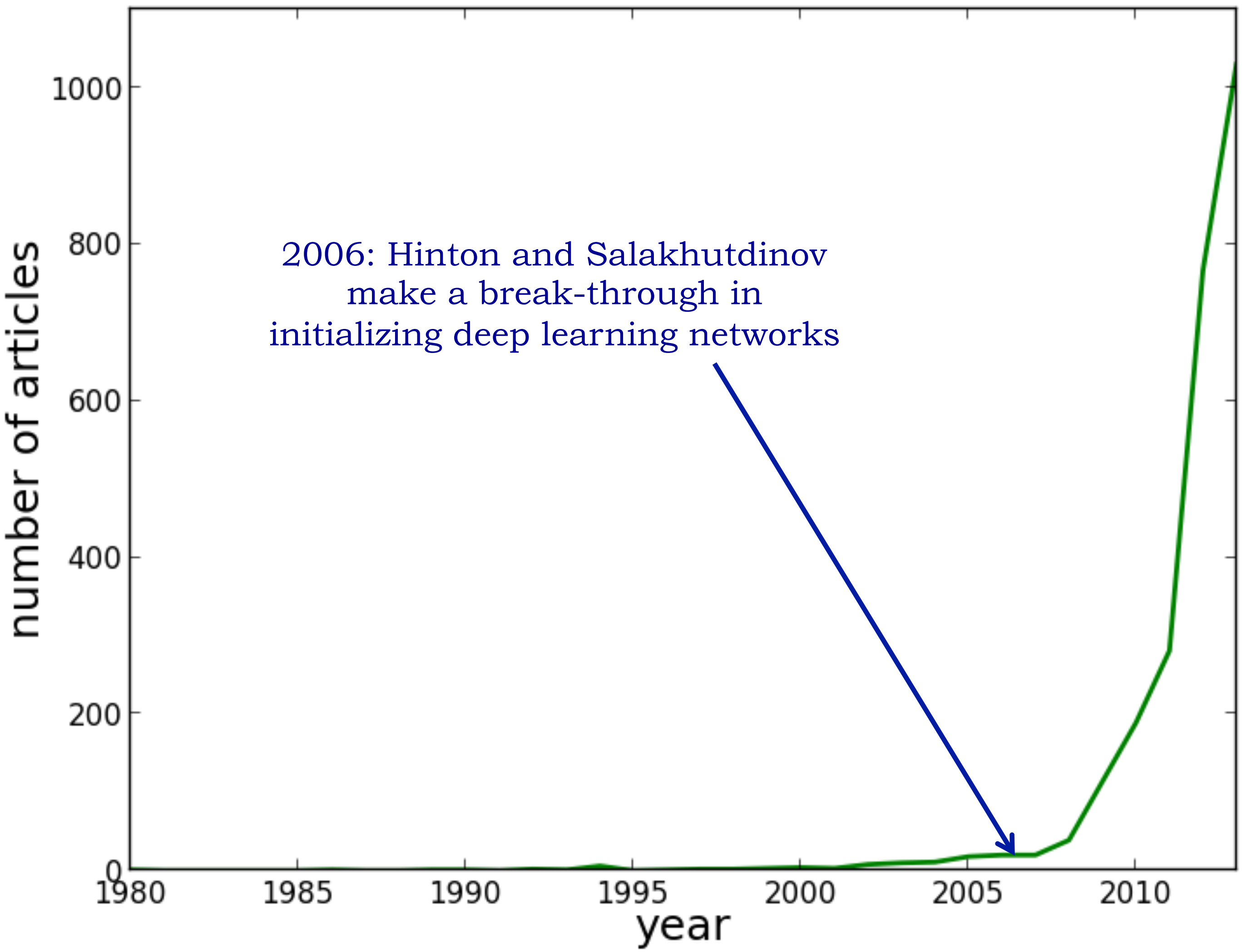
Outline for November 14

- Finish fully connected neural networks
- Convolutional neural networks
- (if time) Backpropagation
- Next week: review

Outline for November 14

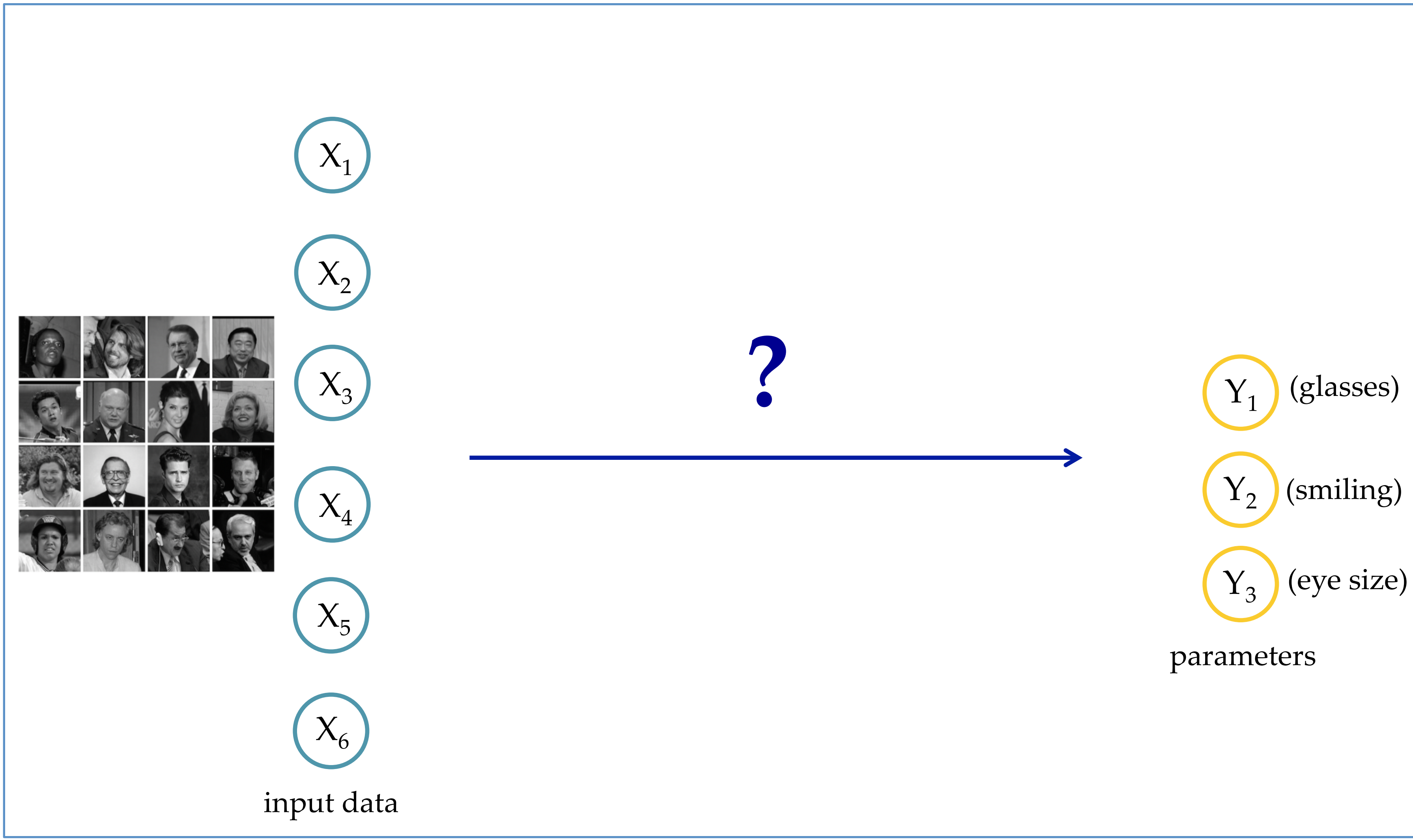
- Finish fully connected neural networks
- Convolutional neural networks
- (if time) Backpropagation

What was this breakthrough in deep learning?

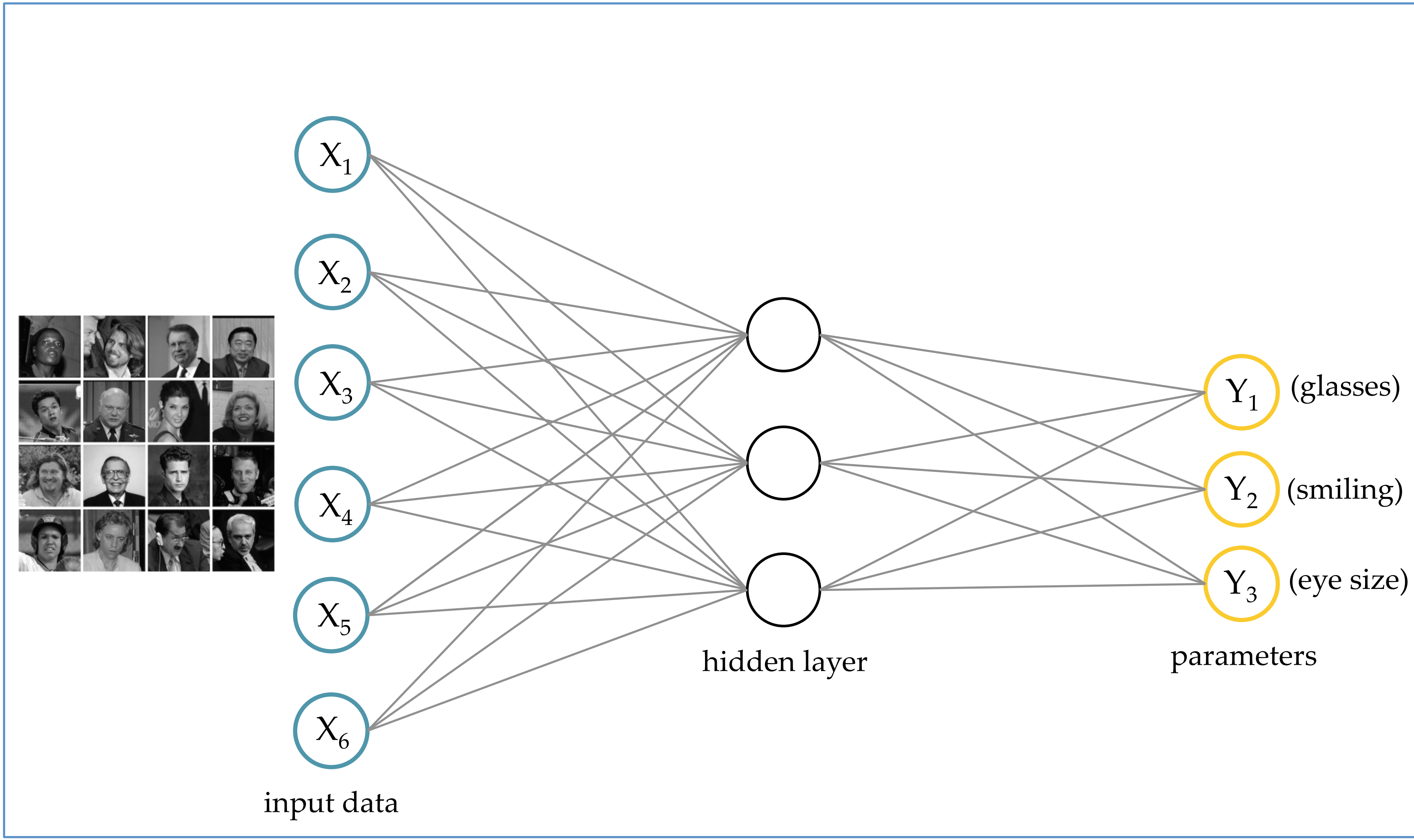


Images on next slides from
Hinton & Salakhutdinov (2006)

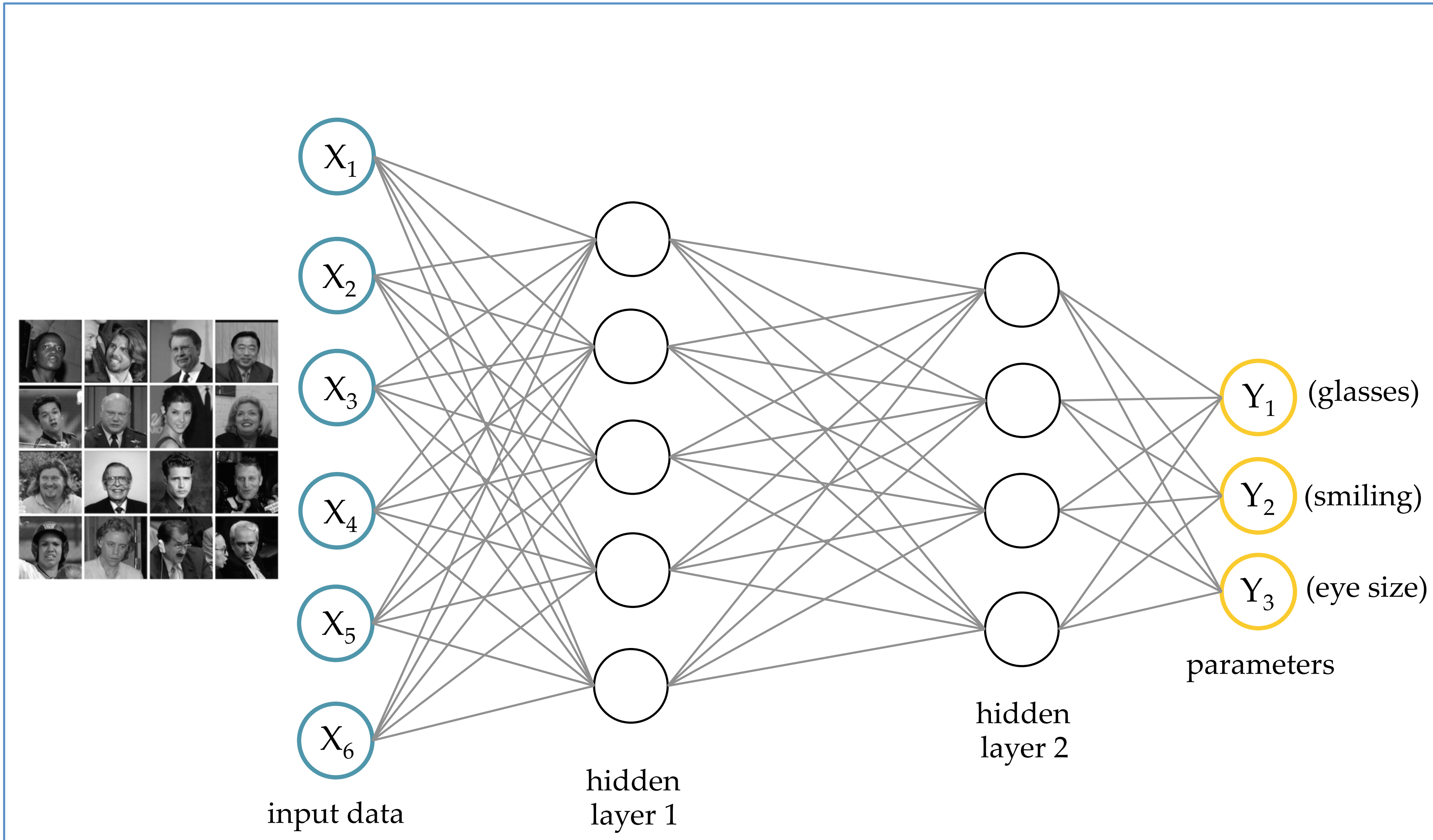
Goal: find a function between input and output



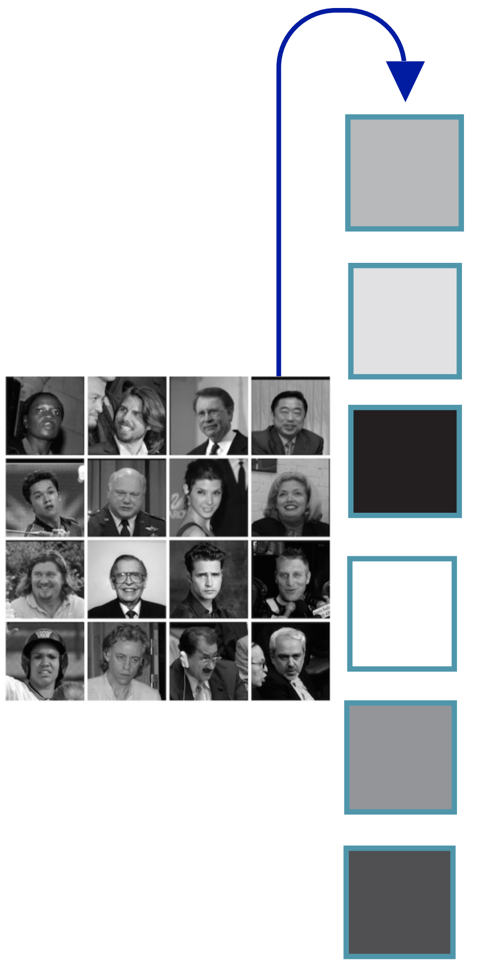
First idea: one hidden layer



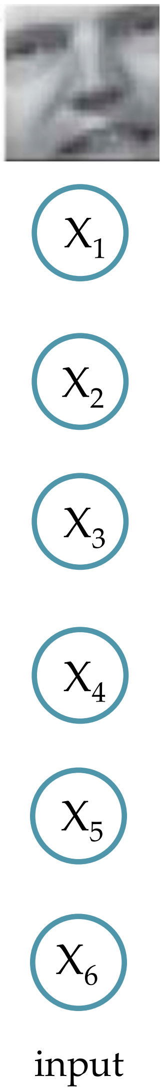
Second idea: more hidden layers ("deep" learning)



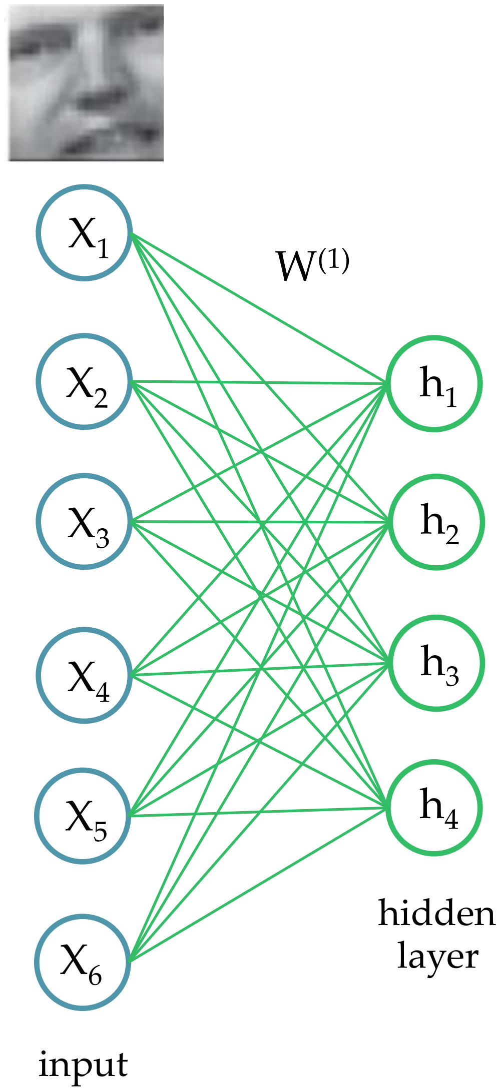
Flatten pixels of image into a single vector



