CS 66: Machine Learning

Prof. Sara Mathieson

Spring 2019

Outline for April 10

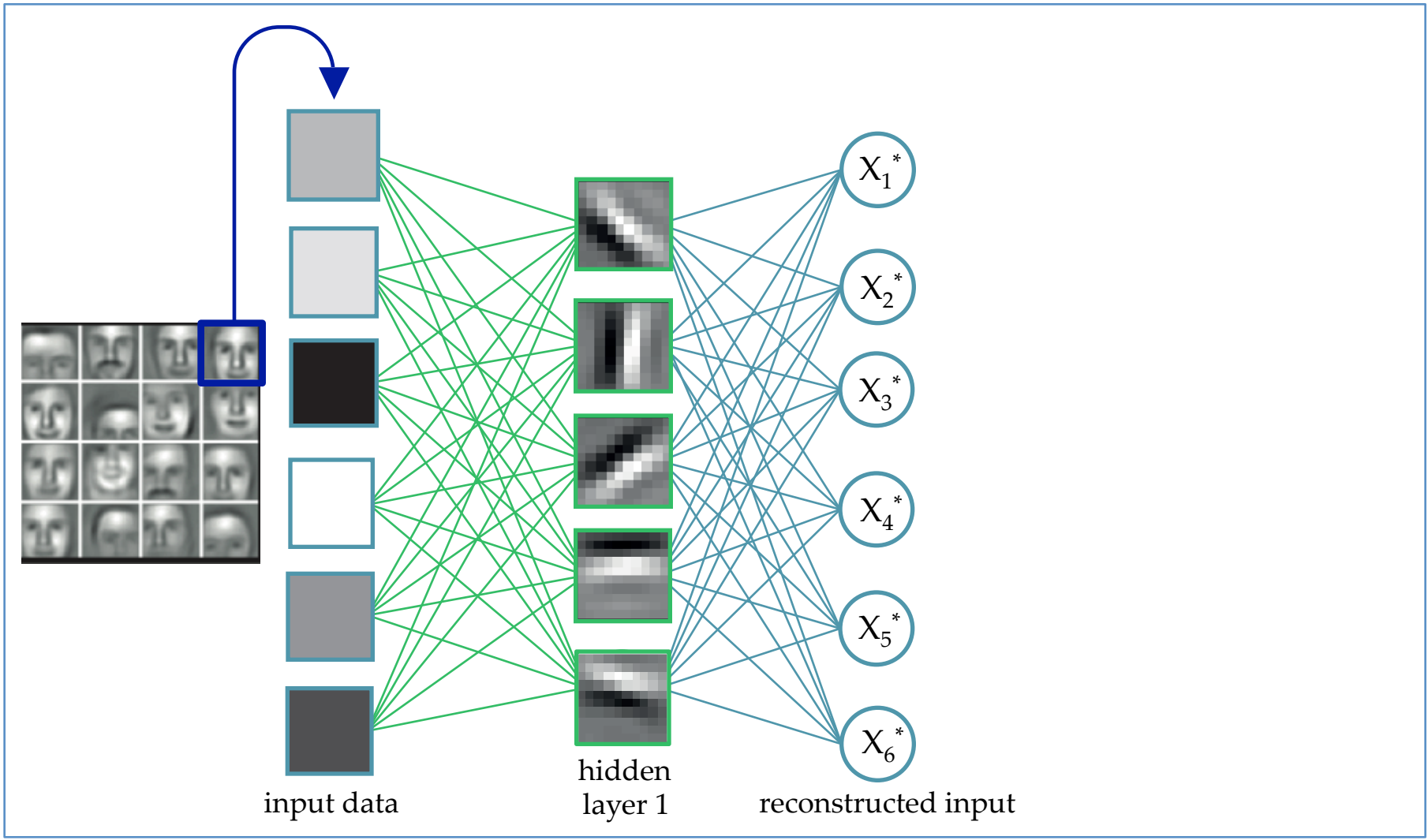
- Autoencoders and unsupervised pre-training
- Notes for lab
  - Scores
  - Dropout
- Introduction to convolutional neural networks
- Convolutional layers
- Dimensionality analysis

- Lab 7 check-in (fully connected NN part) TODAY
- Lab 7 due Monday
- Final project details (partners and proposal) posted tomorrow

Outline for April 10

- Autoencoders and unsupervised pre-training
- Notes for lab
  - Scores
  - Dropout
- Introduction to convolutional neural networks
- Convolutional layers
- Dimensionality analysis

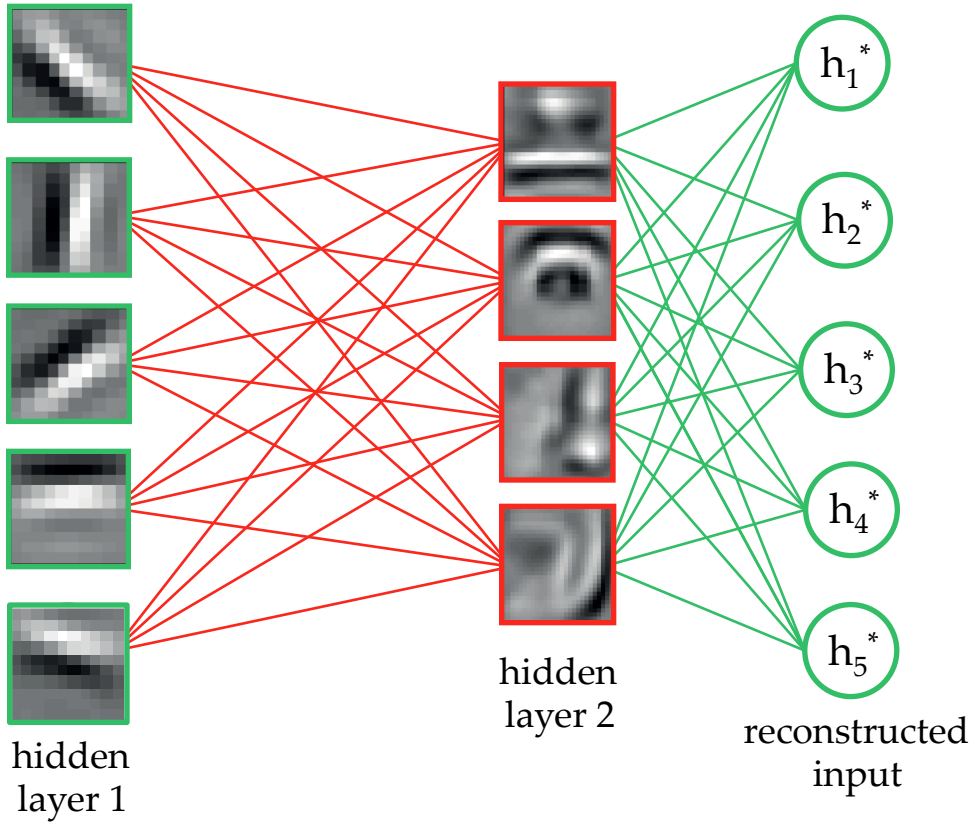
Hidden units can be interpreted as edges



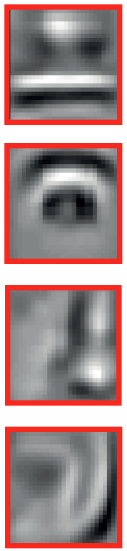
Now: throw away reconstruction and input

hidden
layer 1

Then repeat the entire process for each layer

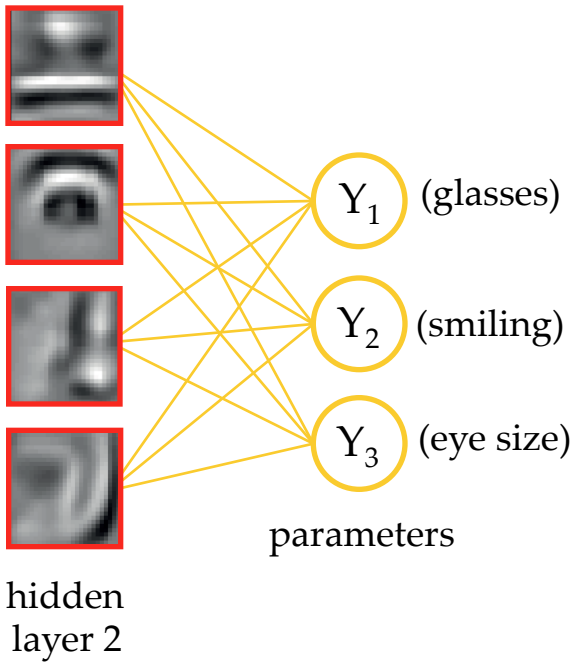


Then repeat the entire process for each layer

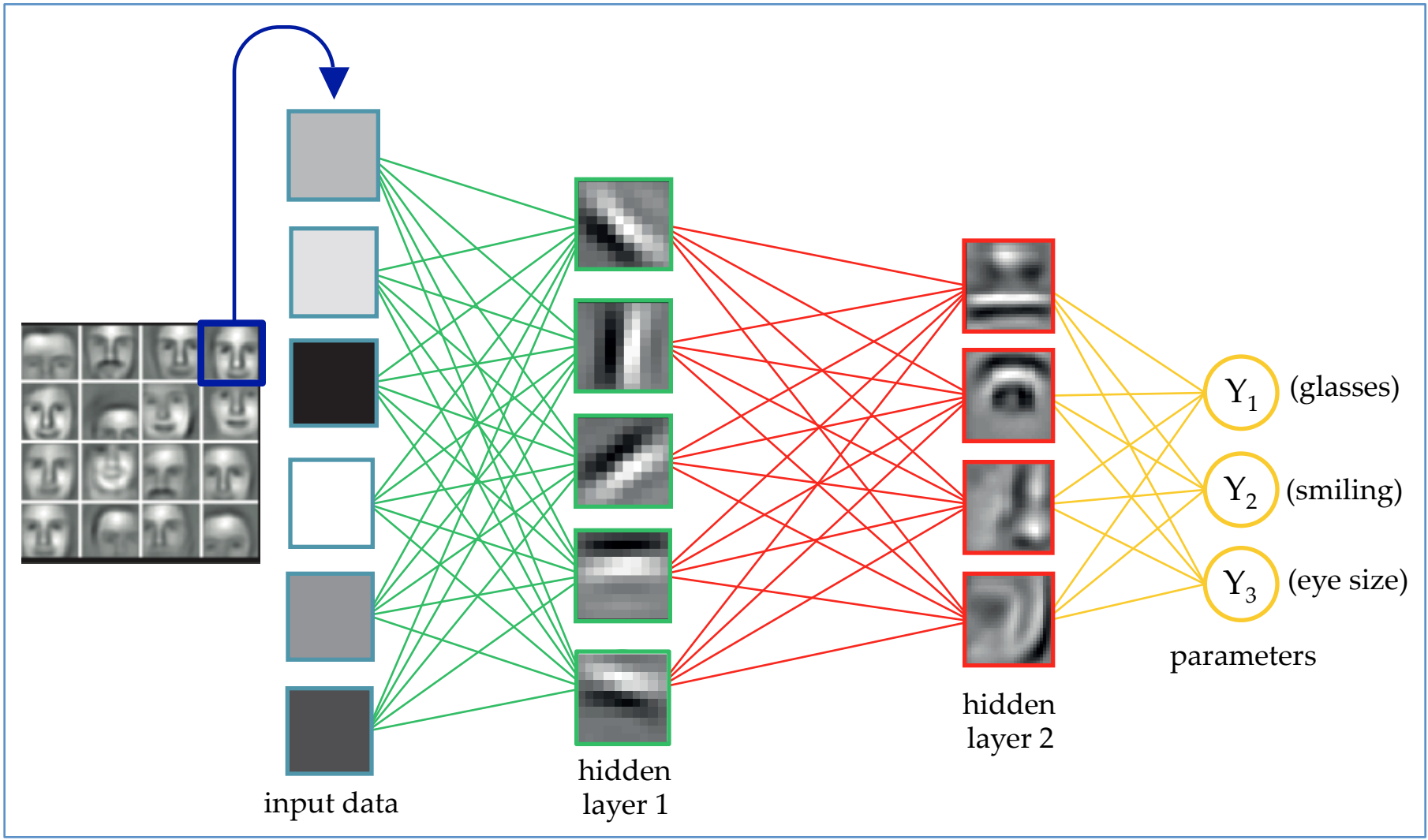


hidden
layer 2

In the last layer, use the outputs (supervised)

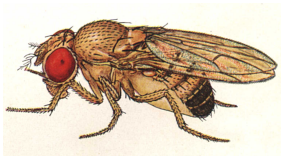


Finally, “fine-tune” the entire network! (supervised)

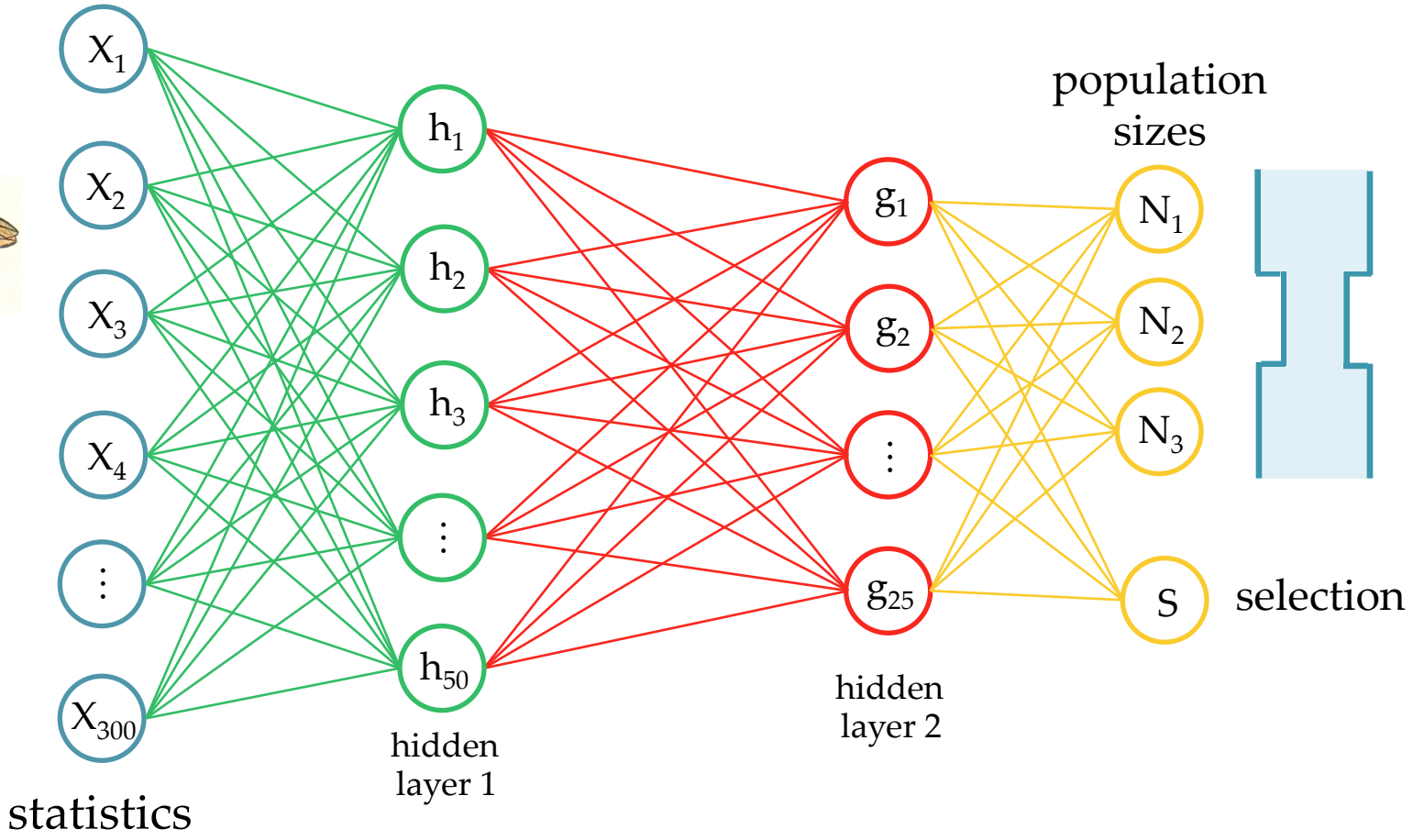


Learning genes under natural selection

(Example of unsupervised pre-training from my research)



