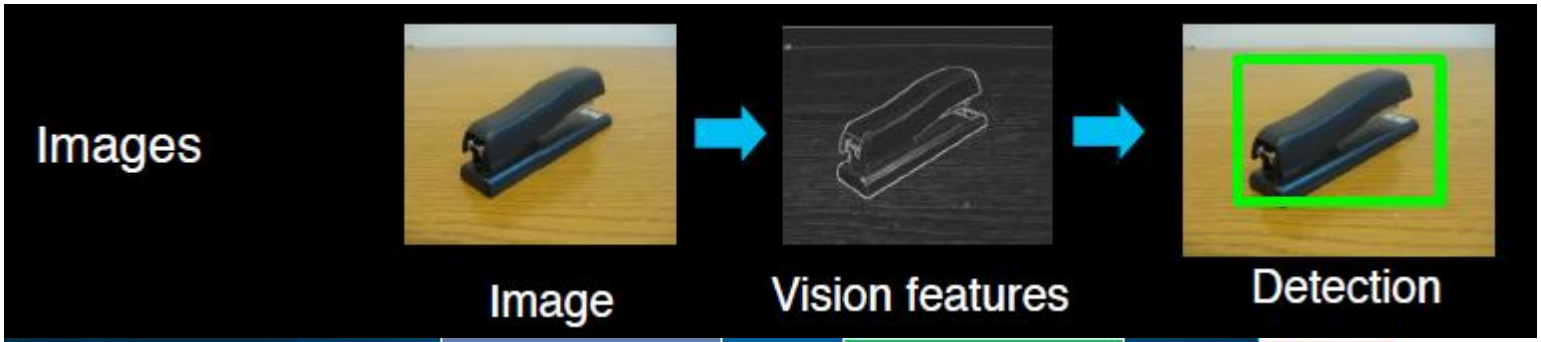
Convolutional Neural Networks & Reinforcement Learning

Convolutional Neural Networks

Computer Vision is Hard

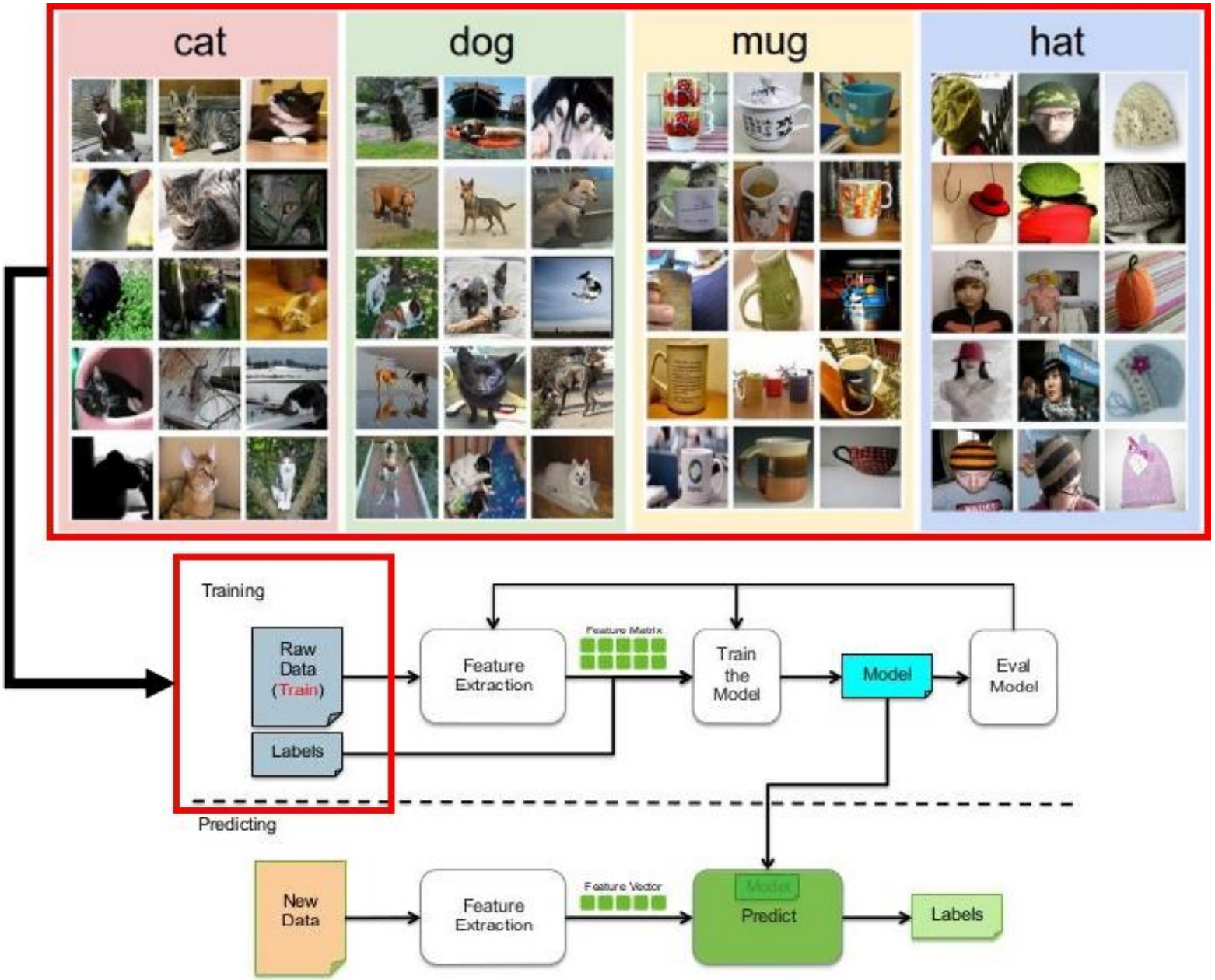
Computer Vision

- Traditional solution: off-line feature extraction
- SIFT, SURF, HOG, RIFT……

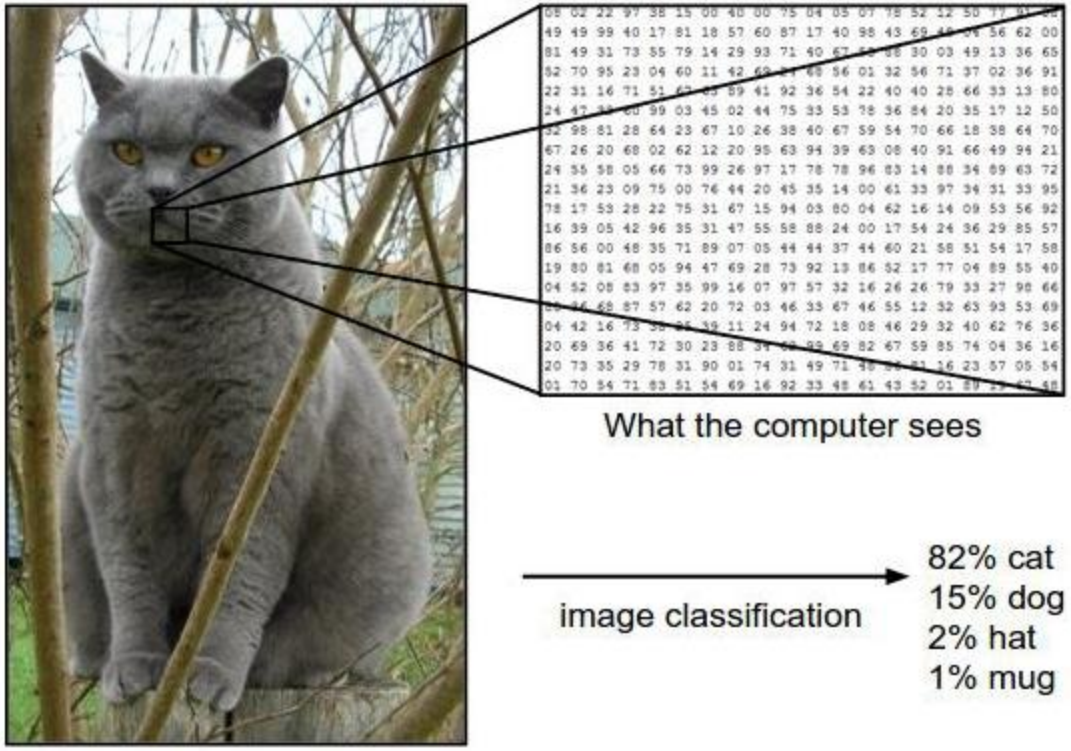


- CNN： use neural network to extract features automatically by convolution(a.k.a. filters)

Image Classification Pipeline



Images are Numbers



- **Regression:** The output variable takes continuous values
- **Classification:** The output variable takes class labels
  - Underneath it may still produce continuous values such as probability of belonging to a particular class.

Famous Computer Vision Datasets

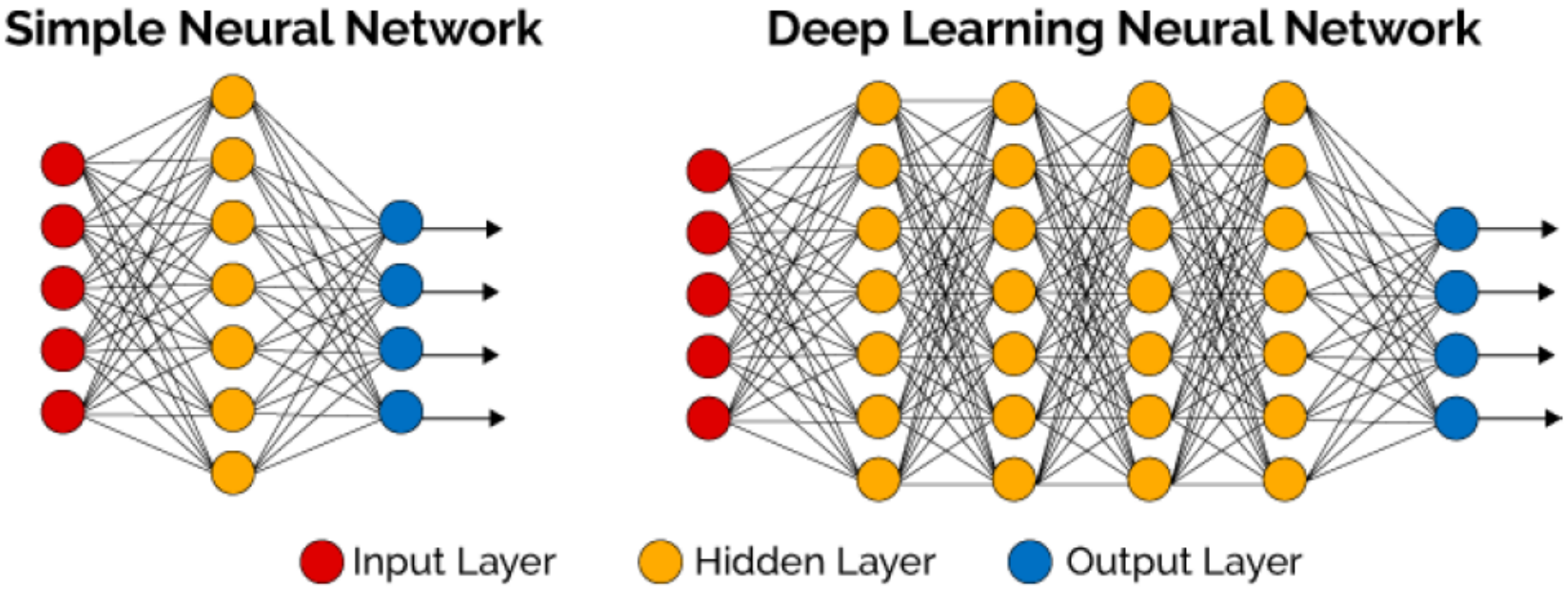
MNIST: handwritten digits

ImageNet: WordNet hierarchy

CIFAR-10(0): tiny images

Places: natural scenes

Combing Neurons in Hidden Layers:
The "Emergent" Power to Approximate



Universality: For any arbitrary function f(x), there exists a neural network that closely approximate it for any input x

Mini-Batch Size

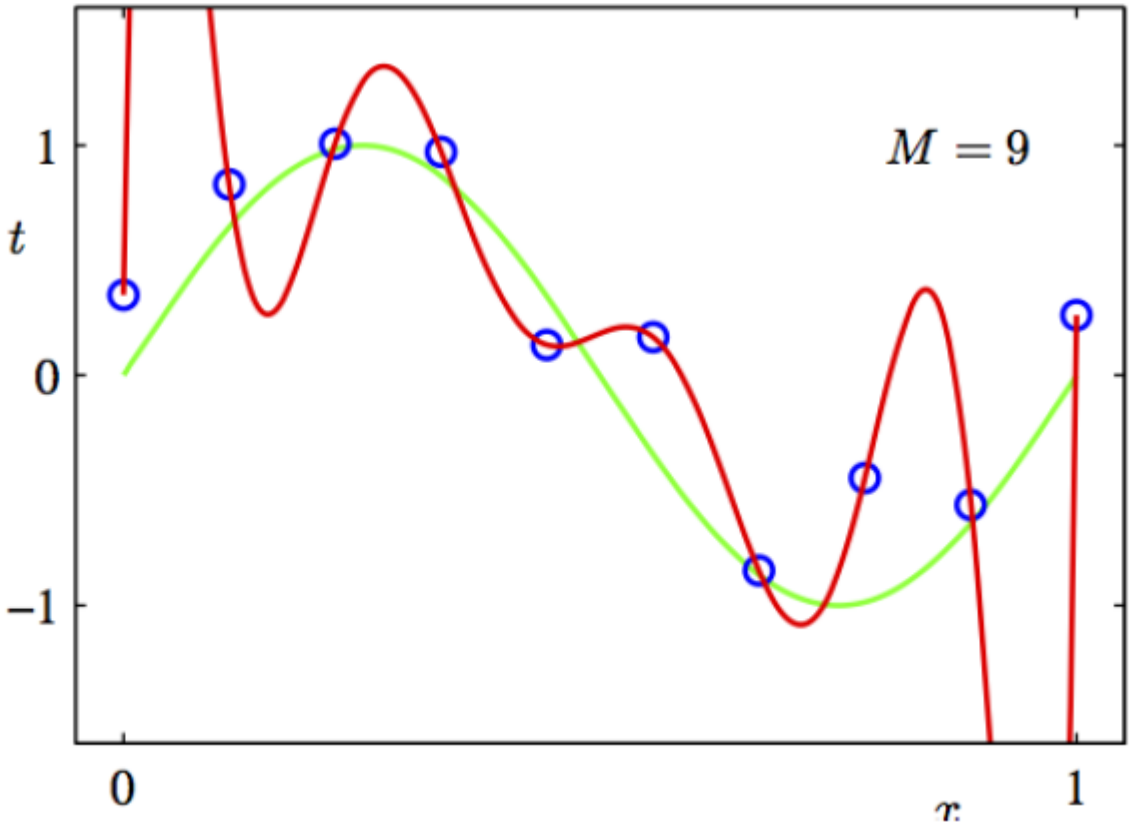
Mini-Batch size: Number of training instances the network evaluates per weight update step.