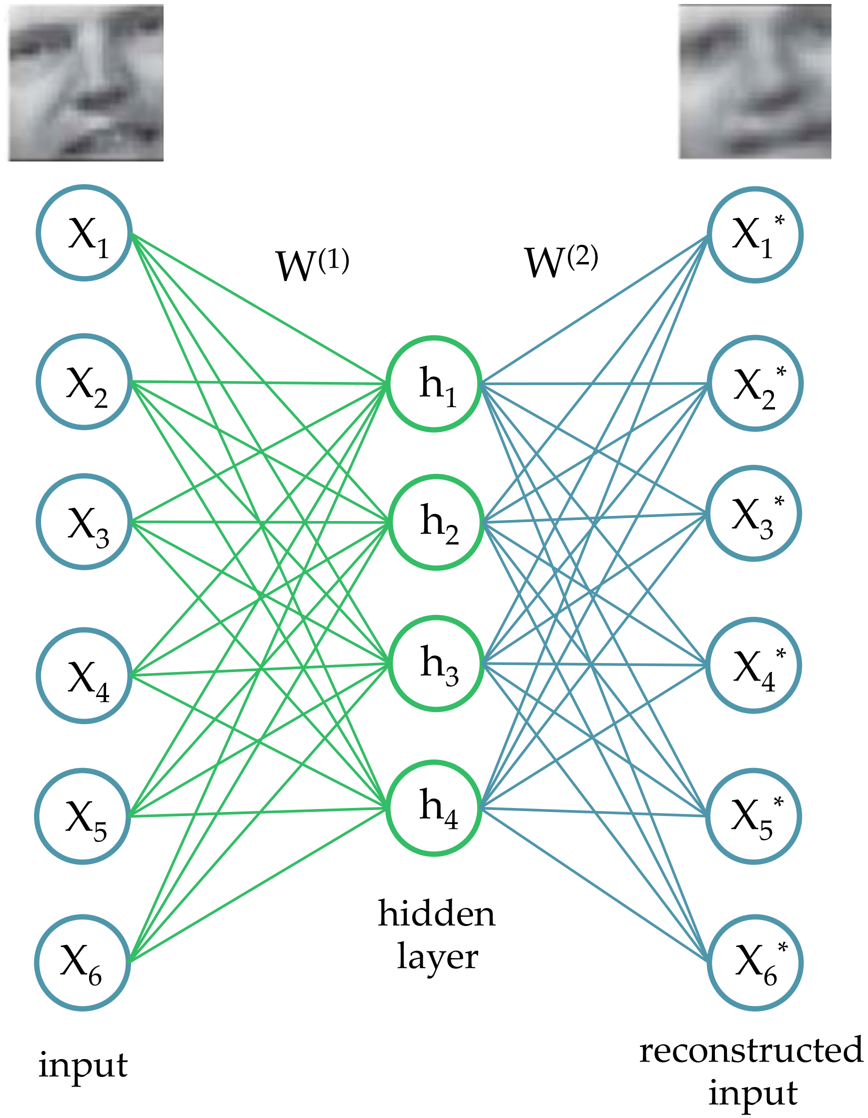
Detour to autoencoders



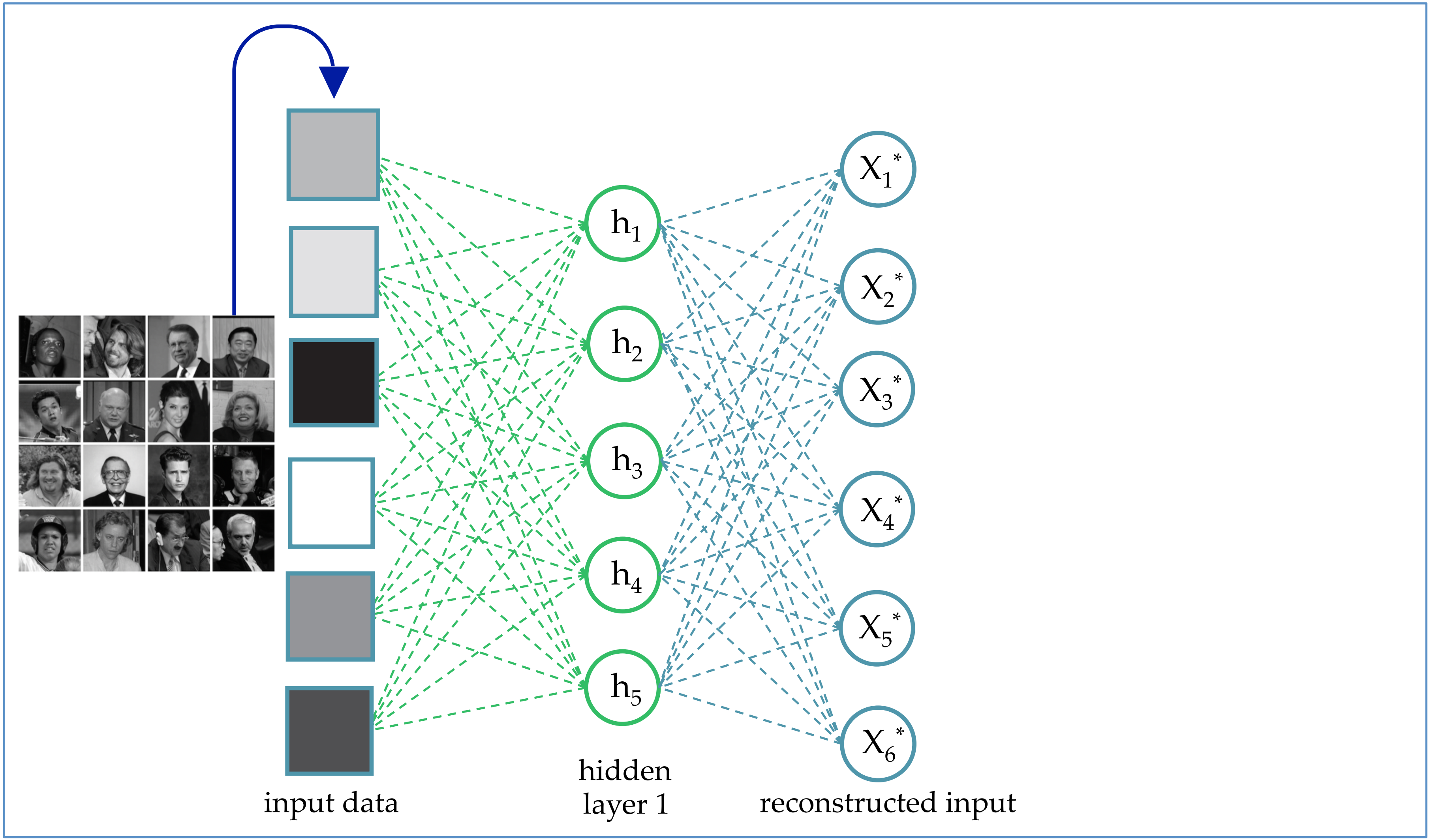
Detour to autoencoders



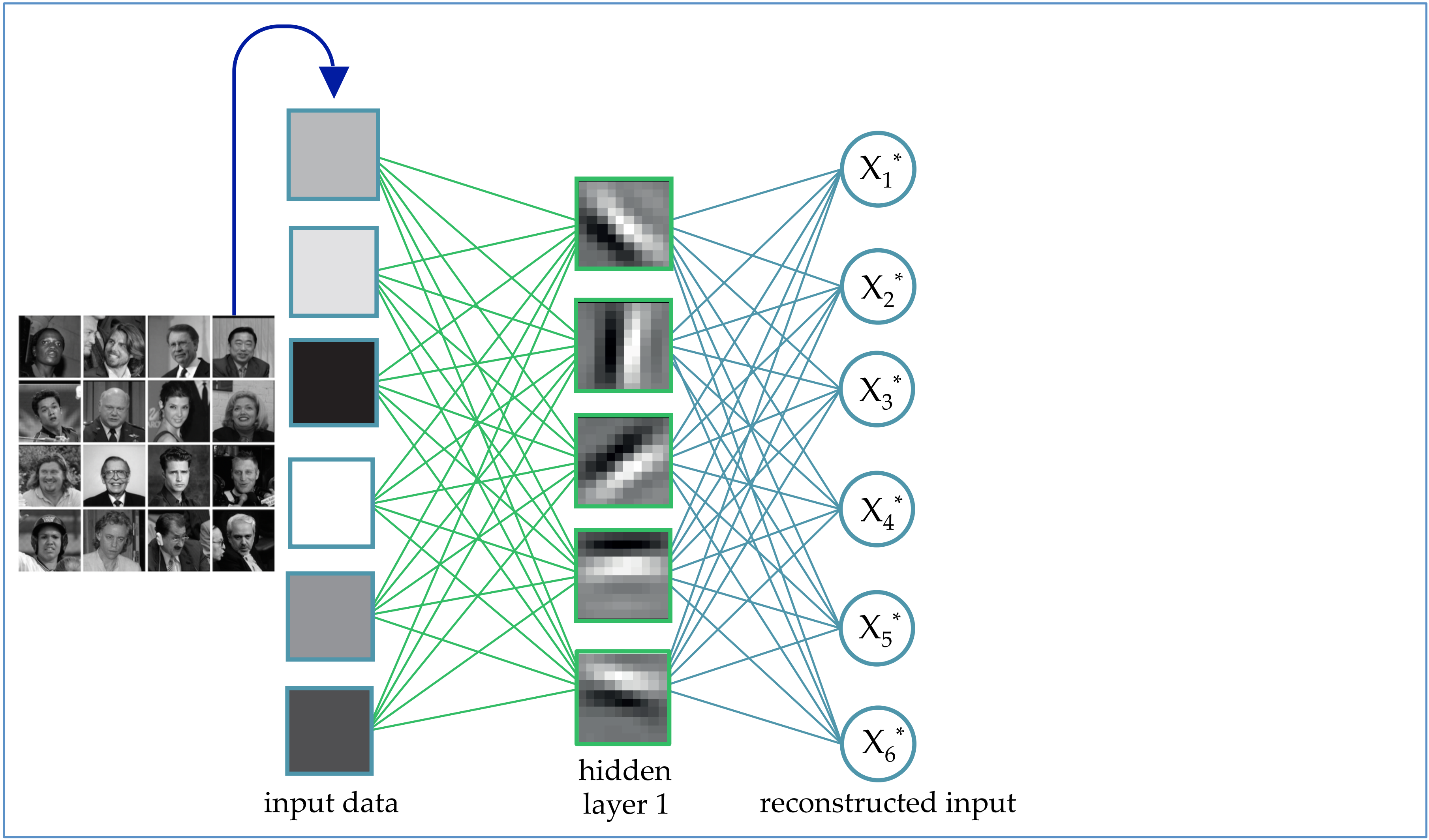
Detour to autoencoders



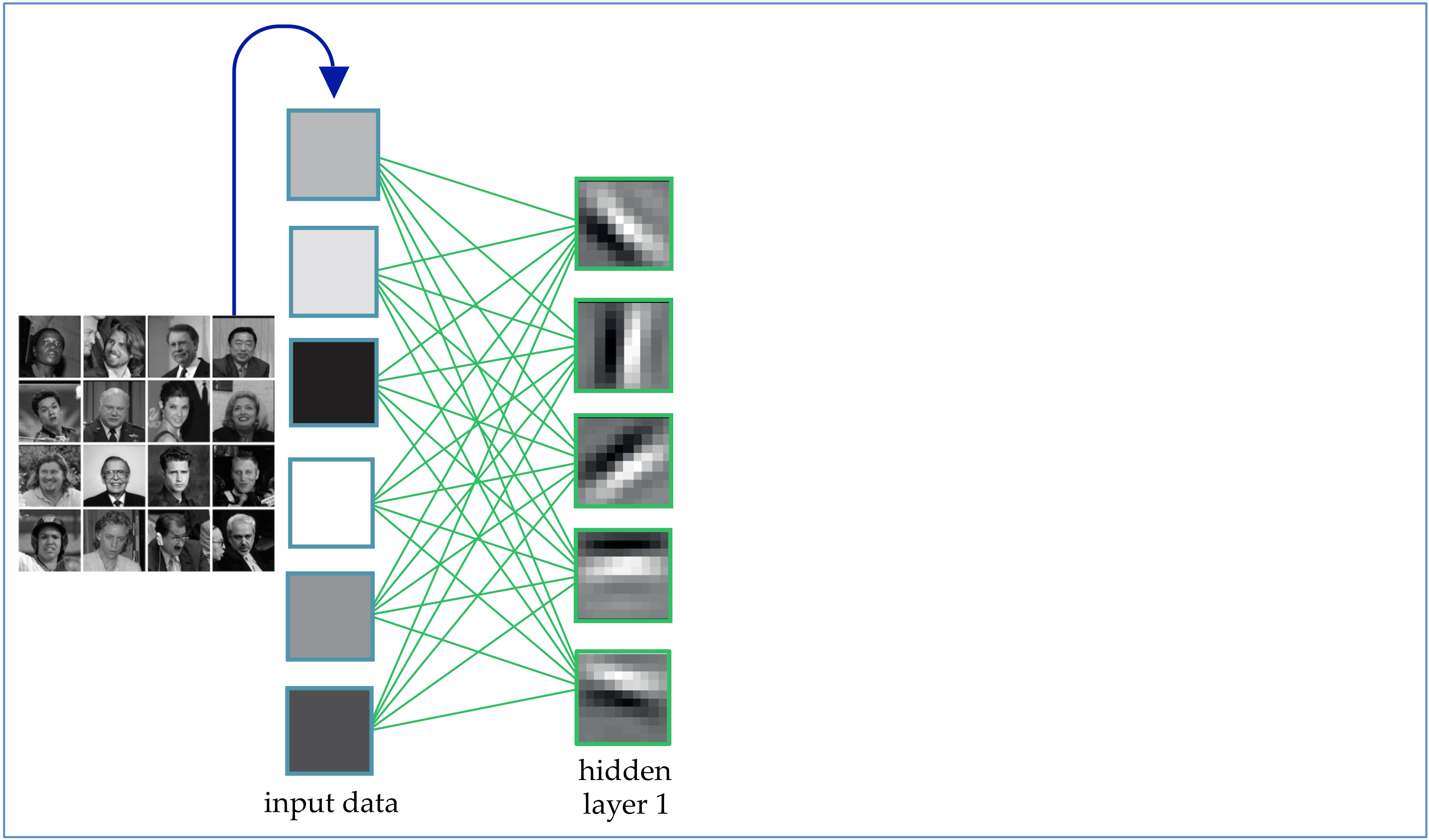
Use unsupervised pre-training to find a function from the input to itself



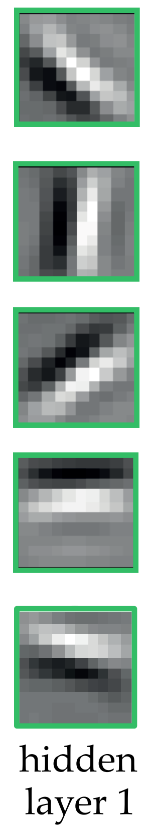
Hidden units can be interpreted as edges



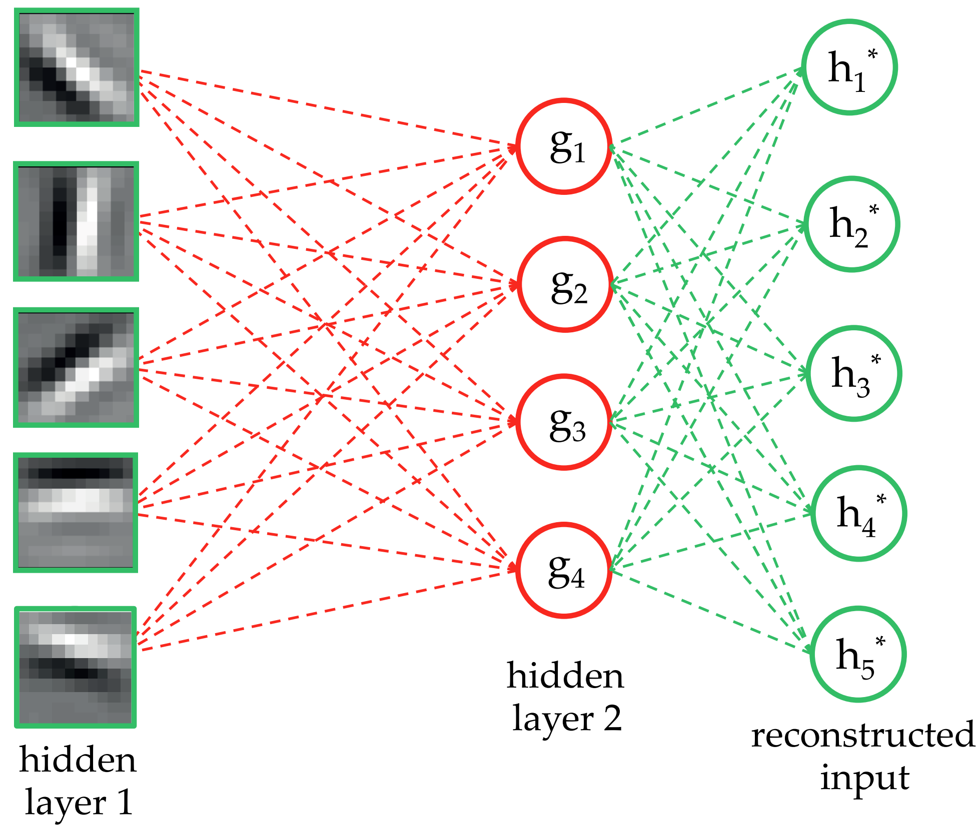
Now: throw away reconstruction and input



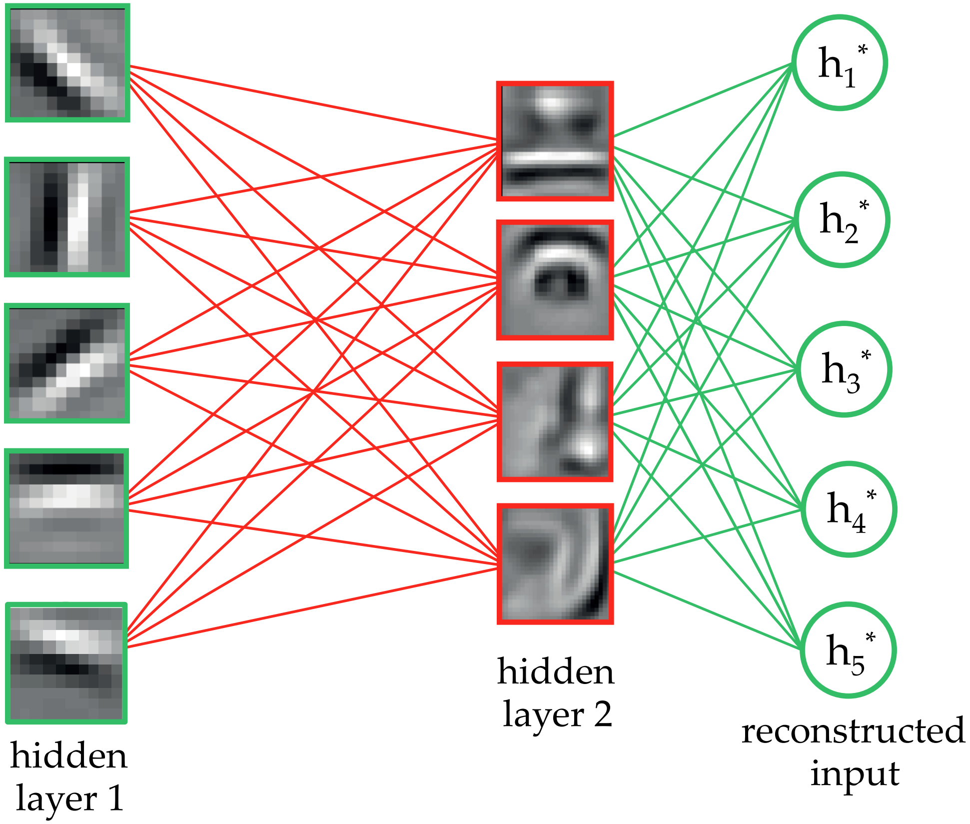
Now: throw away reconstruction and input



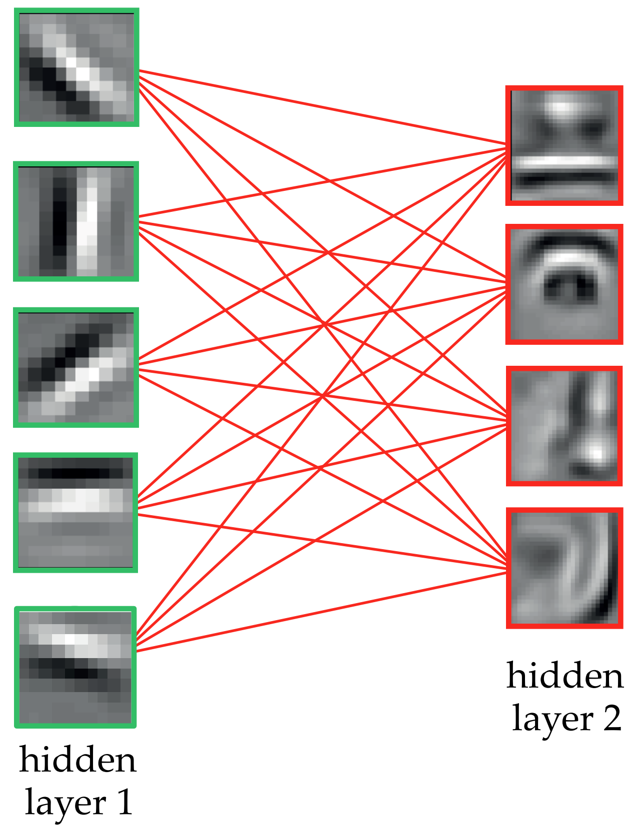
Then repeat the entire process for each layer



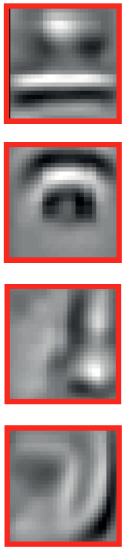
Then repeat the entire process for each layer



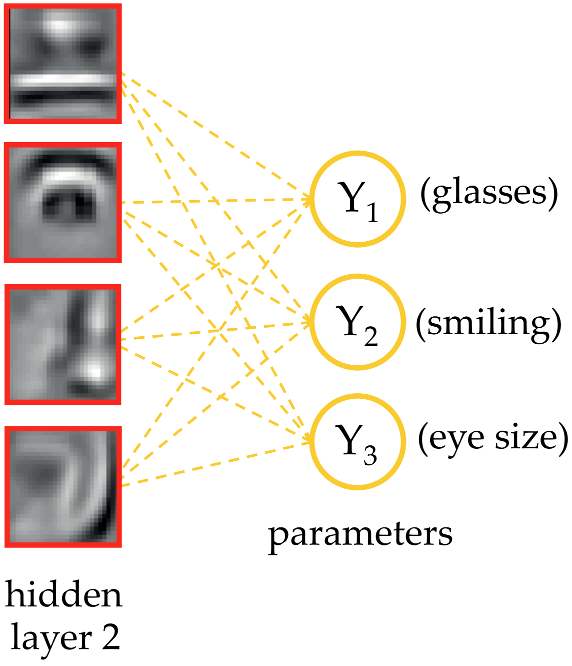
Then repeat the entire process for each layer



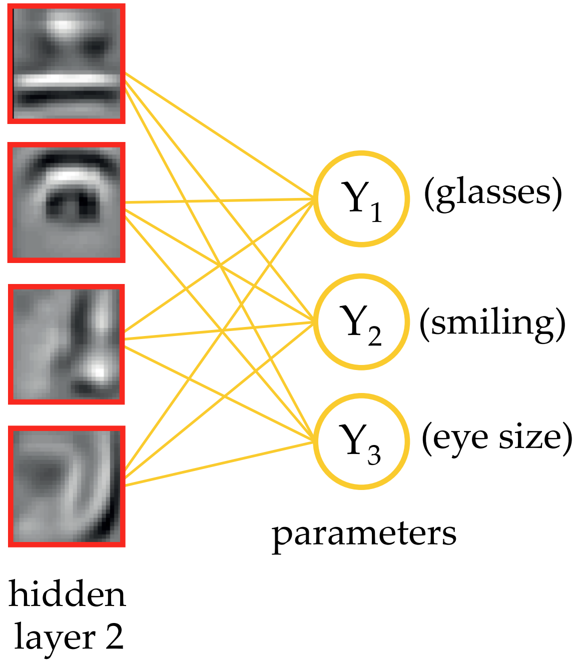
Then repeat the entire process for each layer