GACTGGCTA
AGCTAGCTT
TAATCCGCA



"Deep learning for population genetic inference", Sheehan & Song (2016)

Learning genes under natural selection

(Example of unsupervised pre-training from my research)

True Class \ Predicted Class	No selection	New mutation	Existing mutation	Balancing
No selection				
New mutation				
Existing mutation				
Balancing				

Confusion matrix shows 4 different types of natural selection

"Deep learning for population genetic inference", Sheehan & Song (2016)

Learning genes under natural selection

(Example of unsupervised pre-training from my research)

True Class	Predicted Class: No selection	New mutation	Existing mutation	Balancing
No selection	1.000	0.000	0.000	0.000
New mutation	0.978	0.007	0.000	0.015
Existing mutation	1.000	0.000	0.000	0.000
Balancing	1.000	0.000	0.000	0.000

With random weight initialization

Learning genes under natural selection

(Example of unsupervised pre-training from my research)

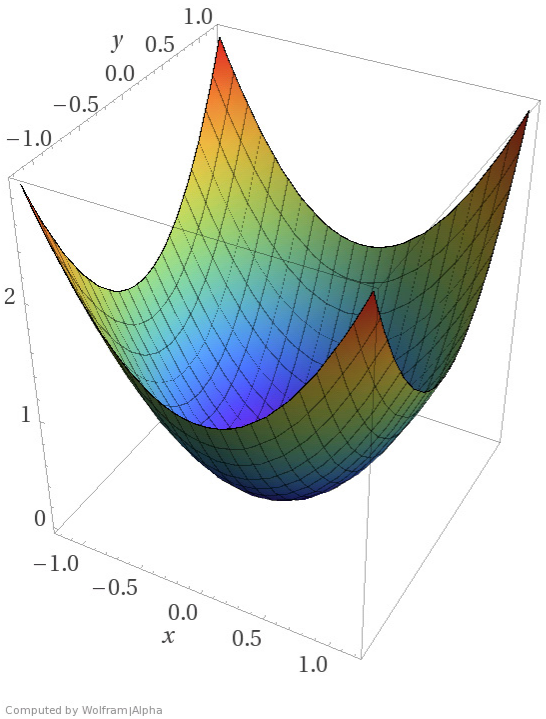
True Class \ Predicted Class	No selection	New mutation	Existing mutation	Balancing
No selection	1.000	0.000	0.000	0.000
New mutation	0.145	0.831	0.004	0.021
Existing mutation	0.011	0.001	0.987	0.000
Balancing	0.030	0.028	0.001	0.941

With unsupervised pre-training using autoencoders

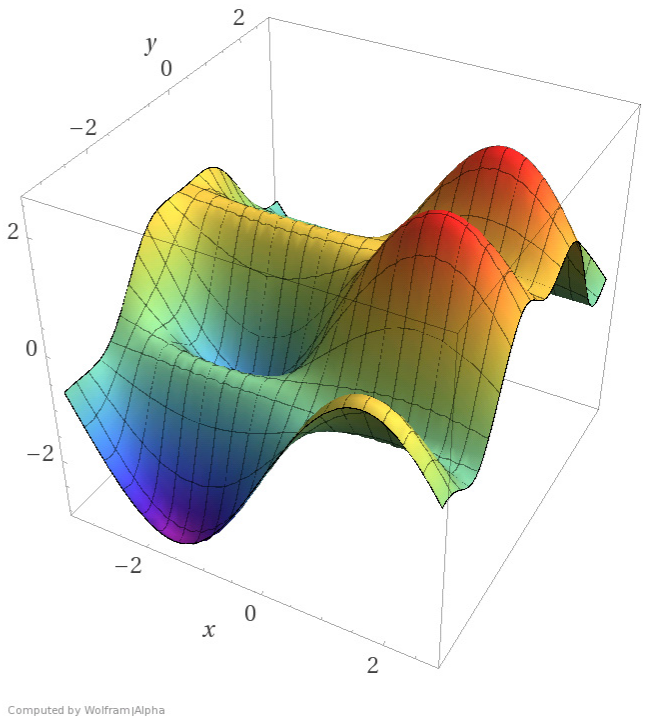
"Deep learning for population genetic inference", Sheehan & Song (2016)

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/

Outline for April 10

- Autoencoders and unsupervised pre-training
- Notes for lab
  - Scores
  - Dropout
- Introduction to convolutional neural networks
- Convolutional layers
- Dimensionality analysis

Notes about scores and softmax

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

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

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

- 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 *C* (number of classes)

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

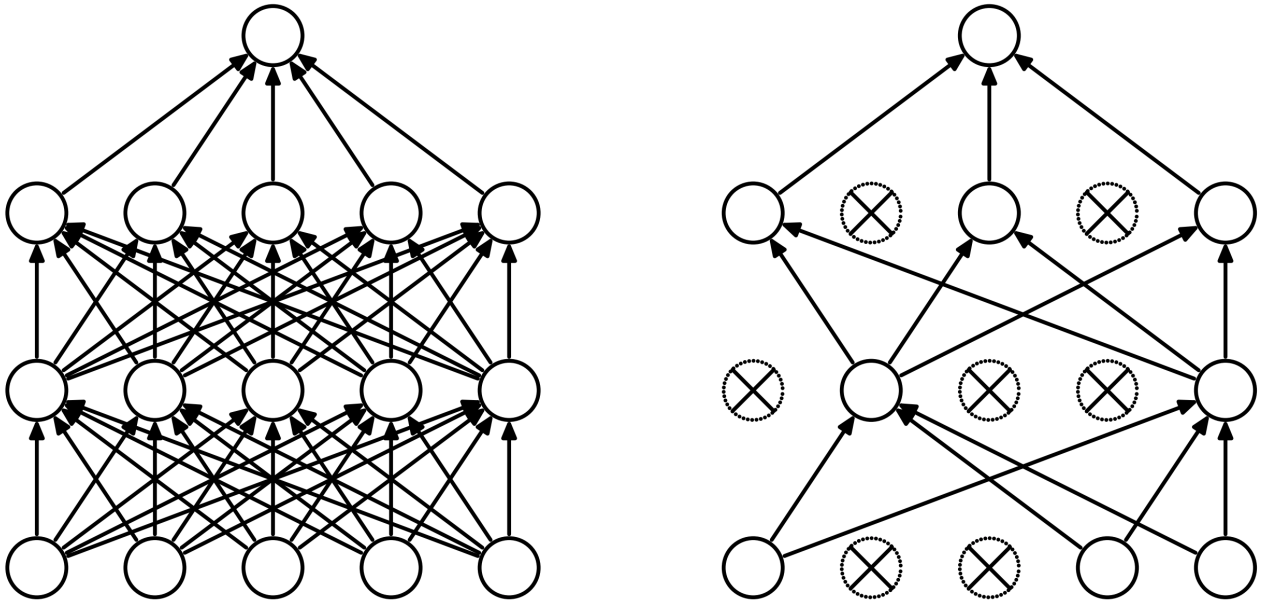
$$\hat{y}_k = \frac{e^{s_k}}{\sum_{j=1}^{C} e^{s_j}}$$

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

Notes about 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 April 10

- Autoencoders and unsupervised pre-training
- Notes for lab
  - Scores
  - Dropout
- Introduction to convolutional neural networks
- Convolutional layers
- Dimensionality analysis

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

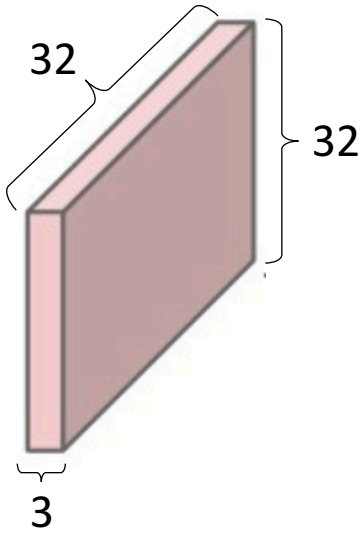


Image: modified from Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

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

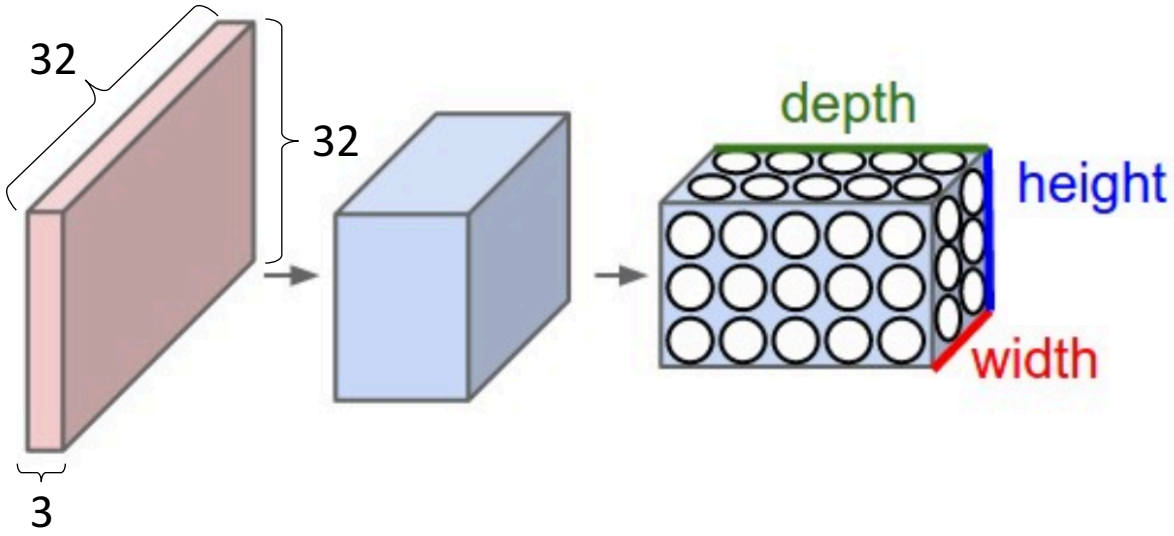


Image: modified from Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

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)