- Larger batch size = more computational speed
- Smaller batch size = (empirically) better generalization

"Training with large minibatches is bad for your health. More importantly, it's bad for your test error. Friends don't let friends use minibatches larger than 32."
- Yann LeCun

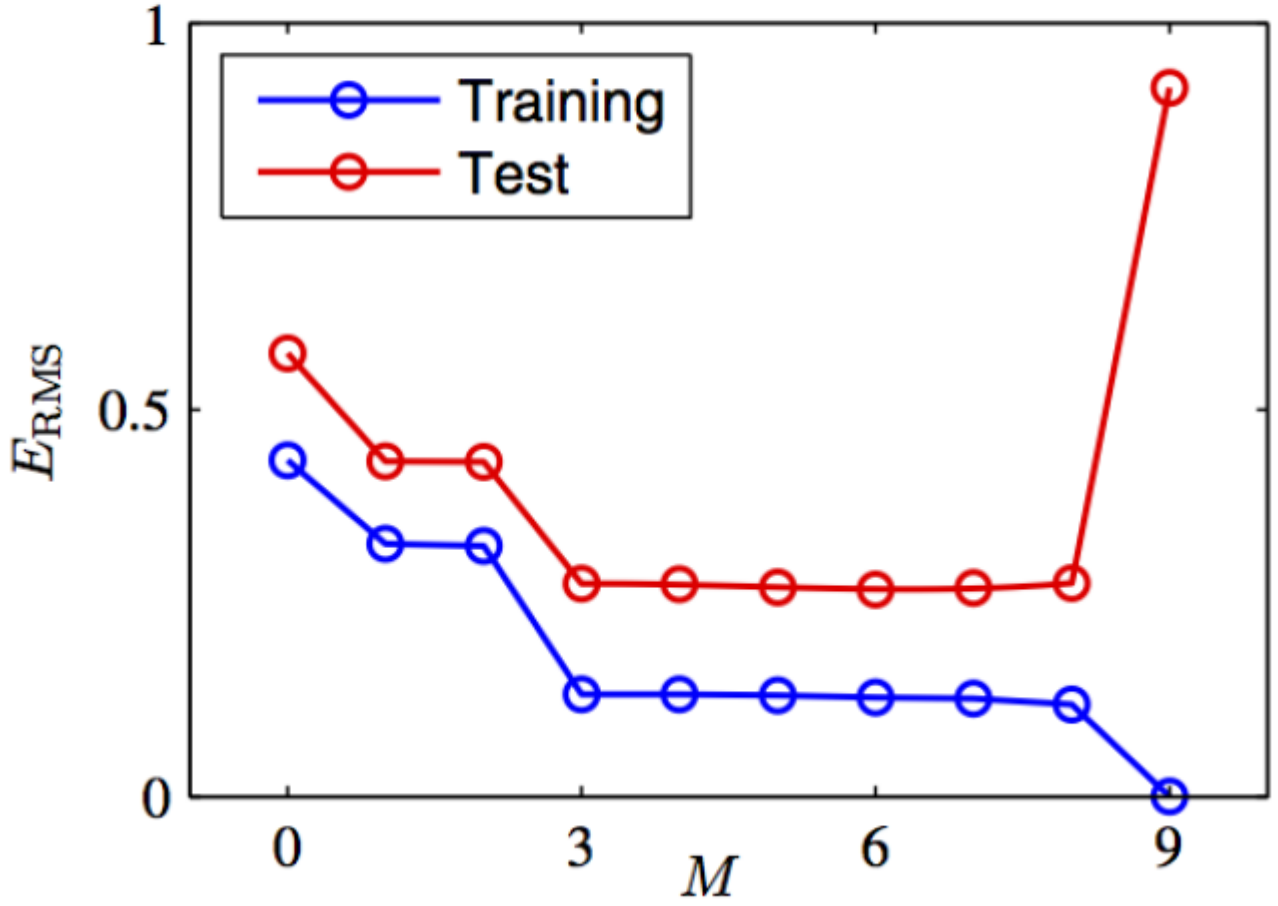
Overfitting and Regularization

- Help the network **generalize** to data it hasn't seen.
- Big problem for **small datasets**.
- Overfitting example (a sine curve vs 9-degree polynomial):

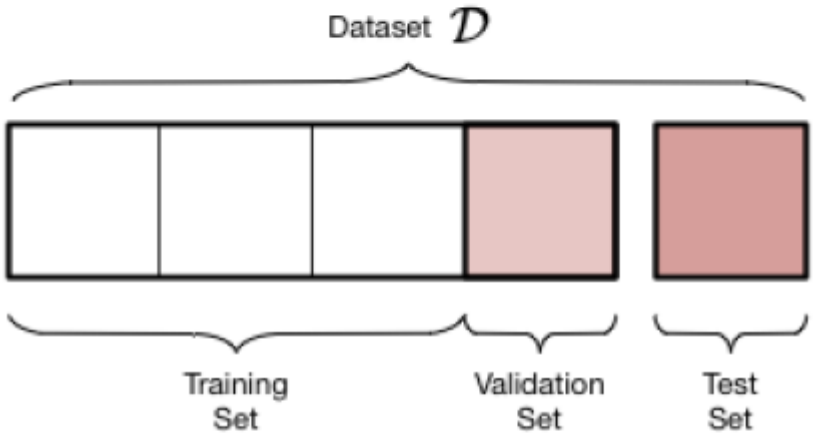


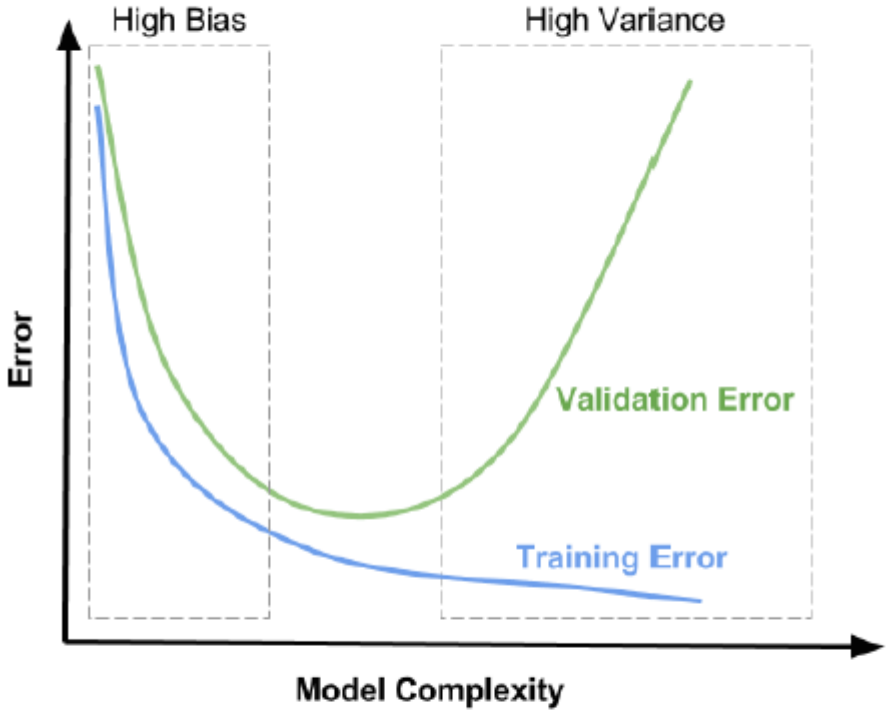
Overfitting and Regularization

- Overfitting: The error decreases in the training set but increases in the test set.



Regularization: Early Stoppage



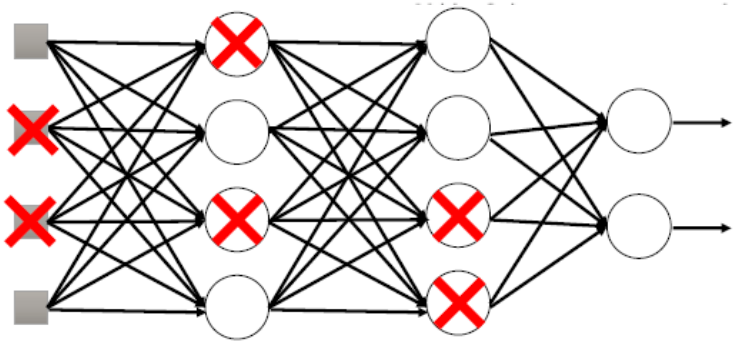


- Create "validation" set (subset of the training set).
  - Validation set is assumed to be a representative of the testing set.
- **Early stoppage:** Stop training (or at least save a checkpoint) when performance on the validation set decreases

Regularization: Dropout

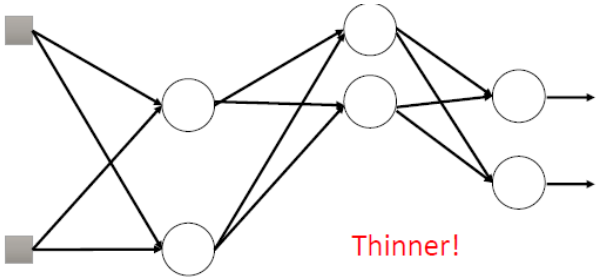
p=0.5

Training:



- **Each time before computing the gradients**
  - Each neuron has p% to dropout

Training:

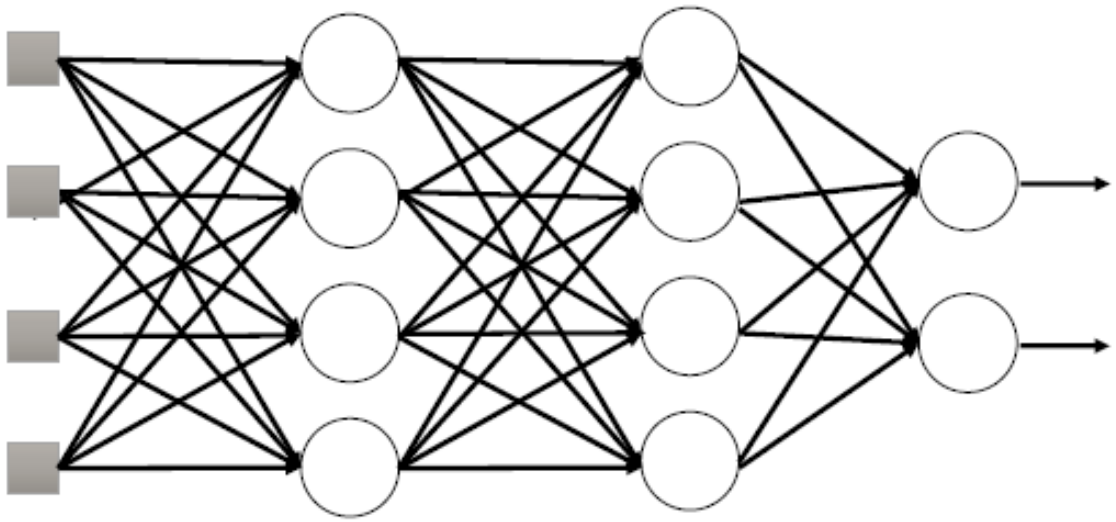


- **Each time before computing the gradients**
  - Each neuron has p% to dropout
  - **The structure of the network is changed.**
  - Using the new network for training

- **Dropout:** Randomly remove some nodes in the network (along with incoming and outgoing edges)
- Notes:
  - Usually $p >= 0.5$ (p is probability of keeping node)
  - Input layers p should be much higher (and use noise instead of dropout)
  - Most deep learning frameworks come with a dropout layer

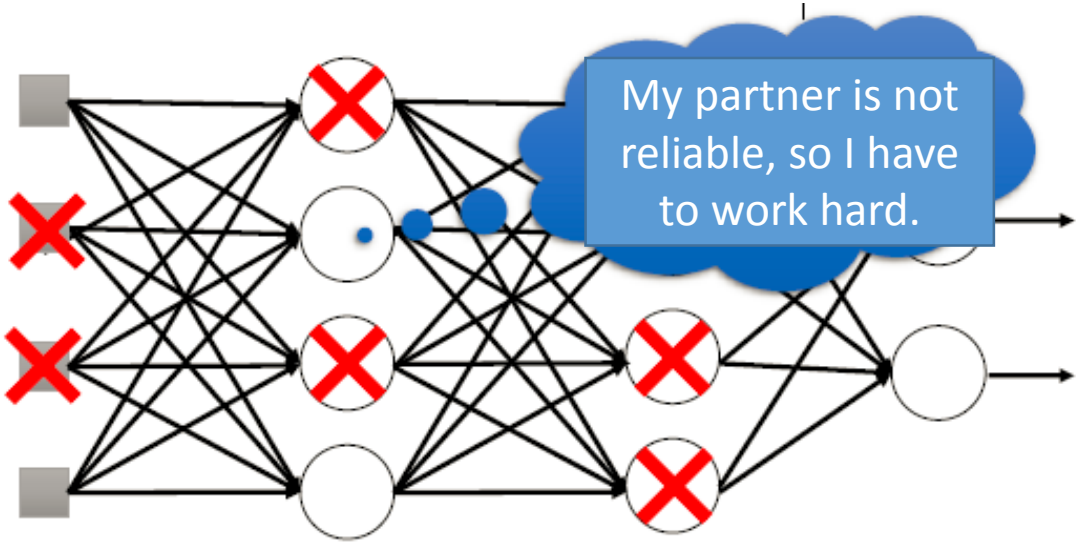
regularization: dropout

Testing:



- **No dropout**
  - If the dropout rate at training is p%, all the weights times (1-p)%
  - Assume that the dropout rate is 50%. If a weight $w = 1$ by training, set $w = 0.5$ for testing.