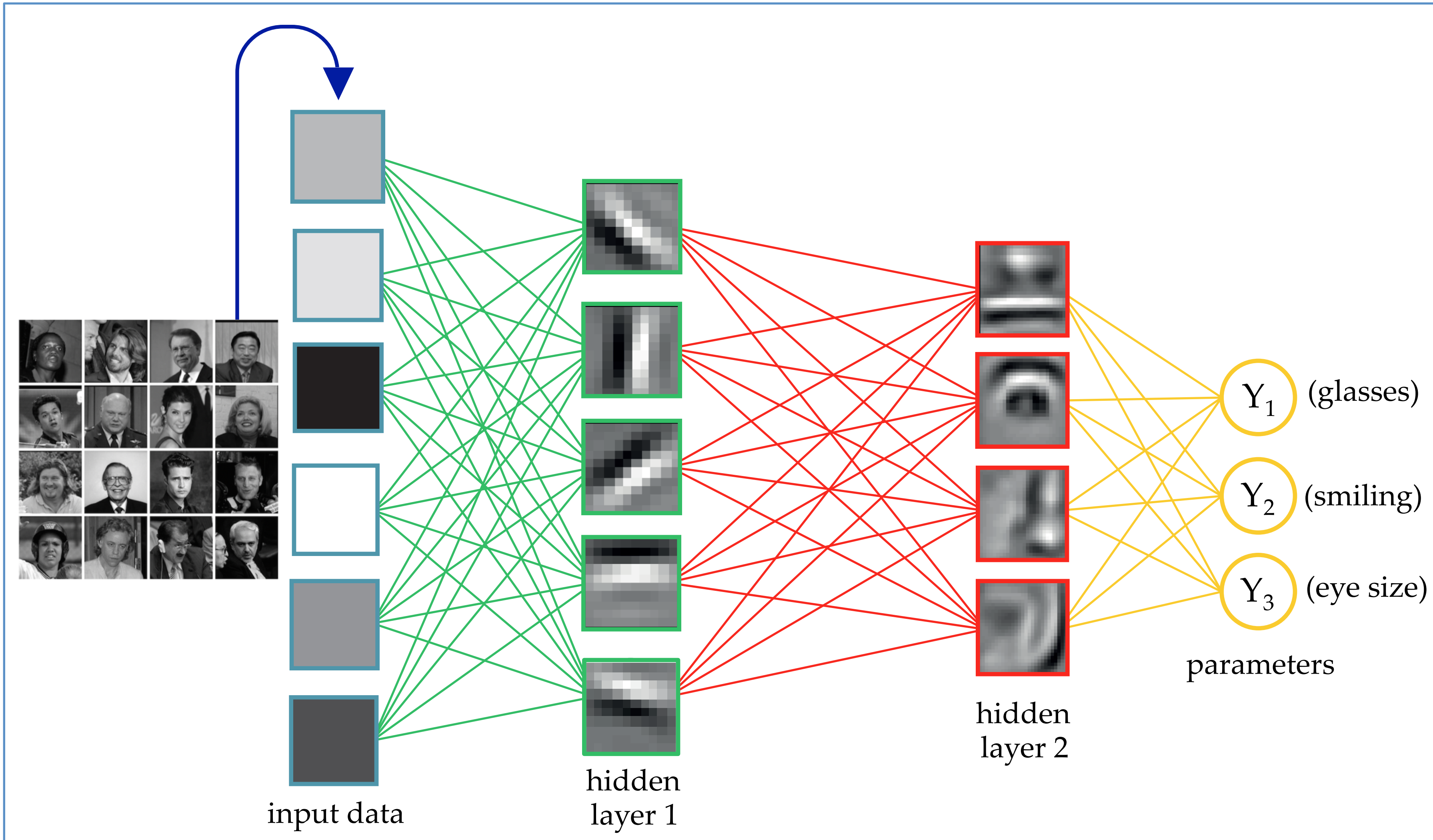
In the last layer, use the outputs (supervised)



In the last layer, use the outputs (supervised)

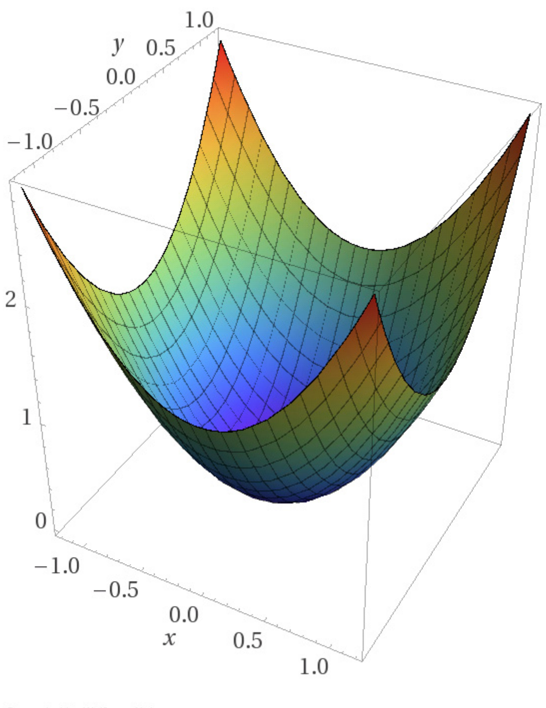


Finally, "fine-tune" the entire network!

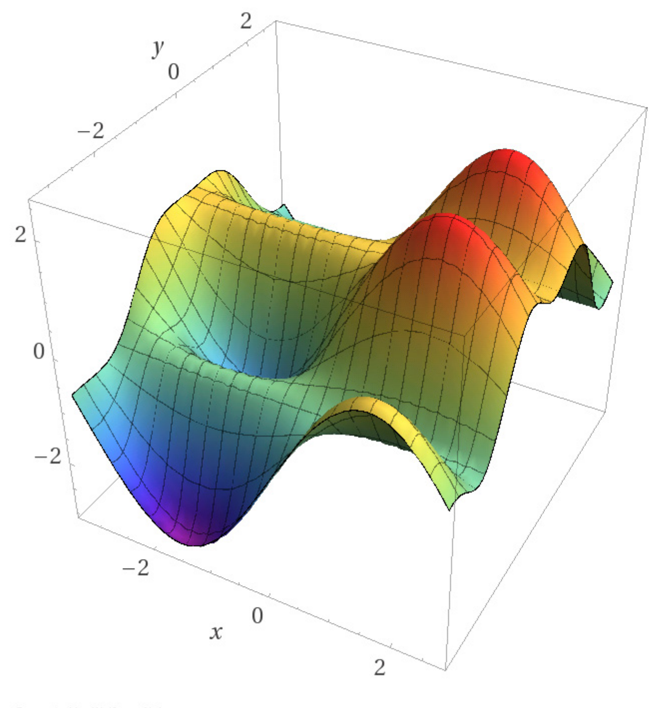


Takeaways

- 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

Takeaways

- 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)

Takeaways

- 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)

Recent Example: OpenAI's GPT-2

- "Language Models are Unsupervised Multitask Learners"
  https://d4mucfpksywv.cloudfront.net/better-language-models/language-models.pdf

- Decision not to release full model: https://openai.com/blog/better-language-models/

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

Mini-batches

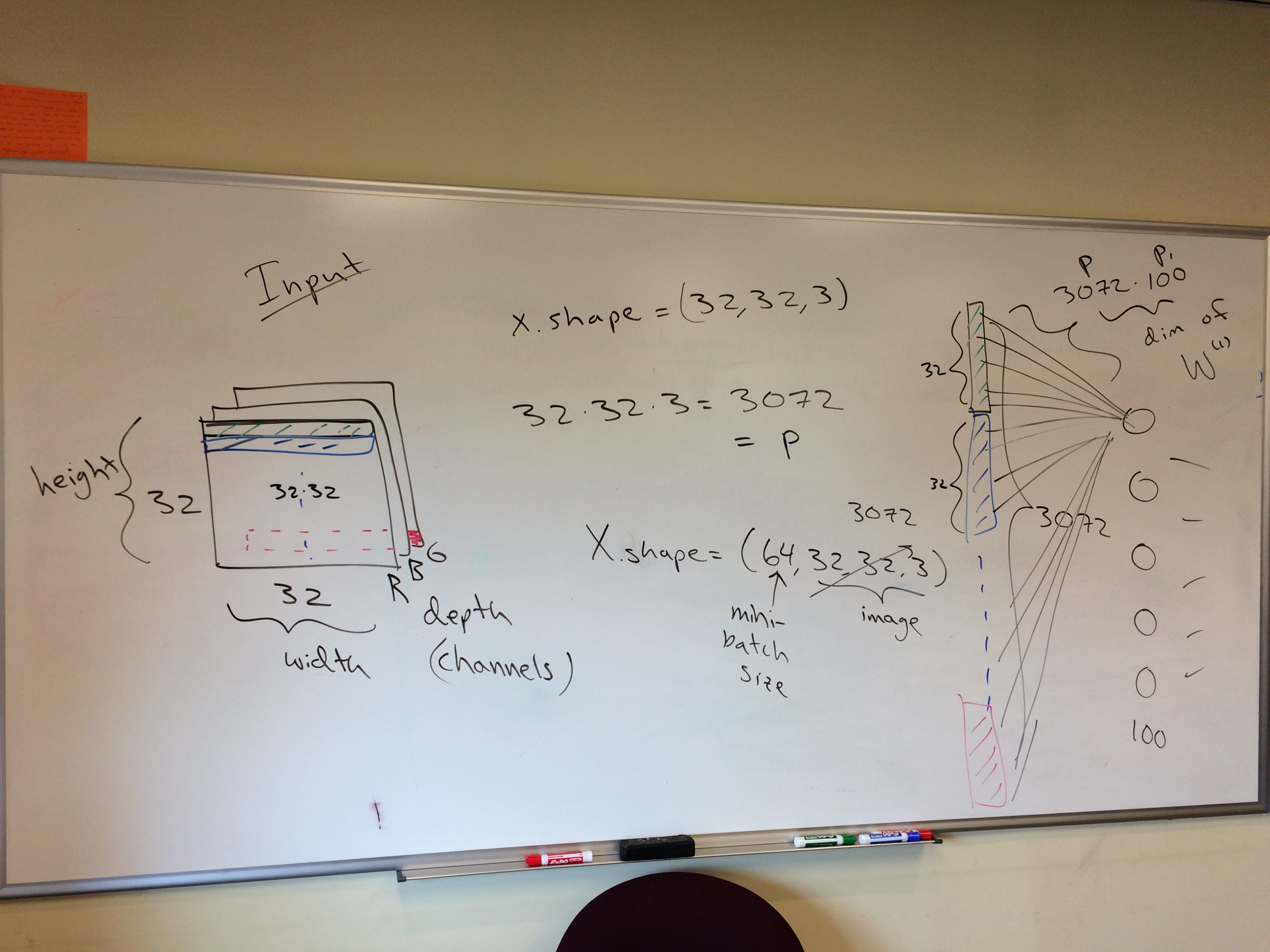
- So far in this class, we have considered *stochastic gradient descent*, where one data point is used to compute the gradient and update the weights
- On the flipside is *batch gradient descent*, where we compute the gradient with respect to all the data, and then update the weights
- A middle ground uses *mini-batches* of examples before updating the weights. This is the approach we will use in Lab 8.

Lab 8 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 8 data pre-processing

- Input is now itself a multi-dimensional array
- 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


Input
X.shape = (32,32,3)
32·32·3 = 3072
= p
height
32
32·32
32
width
R
B
G
depth
(channels)
3072
X.shape = (64, 32, 32, 3)
mini-batch size
image
p
p,
3072 · 100
dim of
W^(1)
32
32
3072
100

Notes about scores and softmax

- The output of the final fully connected layer is a vector of length *K* (number of classes)

- The raw scores are transformed into probabilities using the *softmax function*: (let s_k be the score for class *k*)

$$\hat{y}_k = \frac{e^{s_k}}{\sum_{j=1}^{K} e^{s_j}}$$

- Then we apply *cross-entropy loss* to these probabilities

Notes about scores and softmax

- The output of the final fully connected layer is a vector of length *K* (number of classes)

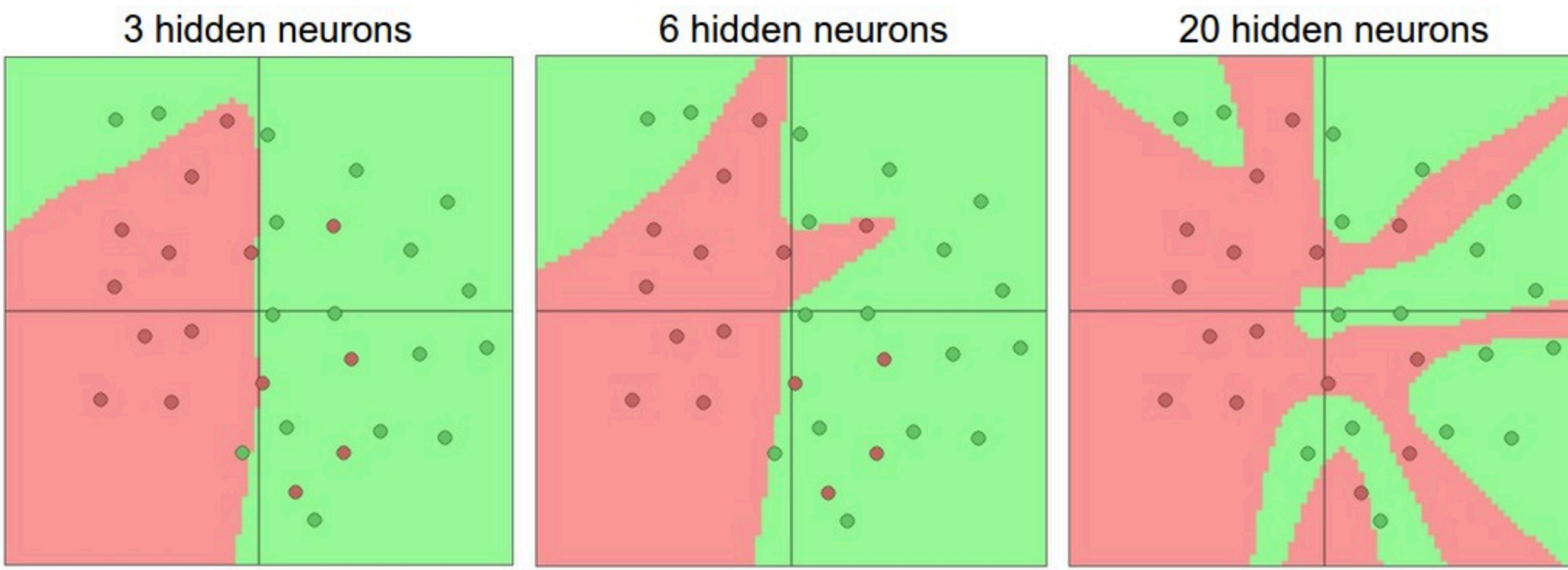
- The raw scores are transformed into probabilities using the *softmax function*: (let s_k be the score for class *k*)

$$\hat{y}_k = \frac{e^{s_k}}{\sum_{j=1}^{K} e^{s_j}}$$

Think about outside of class:
- Why do we use exp?
- Why don't we just take the max score?

- Then we apply *cross-entropy loss* to these probabilities

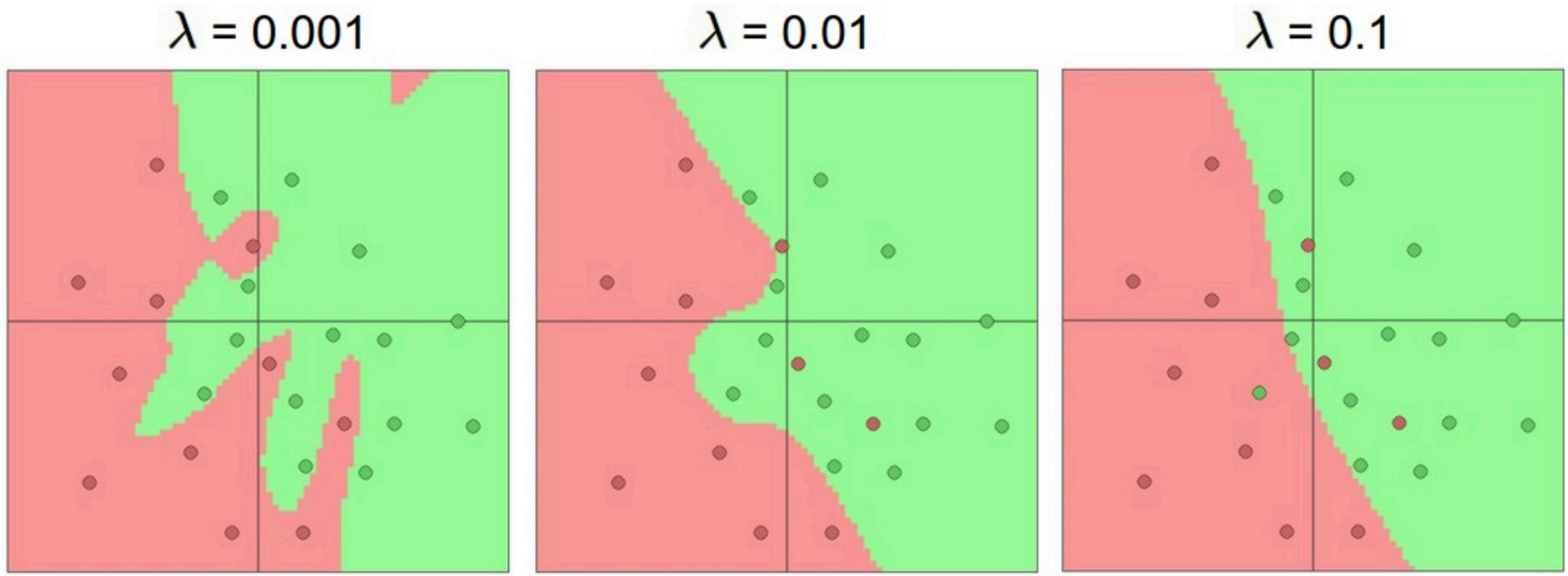
More hidden units can contribute to overfitting



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

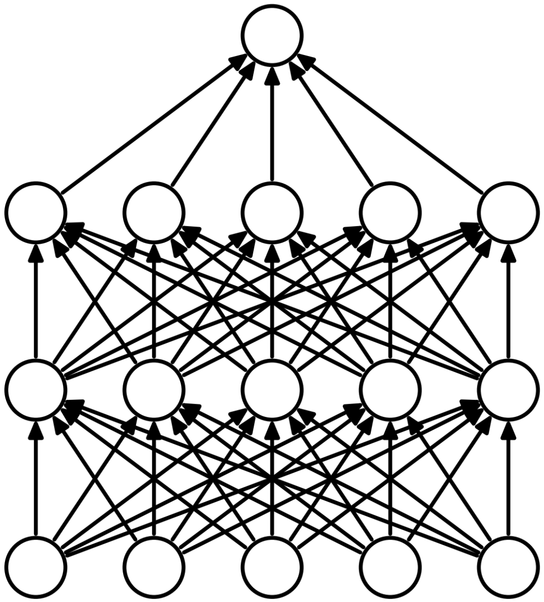


The effects of regularization strength: Each neural network above has 20 hidden neurons, but changing the regularization strength makes its final decision regions smoother with a higher regularization. You can play with these examples in this ConvNetsJS demo.