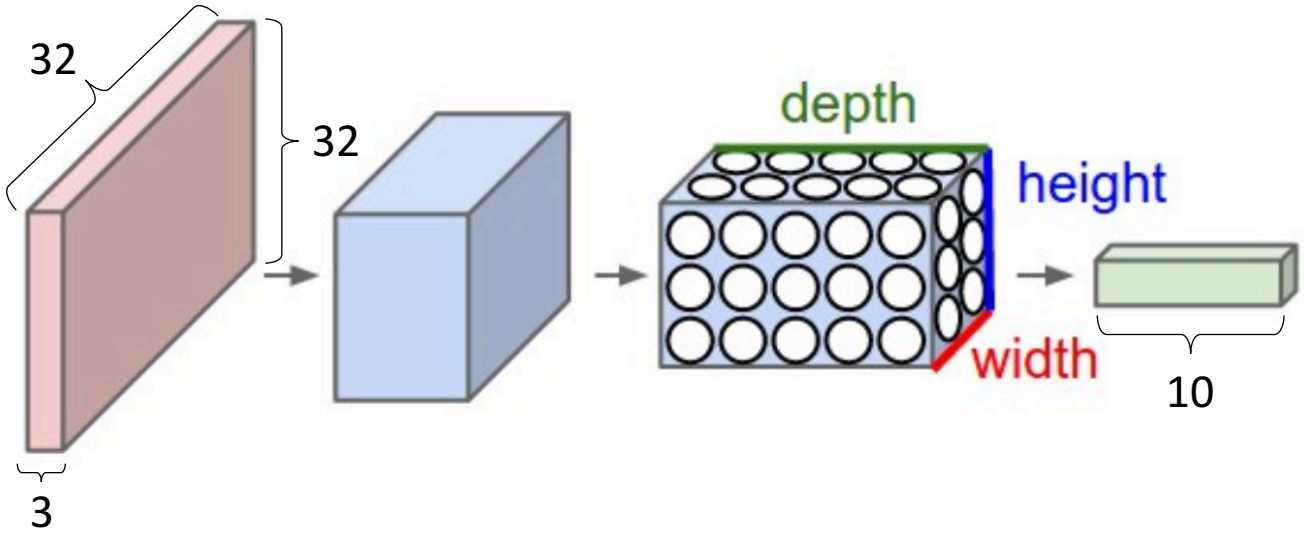


Image: modified from Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

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

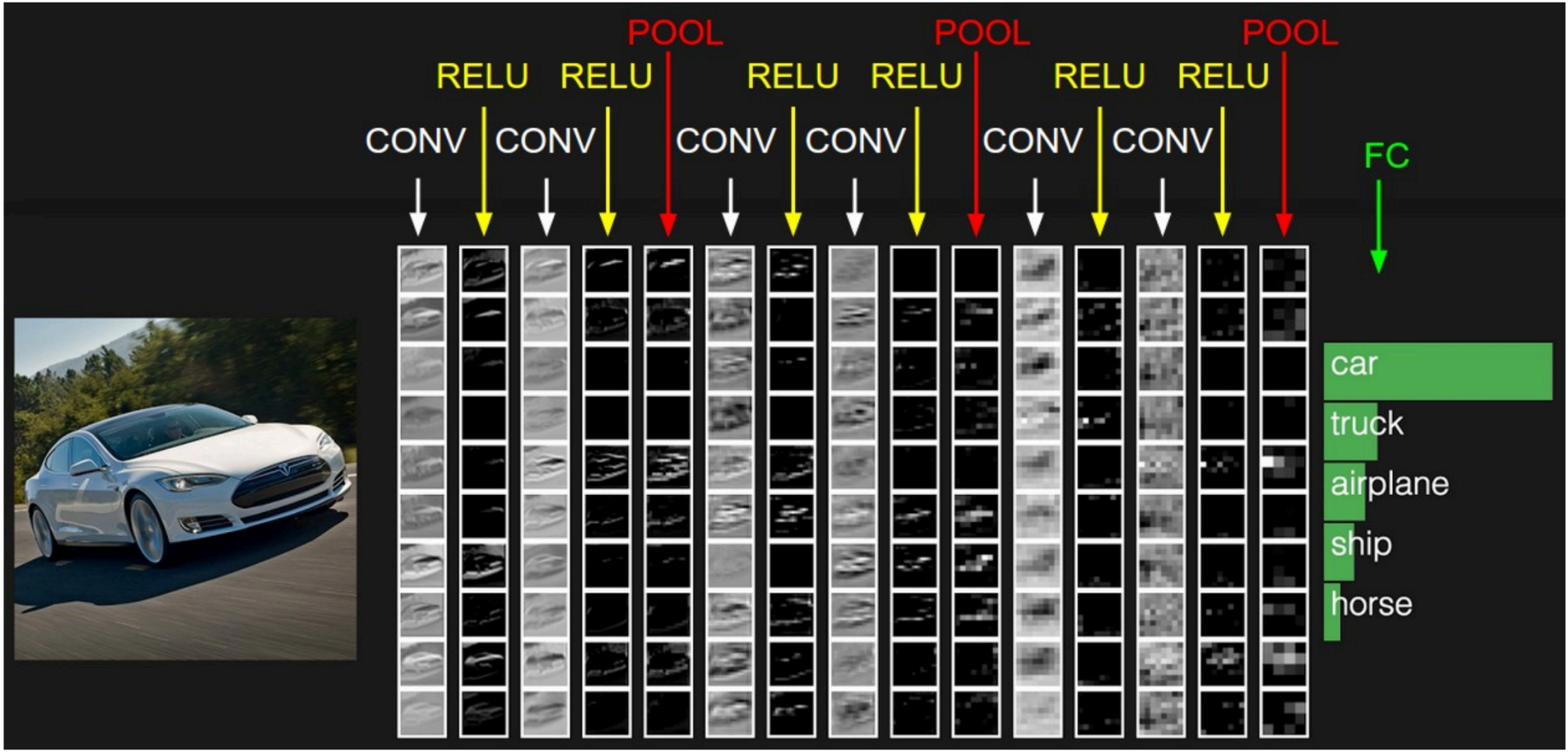
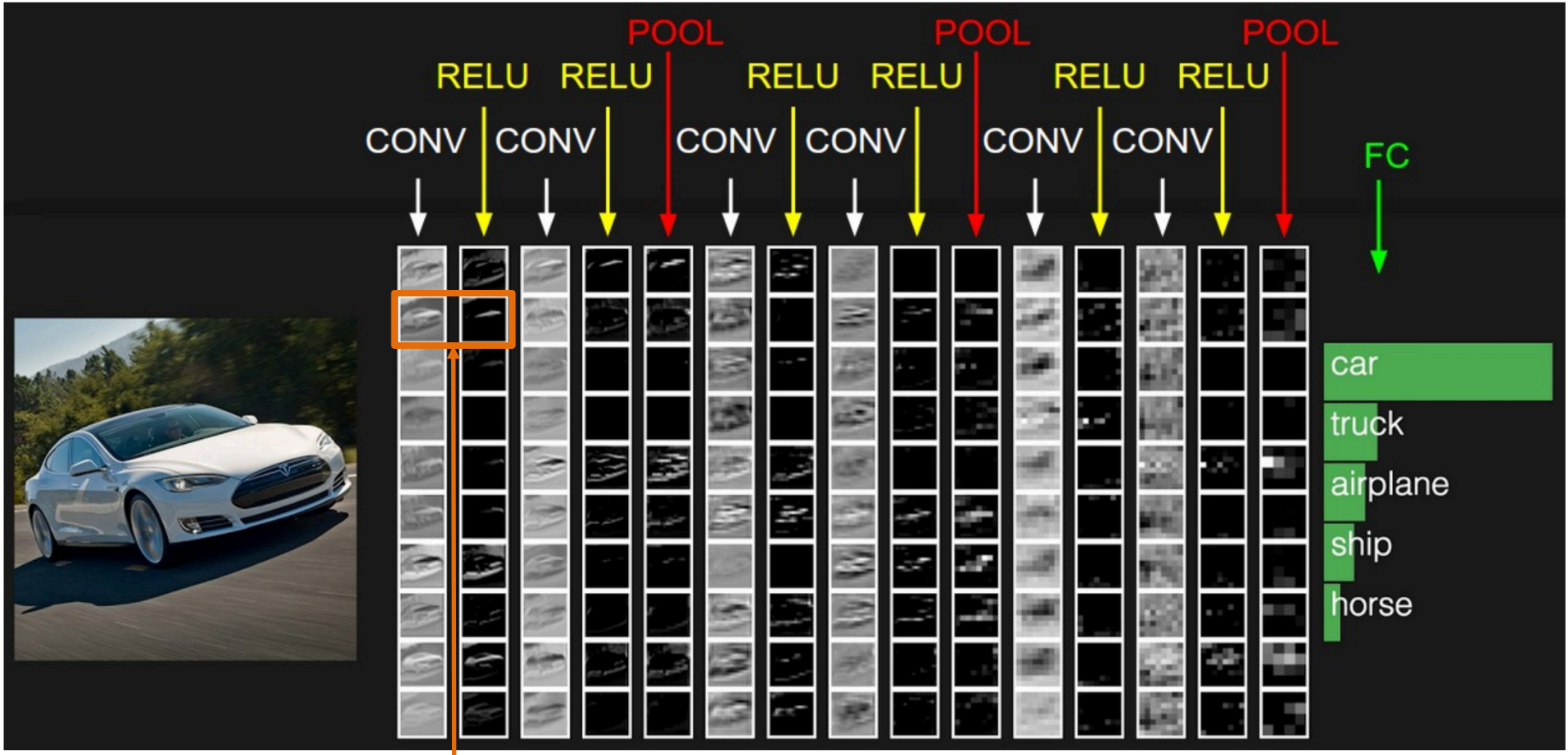


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

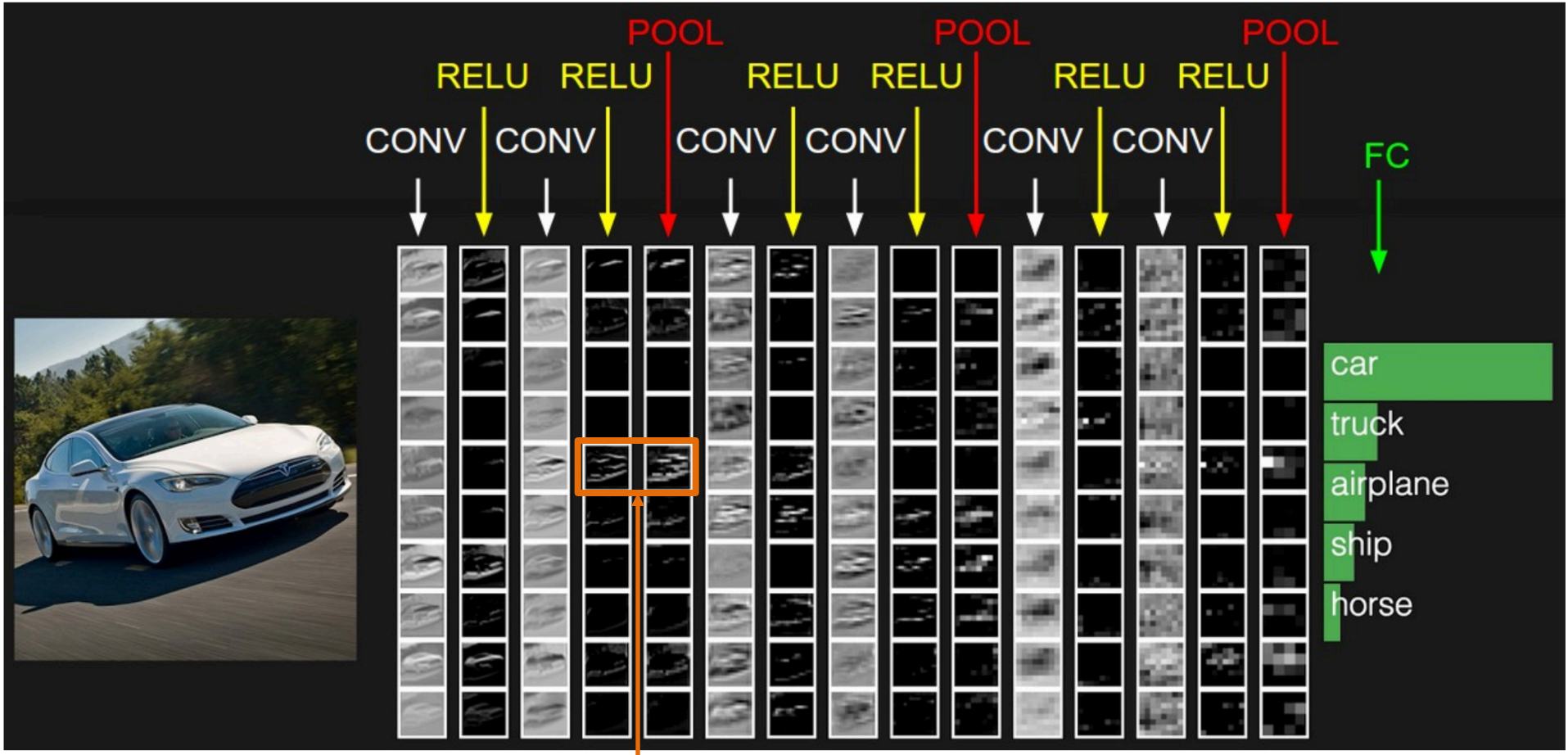
Example CNN architecture



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

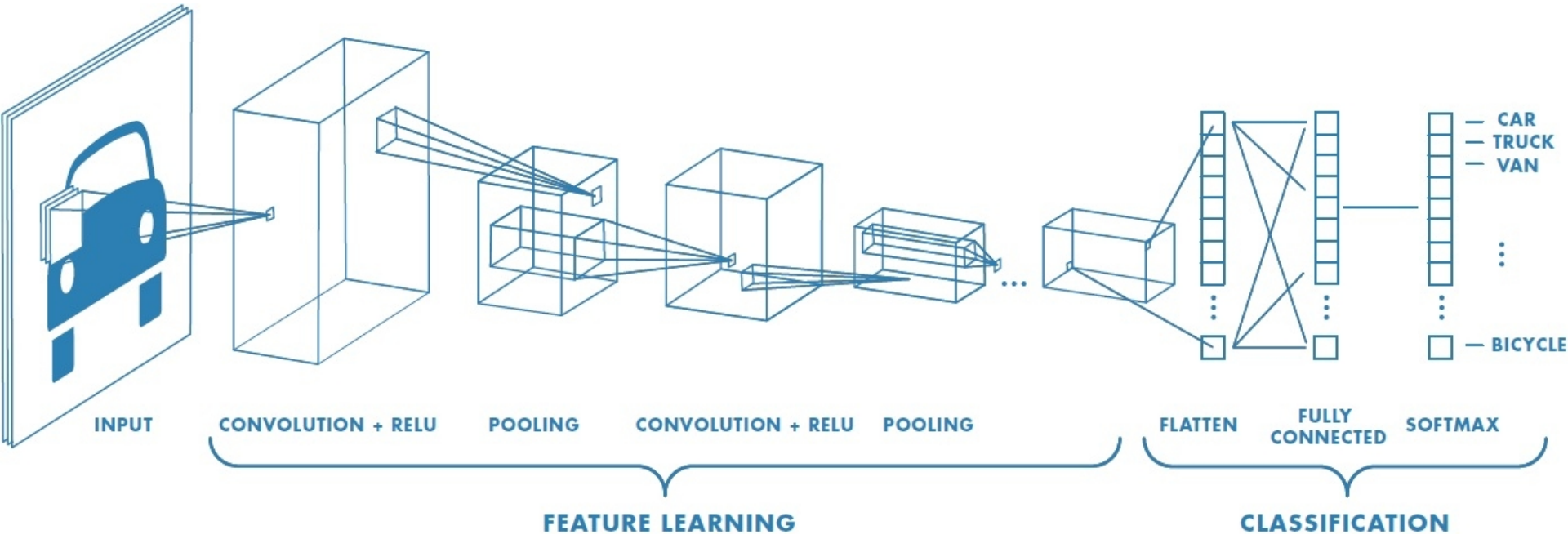


Image from MathWorks: https://www.mathworks.com/solutions/deep-learning/convolutional-neural-network.html

Outline for April 10

- Autoencoders and unsupervised pre-training
- Notes for lab
  - Scores
  - Dropout
- Introduction to convolutional neural networks
- Convolutional layers
- Dimensionality analysis

Idea: local “receptive field”

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

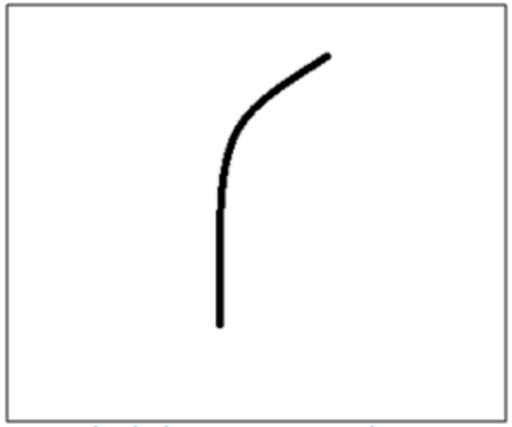
- Note an important *asymmetry*: we will look at a small “patch” of the image relative to its width and height, but we will look all the way through the depth!