Dropout -Intuitive Reason

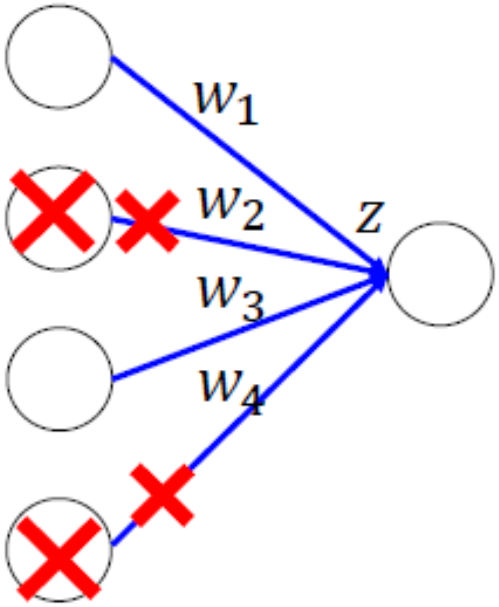


- When teams up, if everyone expect the partner will do the work, nothing will be done finally.
- However, if you know your partner will dropout, you will do better.
- When testing, no one dropout actually, so obtaining good results eventually.

- Why the weights should multiply (1-p)% (dropout rate) when testing?