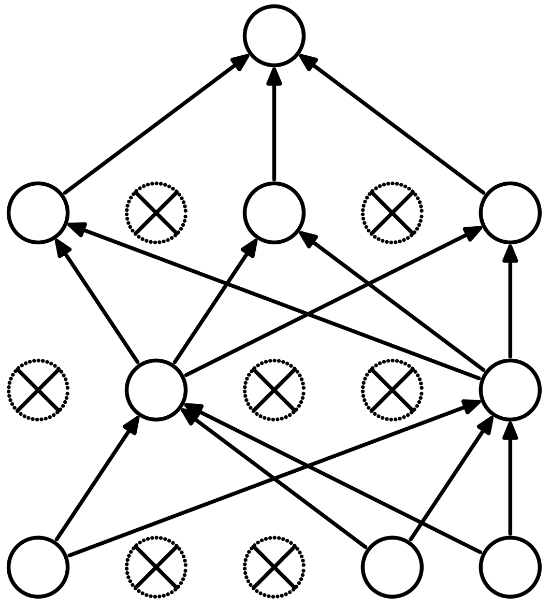
Image from: http://cs231n.github.io/neural-networks-1/

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

Outline for November 14

- Finish fully connected neural networks
- Convolutional neural networks
- (if time) Backpropagation

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*

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 *width*, *height*, and *depth*

Idea: 3D volumes of neurons

- Do not "flatten" image, keep it as a volume with *width*, *height*, and *depth*
- For **CIFAR-10**, we would have:
  - Width=32, Height=32, Depth=3

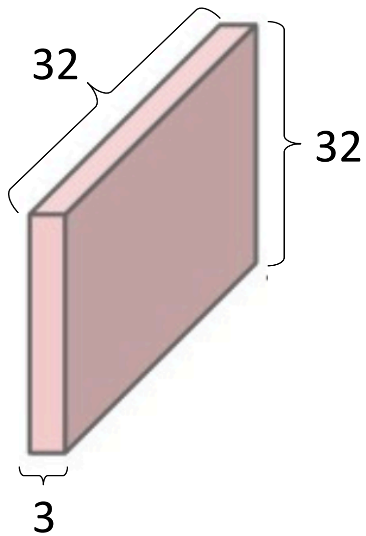


Image: modified from Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

Idea: 3D volumes of neurons

- Do not "flatten" image, keep it as a volume with *width*, *height*, and *depth*
- For **CIFAR-10**, we would have:
  - Width=32, Height=32, Depth=3
- Each layer is also a 3 dimensional volume

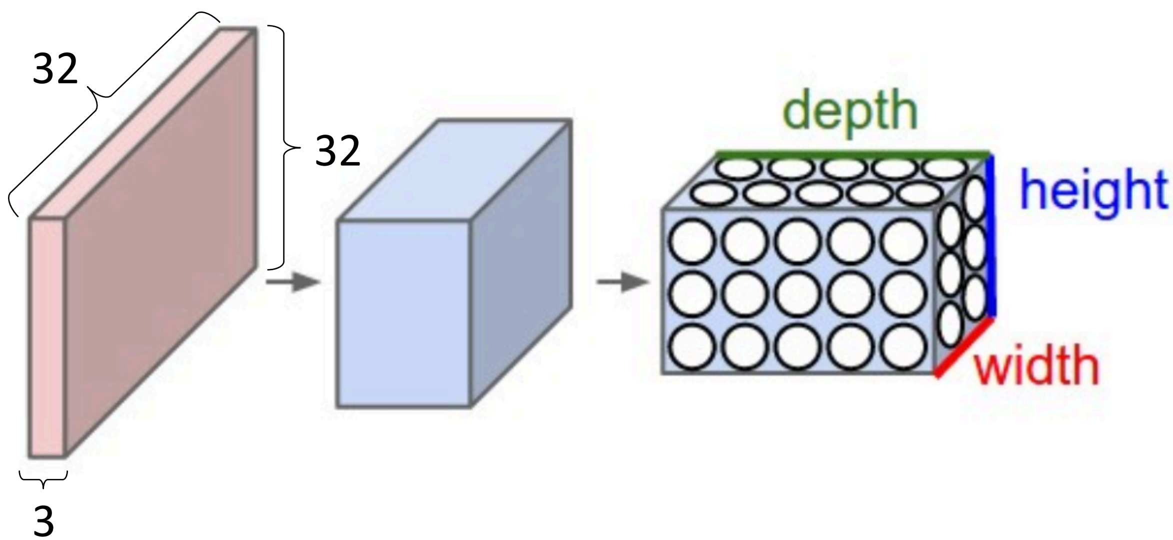


Image: modified from Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

Idea: 3D volumes of neurons

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

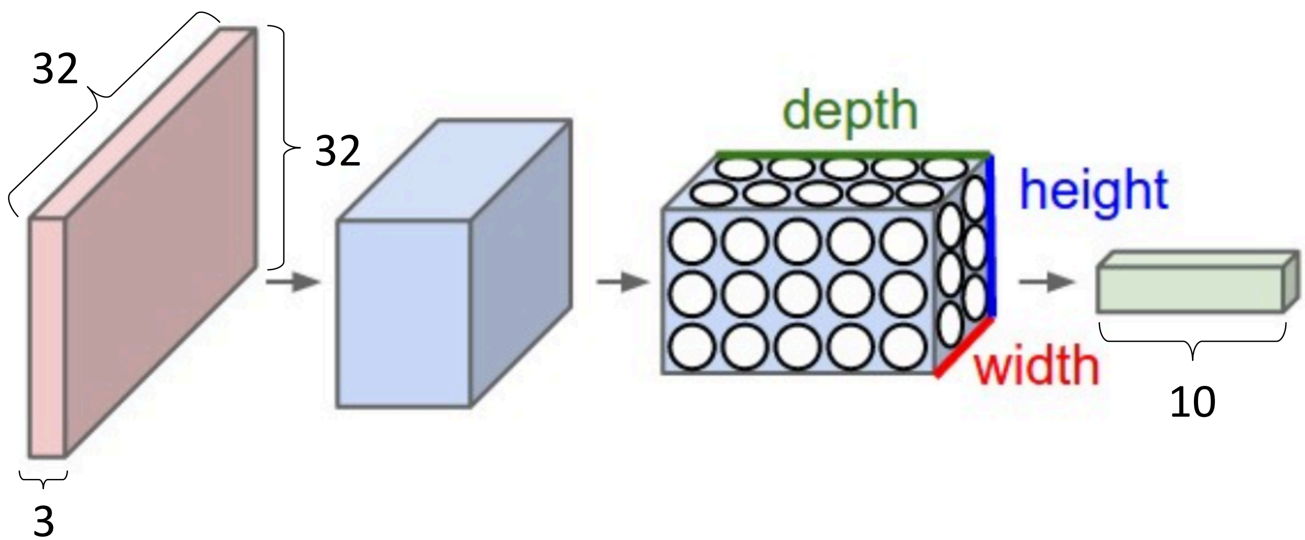


Image: modified from Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

Layers of a Convolutional Neural Network (CNN)

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

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

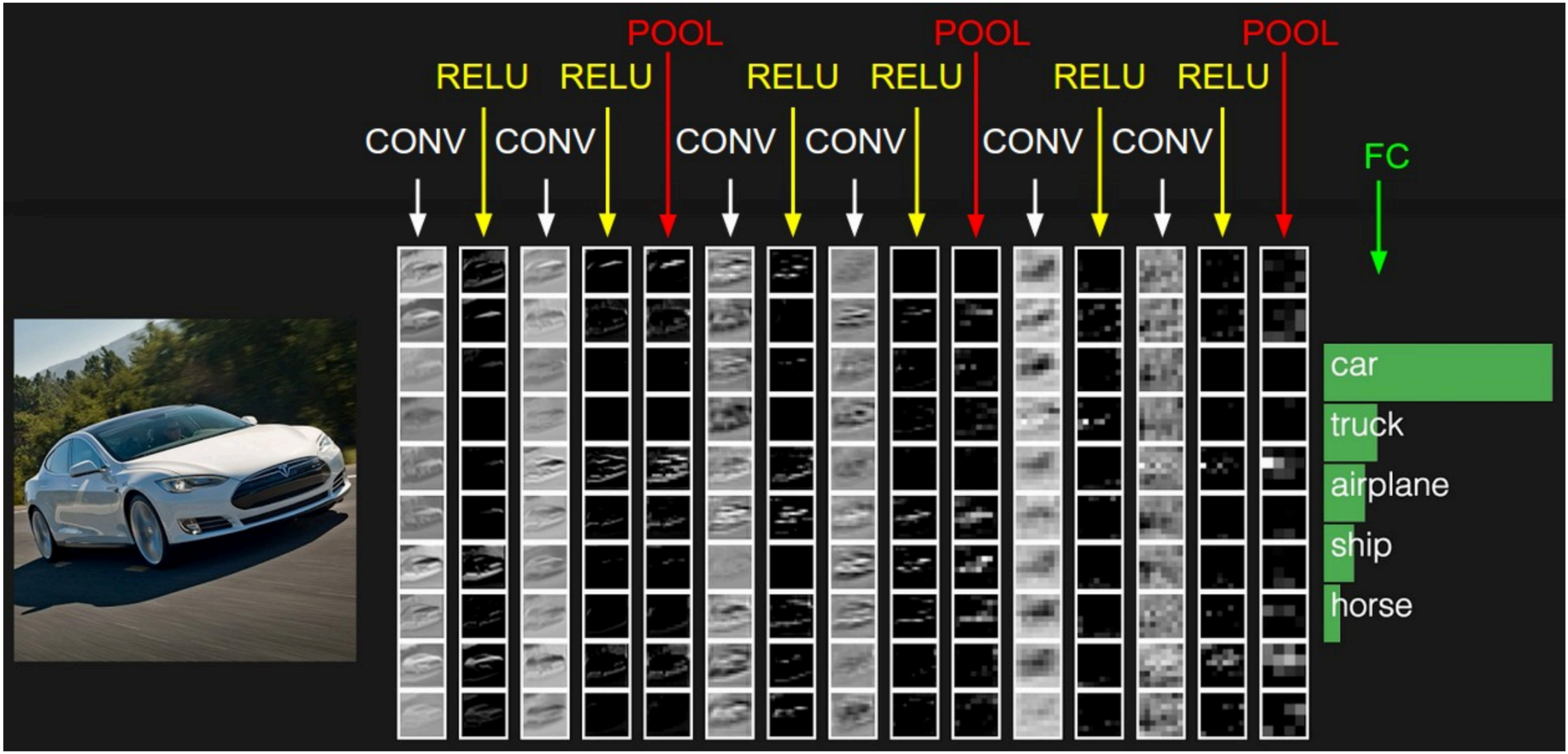
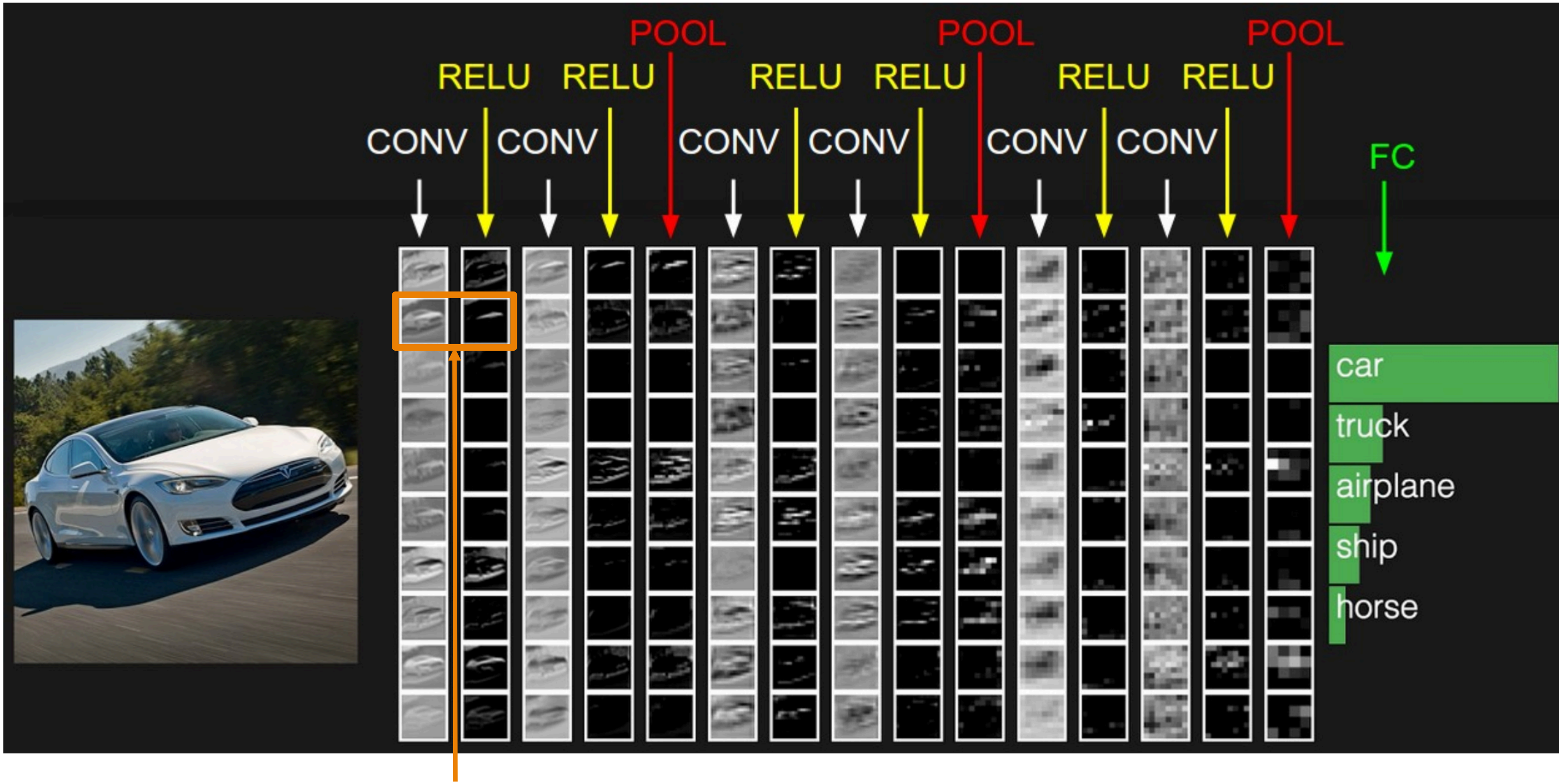


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

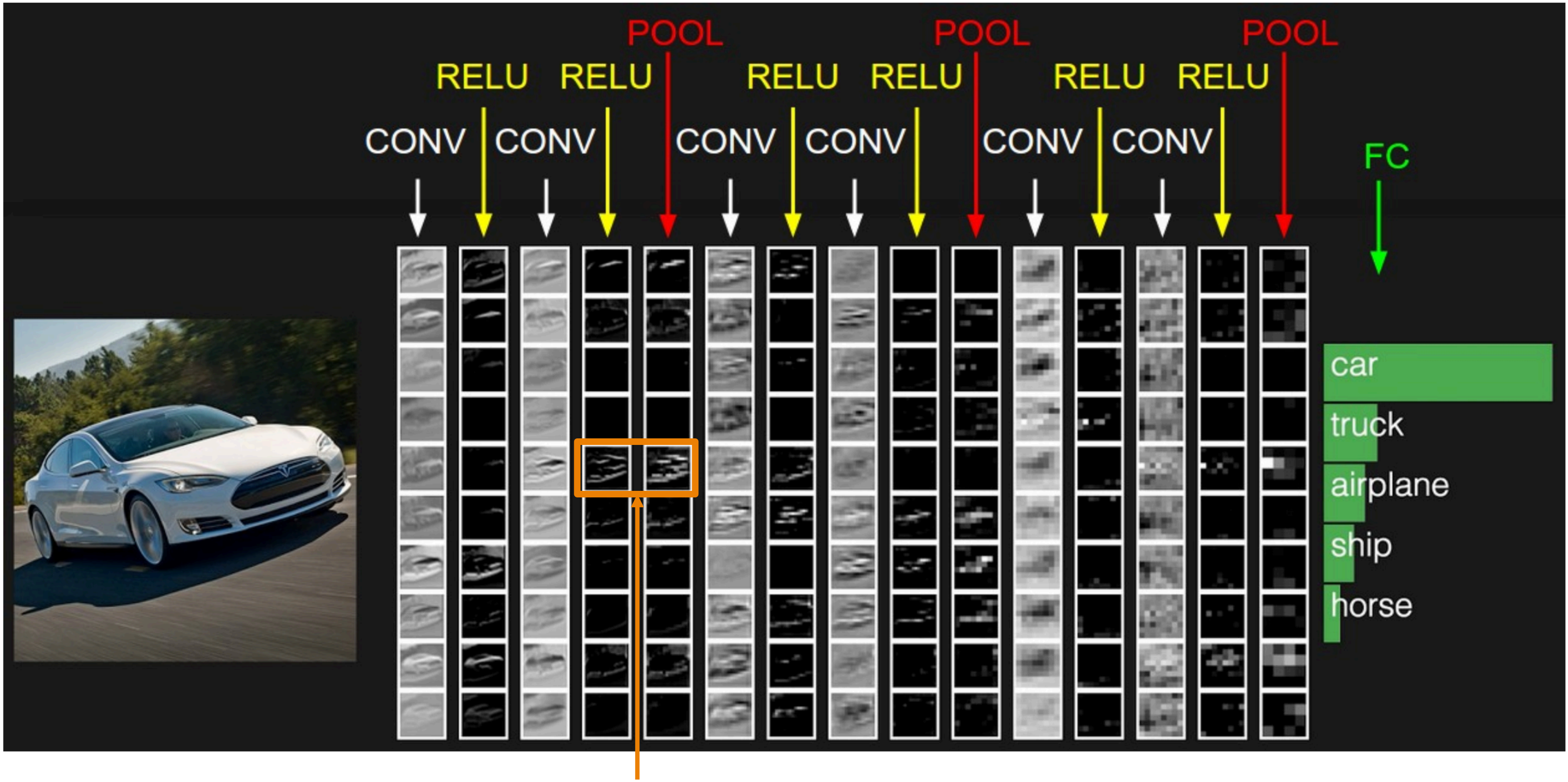
Example CNN architecture



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

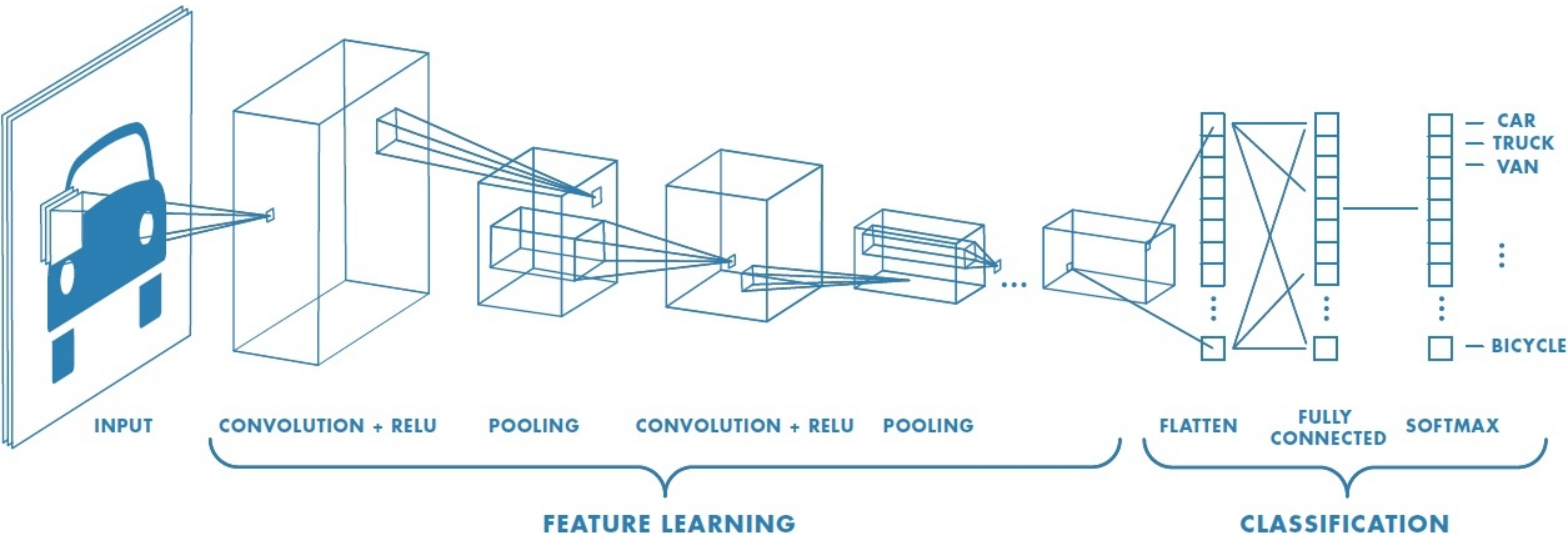


Image from MathWorks: https://www.mathworks.com/solutions/deep-learning/convolutional-neural-netwo

Idea: local "receptive field"

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

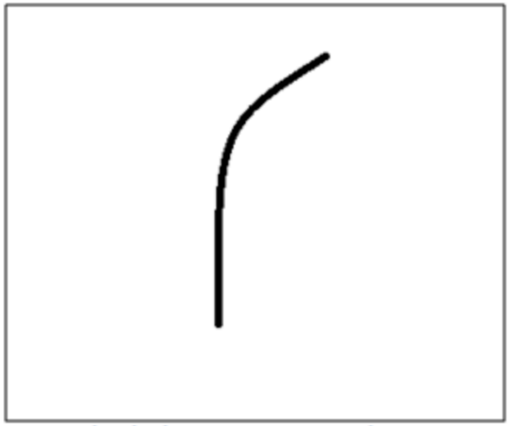
- Note an important *asymmetry*: we will look at a small "patch" of the image relative to its width and height, but we will look all the way through the depth!