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

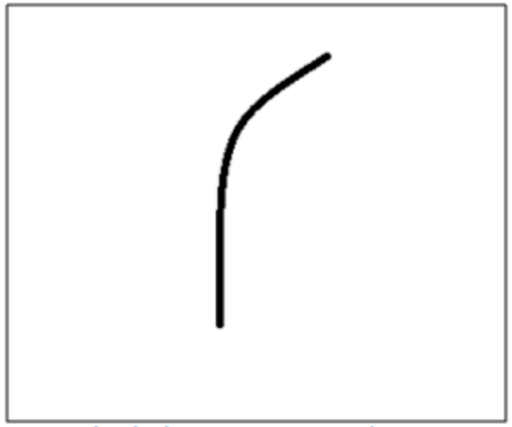
Images by: Adit Deshpande

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

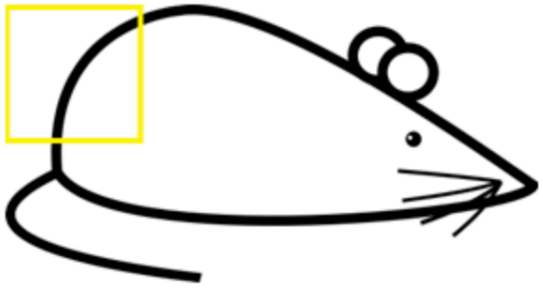
Pixel representation of filter



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

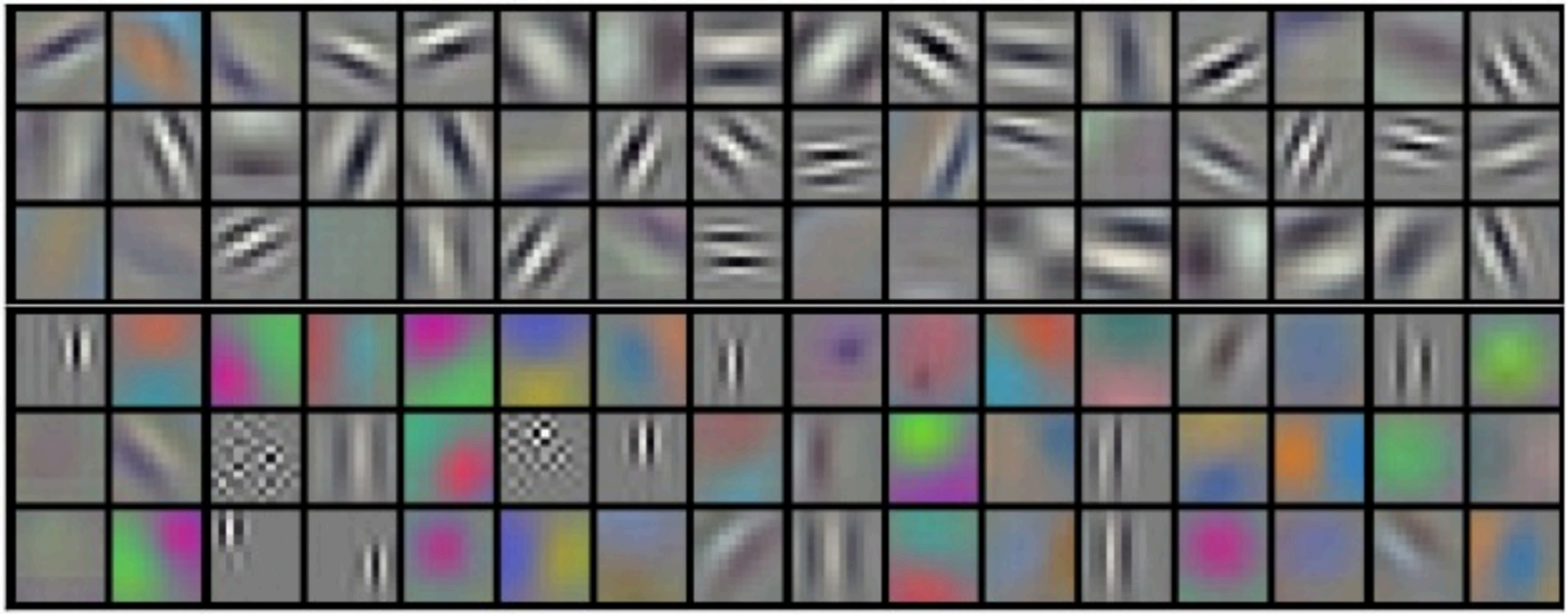
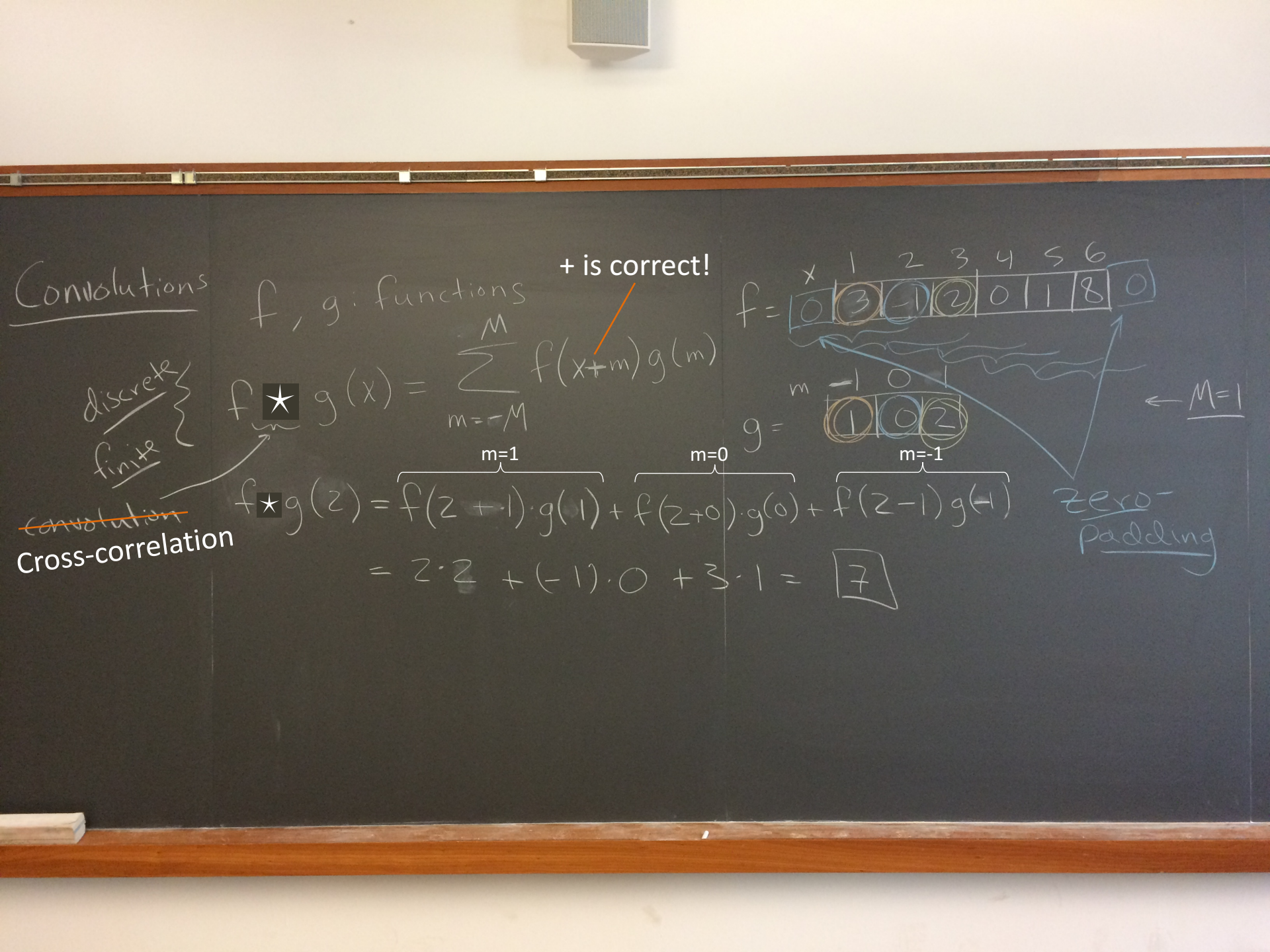


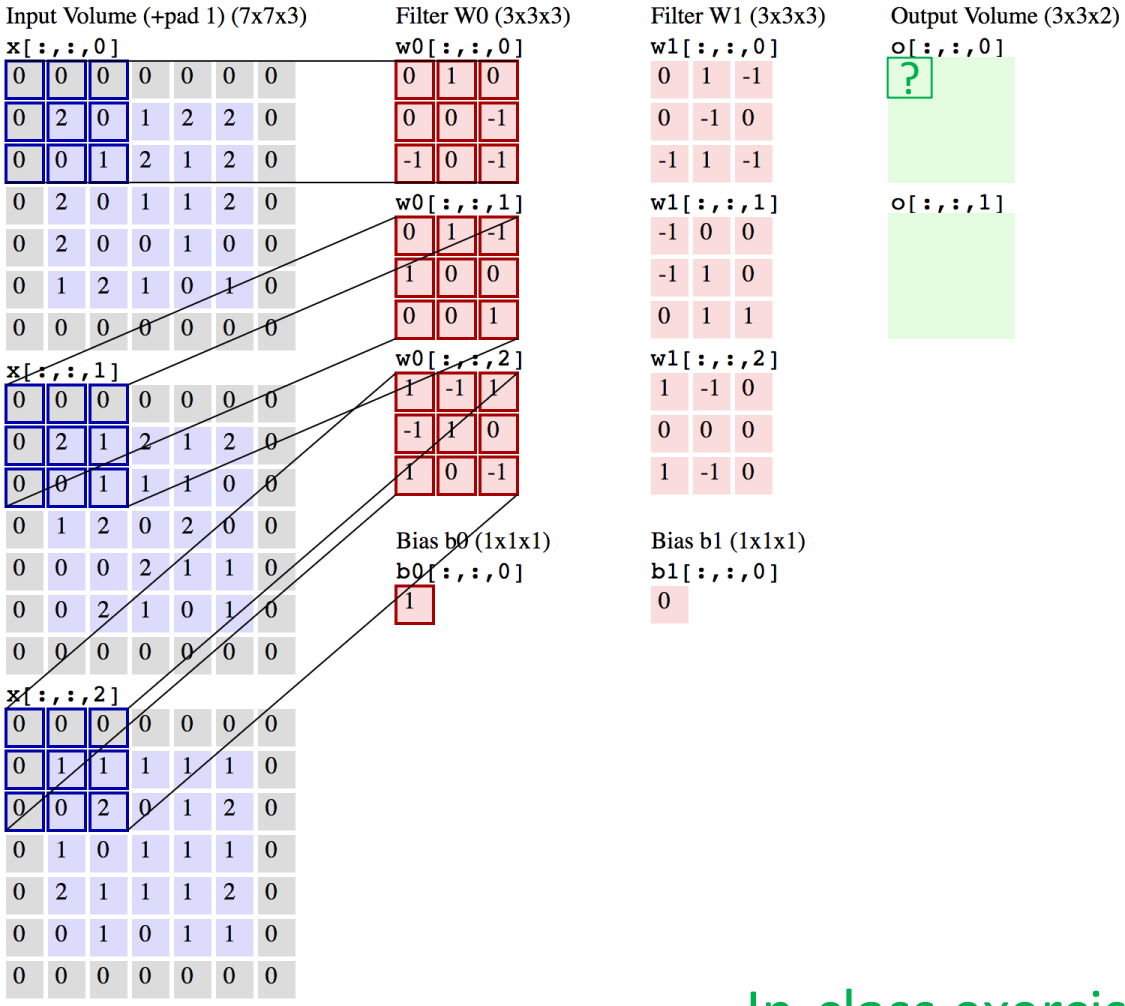
Image: Krizhevsky et al. (2012) https://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

Math behind convolutions

(actually cross-correlations!)

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


Convolutions
f, g : functions
+ is correct!
discrete
finite
$f \star g(x) = \sum_{m=-M}^{M} f(x+m)\, g(m)$
~~convolution~~
Cross-correlation
x 1 2 3 4 5 6
f = 0 3 -1 2 0 1 8 0
m -1 0 1
g = 1 0 2
← M=1
m=1
m=0
m=-1
$f \star g(2) = f(2+1)\cdot g(1) + f(2+0)\cdot g(0) + f(2-1)\, g(-1)$
$= 2\cdot 2 + (-1)\cdot 0 + 3\cdot 1 = 7$
zero-padding