Training of Dropout

Assume dropout rate is 50%

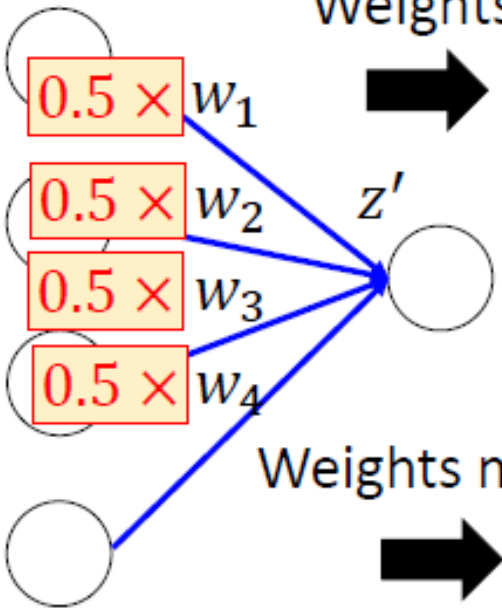


Testing of Dropout

No dropout

Weights from training

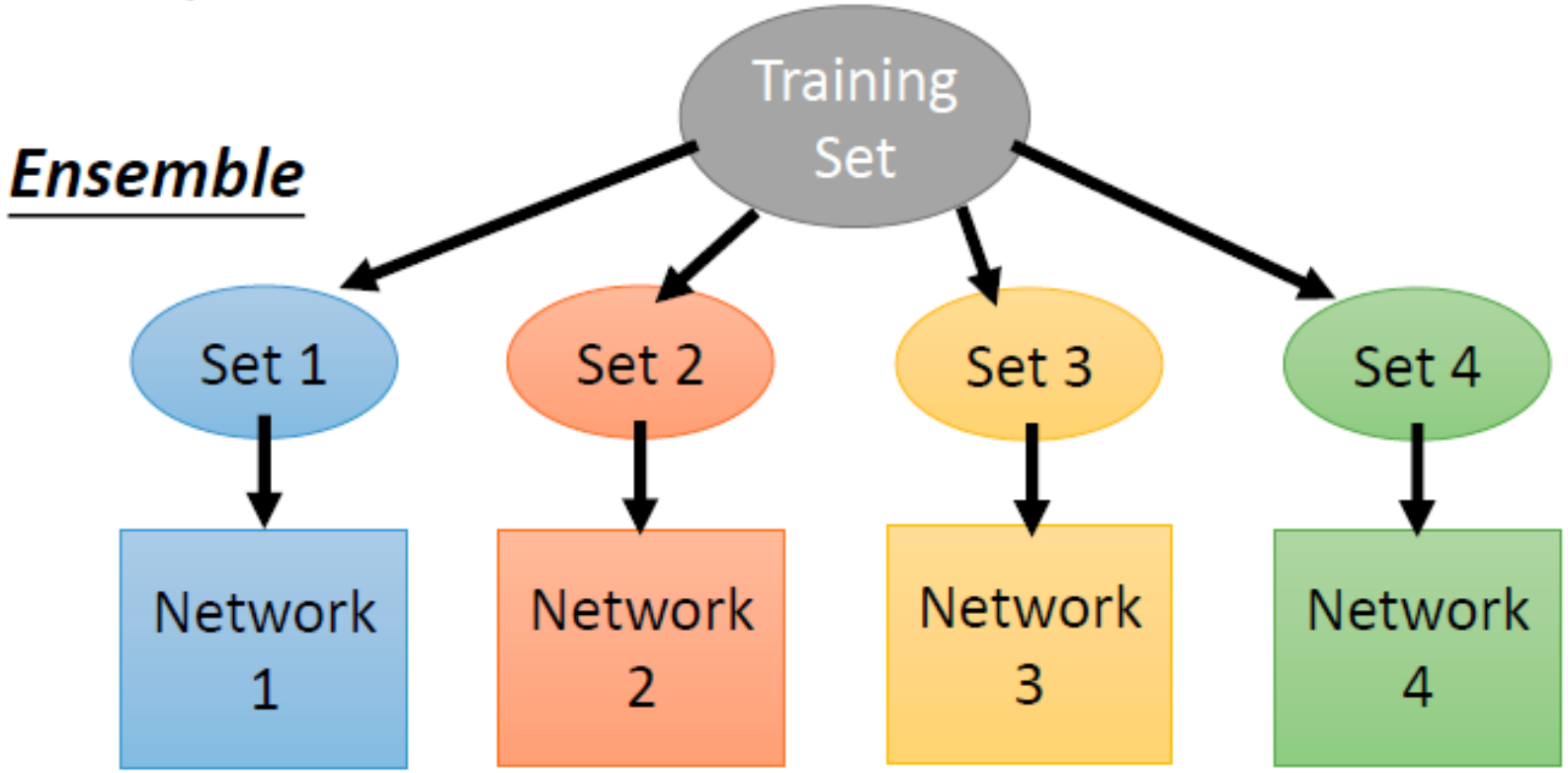
$z' \approx 2z$



Weights multiply (1-p)%

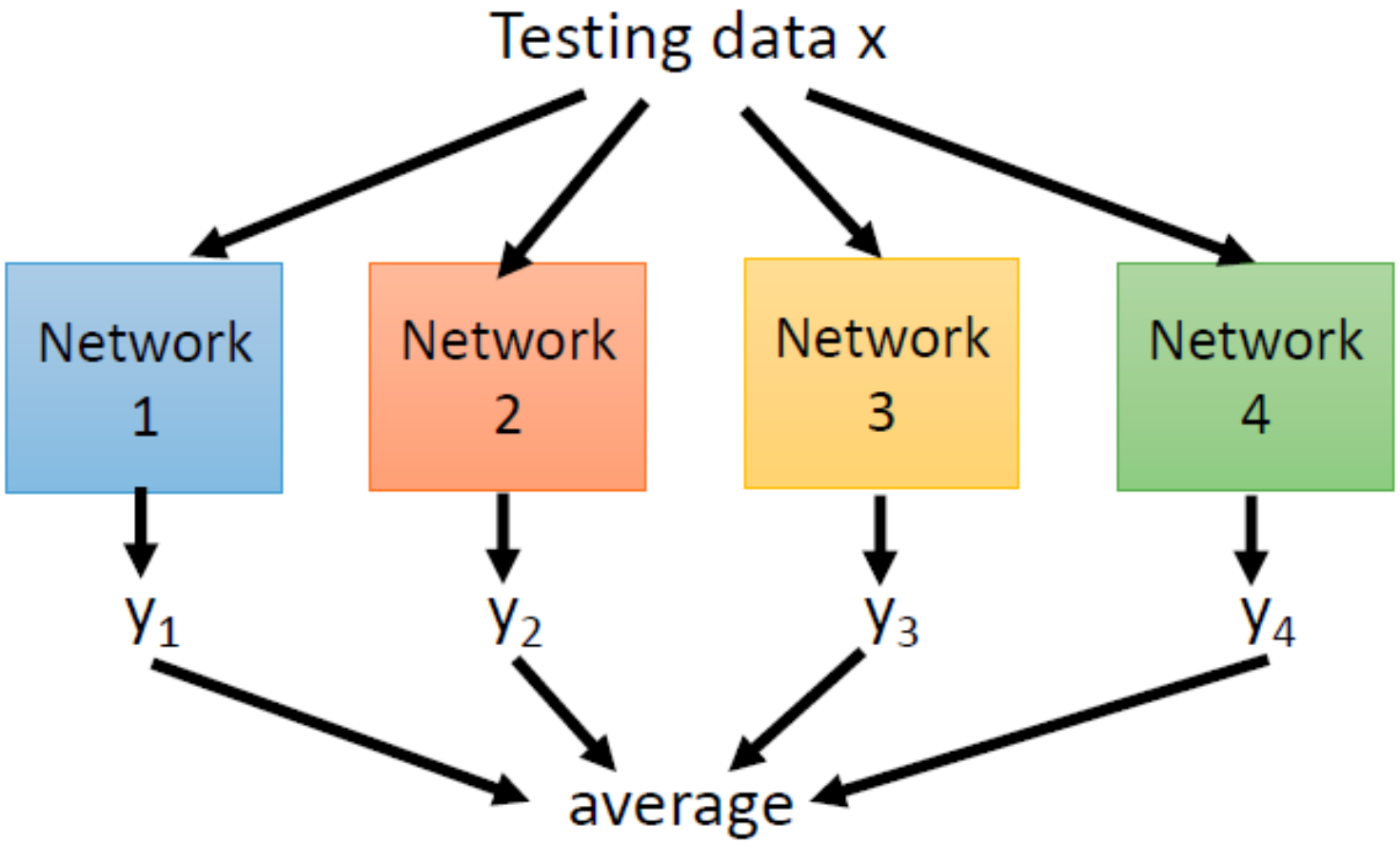
$z' \approx z$

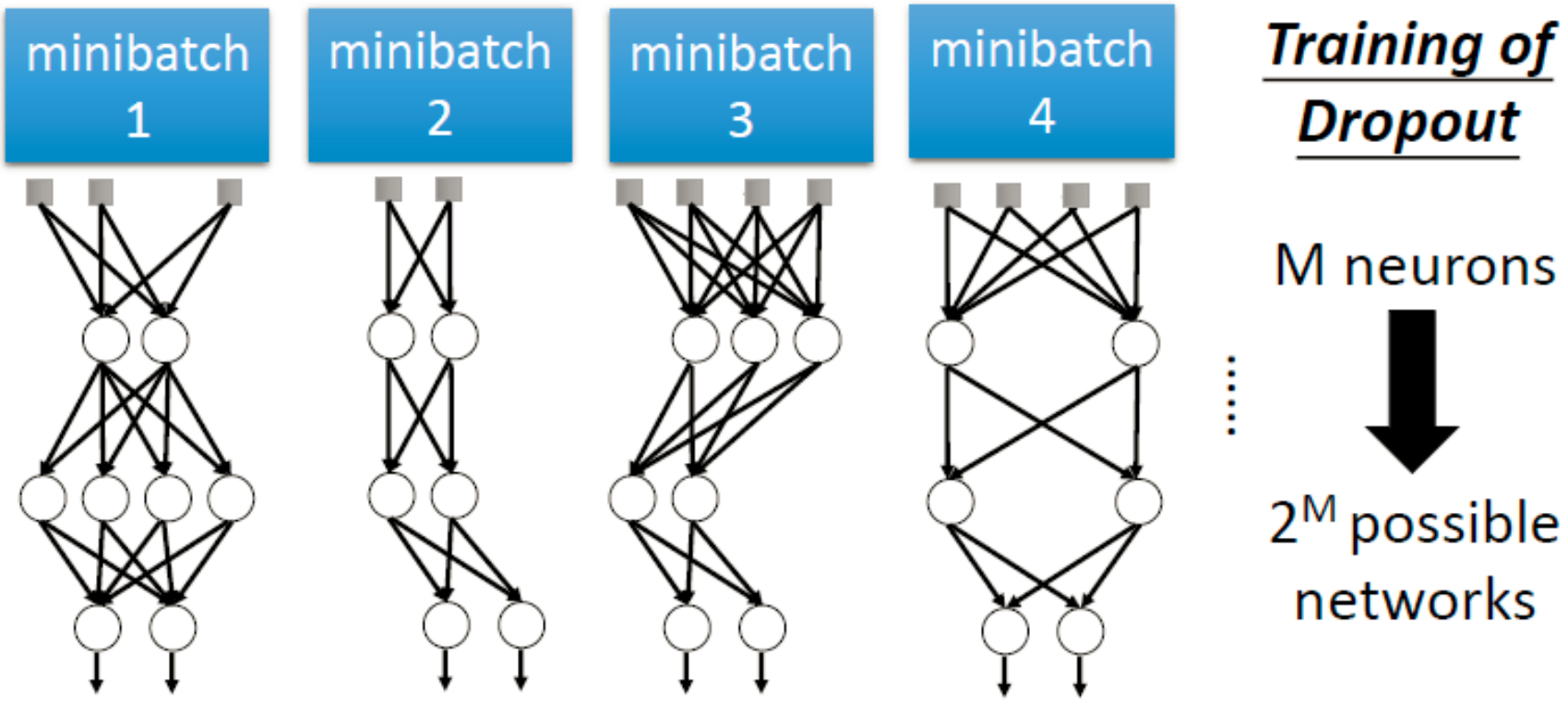
Dropout is a kind of ensemble



Train a bunch of networks with different structures

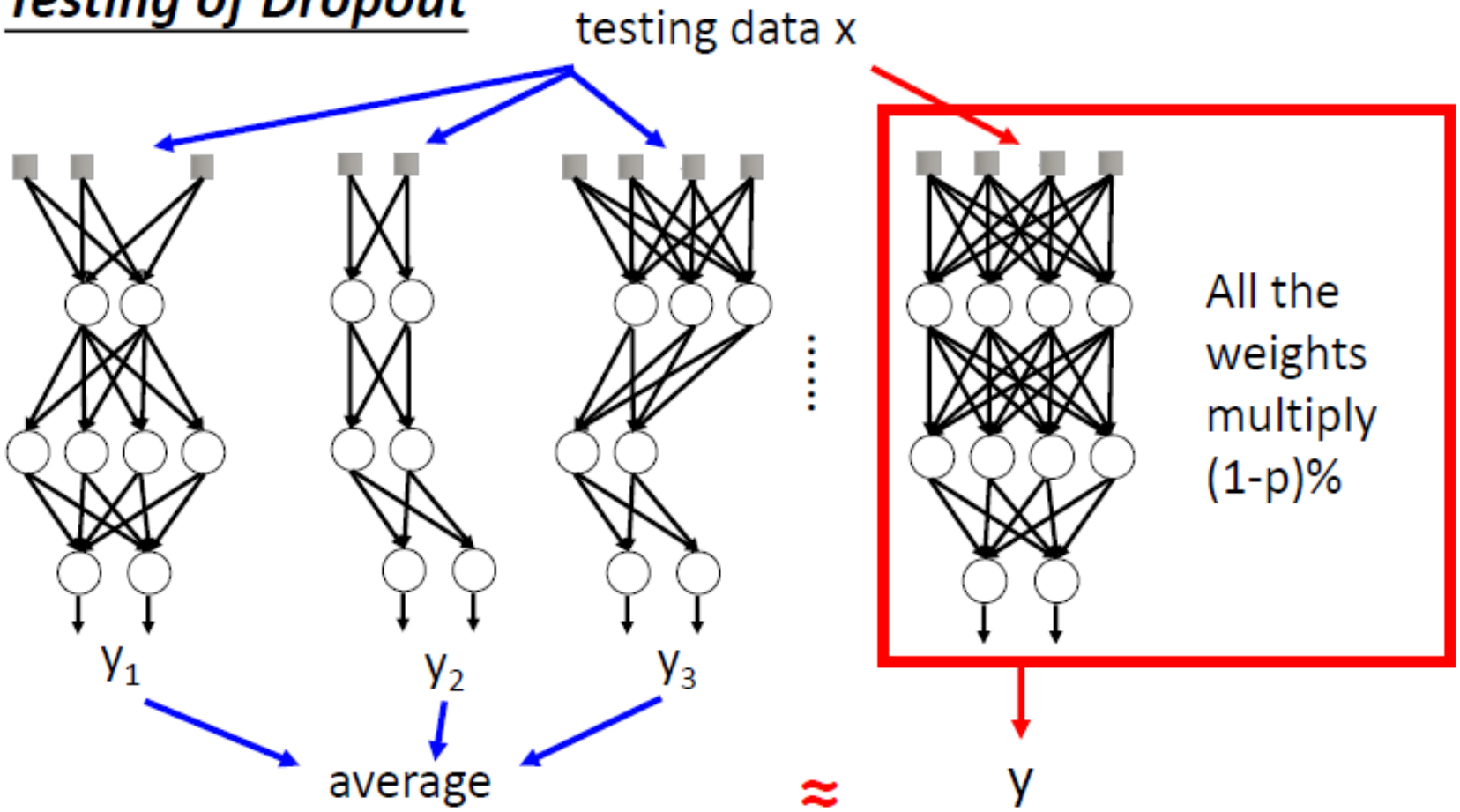
Ensemble


Testing data x
Network 1
Network 2
Network 3
Network 4
y_1
y_2
y_3
y_4
average



- Using one mini-batch to train one network
- Some parameters in the network are shared

Testing of Dropout

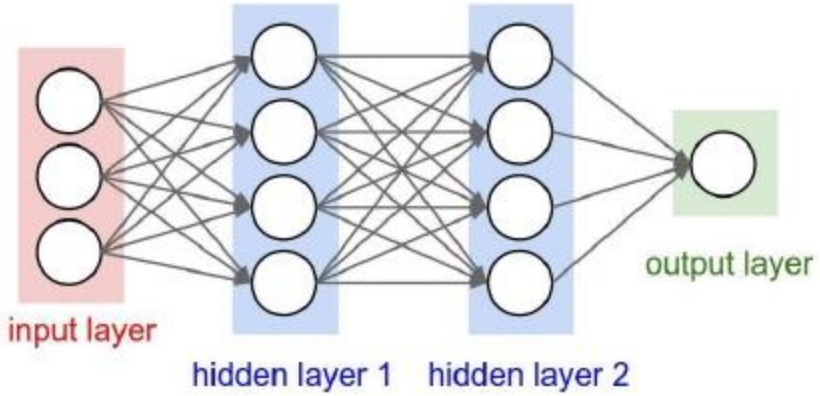


Normalization

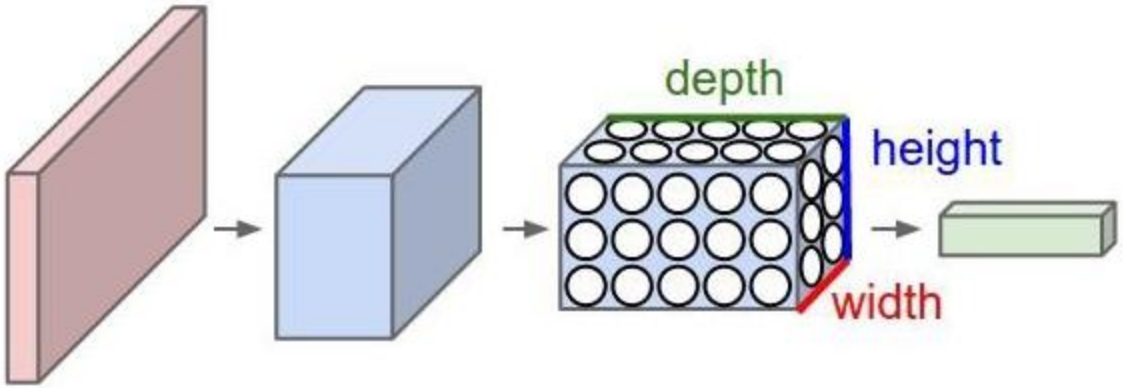
- Network Input Normalization
  - *Example:* Pixel to [0, 1] or [-1, 1] or according to mean and std.
- Batch Normalization (BatchNorm, BN)
  - Normalize hidden layer inputs to mini-batch mean & variance
  - Reduces impact of earlier layers on later layers
- Batch Renormalization (BatchRenorm, BR)
  - Fixes difference b/w training and inference by keeping a moving average asymptotically approaching a global normalization.
- Other options:
  - Layer normalization (LN) – conceived for RNNs
  - Instance normalization (IN) – conceived for Style Transfer
  - Group normalization (GN) – conceived for CNNs

Convolutional Neural Networks

Regular neural network (fully connected):



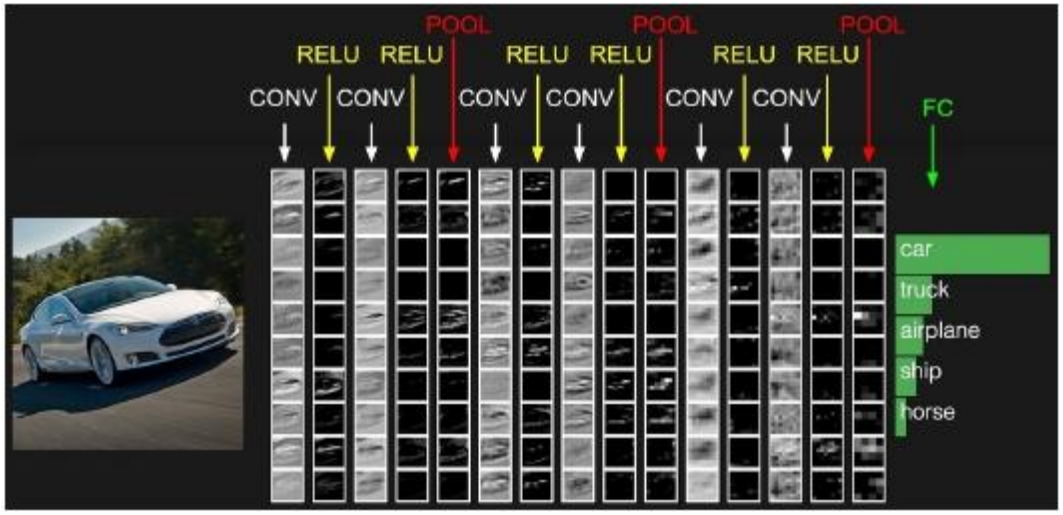
Convolutional neural network:



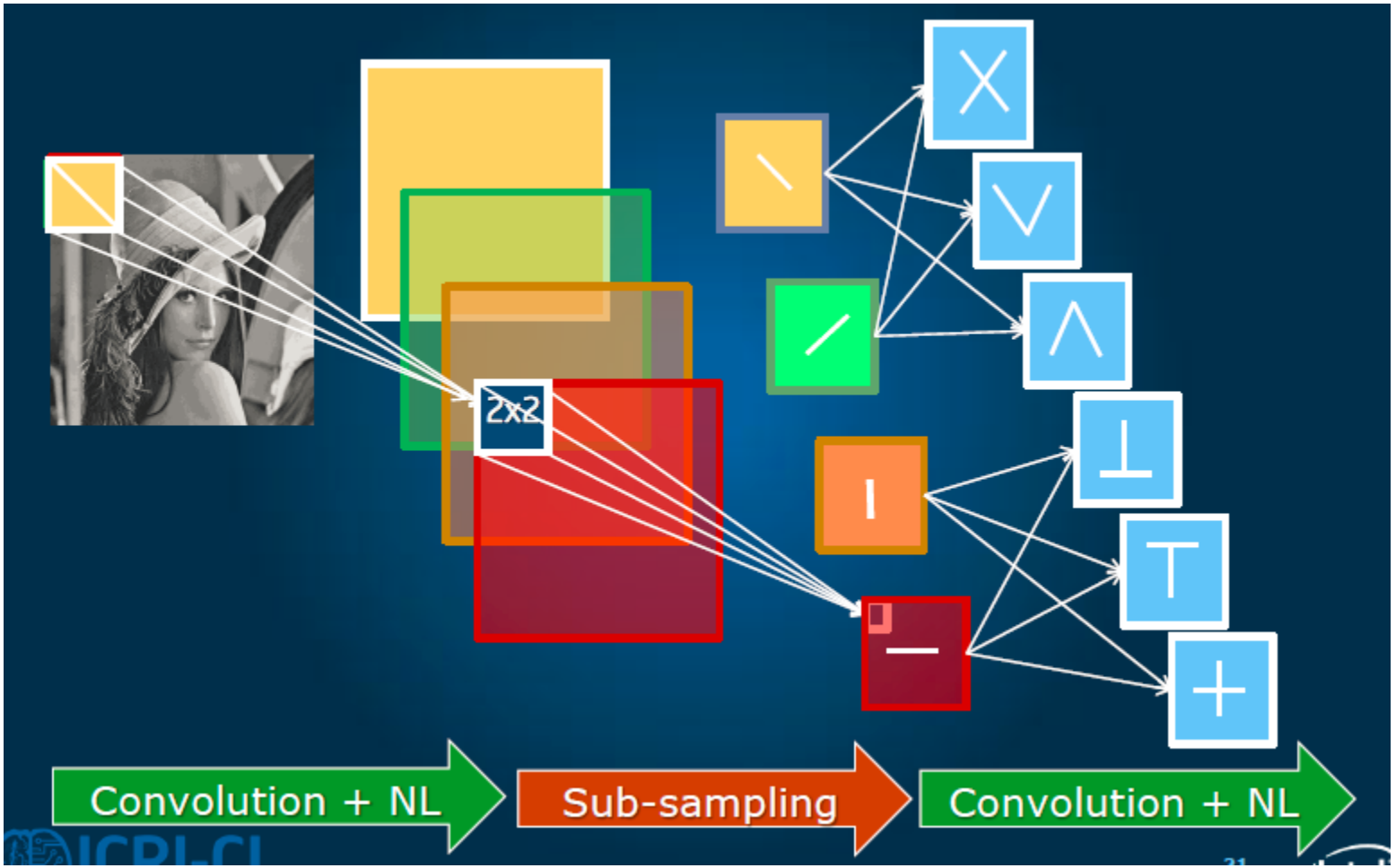
Each layer takes a 3d volume, produces 3d volume with some smooth function that may or may not have parameters.