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

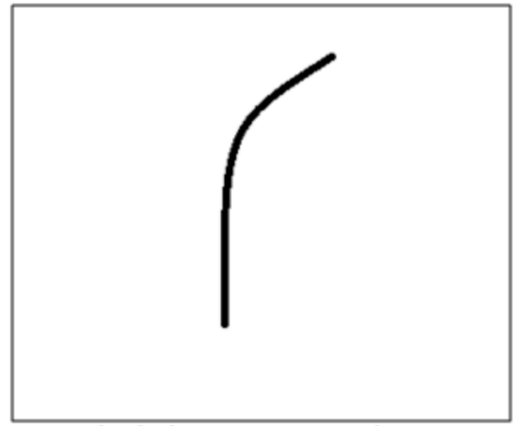
Images by: Adit Deshpande

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

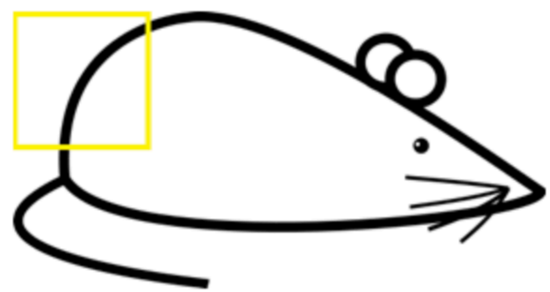
Pixel representation of filter



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

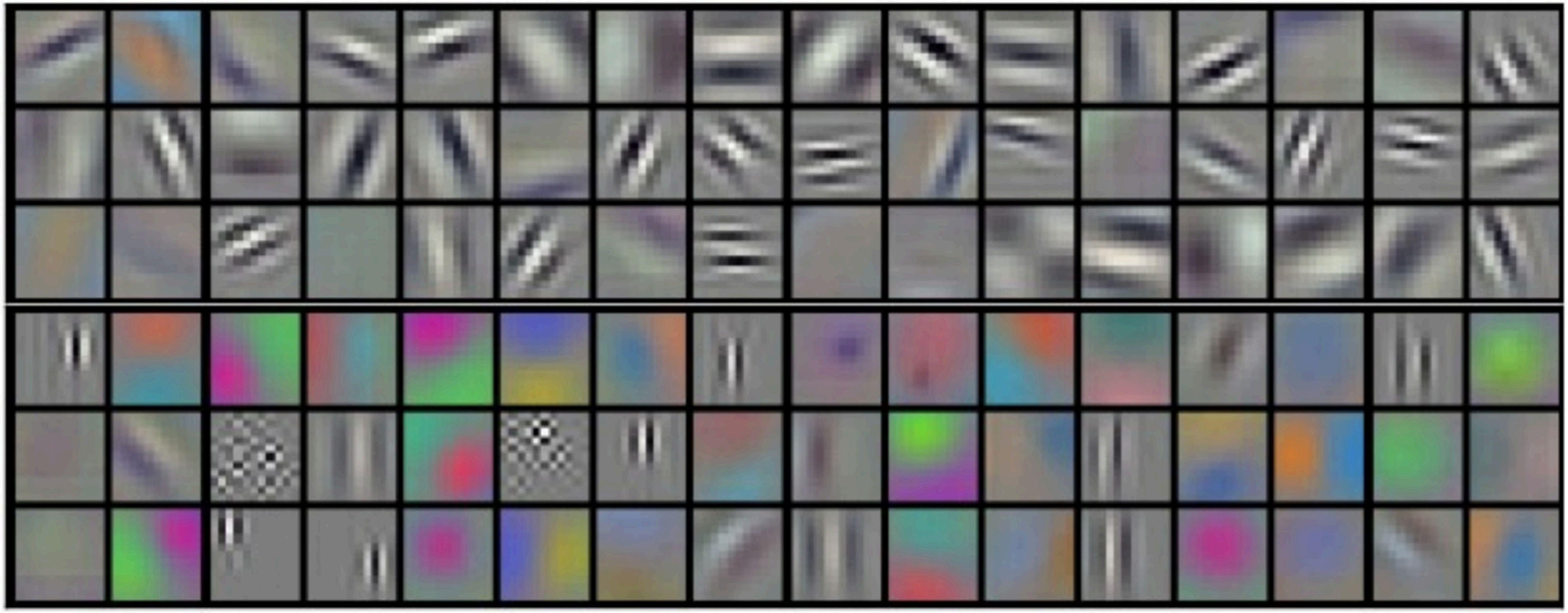
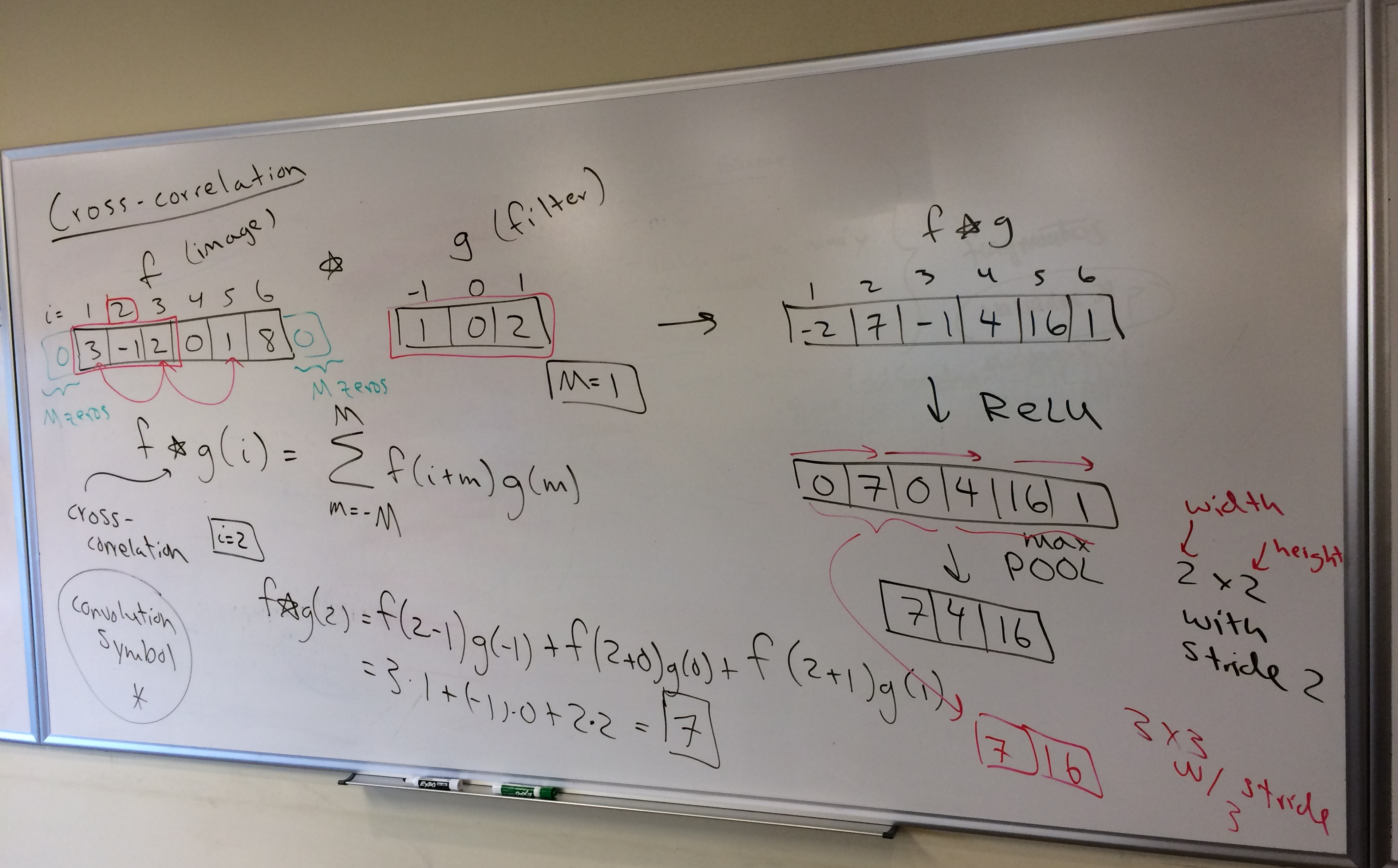


Image: Krizhevsky et al. (2012) https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

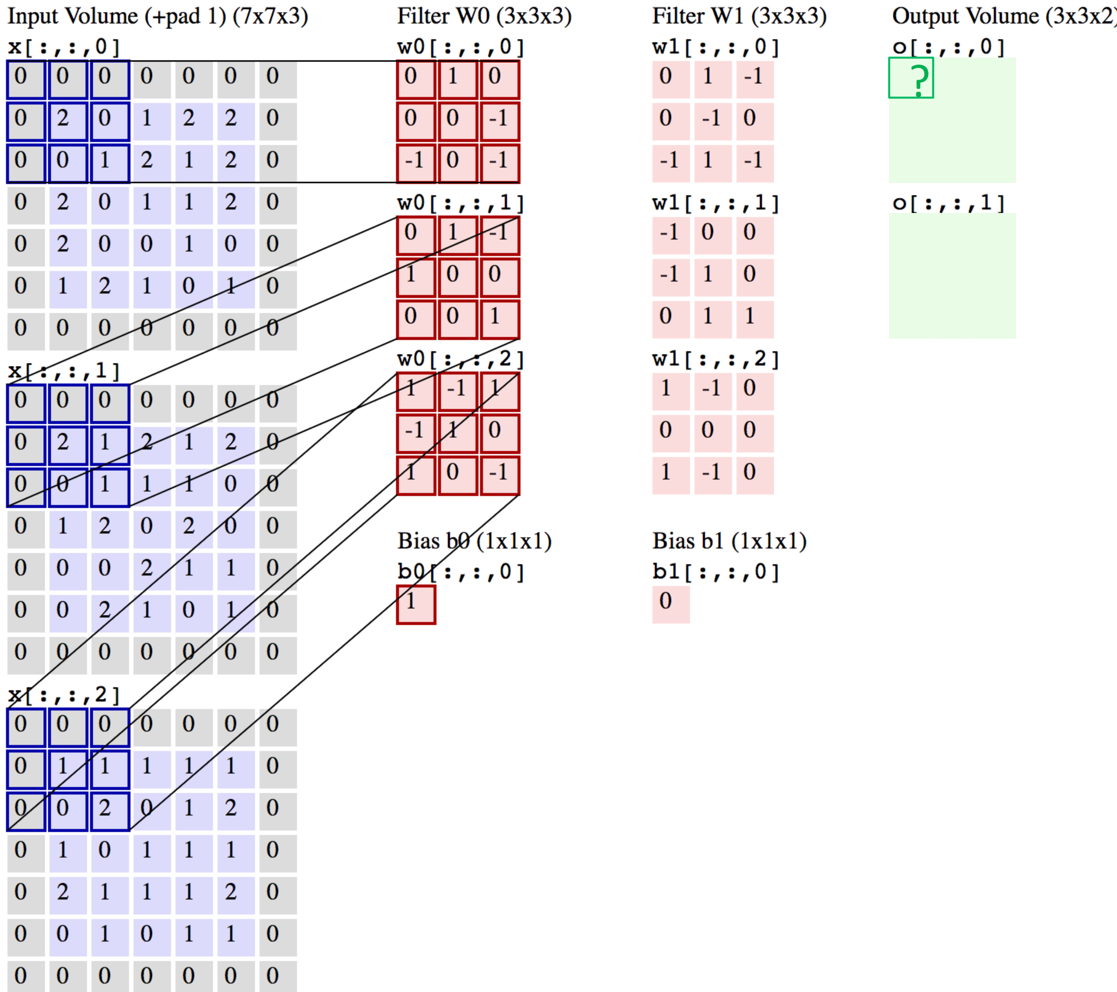
Math behind convolutions

(actually cross-correlations!)

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



Cross-correlation
f (image)
g (filter)
i = 1 2 3 4 5 6
0 | 3 | -1 | 2 | 0 | 1 | 8 | 0
M zeros
M zeros
-1 0 1
1 | 0 | 2
M=1
f ✡ g
1 2 3 4 5 6
-2 | 7 | -1 | 4 | 16 | 1
↓ ReLU
0 | 7 | 0 | 4 | 16 | 1
max
↓ POOL
7 | 4 | 16
width
height
2 x 2
with
stride 2
7 | 16
3 x 3
w/ stride 3
f ✡ g(i) = Σ_{m=-M}^{M} f(i+m) g(m)
cross-correlation
i=2
Convolution Symbol
*
f✡g(2) = f(2-1)g(-1) + f(2+0)g(0) + f(2+1)g(1)
= 3·1 + (-1)·0 + 2·2 = 7