In-class exercise

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

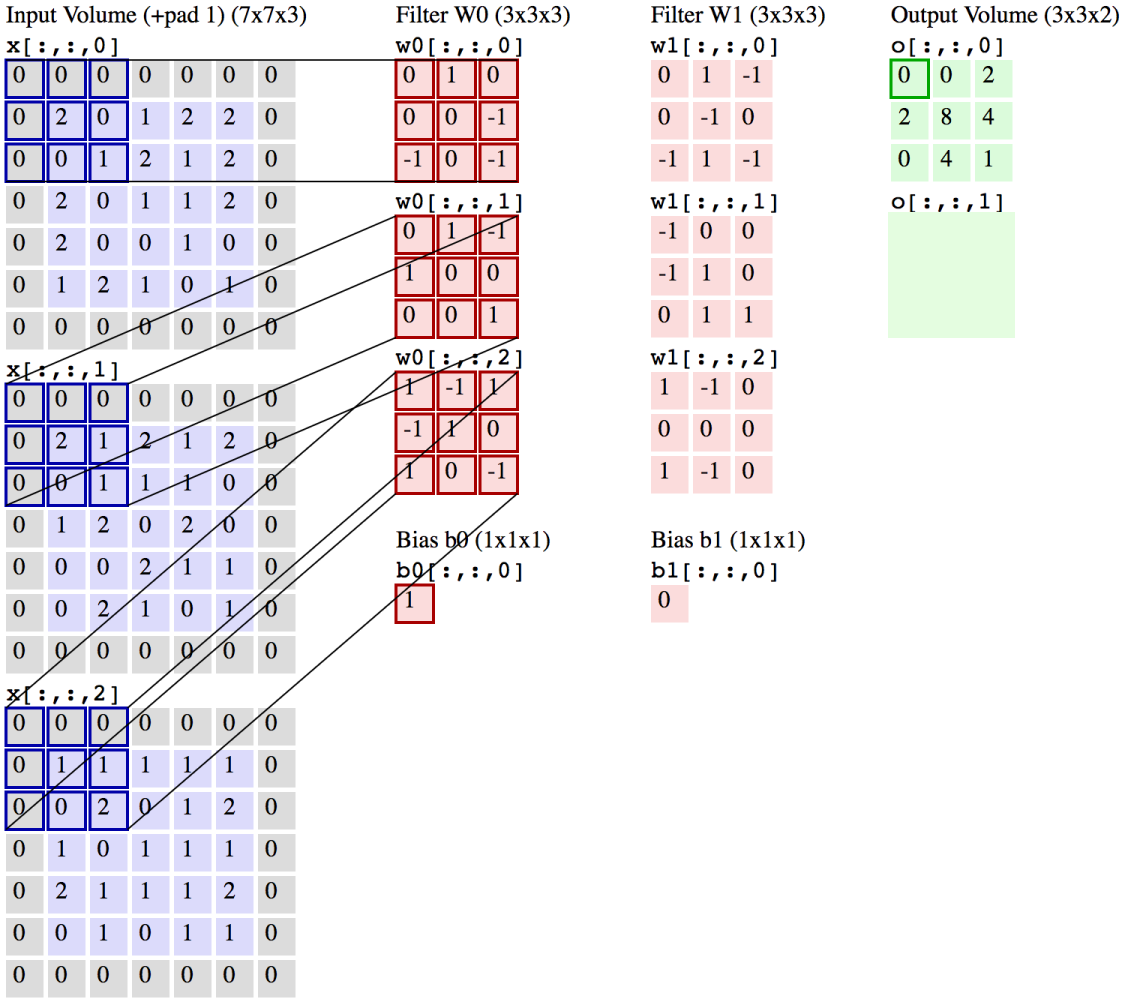


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

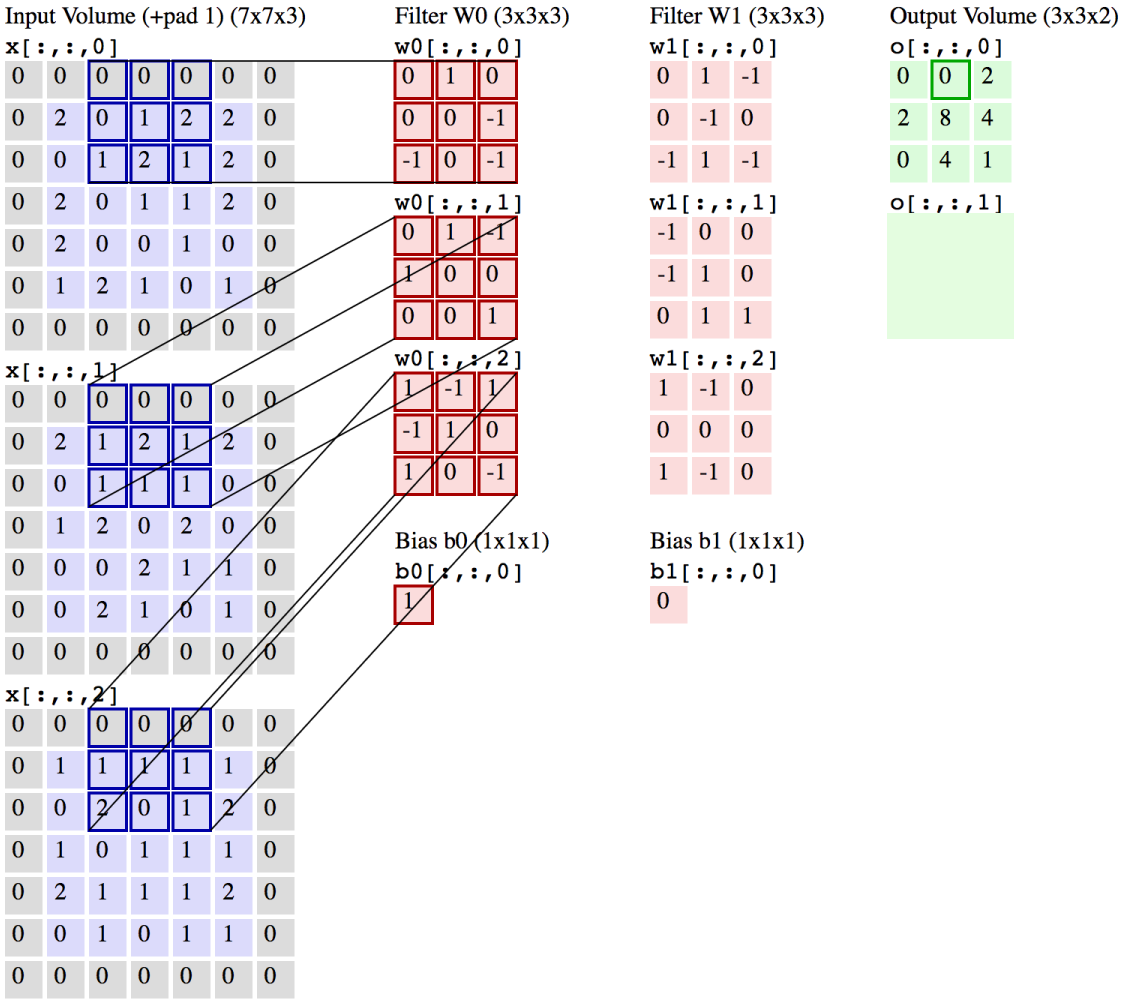


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

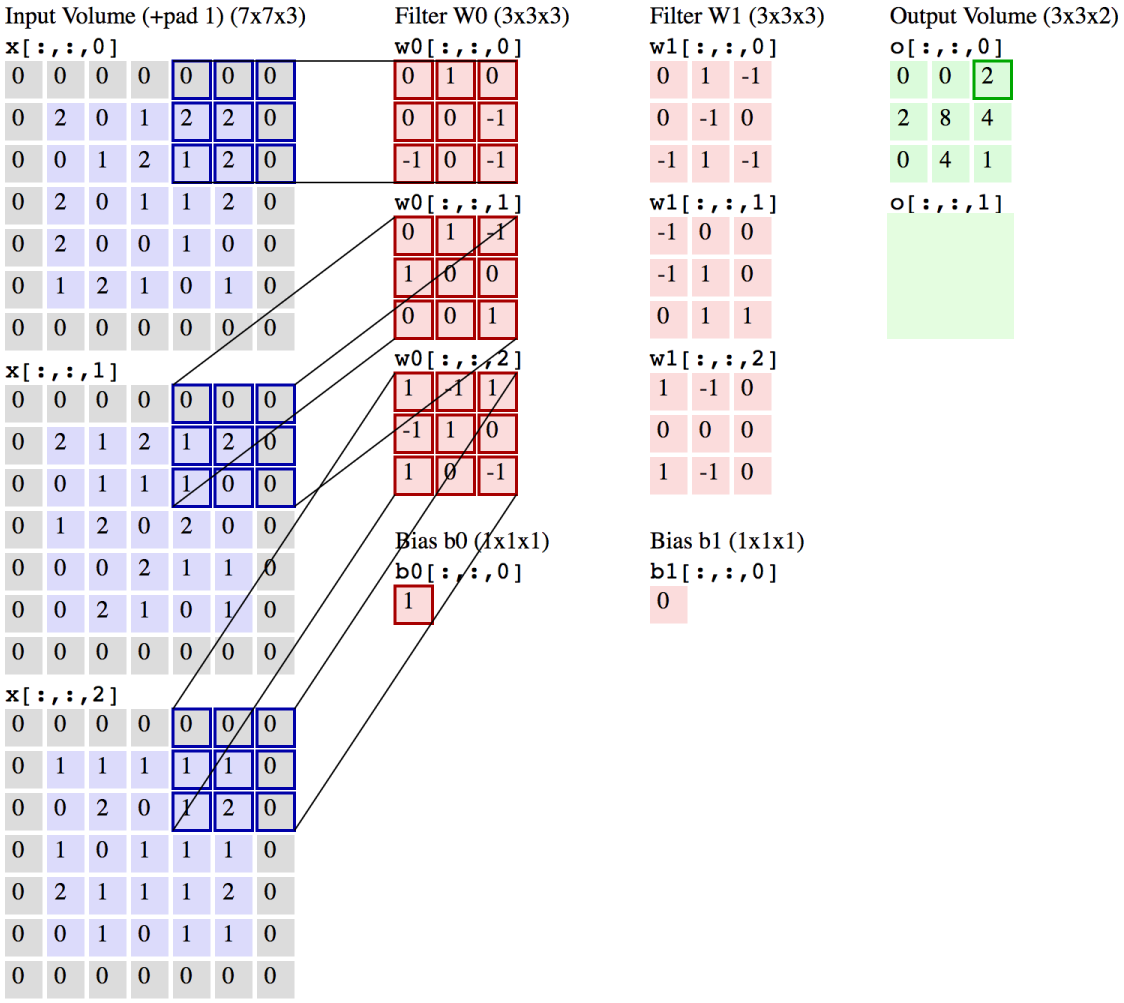


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

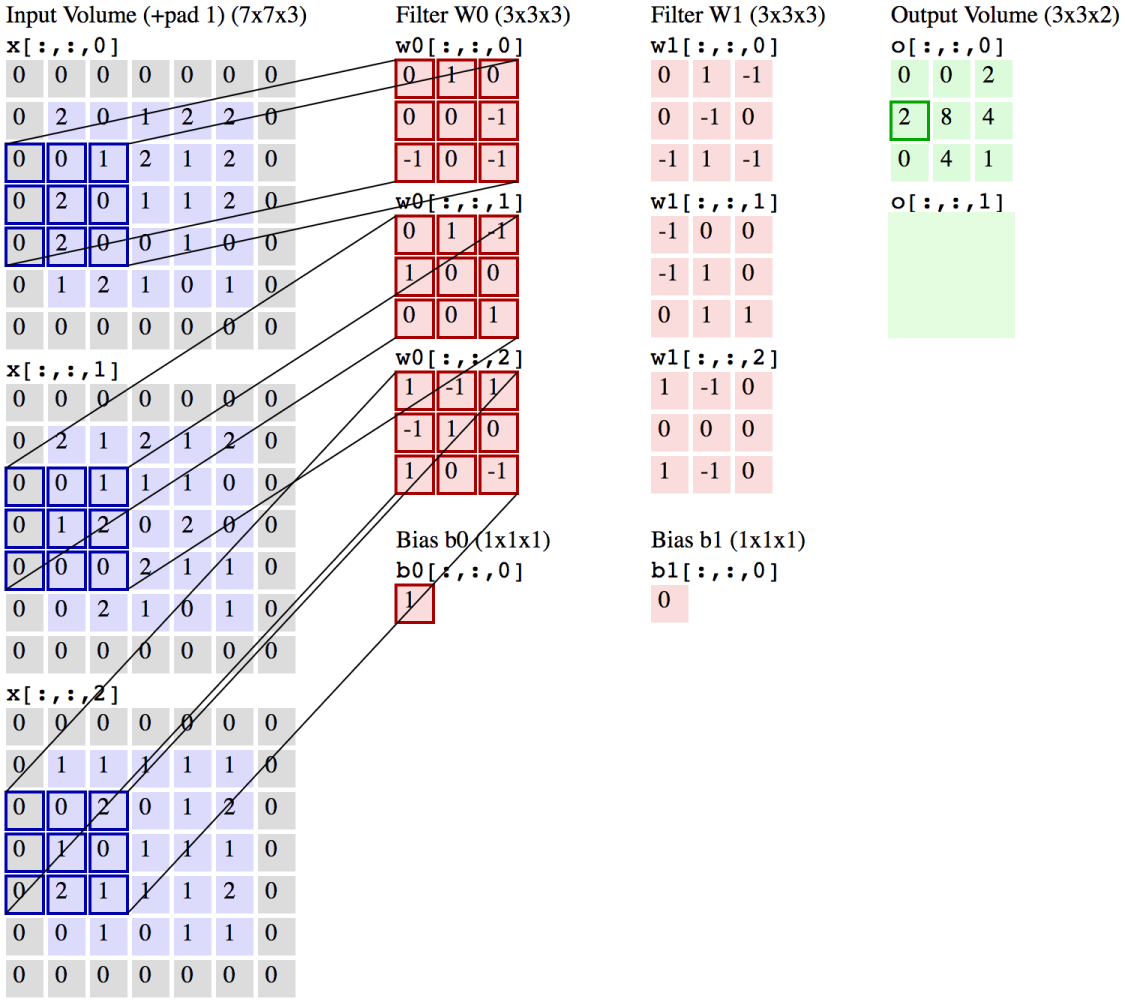


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

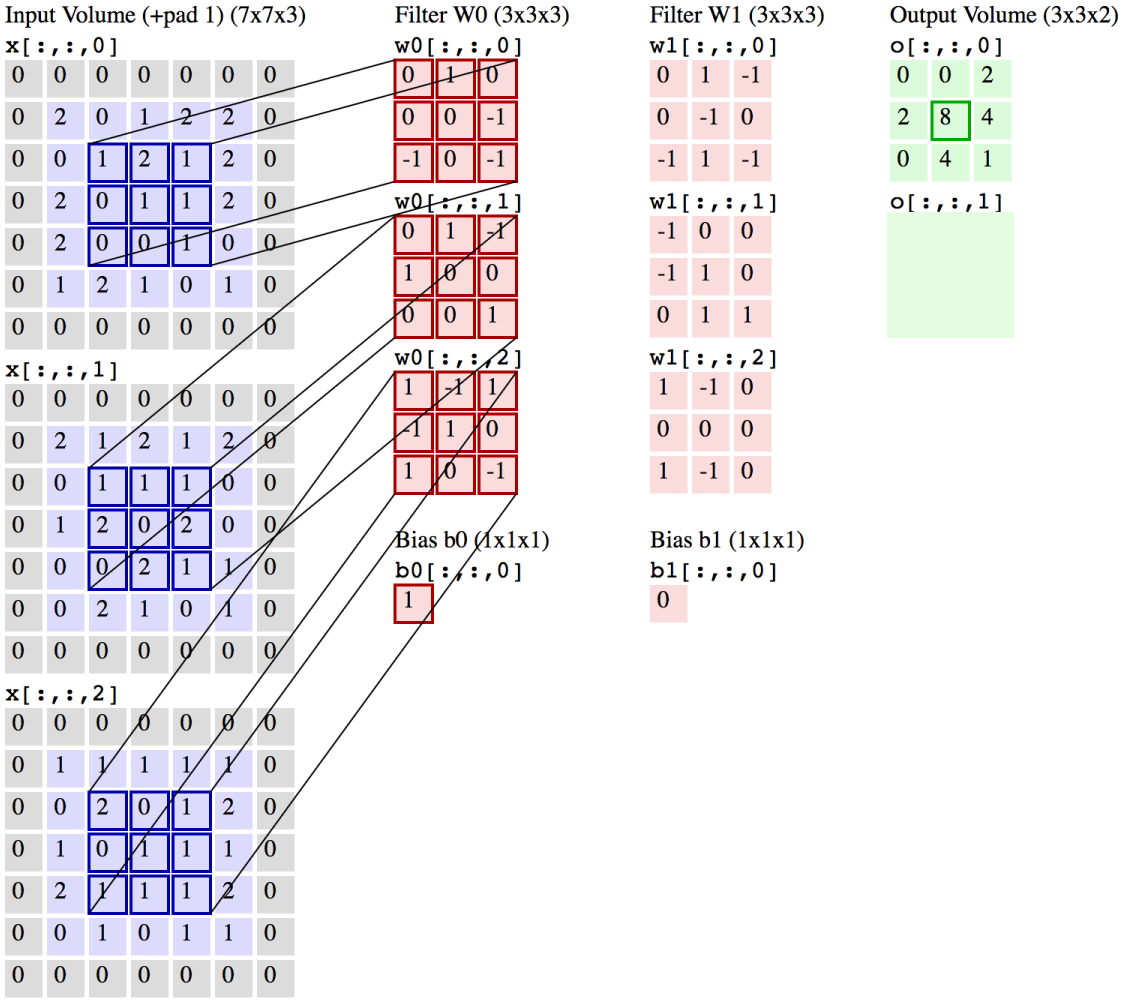


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