Convolutional Neural Networks: Layers

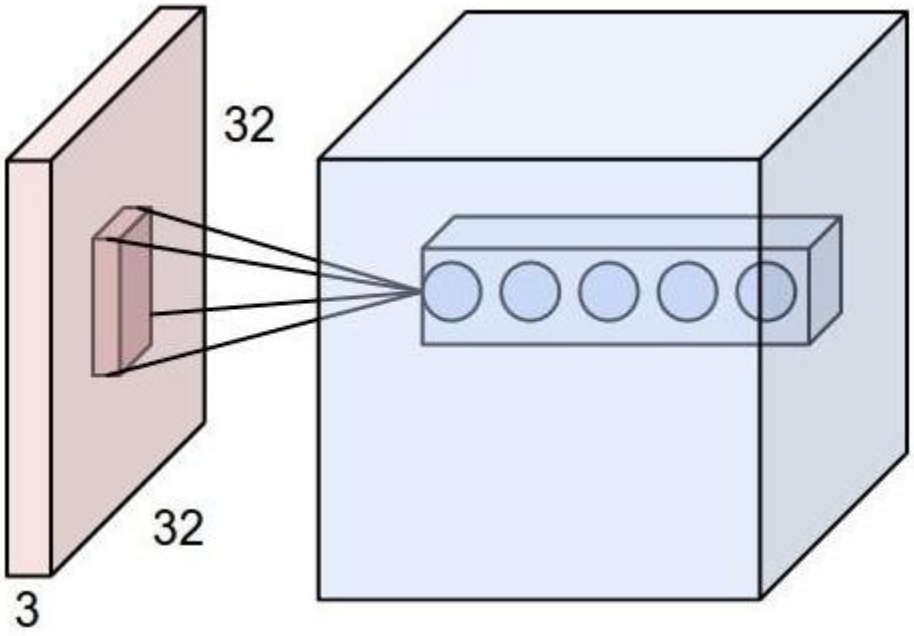
- **INPUT** [32x32x3] will hold the raw pixel values of the image, in this case an image of width 32, height 32, and with three color channels R,G,B.
- **CONV** layer will compute the output of neurons that are connected to local regions in the input, each computing a dot product between their weights and a small region they are connected to in the input volume. This may result in volume such as [32x32x12] if we decided to use 12 filters.
- **RELU** layer will apply an elementwise activation function, such as the *max(0,x)* thresholding at zero. This leaves the size of the volume unchanged ([32x32x12]).
- **POOL** layer will perform a downsampling operation along the spatial dimensions (width, height), resulting in volume such as [16x16x12].
- **FC** (i.e. fully-connected) layer will compute the class scores, resulting in volume of size [1x1x10], where each of the 10 numbers correspond to a class score, such as among the 10 categories of CIFAR-10. As with ordinary Neural Networks and as the name implies, each neuron in this layer will be connected to all the numbers in the previous volume.

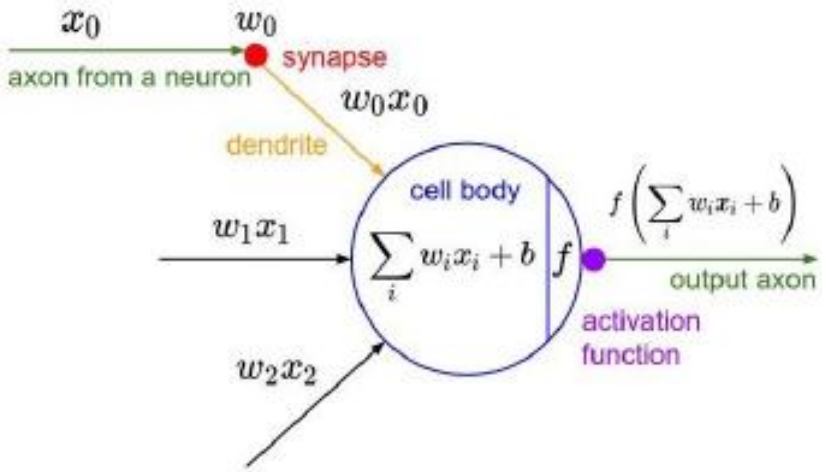


What is Convolutional NN ?



Dealing with Images: Local Connectivity



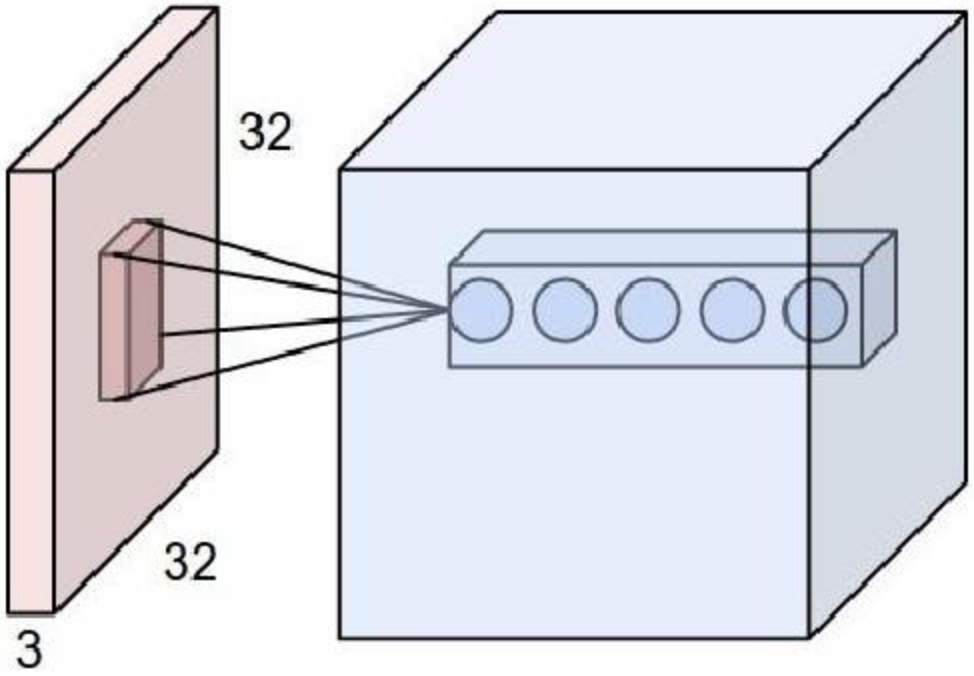


Same neuron. Just more focused (narrow “receptive field”).

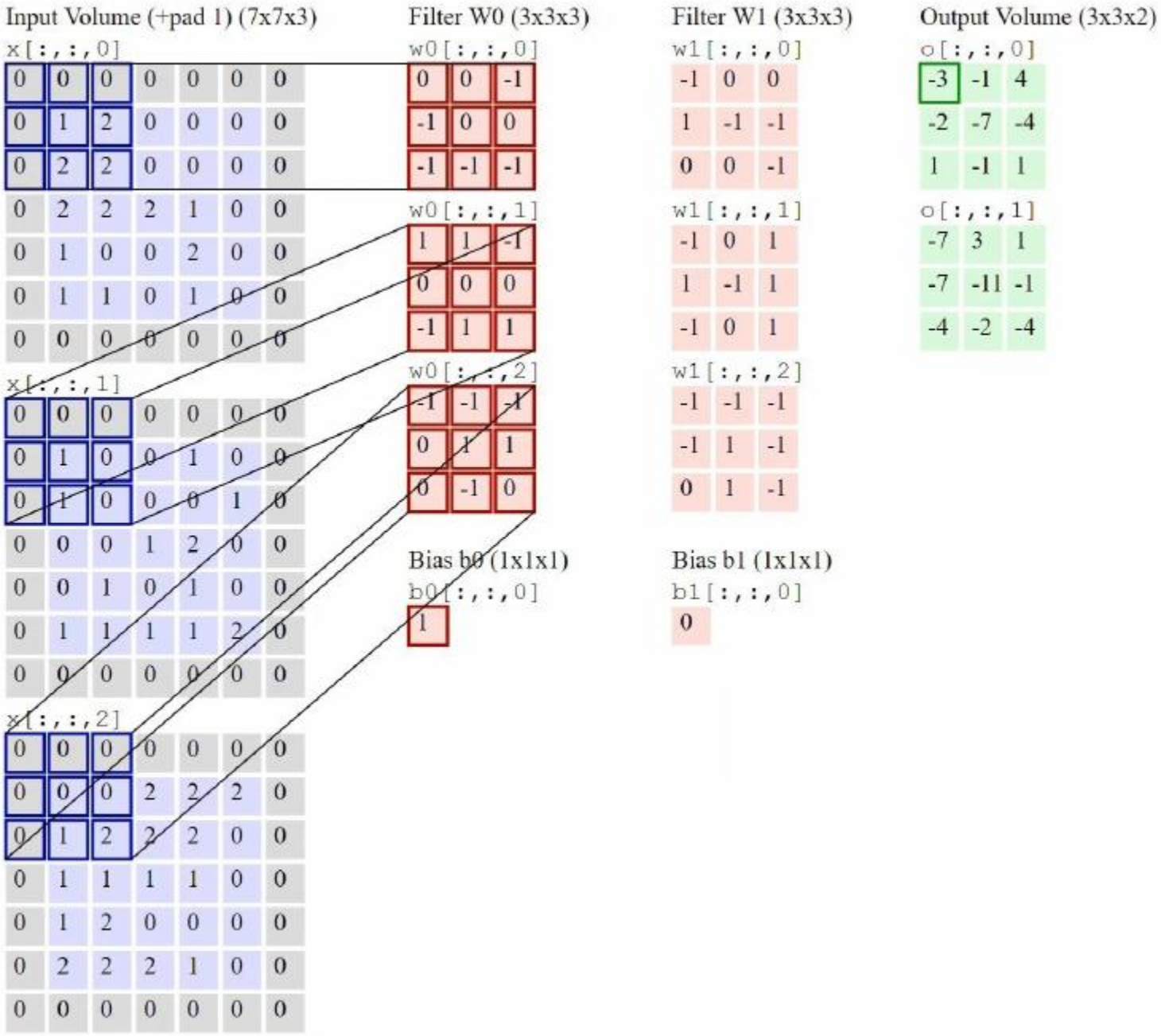
The parameters on each filter are spatially “shared”

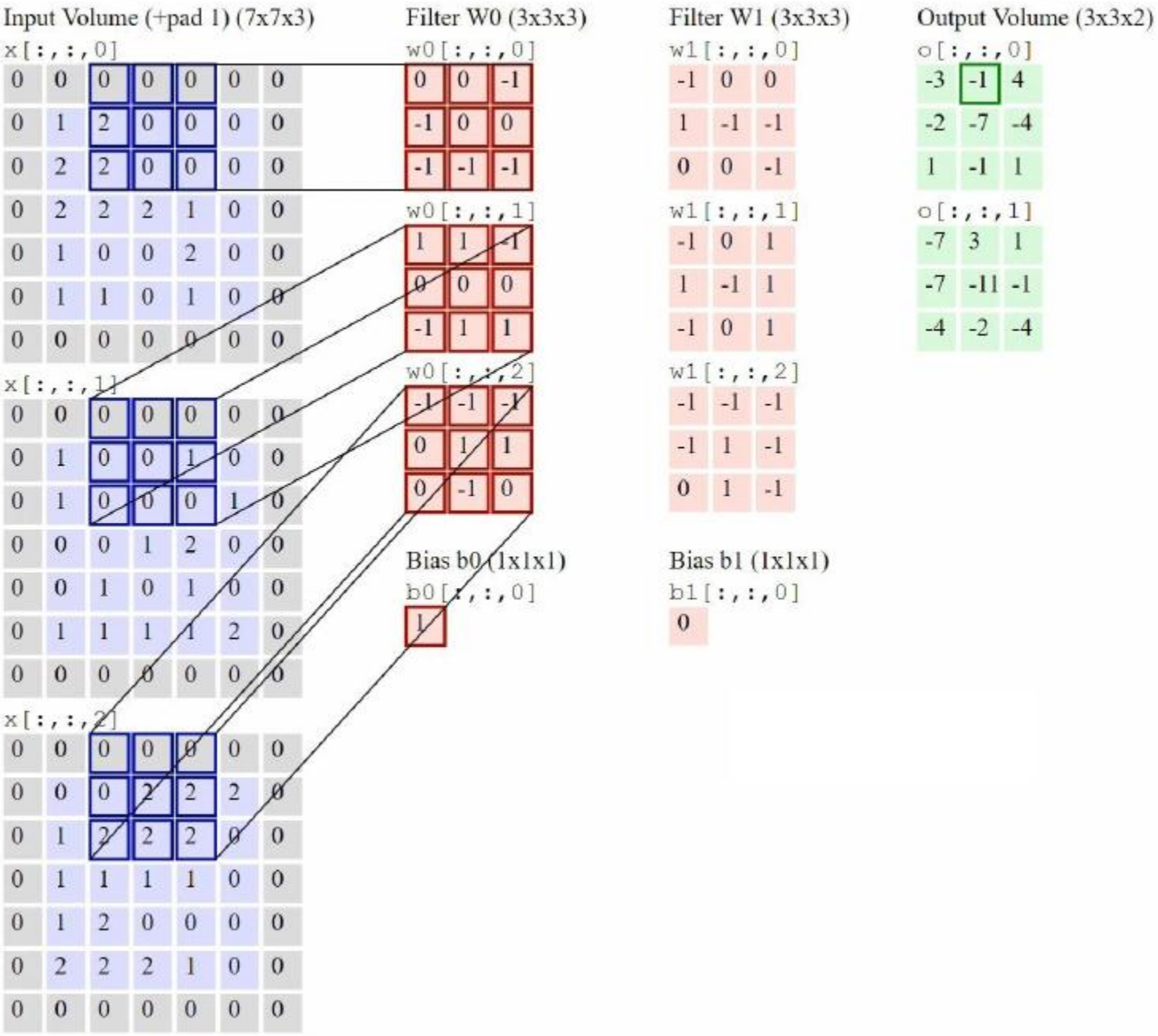
(if a feature is useful in one place, it’s useful elsewhere)