In-class exercise

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

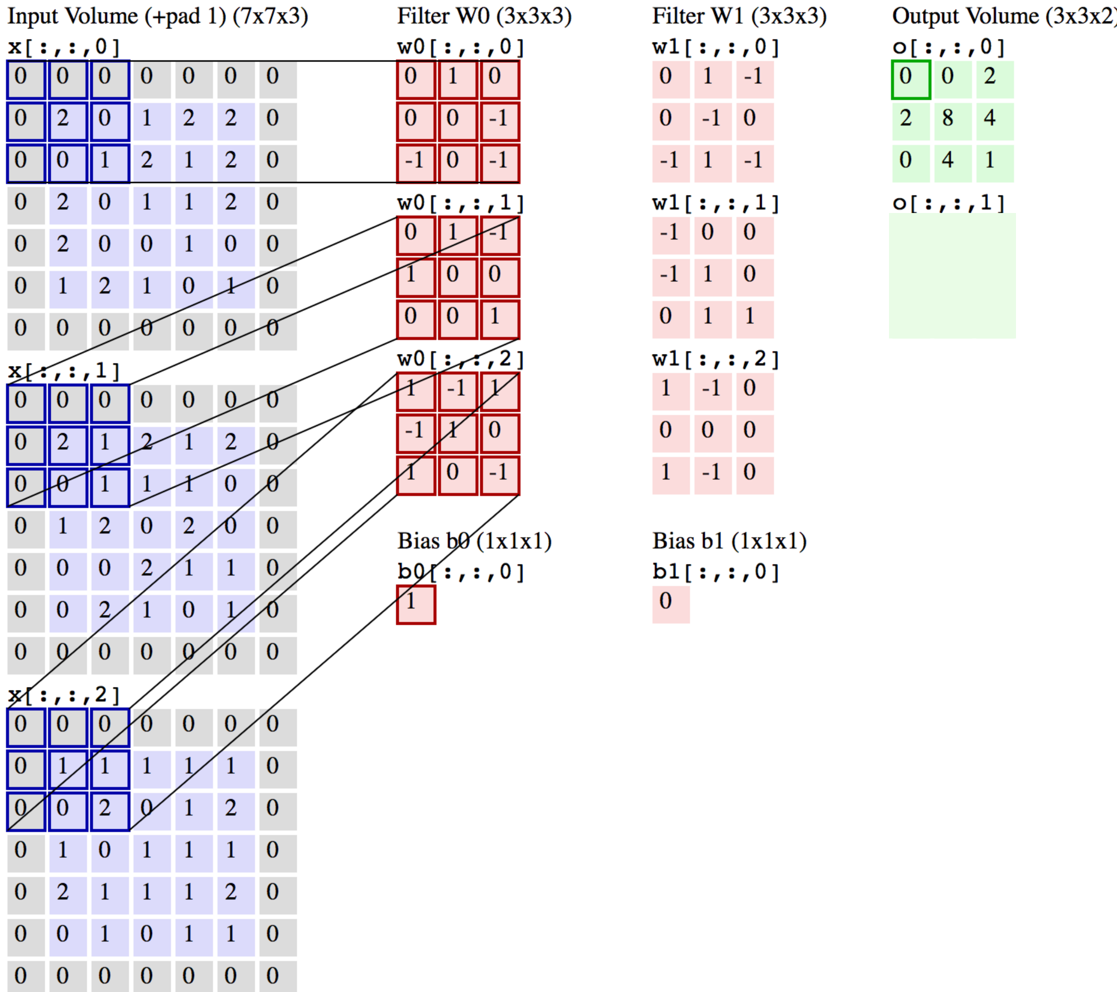


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

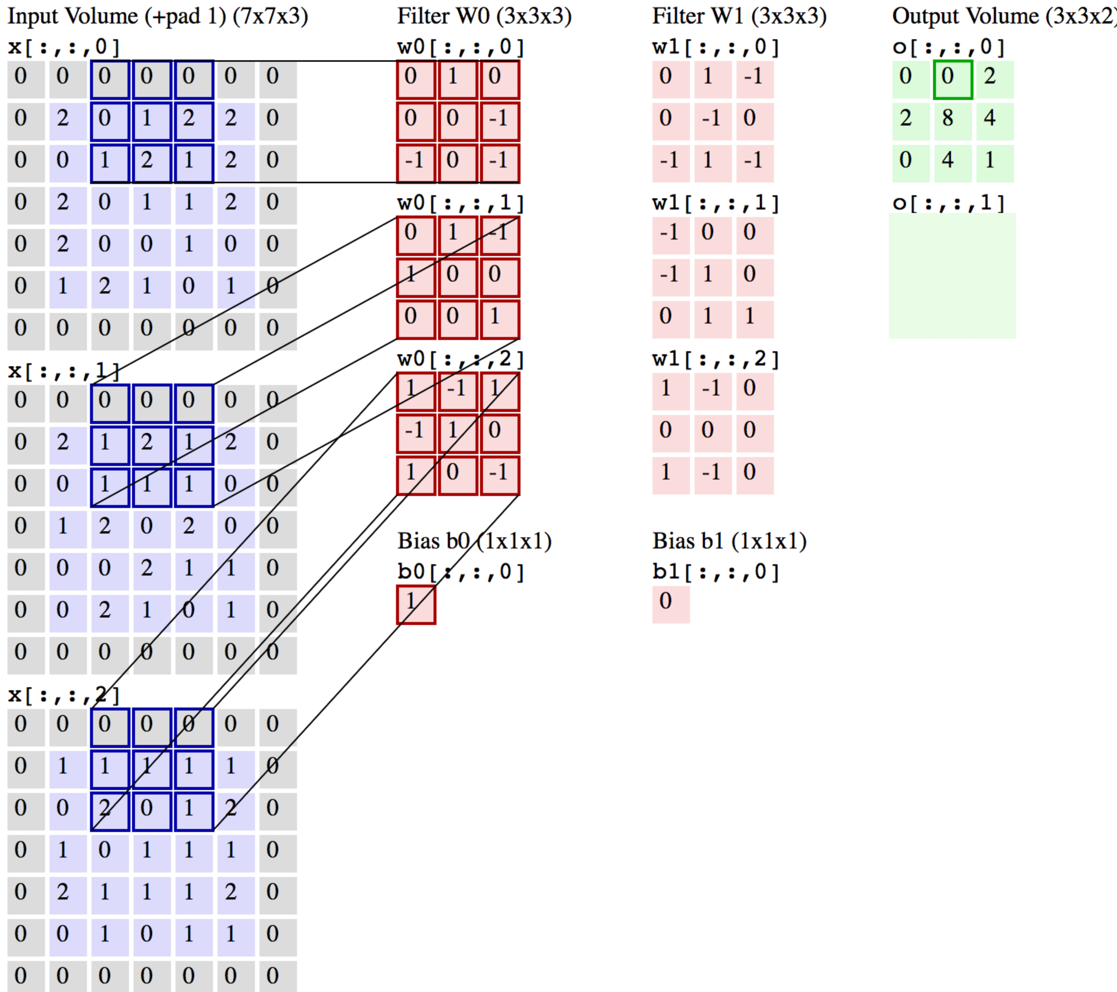


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

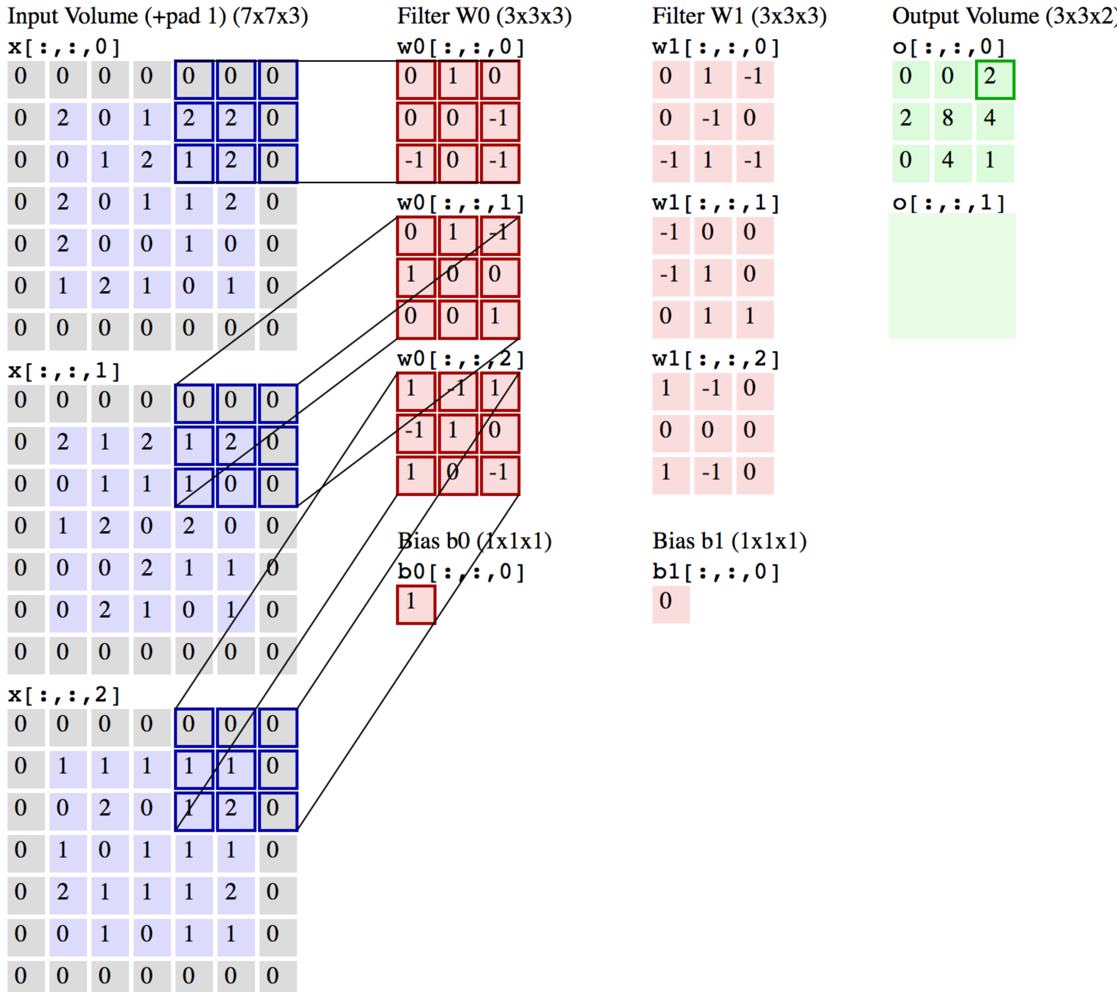


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

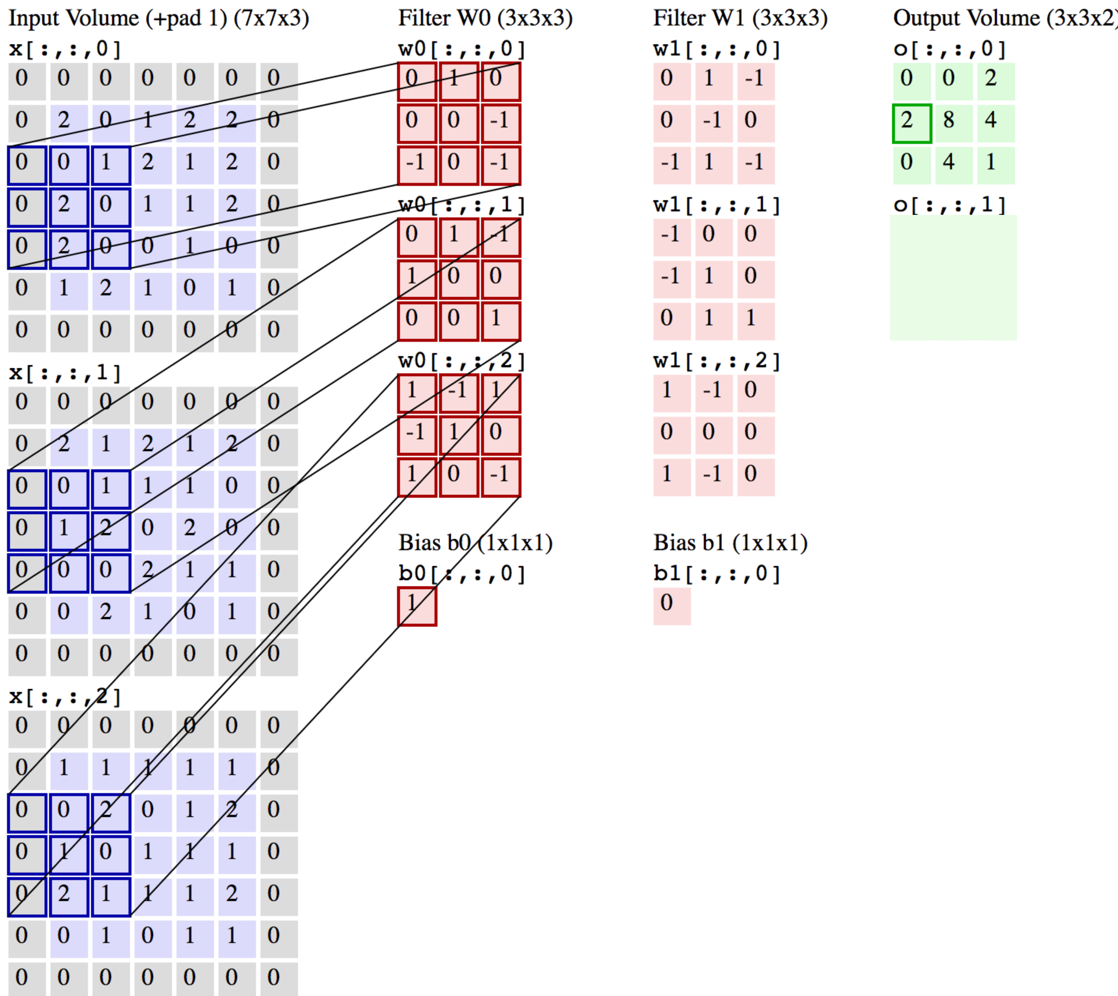


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

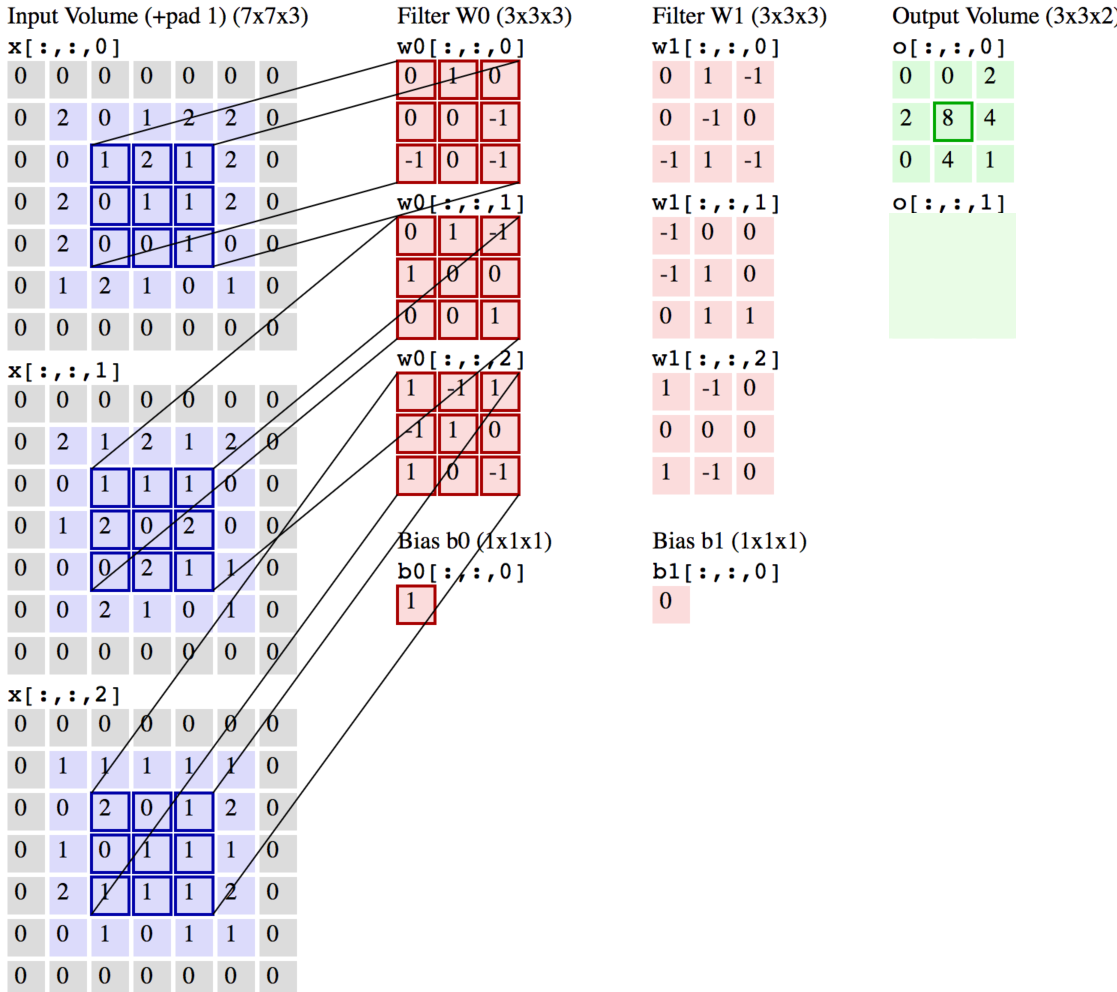


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

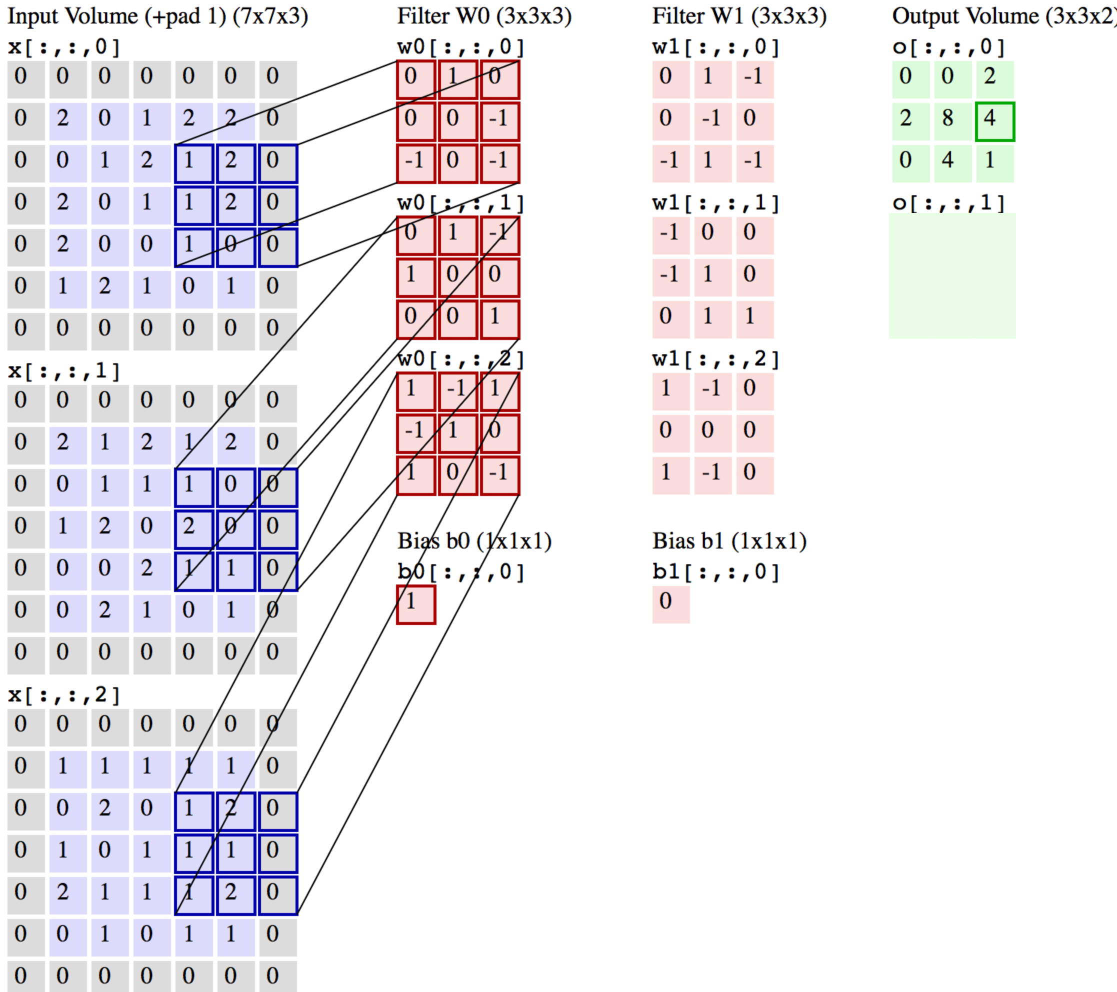


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

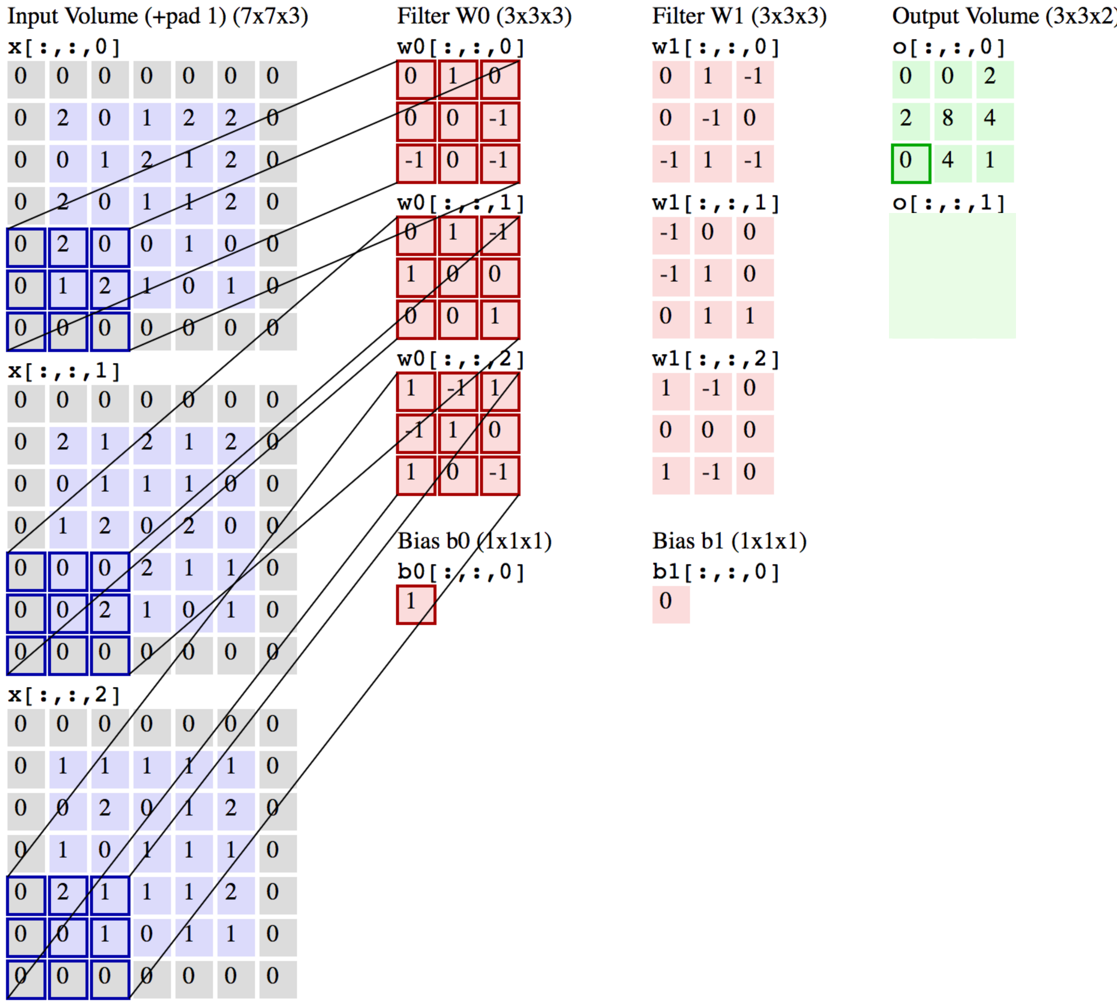


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

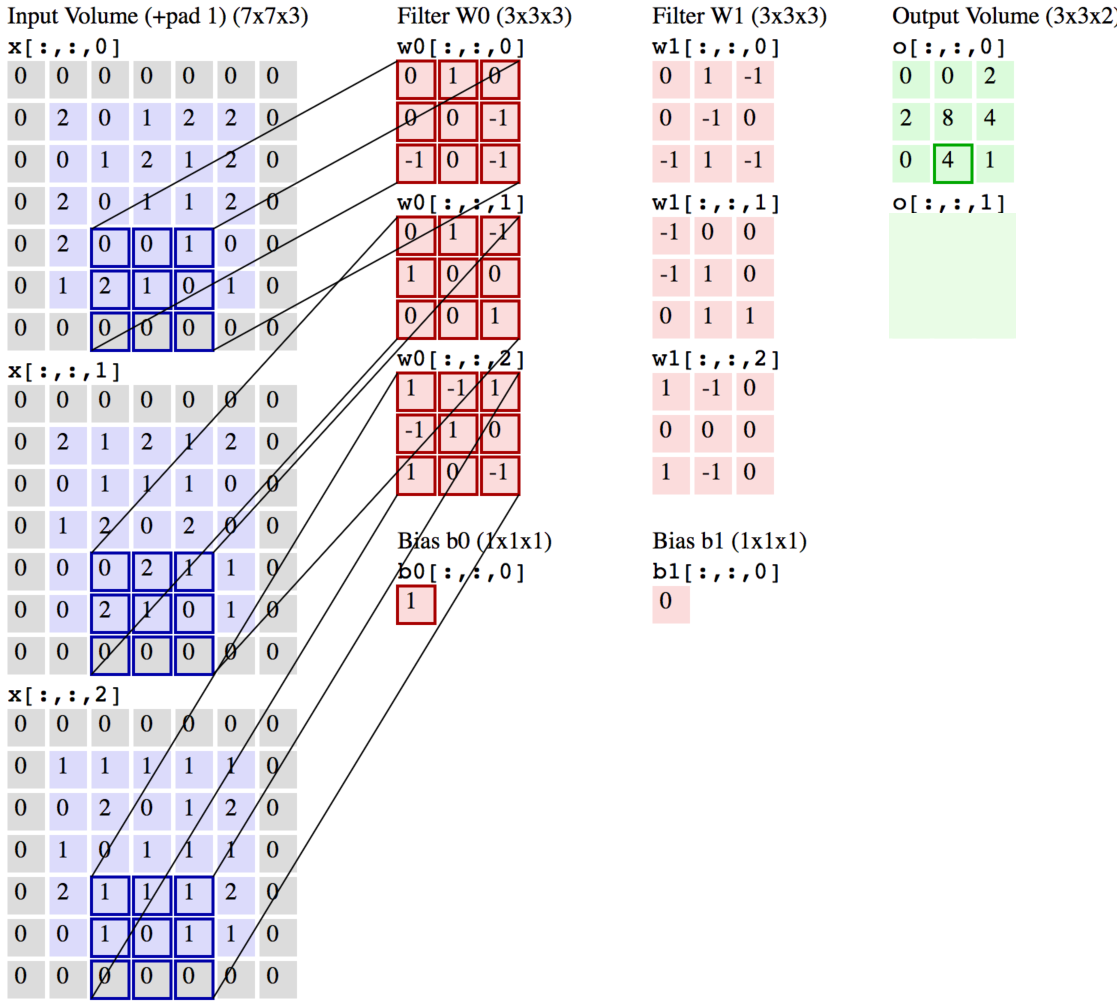


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