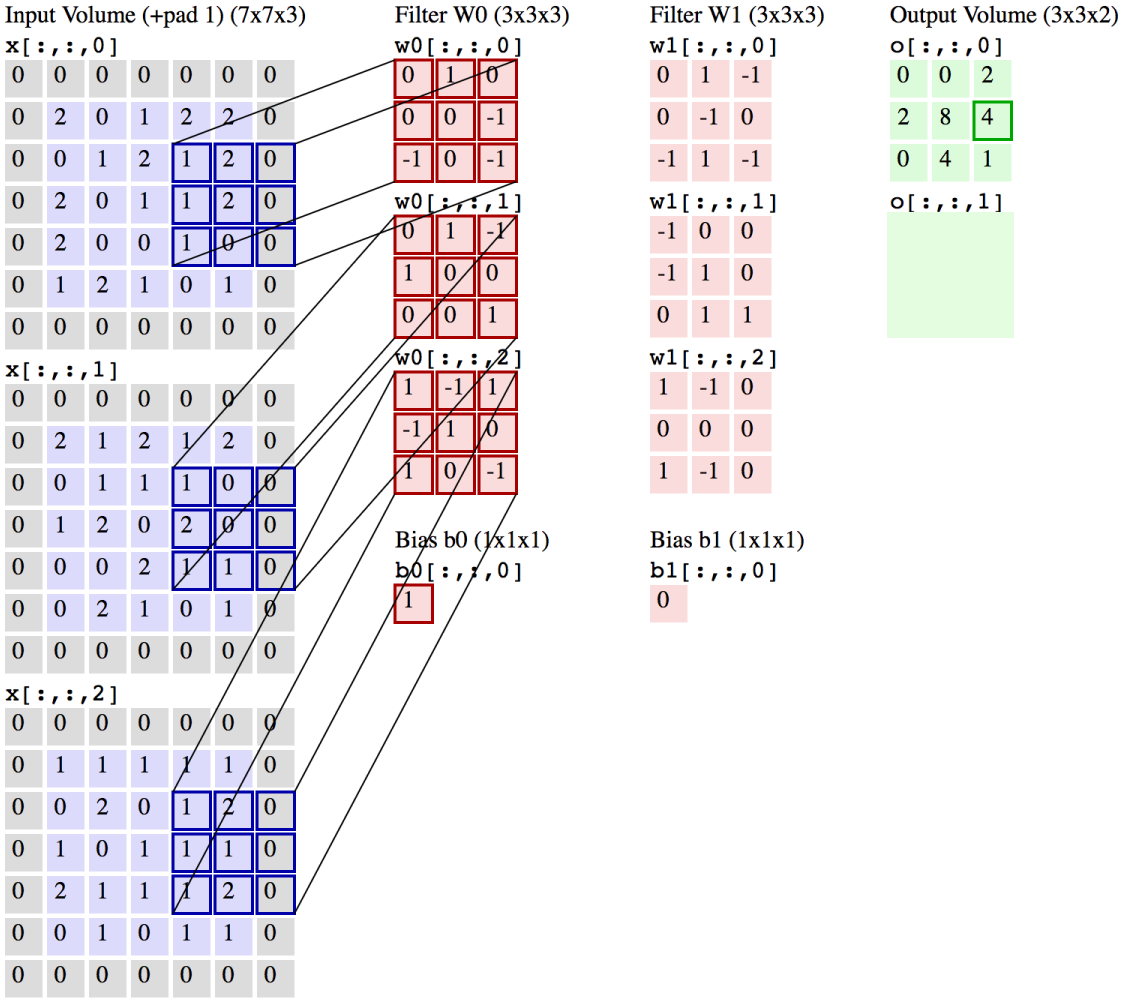


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

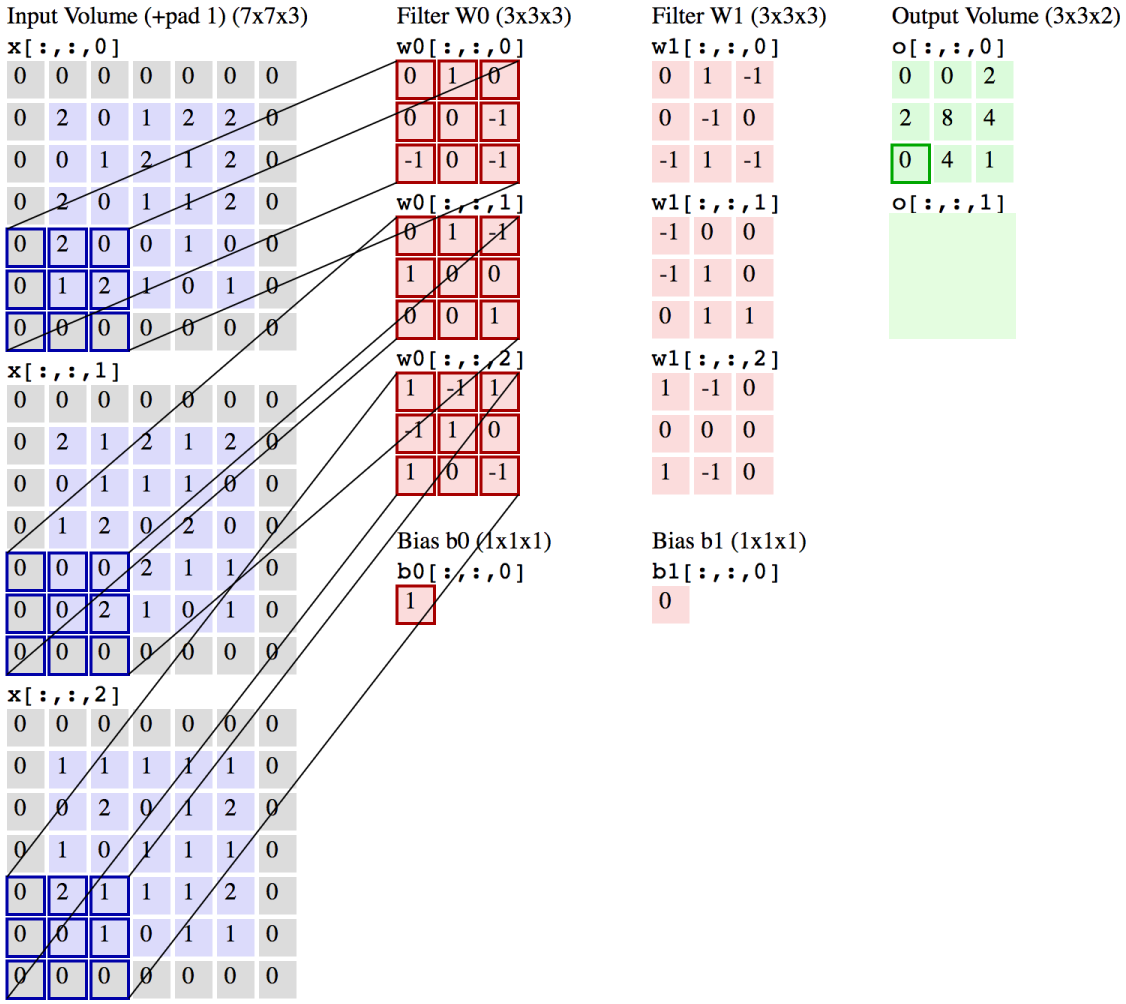


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

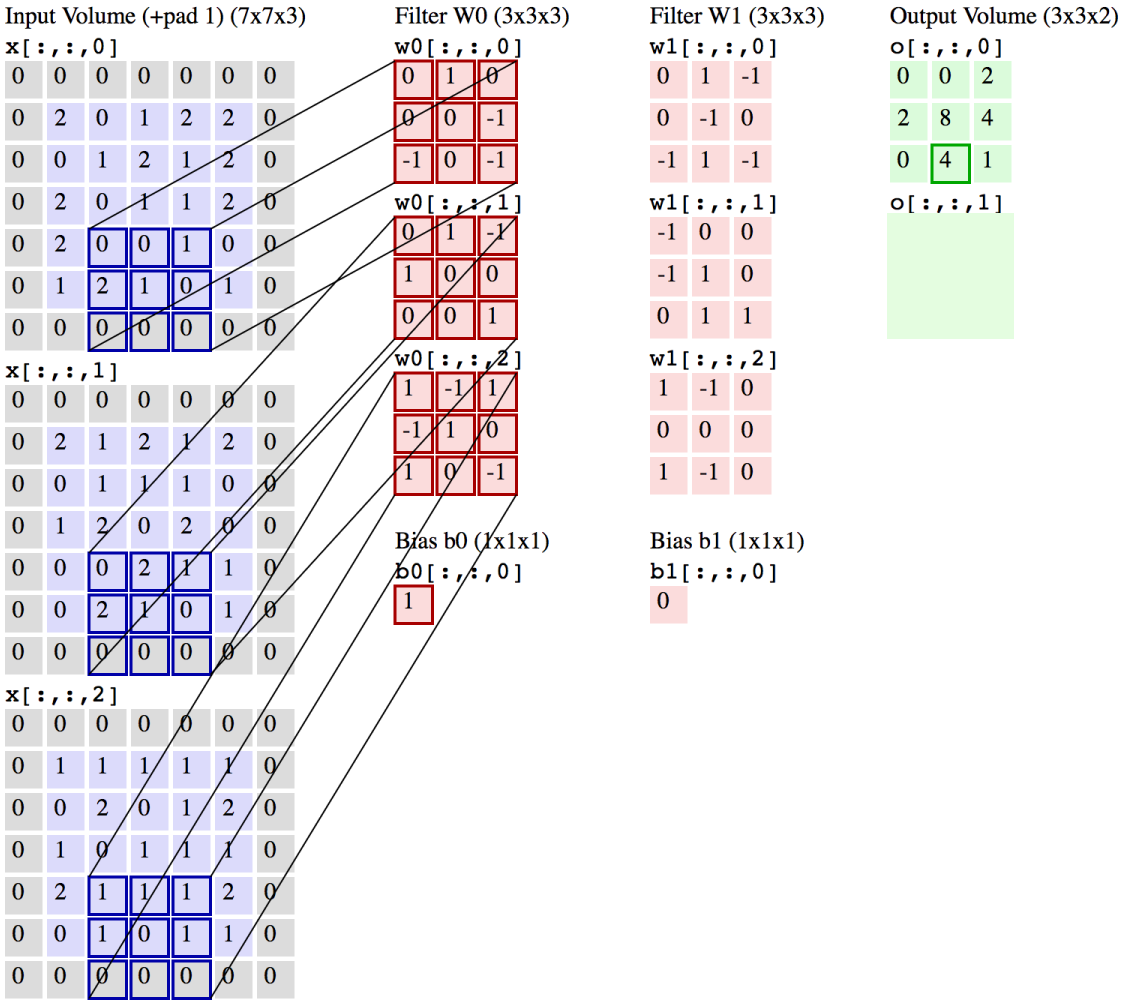


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

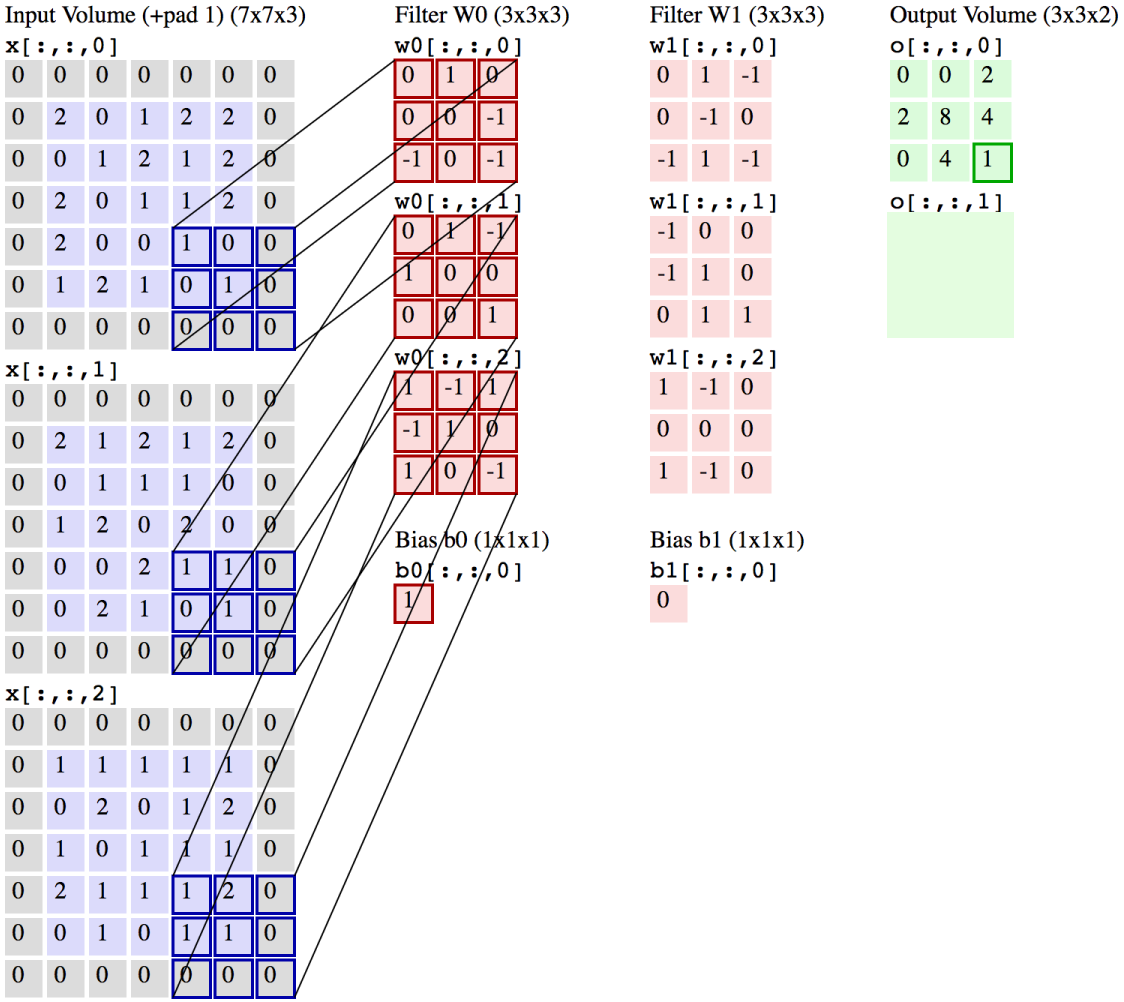


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

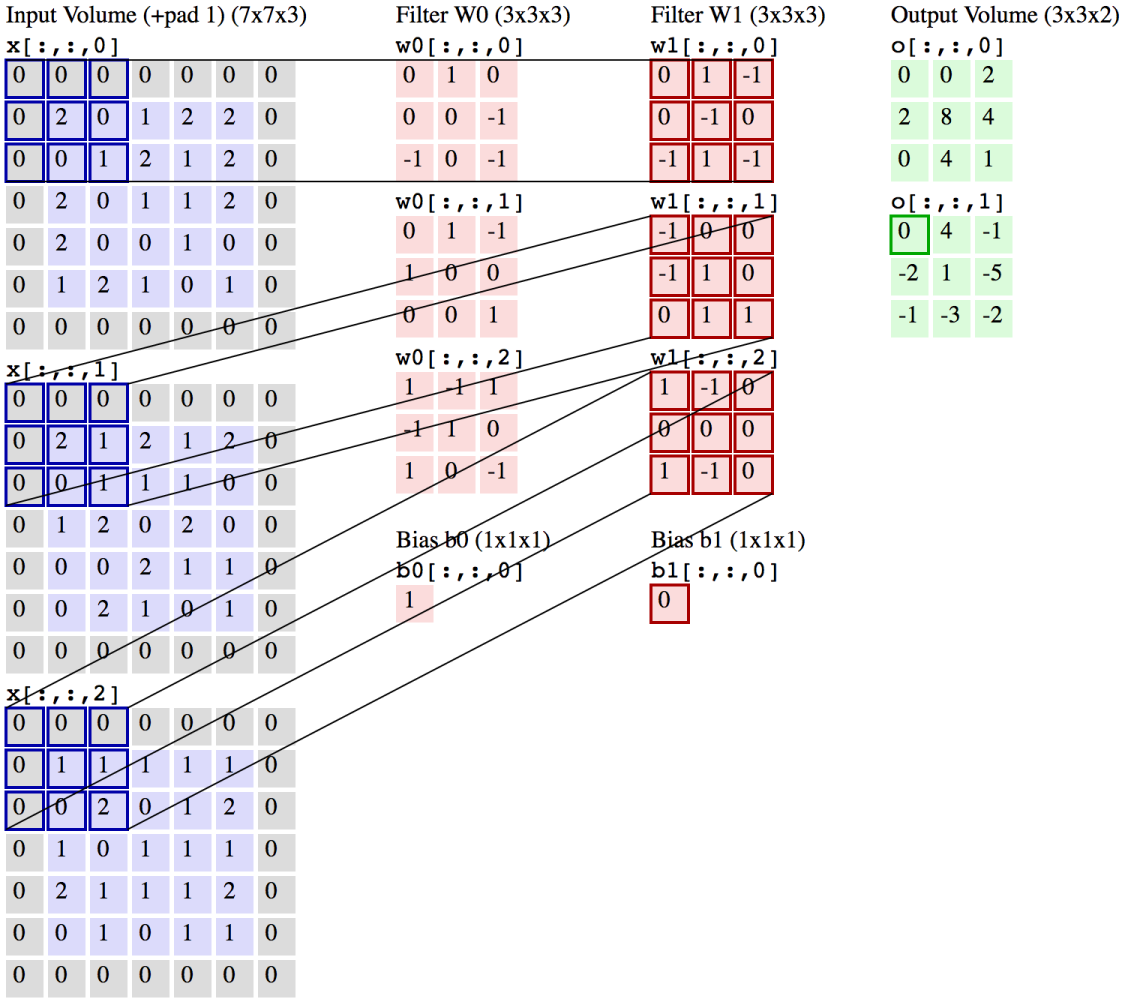


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

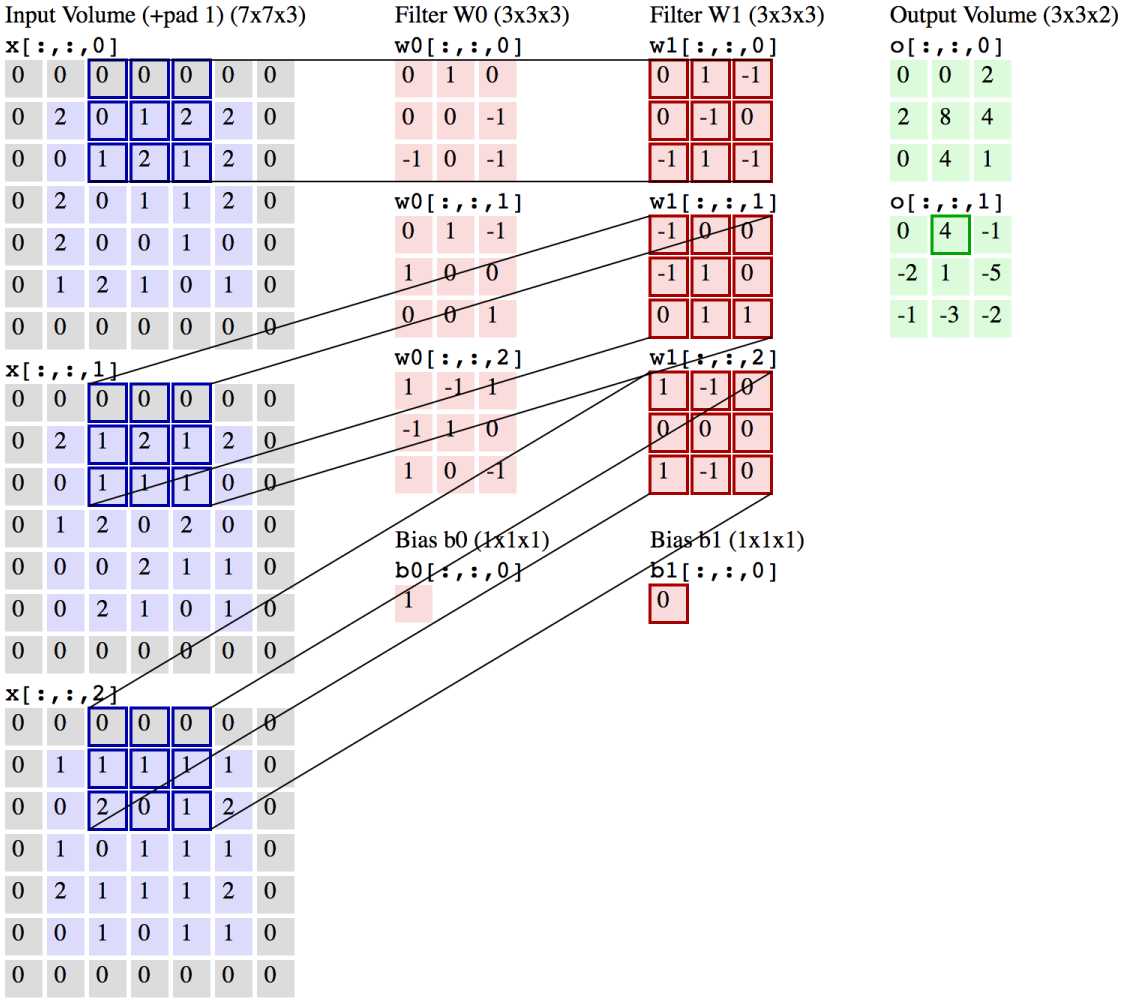