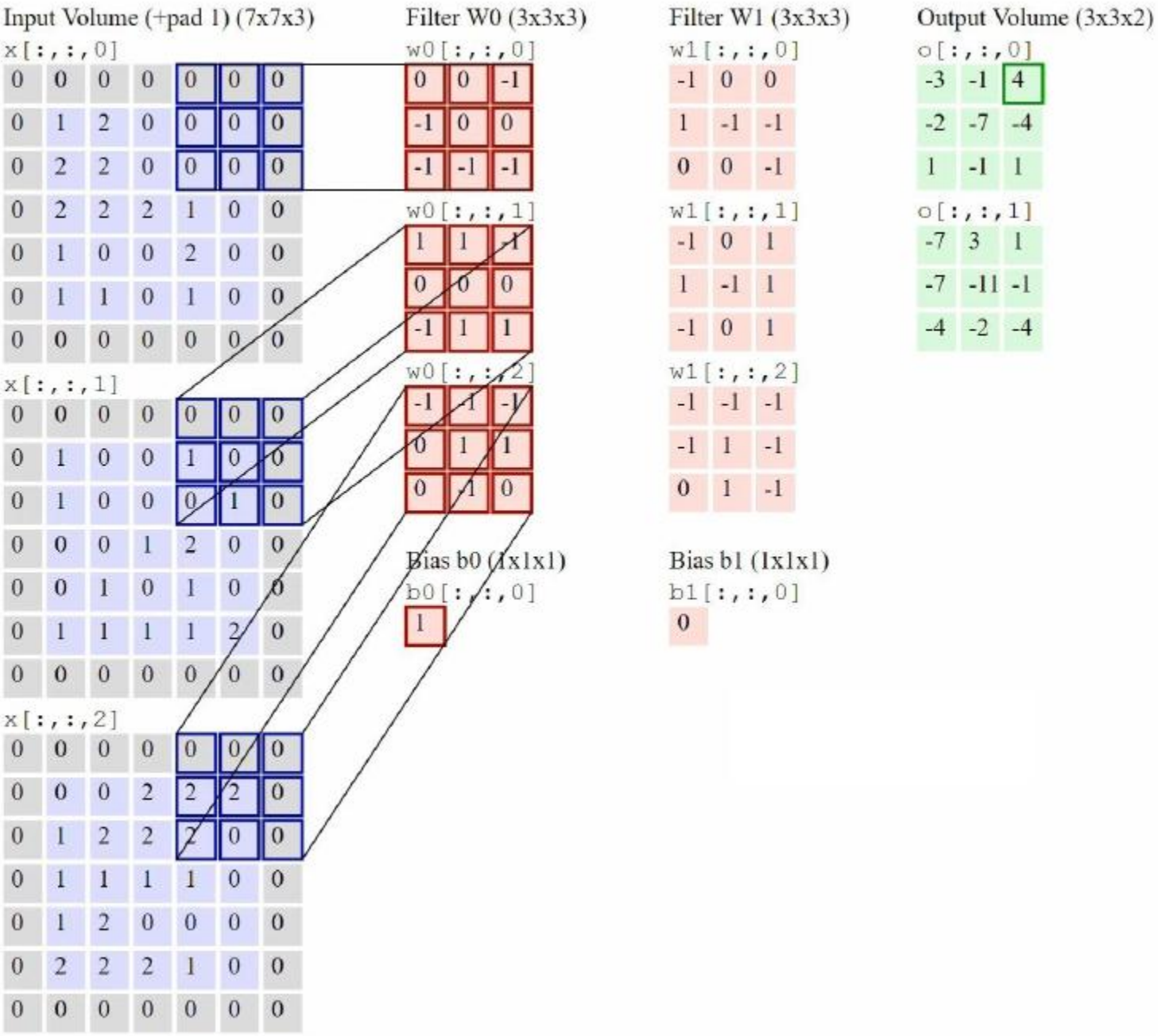
ConvNets: Spatial Arrangement of Output Volume

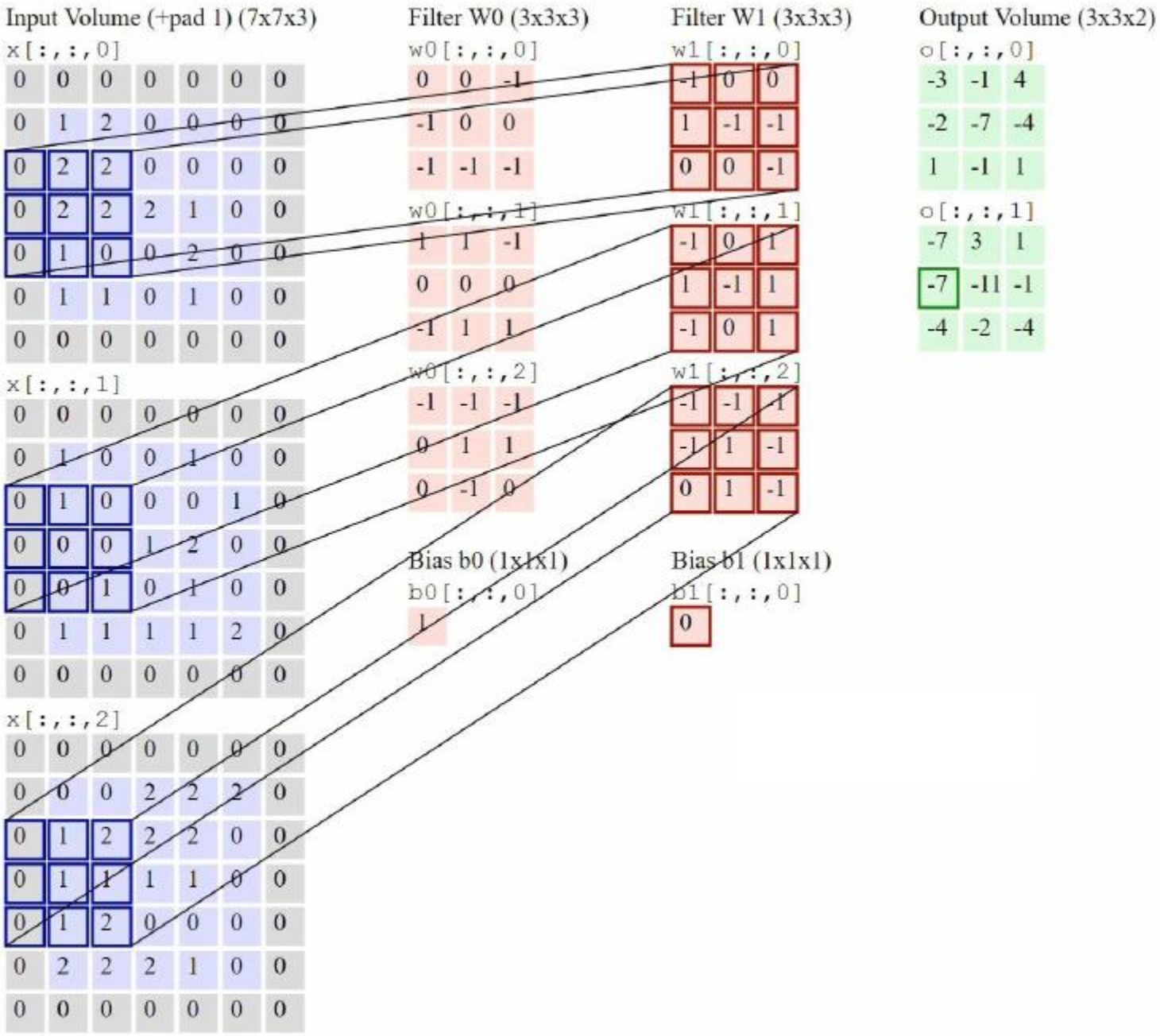


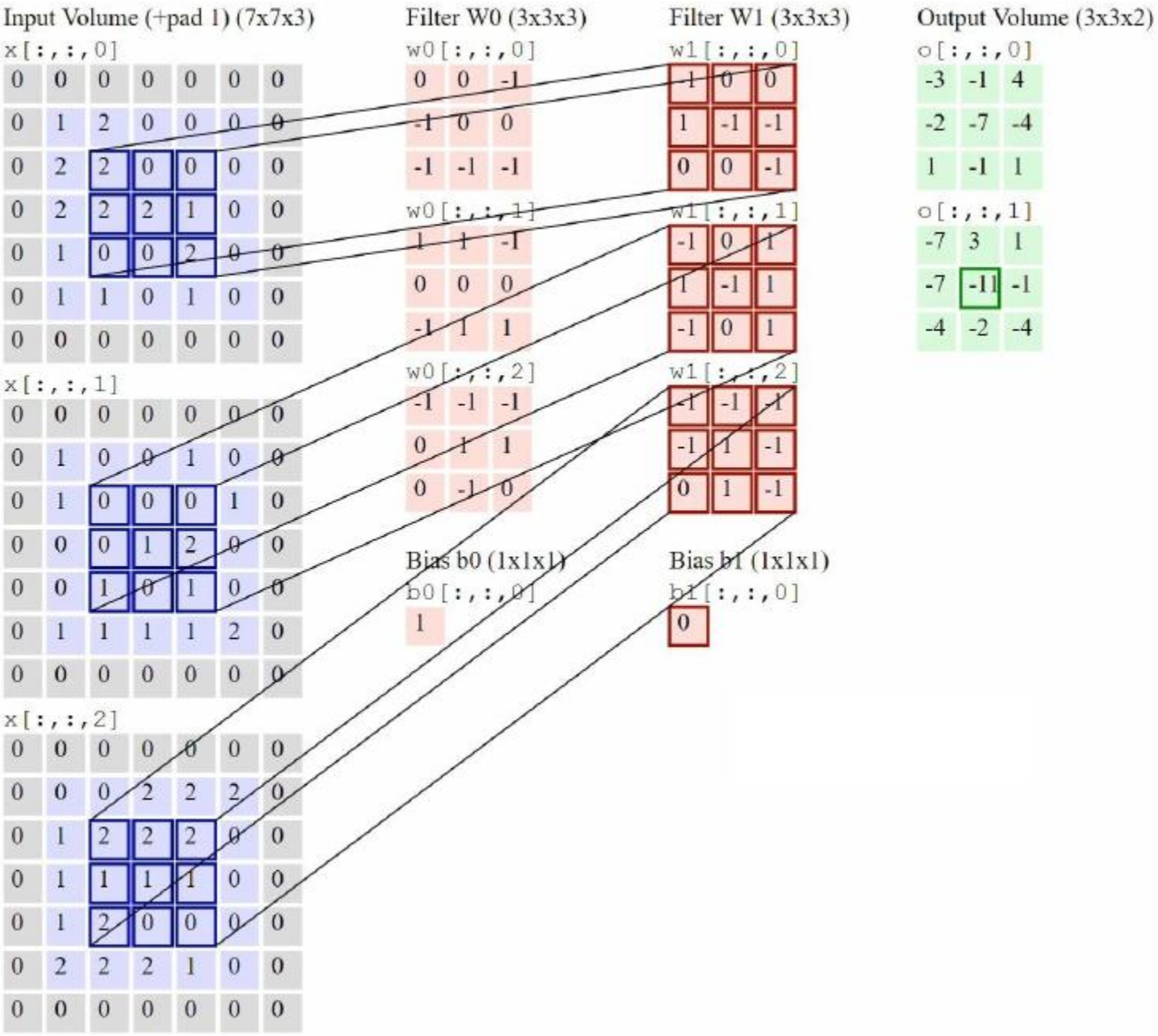
- **Depth:** number of filters
- **Stride:** filter step size (when we “slide” it)
- **Padding:** zero-pad the input