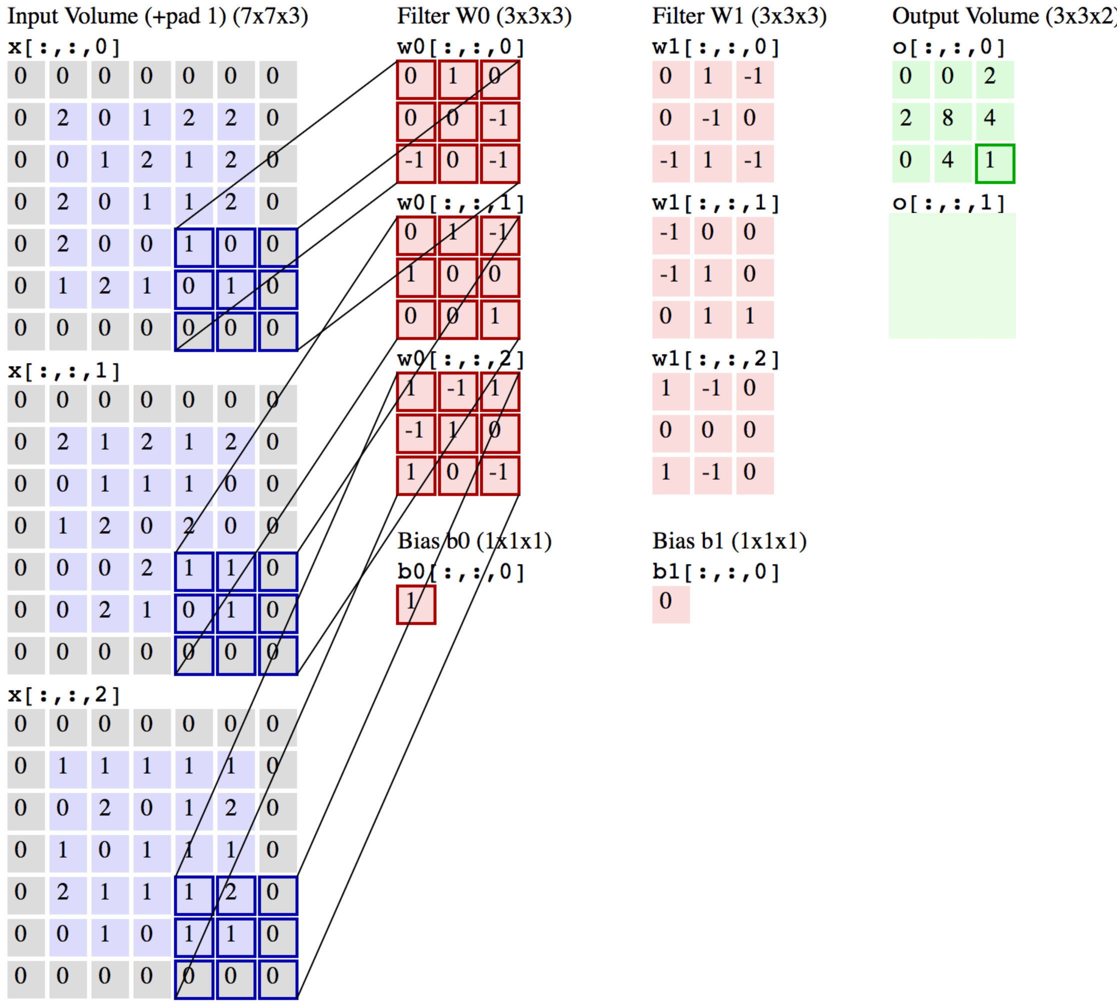


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

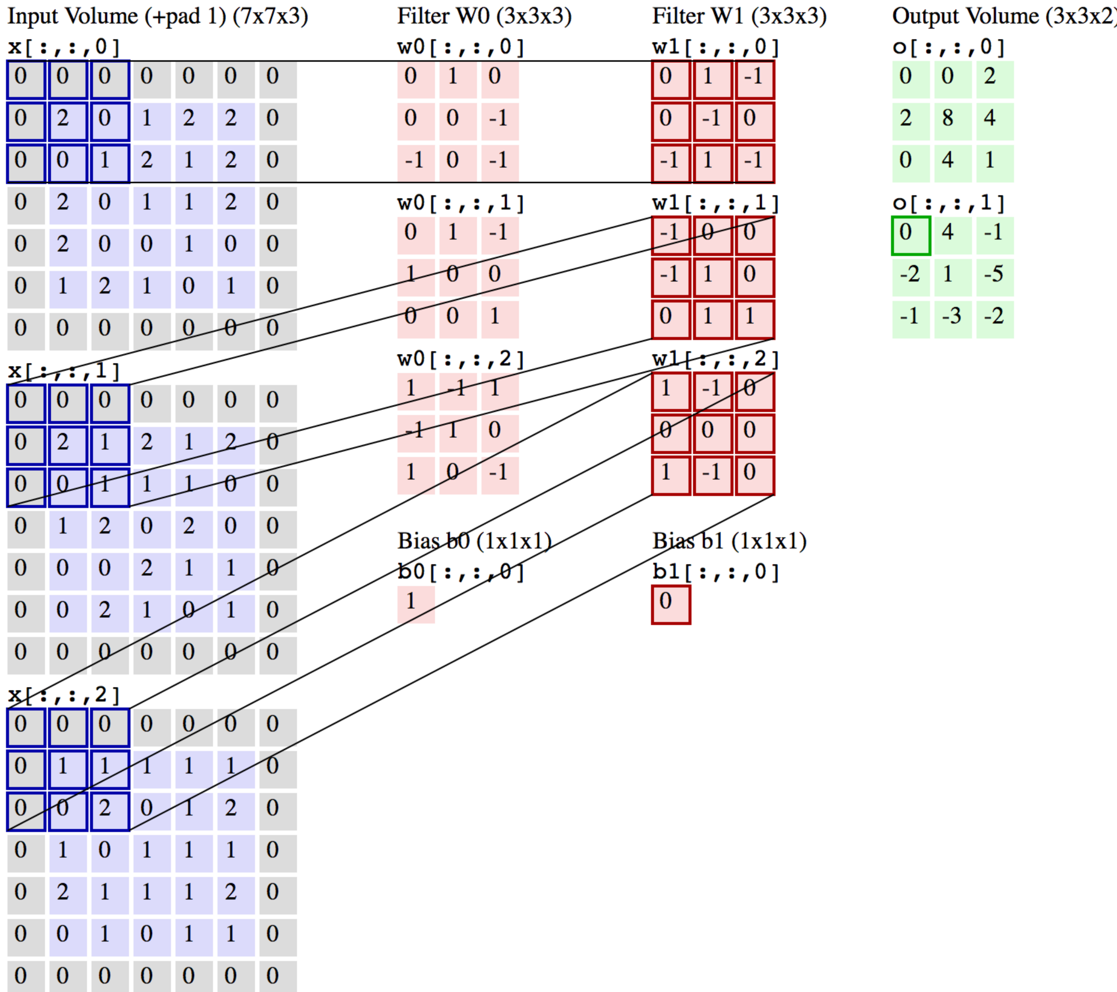


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

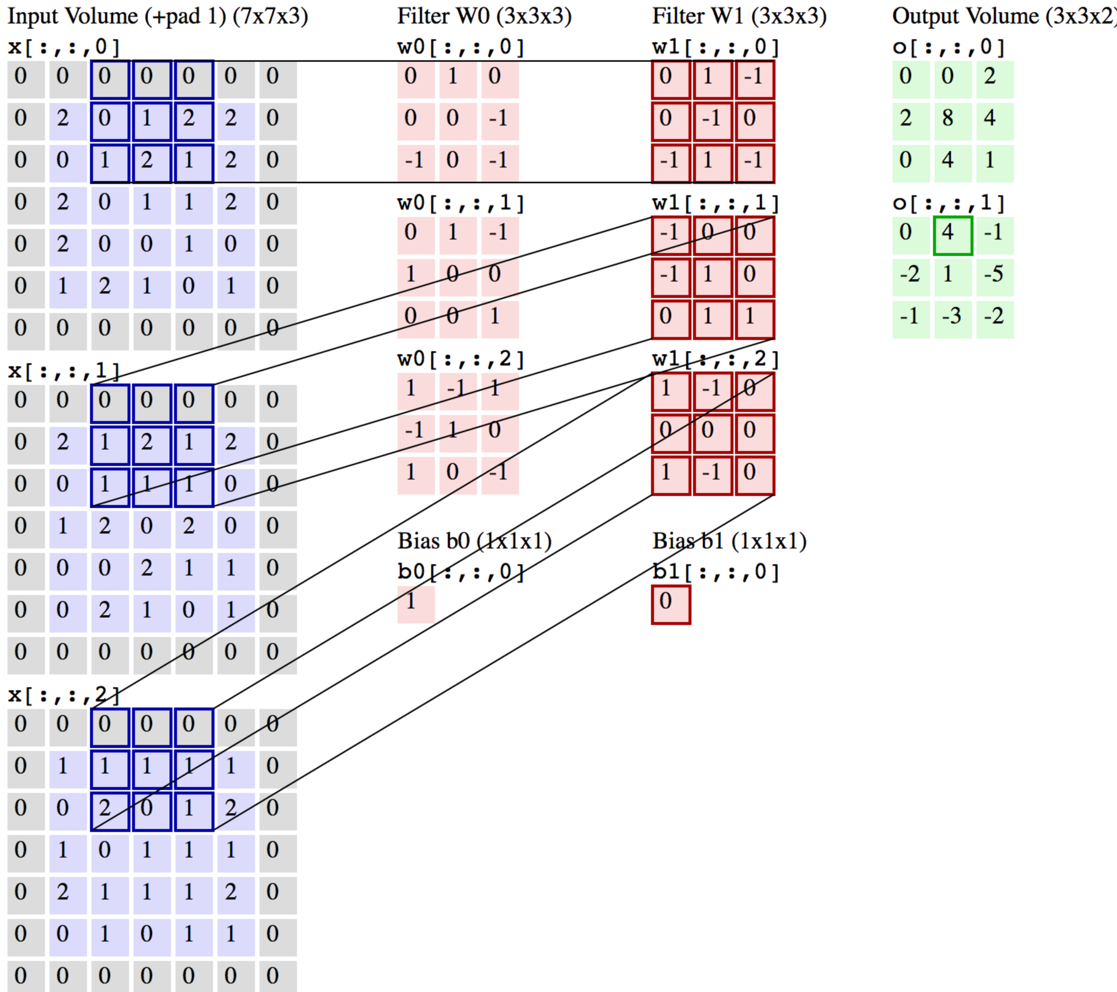


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

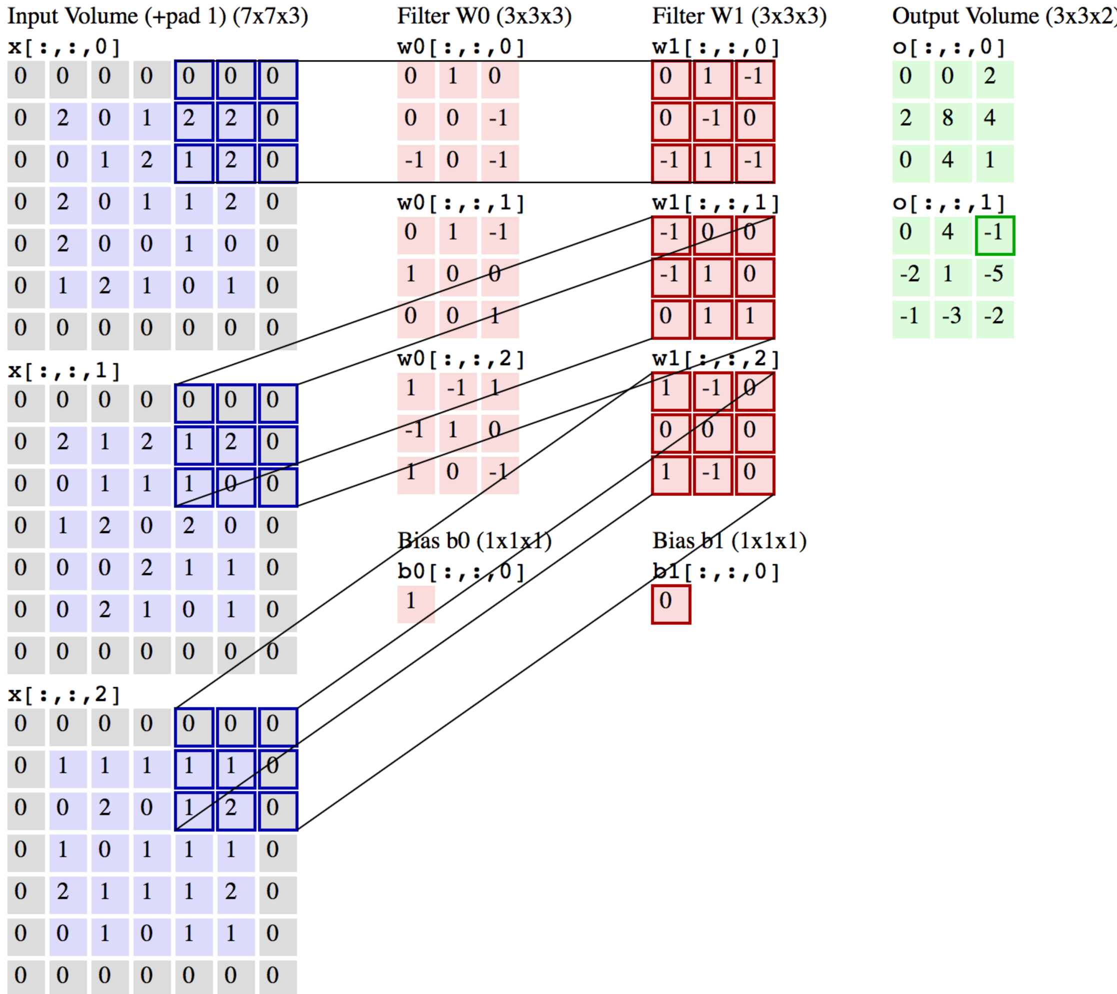


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

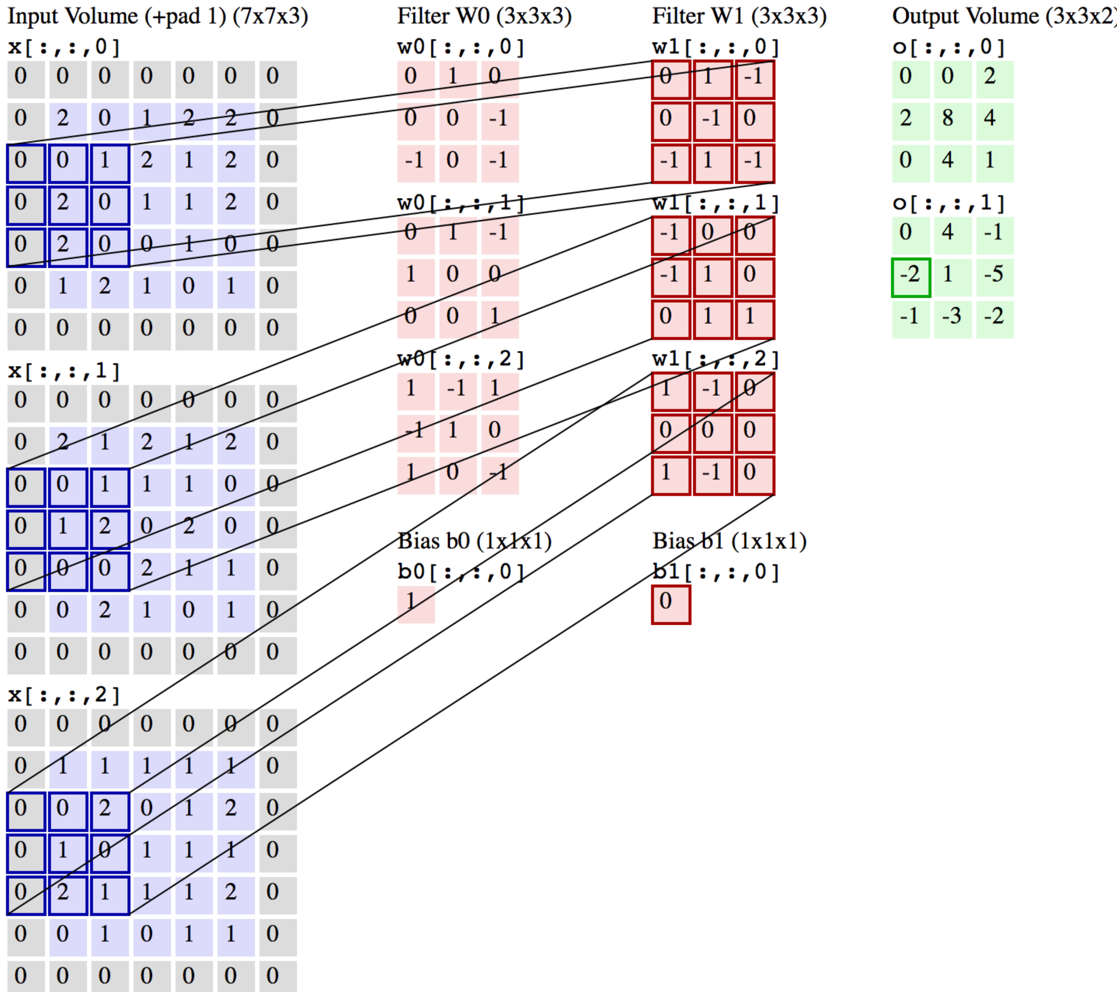


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

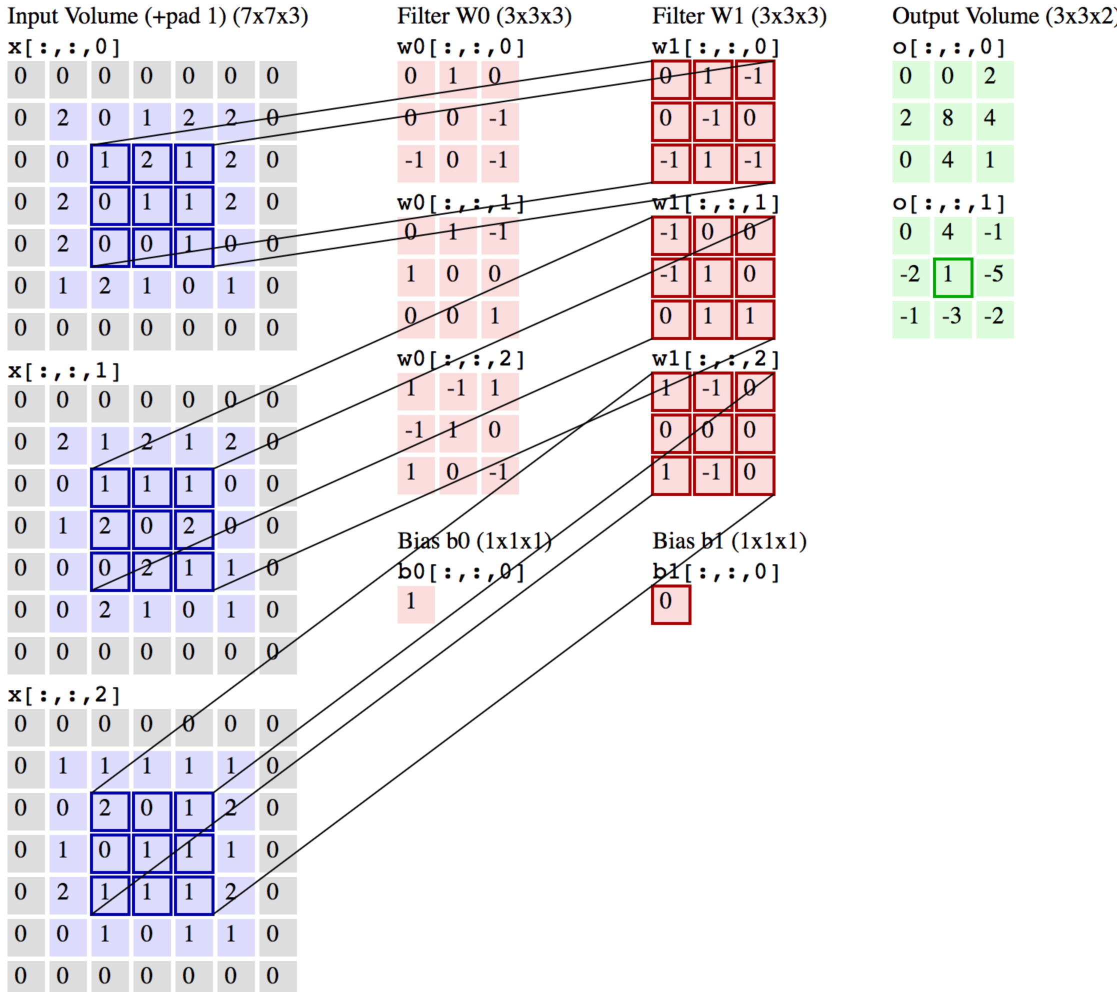


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

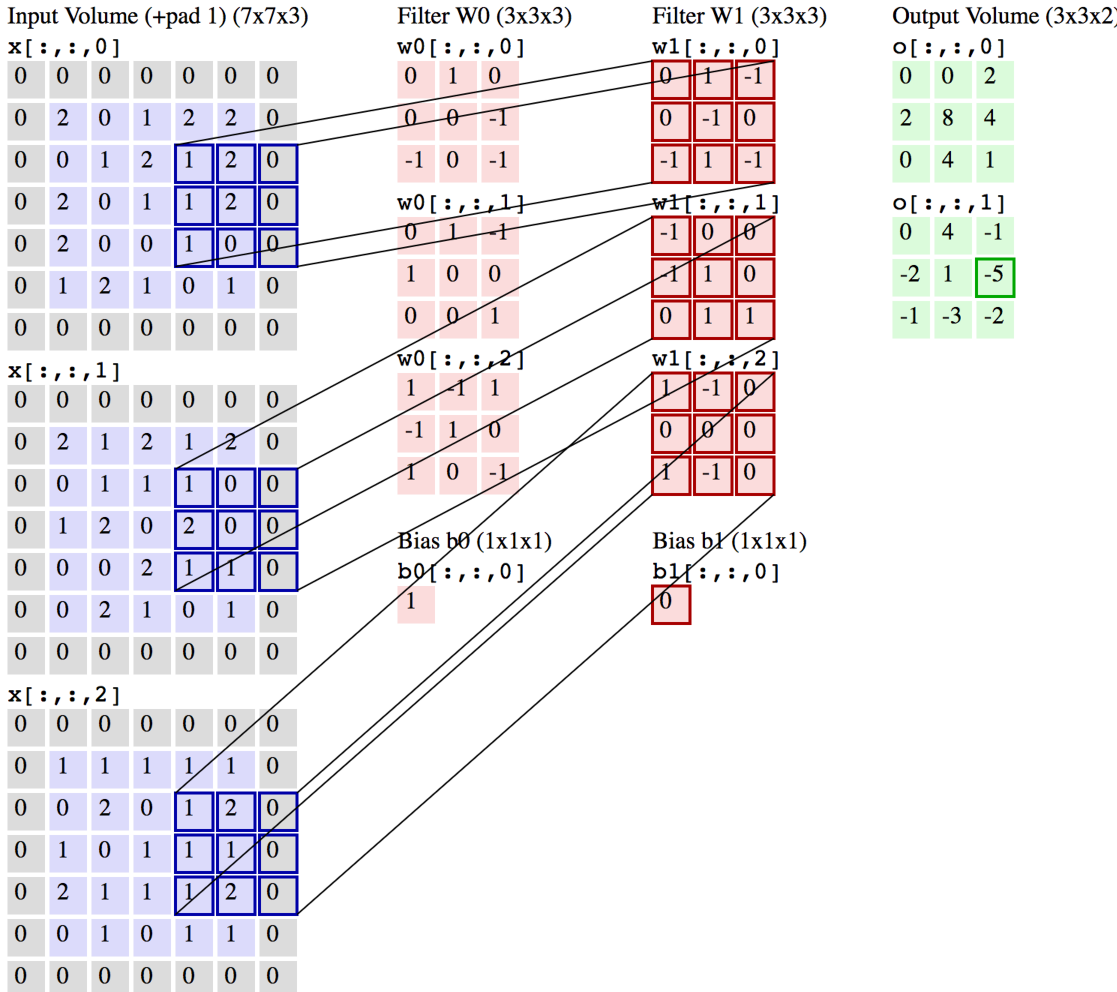


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

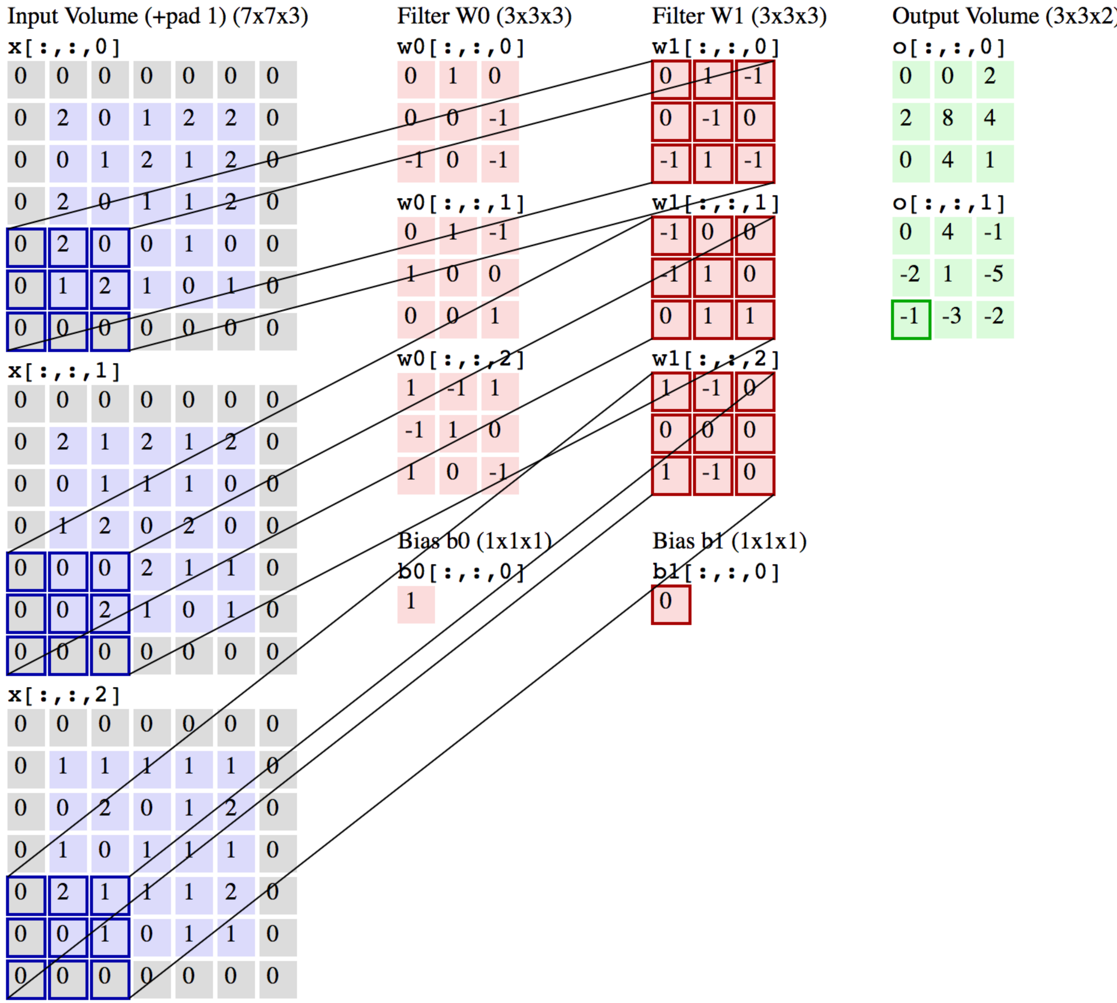


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

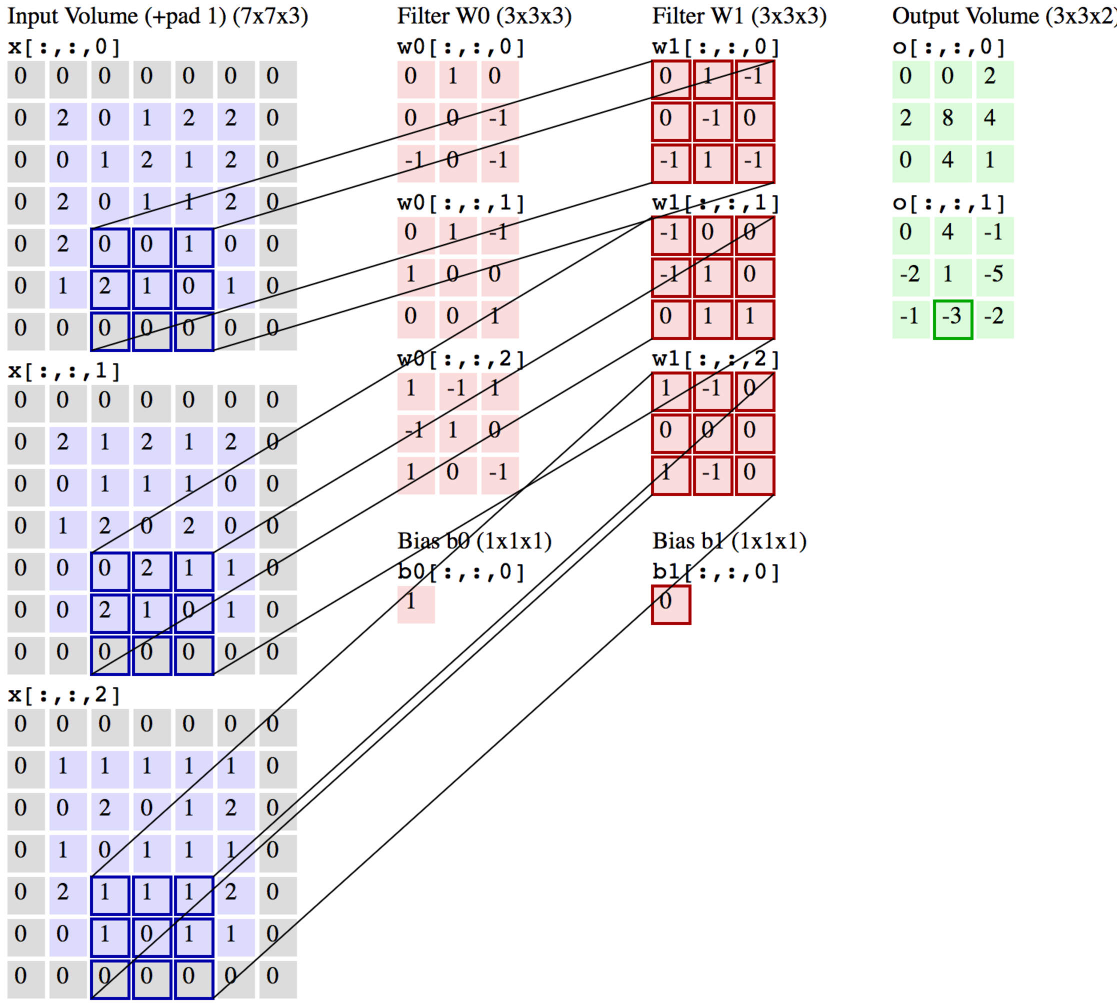


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