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

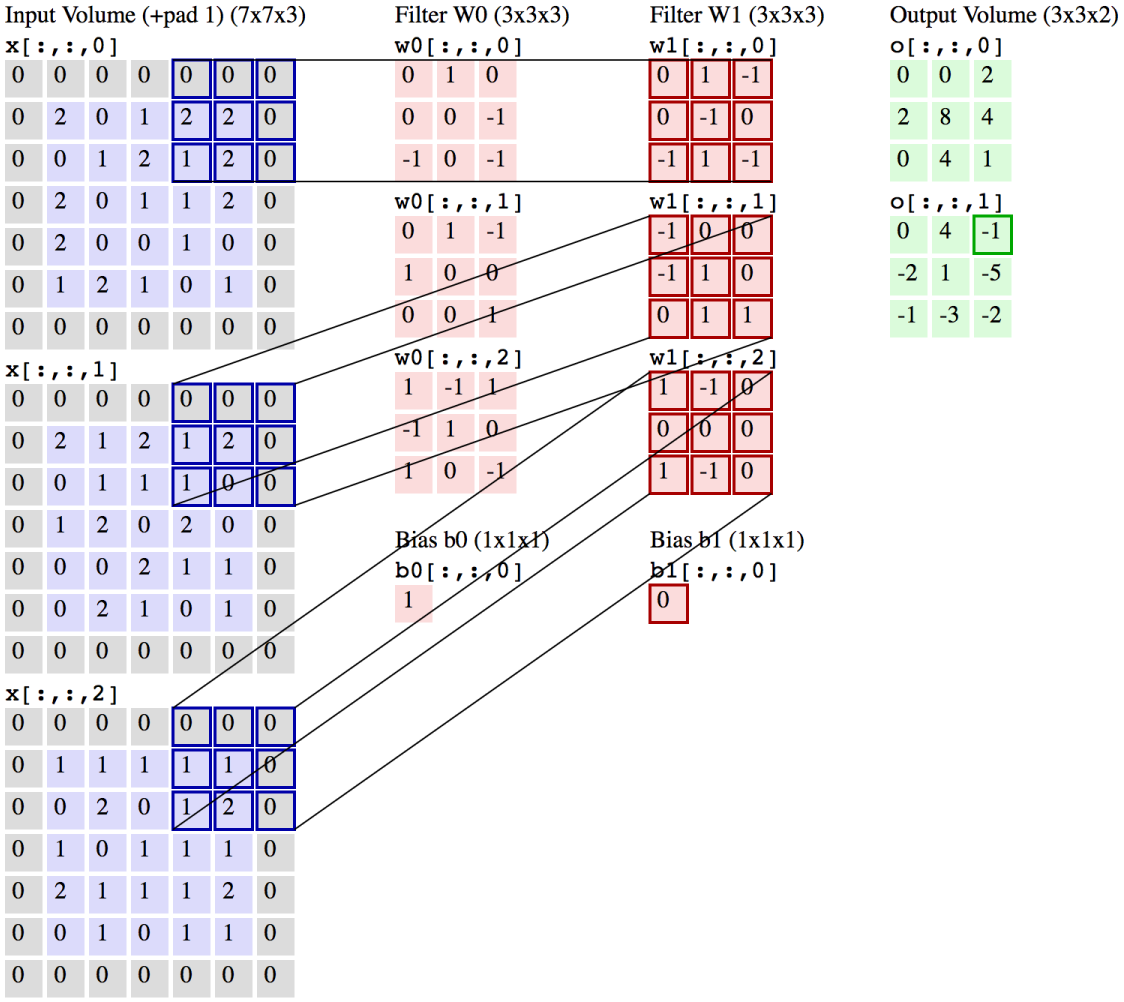


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

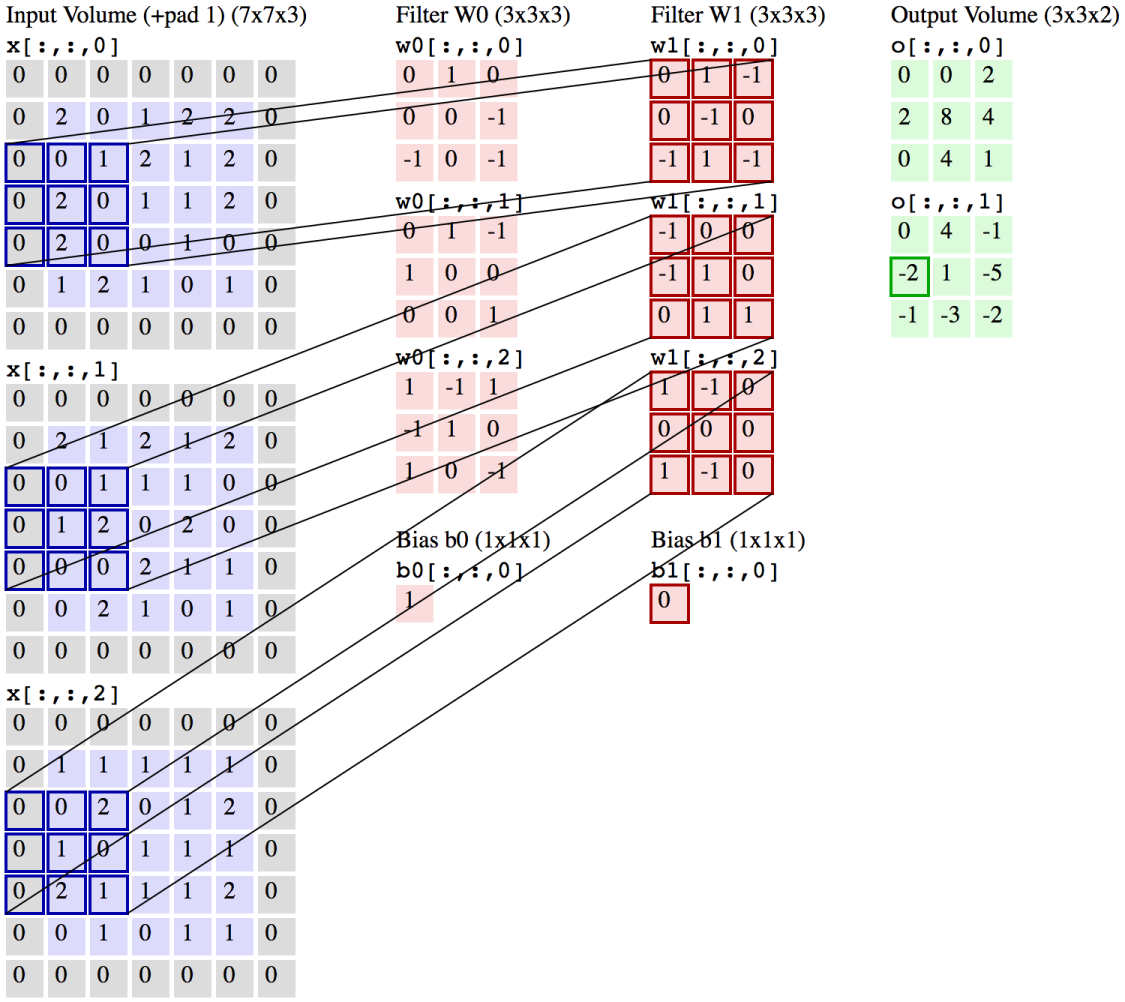


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

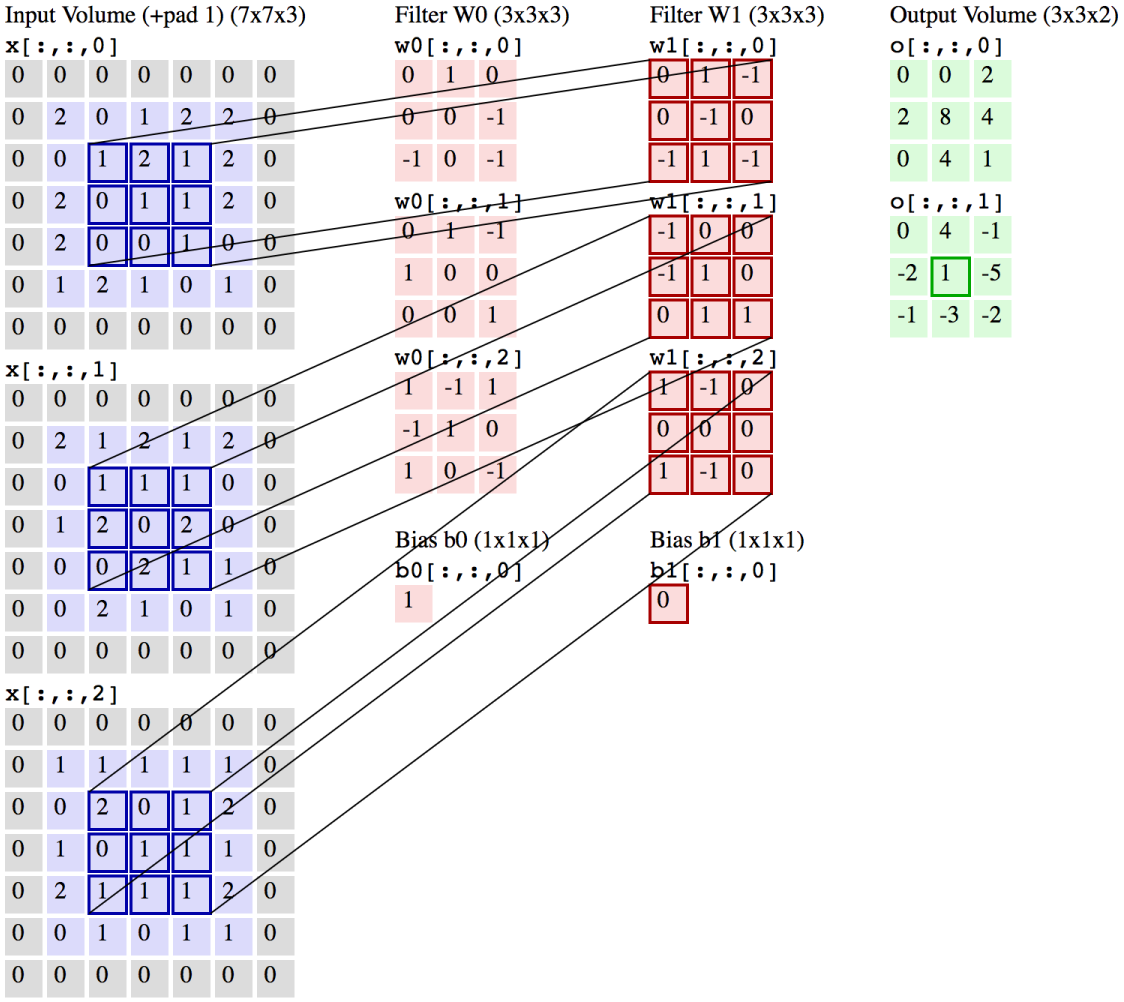


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

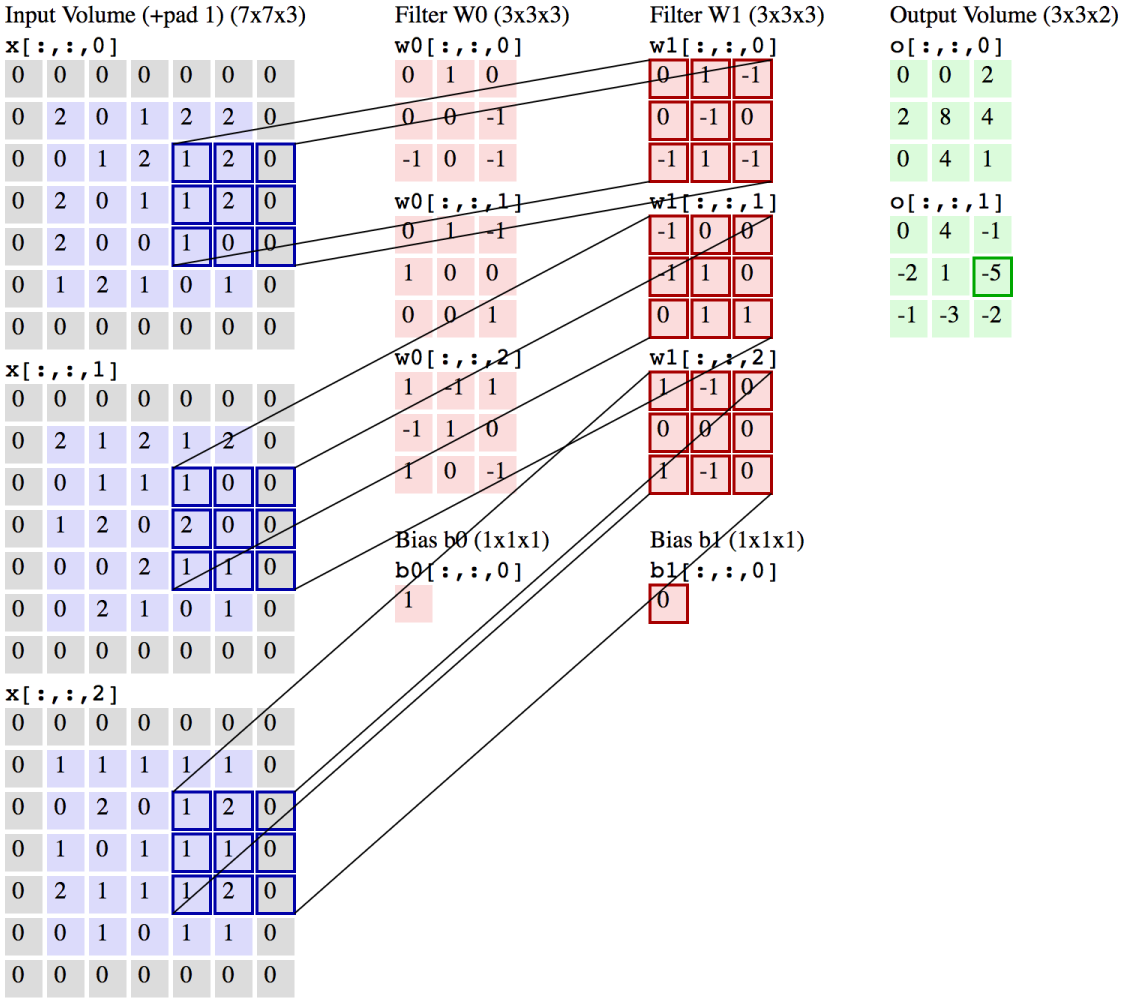


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

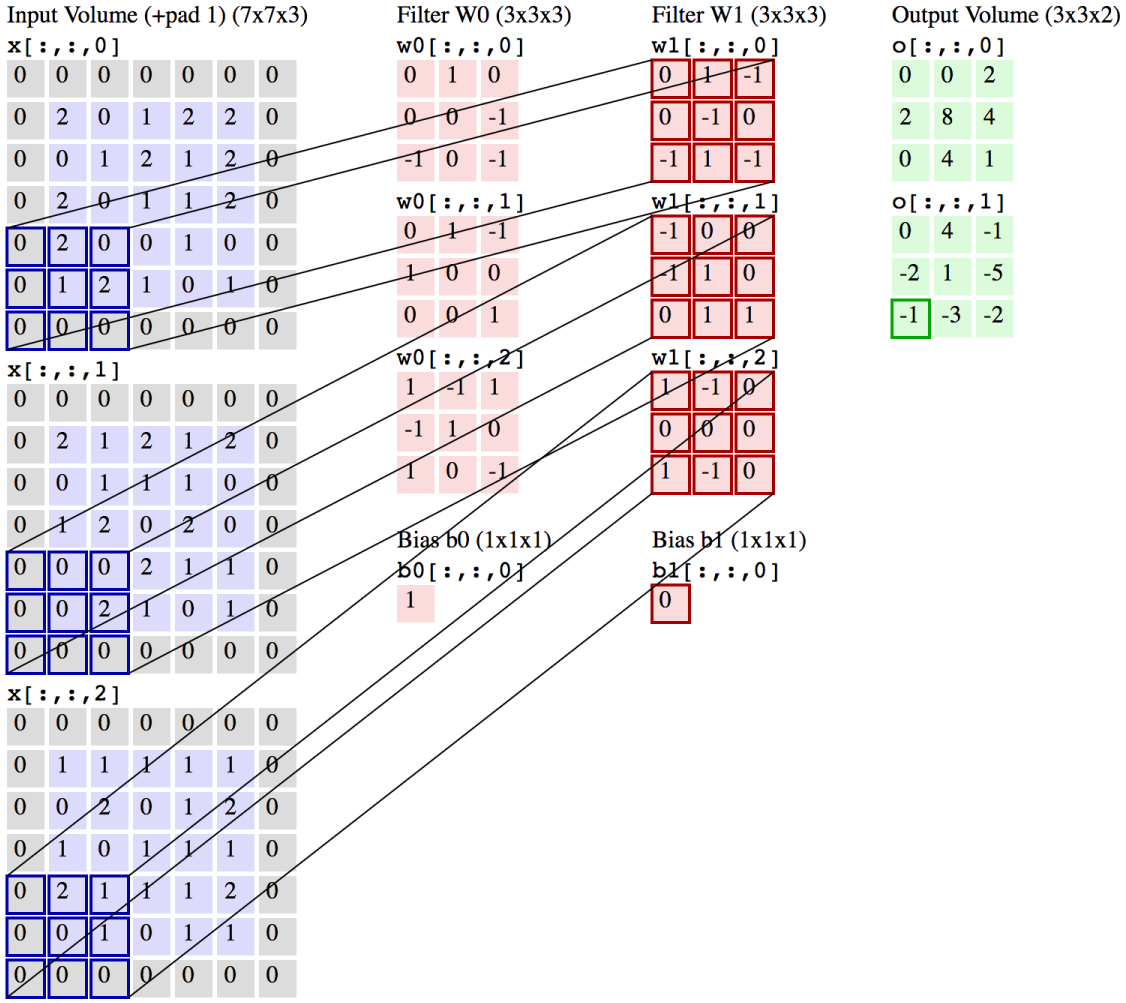


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