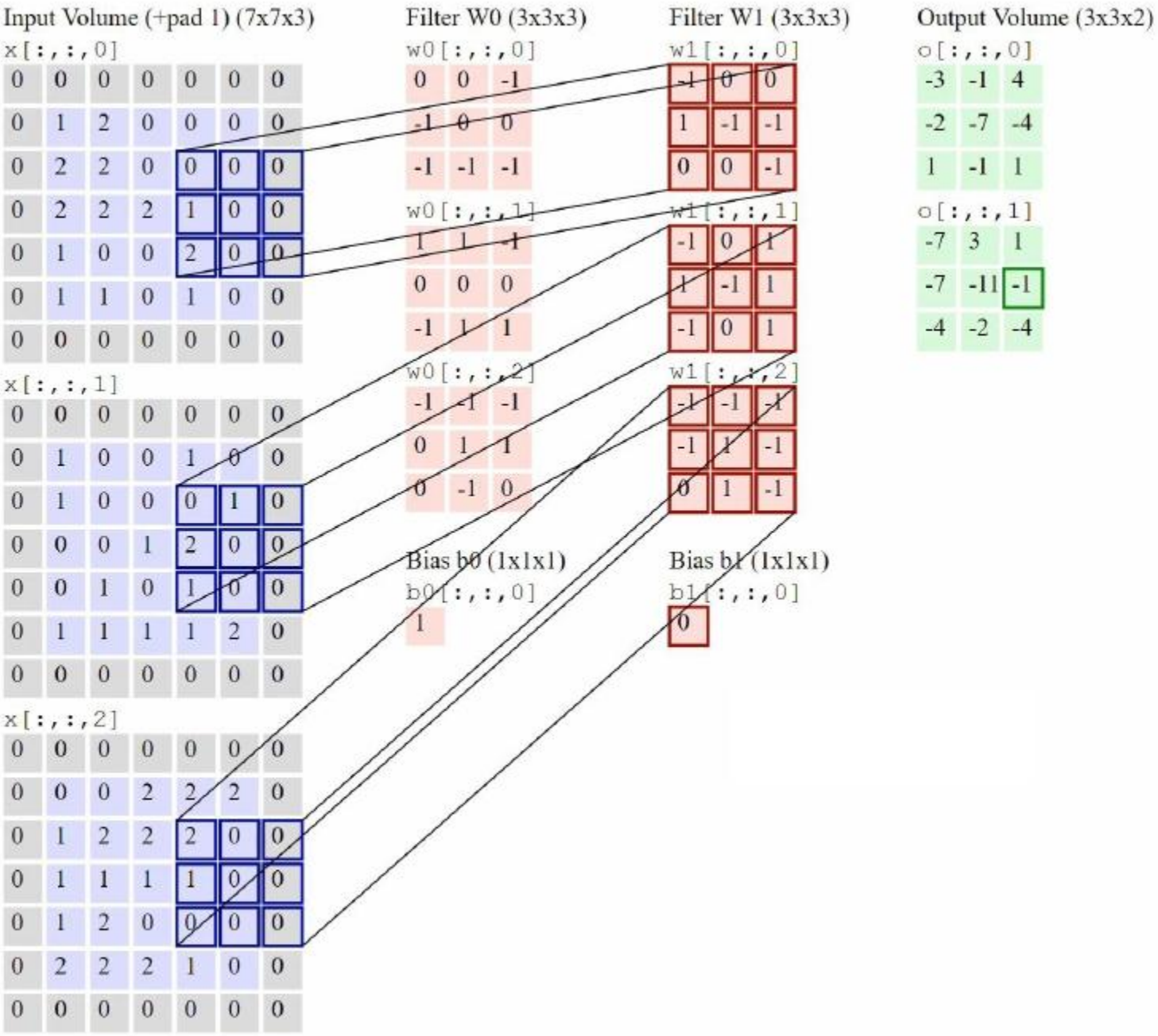

Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 1 2 0 0 0 0
0 2 2 0 0 0 0
0 2 2 2 1 0 0
0 1 0 0 2 0 0
0 1 1 0 1 0 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 0 0 1 0 0
0 1 0 0 0 1 0
0 0 0 1 2 0 0
0 0 1 0 1 0 0
0 1 1 1 1 2 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 0 2 2 2 0
0 1 2 2 2 0 0
0 1 1 1 1 0 0
0 1 2 0 0 0 0
0 2 2 2 1 0 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
0 0 -1
-1 0 0
-1 -1 -1
w0[:,:,1]
1 1 -1
0 0 0
-1 1 1
w0[:,:,2]
-1 -1 -1
0 1 1
0 -1 0
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 0 0
1 -1 -1
0 0 -1
w1[:,:,1]
-1 0 1
1 -1 1
-1 0 1
w1[:,:,2]
-1 -1 -1
-1 1 -1
0 1 -1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
-3 -1 4
-2 -7 -4
1 -1 1
o[:,:,1]
-7 3 1
-7 -11 -1
-4 -2 -4

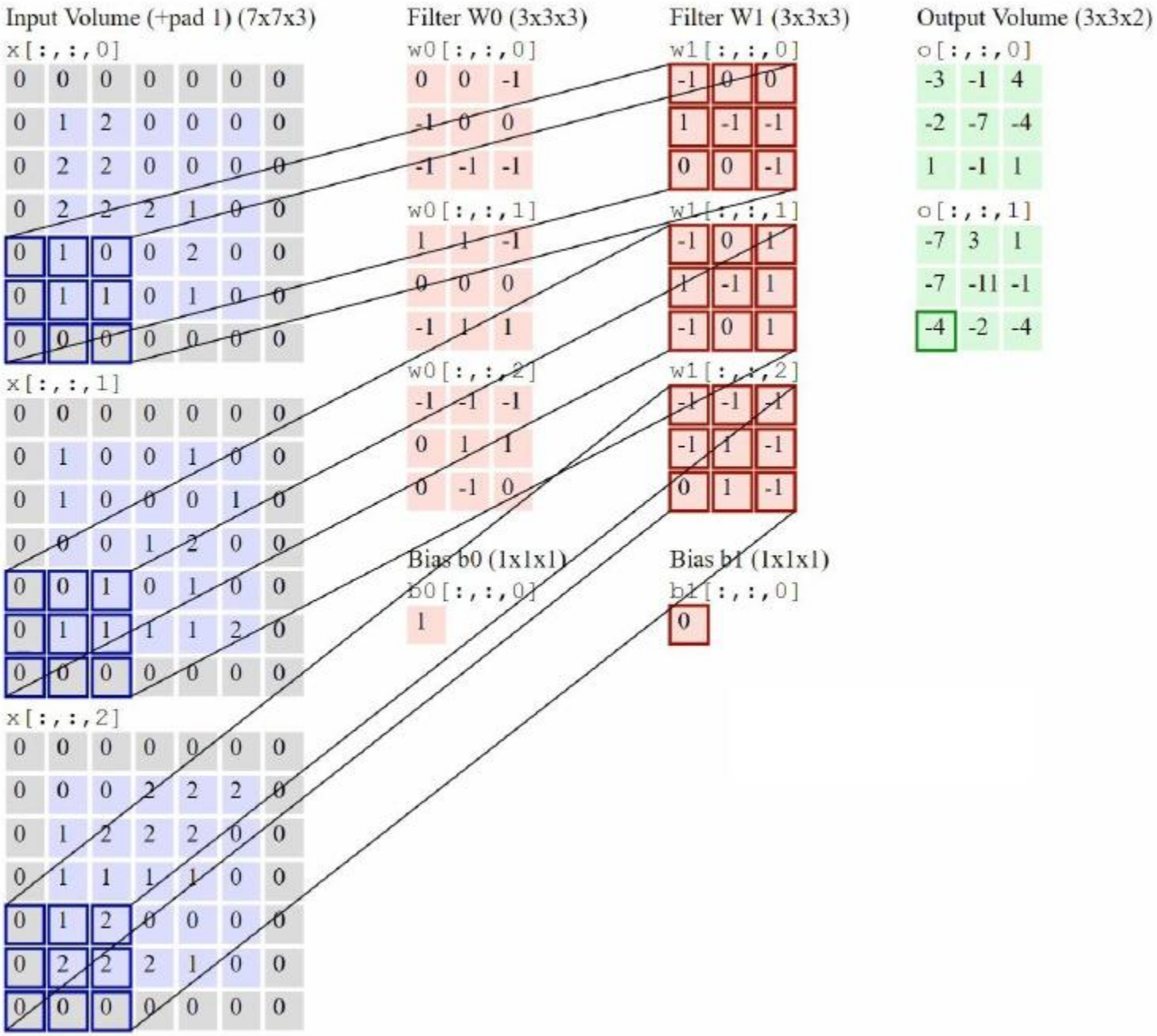

Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 1 2 0 0 0 0
0 2 2 0 0 0 0
0 2 2 2 1 0 0
0 1 0 0 2 0 0
0 1 1 0 1 0 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 0 0 1 0 0
0 1 0 0 0 1 0
0 0 0 1 2 0 0
0 0 1 0 1 0 0
0 1 1 1 1 2 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 0 2 2 2 0
0 1 2 2 2 0 0
0 1 1 1 1 0 0
0 1 2 0 0 0 0
0 2 2 2 1 0 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
0 0 -1
-1 0 0
-1 -1 -1
w0[:,:,1]
1 1 -1
0 0 0
-1 1 1
w0[:,:,2]
-1 -1 -1
0 1 1
0 -1 0
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 0 0
1 -1 -1
0 0 -1
w1[:,:,1]
-1 0 1
1 -1 1
-1 0 1
w1[:,:,2]
-1 -1 -1
-1 1 -1
0 1 -1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
-3 -1 4
-2 -7 -4
1 -1 1
o[:,:,1]
-7 3 1
-7 -11 -1
-4 -2 -4

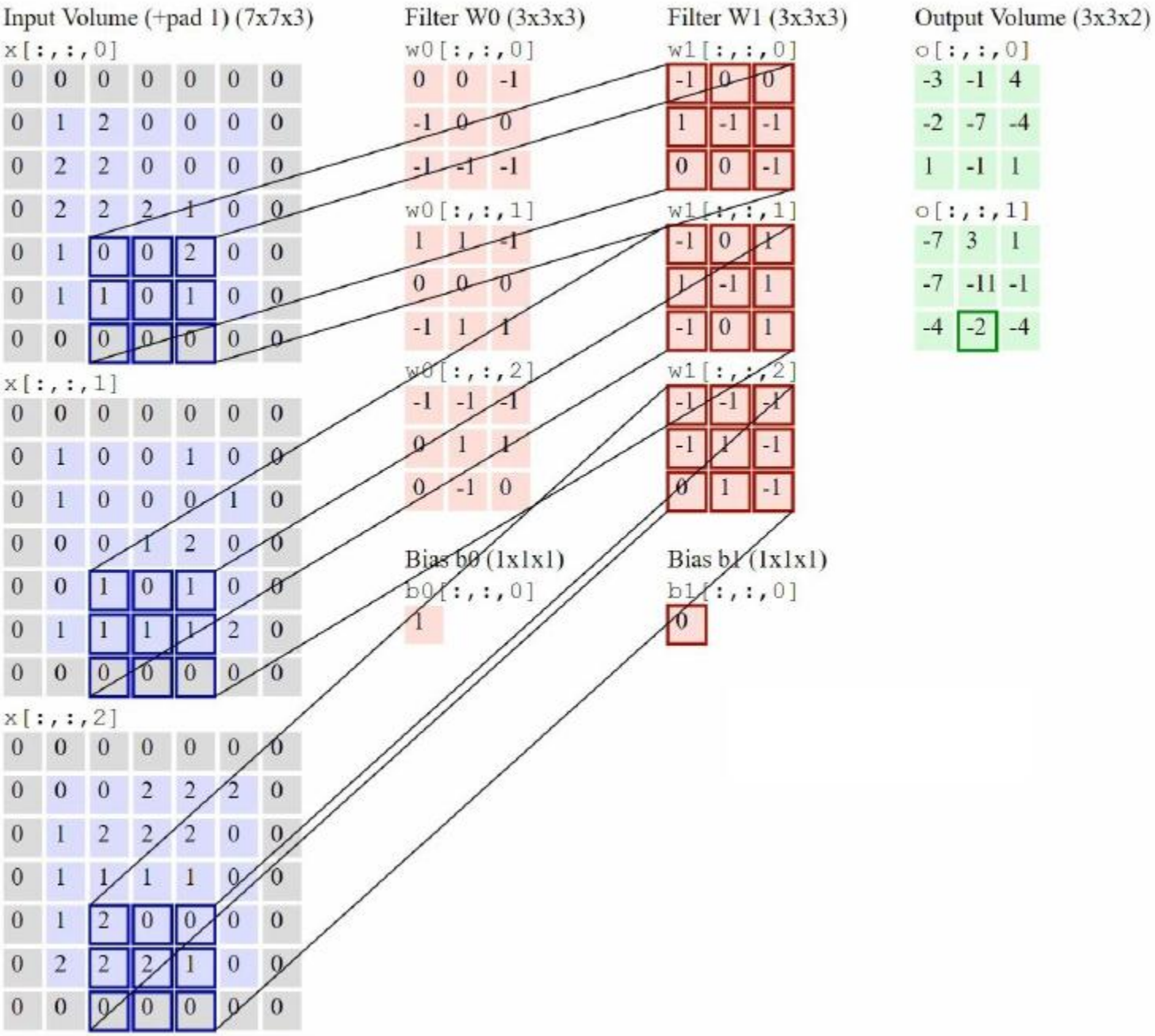

Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 1 2 0 0 0 0
0 2 2 0 0 0 0
0 2 2 2 1 0 0
0 1 0 0 2 0 0
0 1 1 0 1 0 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 0 0 1 0 0
0 1 0 0 0 1 0
0 0 0 1 2 0 0
0 0 1 0 1 0 0
0 1 1 1 1 2 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 0 2 2 2 0
0 1 2 2 2 0 0
0 1 1 1 1 0 0
0 1 2 0 0 0 0
0 2 2 2 1 0 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
0 0 -1
-1 0 0
-1 -1 -1
w0[:,:,1]
1 1 -1
0 0 0
-1 1 1
w0[:,:,2]
-1 -1 -1
0 1 1
0 -1 0
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 0 0
1 -1 -1
0 0 -1
w1[:,:,1]
-1 0 1
1 -1 1
-1 0 1
w1[:,:,2]
-1 -1 -1
-1 1 -1
0 1 -1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
-3 -1 4
-2 -7 -4
1 -1 1
o[:,:,1]
-7 3 1
-7 -11 -1
-4 -2 -4