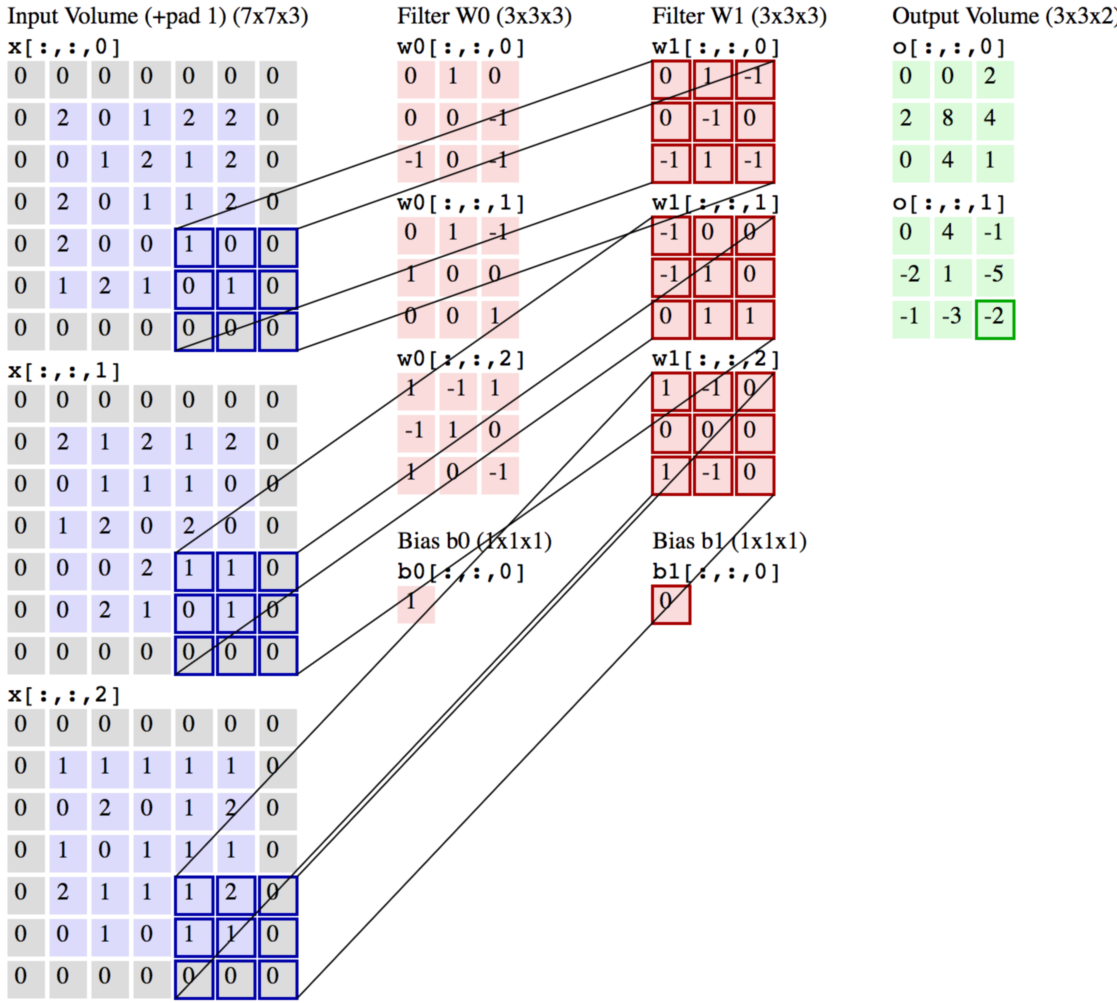
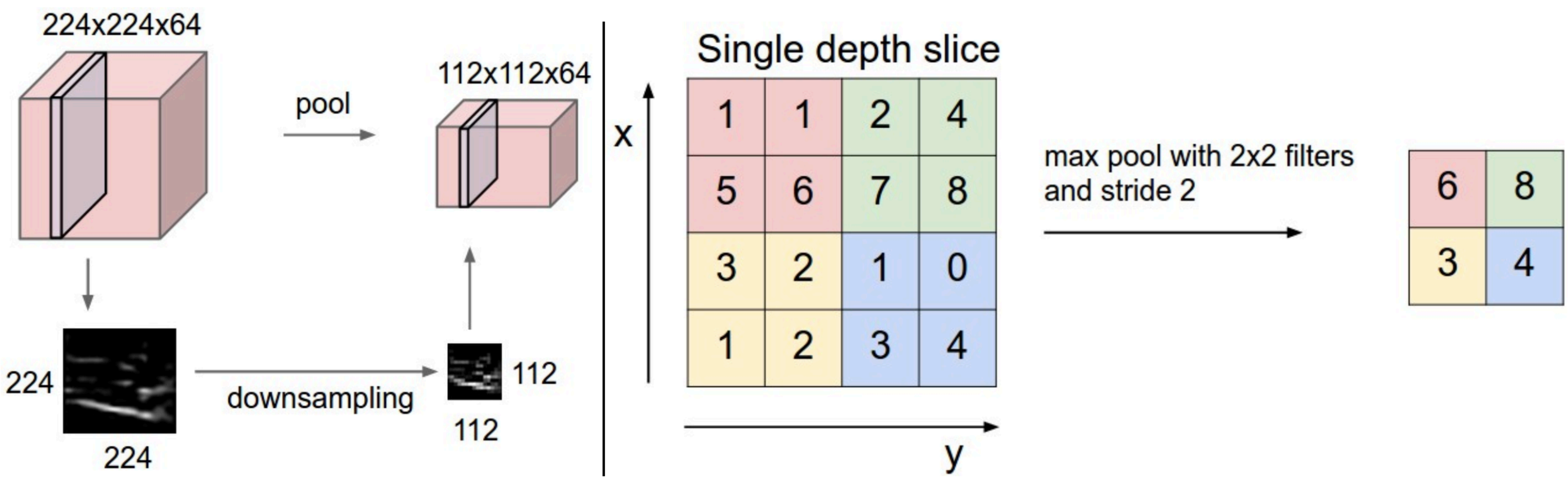


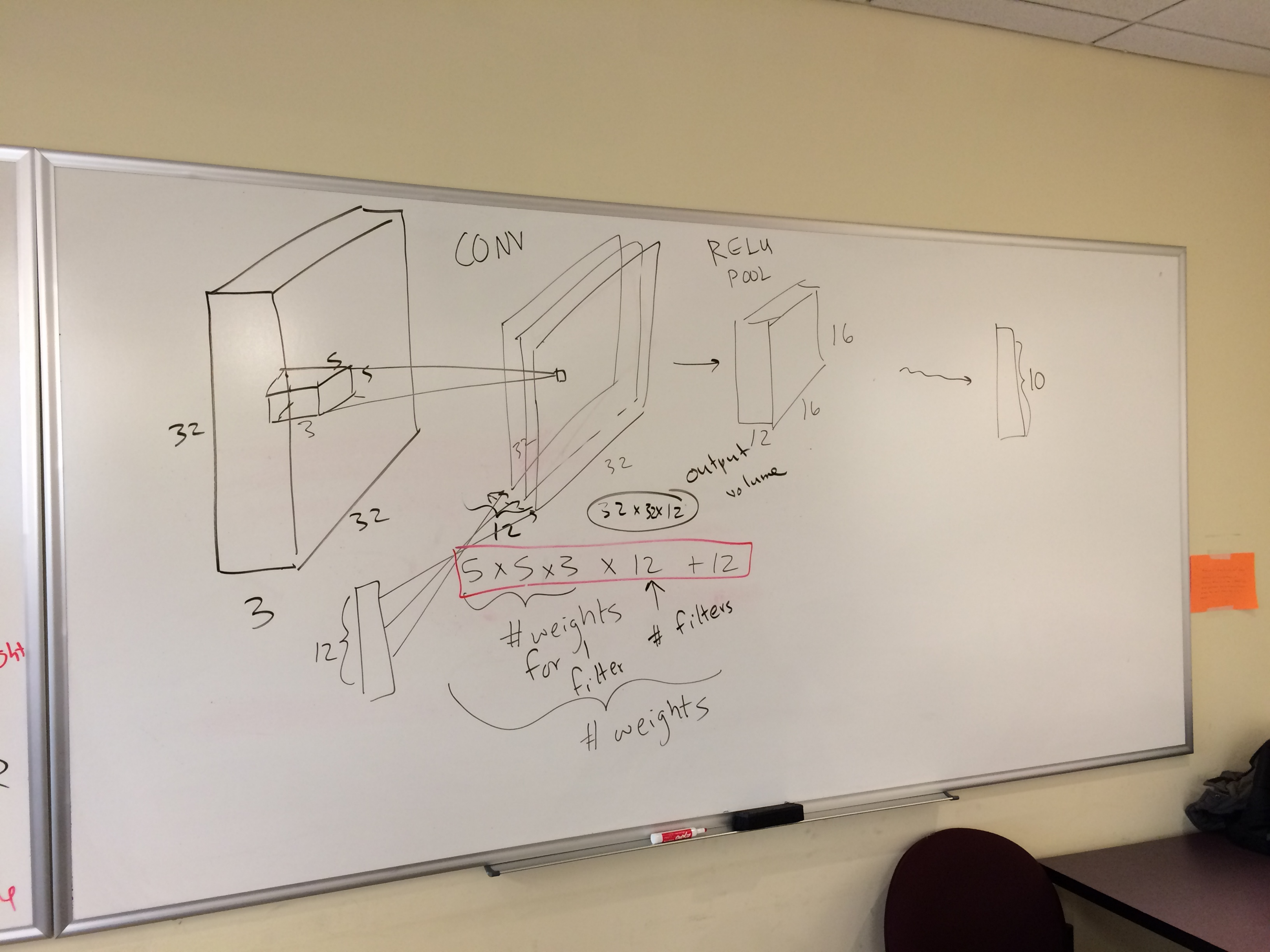
Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

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/


CONV
RELU
POOL
32
32
3
5
5
3
32
32
12
16
16
12
10
output volume
32 x 32 x 12
5 x 5 x 3 x 12 + 12
weights for 1 filter
filters
12
weights

Handout 18, Q4

(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, Q4

(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, Q4

(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, Q4

(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!

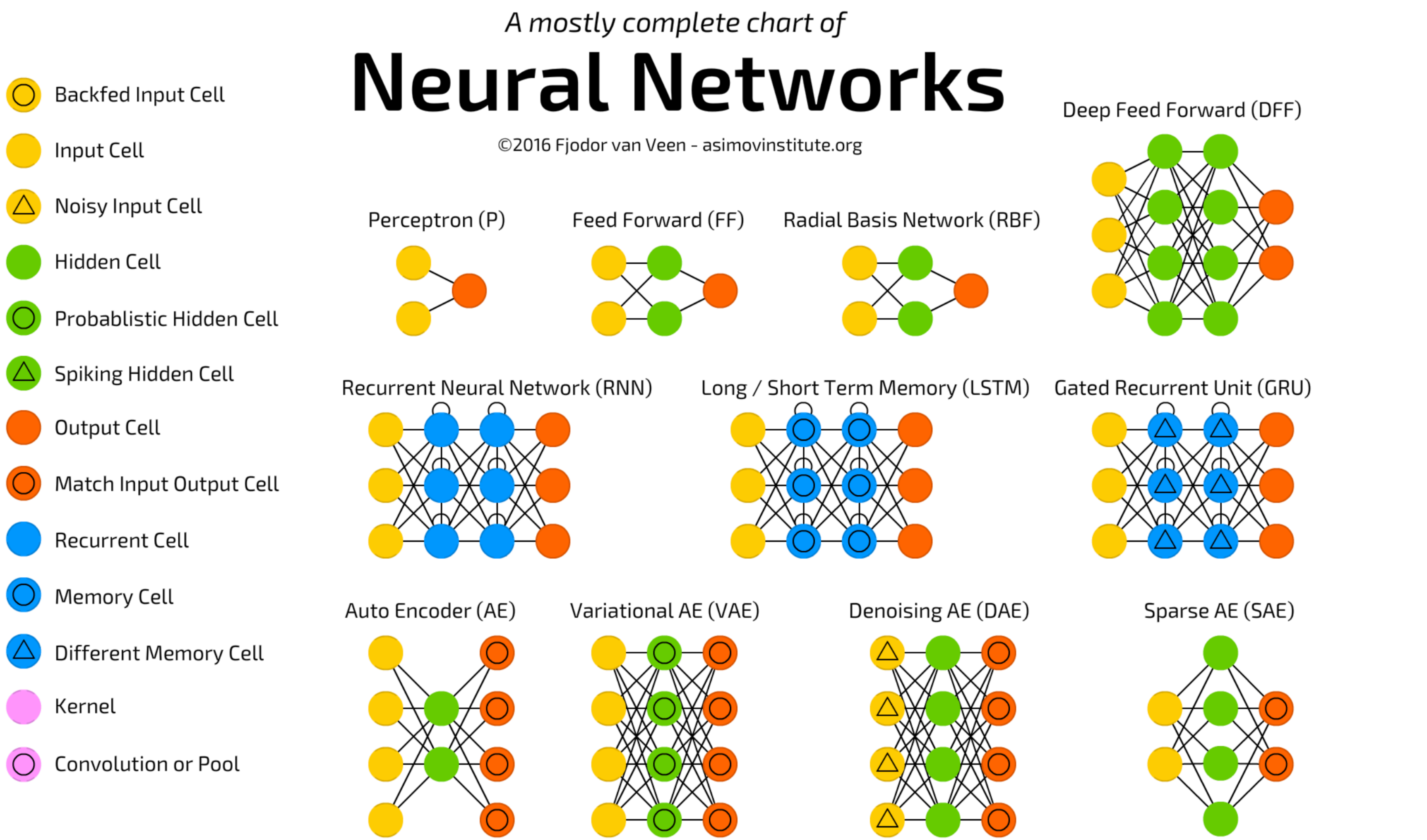


Image from Neural Network Zoo: http://www.asimovinstitute.org/neural-network-z

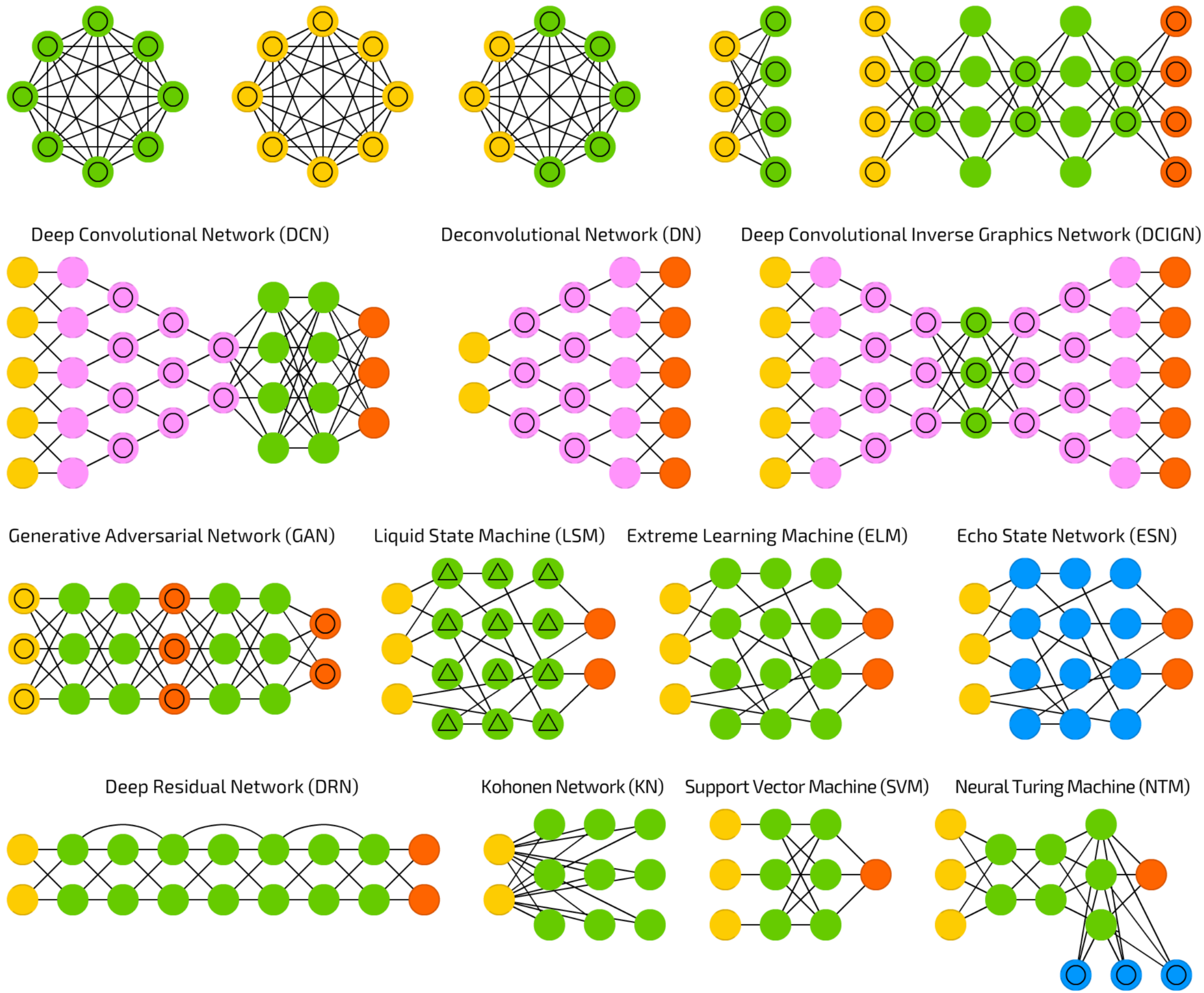


Image from Neural Network Zoo: http://www.asimovinstitute.org/neural-network-z