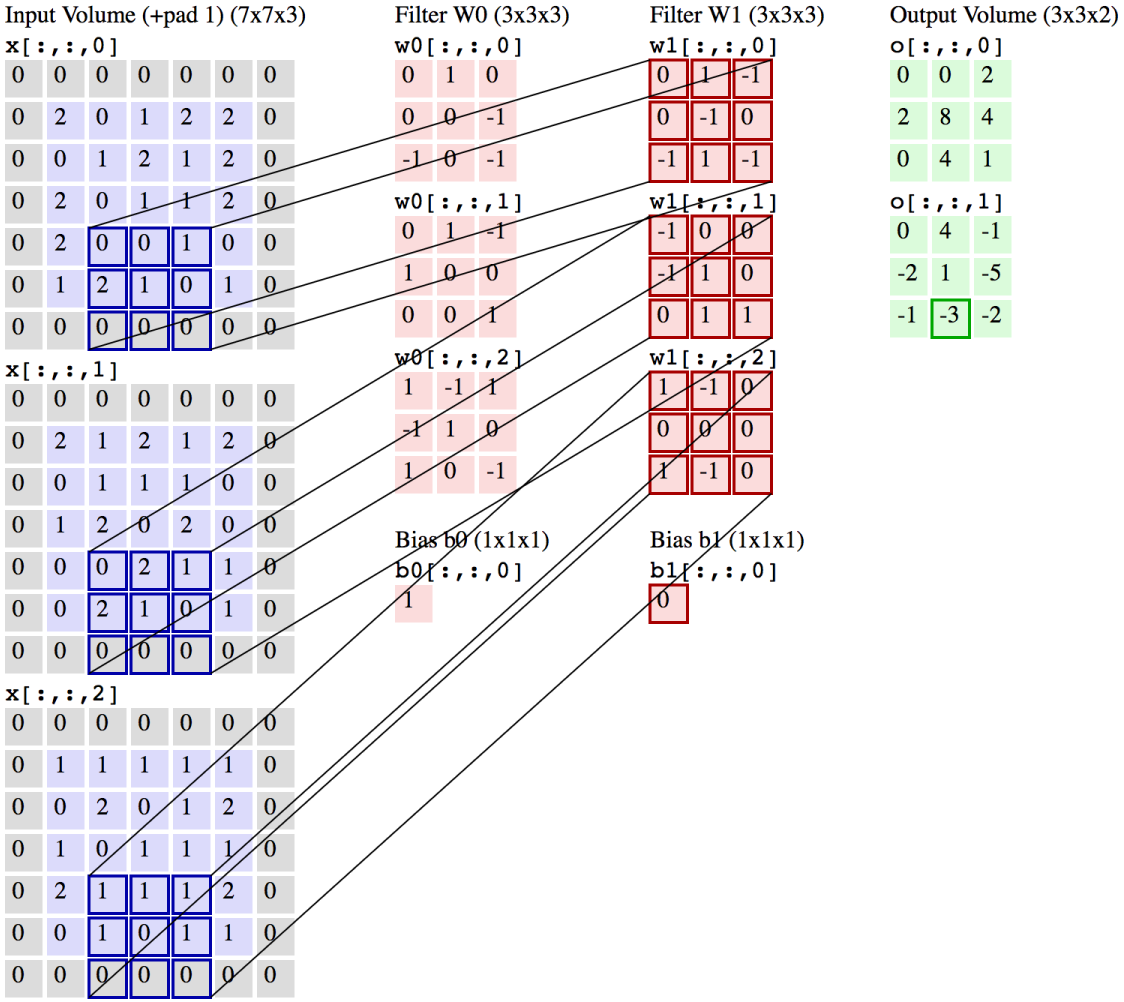


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

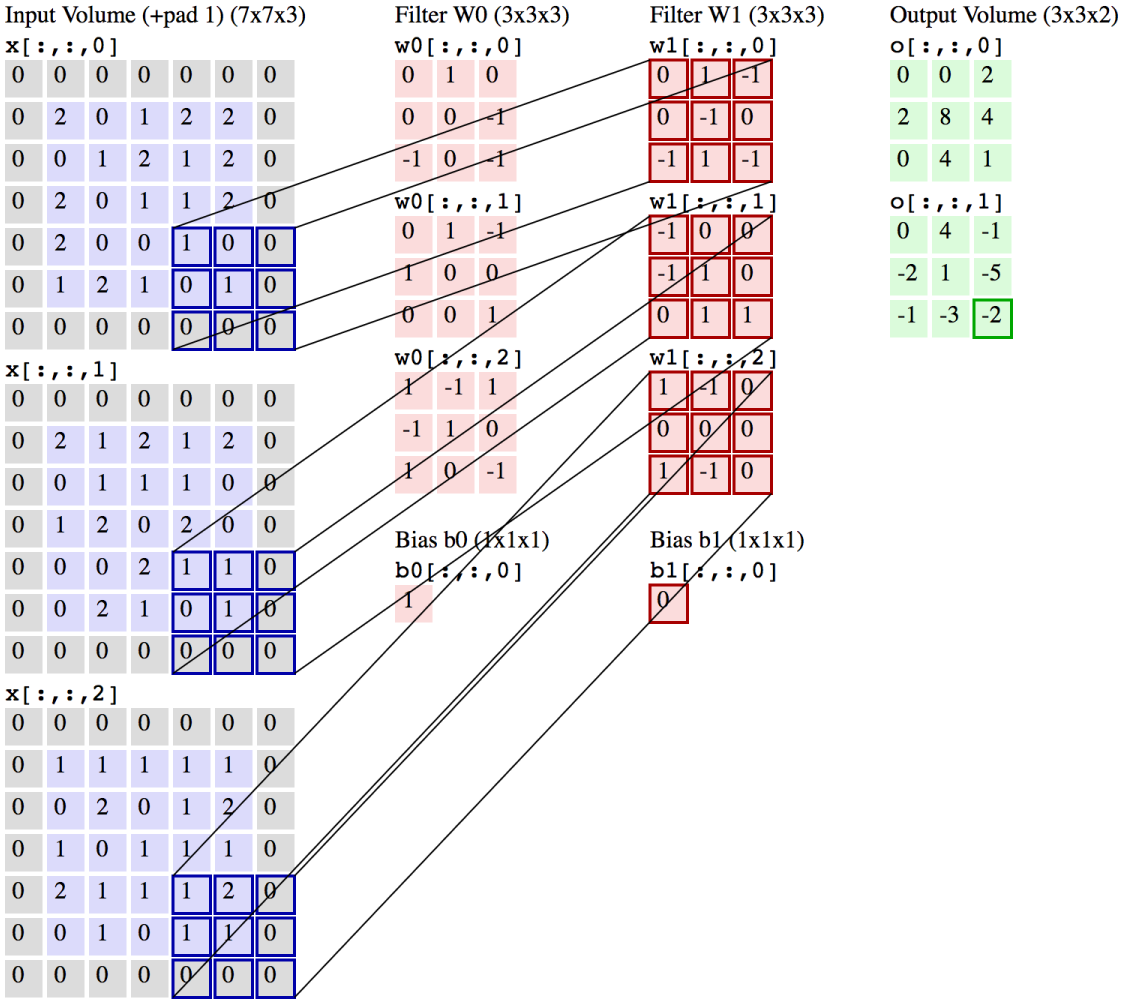
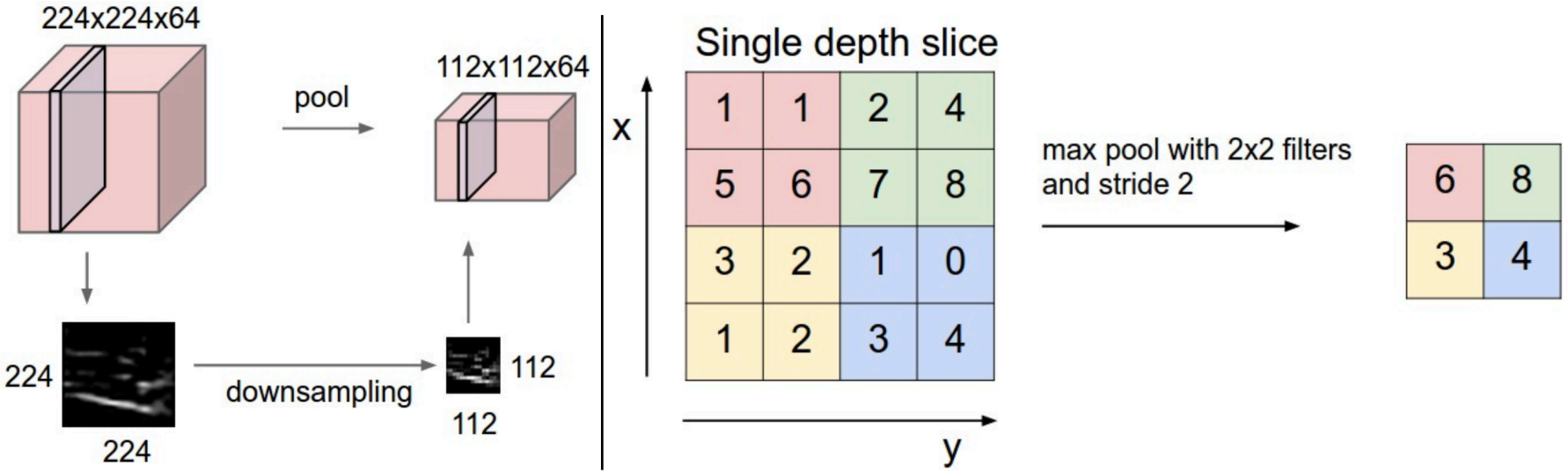


Image: Stanford Course CS231n: http://cs231n.github.io/convolutional-networks/

Outline for April 10

- Autoencoders and unsupervised pre-training
- Notes for lab
  - Scores
  - Dropout
- Introduction to convolutional neural networks
- Convolutional layers
- Dimensionality analysis

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/