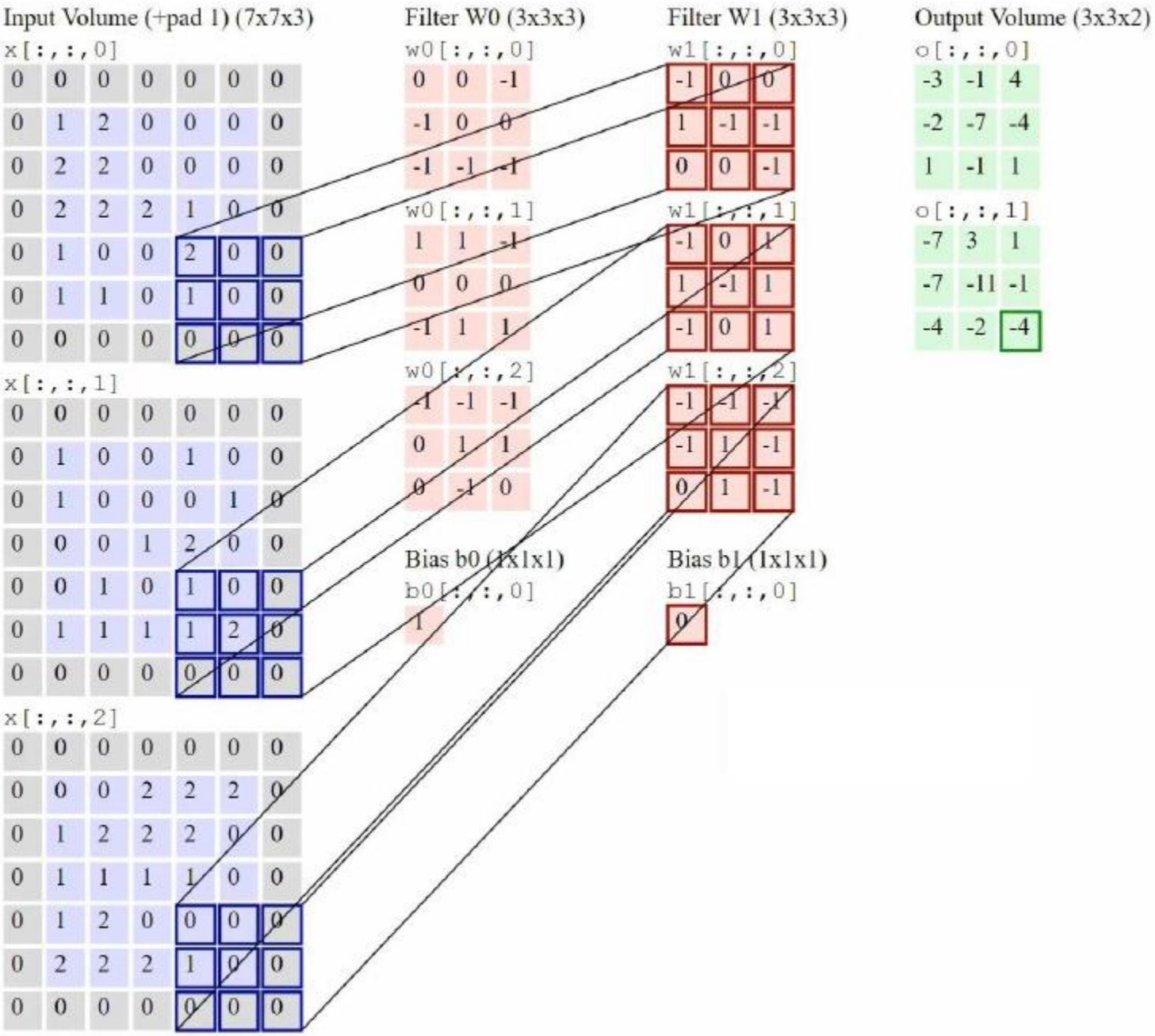

Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 1 2 0 0 0 0
0 2 2 0 0 0 0
0 2 2 2 1 0 0
0 1 0 0 2 0 0
0 1 1 0 1 0 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 0 0 1 0 0
0 1 0 0 0 1 0
0 0 0 1 2 0 0
0 0 1 0 1 0 0
0 1 1 1 1 2 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 0 2 2 2 0
0 1 2 2 2 0 0
0 1 1 1 1 0 0
0 1 2 0 0 0 0
0 2 2 2 1 0 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
0 0 -1
-1 0 0
-1 -1 -1
w0[:,:,1]
1 1 -1
0 0 0
-1 1 1
w0[:,:,2]
-1 -1 -1
0 1 1
0 -1 0
Filter W1 (3x3x3)
w1[:,:,0]
-1 0 0
1 -1 -1
0 0 -1
w1[:,:,1]
-1 0 1
1 -1 1
-1 0 1
w1[:,:,2]
-1 -1 -1
-1 1 -1
0 1 -1
Output Volume (3x3x2)
o[:,:,0]
-3 -1 4
-2 -7 -4
1 -1 1
o[:,:,1]
-7 3 1
-7 -11 -1
-4 -2 -4
Bias b0 (1x1x1)
b0[:,:,0]
1
Bias b1 (1x1x1)
b1[:,:,0]
0


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 1 2 0 0 0 0
0 2 2 0 0 0 0
0 2 2 2 1 0 0
0 1 0 0 2 0 0
0 1 1 0 1 0 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 0 0 1 0 0
0 1 0 0 0 1 0
0 0 0 1 2 0 0
0 0 1 0 1 0 0
0 1 1 1 1 2 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 0 2 2 2 0
0 1 2 2 2 0 0
0 1 1 1 1 0 0
0 1 2 0 0 0 0
0 2 2 2 1 0 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
0 0 -1
-1 0 0
-1 -1 -1
w0[:,:,1]
1 1 -1
0 0 0
-1 1 1
w0[:,:,2]
-1 -1 -1
0 1 1
0 -1 0
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 0 0
1 -1 -1
0 0 -1
w1[:,:,1]
-1 0 1
1 -1 1
-1 0 1
w1[:,:,2]
-1 -1 -1
-1 1 -1
0 1 -1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
-3 -1 4
-2 -7 -4
1 -1 1
o[:,:,1]
-7 3 1
-7 -11 -1
-4 -2 -4


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 1 2 0 0 0 0
0 2 2 0 0 0 0
0 2 2 2 1 0 0
0 1 0 0 2 0 0
0 1 1 0 1 0 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 0 0 1 0 0
0 1 0 0 0 1 0
0 0 0 1 2 0 0
0 0 1 0 1 0 0
0 1 1 1 1 2 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 0 2 2 2 0
0 1 2 2 2 0 0
0 1 1 1 1 0 0
0 1 2 0 0 0 0
0 2 2 2 1 0 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
0 0 -1
-1 0 0
-1 -1 -1
w0[:,:,1]
1 1 -1
0 0 0
-1 1 1
w0[:,:,2]
-1 -1 -1
0 1 1
0 -1 0
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 0 0
1 -1 -1
0 0 -1
w1[:,:,1]
-1 0 1
1 -1 1
-1 0 1
w1[:,:,2]
-1 -1 -1
-1 1 -1
0 1 -1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
-3 -1 4
-2 -7 -4
1 -1 1
o[:,:,1]
-7 3 1
-7 -11 -1
-4 -2 -4


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 1 2 0 0 0 0
0 2 2 0 0 0 0
0 2 2 2 1 0 0
0 1 0 0 2 0 0
0 1 1 0 1 0 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 0 0 1 0 0
0 1 0 0 0 1 0
0 0 0 1 2 0 0
0 0 1 0 1 0 0
0 1 1 1 1 2 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 0 2 2 2 0
0 1 2 2 2 0 0
0 1 1 1 1 0 0
0 1 2 0 0 0 0
0 2 2 2 1 0 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
0 0 -1
-1 0 0
-1 -1 -1
w0[:,:,1]
1 1 -1
0 0 0
-1 1 1
w0[:,:,2]
-1 -1 -1
0 1 1
0 -1 0
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 0 0
1 -1 -1
0 0 -1
w1[:,:,1]
-1 0 1
1 -1 1
-1 0 1
w1[:,:,2]
-1 -1 -1
-1 1 -1
0 1 -1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
-3 -1 4
-2 -7 -4
1 -1 1
o[:,:,1]
-7 3 1
-7 -11 -1
-4 -2 -4

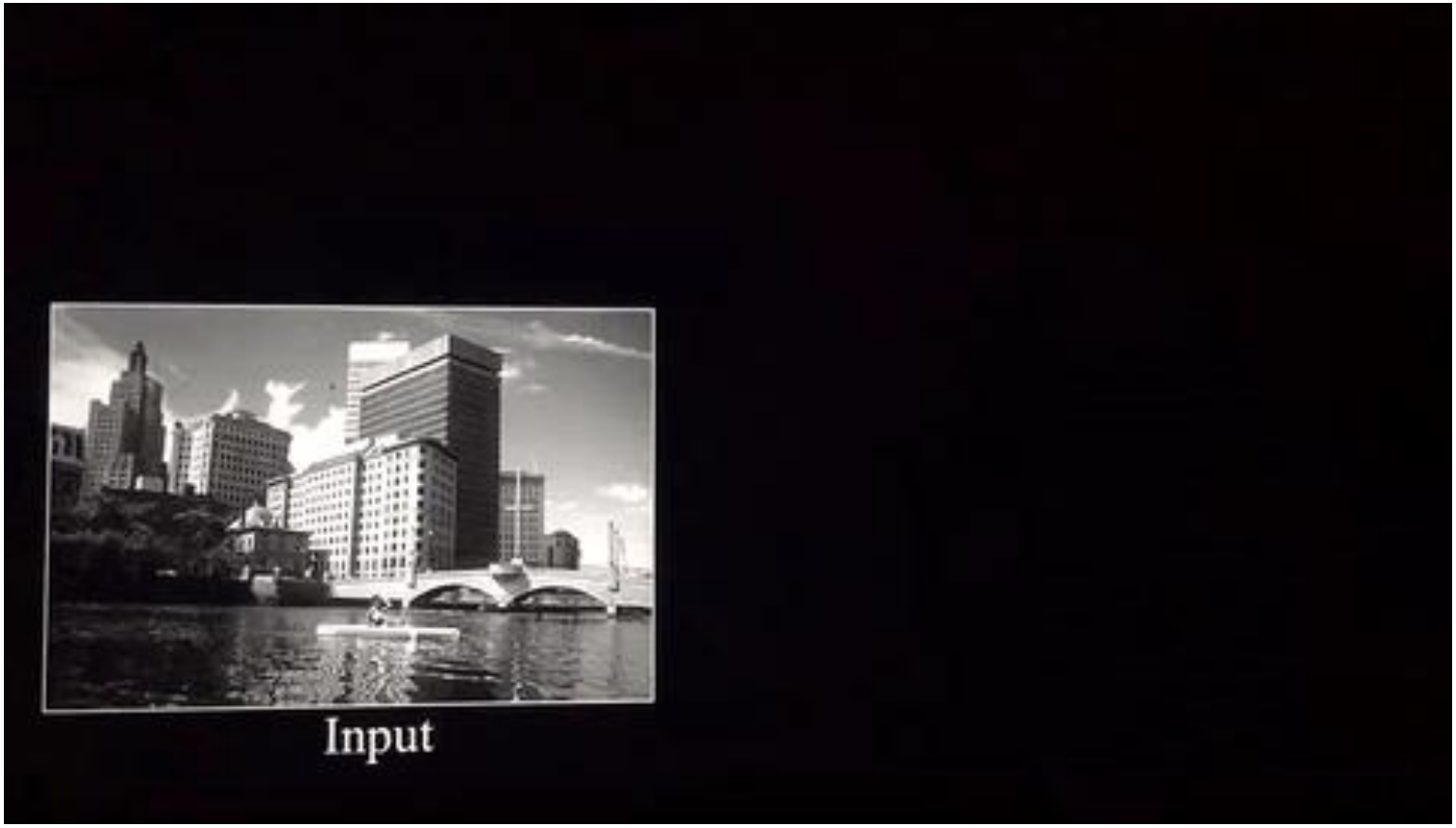

Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 1 2 0 0 0 0
0 2 2 0 0 0 0
0 2 2 2 1 0 0
0 1 0 0 2 0 0
0 1 1 0 1 0 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 0 0 1 0 0
0 1 0 0 0 1 0
0 0 0 1 2 0 0
0 0 1 0 1 0 0
0 1 1 1 1 2 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 0 2 2 2 0
0 1 2 2 2 0 0
0 1 1 1 1 0 0
0 1 2 0 0 0 0
0 2 2 2 1 0 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
0 0 -1
-1 0 0
-1 -1 -1
w0[:,:,1]
1 1 -1
0 0 0
-1 1 1
w0[:,:,2]
-1 -1 -1
0 1 1
0 -1 0
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 0 0
1 -1 -1
0 0 -1
w1[:,:,1]
-1 0 1
1 -1 1
-1 0 1
w1[:,:,2]
-1 -1 -1
-1 1 -1
0 1 -1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
-3 -1 4
-2 -7 -4
1 -1 1
o[:,:,1]
-7 3 1
-7 -11 -1
-4 -2 -4


Input Volume (+pad 1) (7x7x3)
x[:,:,0]
0 0 0 0 0 0 0
0 1 2 0 0 0 0
0 2 2 0 0 0 0
0 2 2 2 1 0 0
0 1 0 0 2 0 0
0 1 1 0 1 0 0
0 0 0 0 0 0 0
x[:,:,1]
0 0 0 0 0 0 0
0 1 0 0 1 0 0
0 1 0 0 0 1 0
0 0 0 1 2 0 0
0 0 1 0 1 0 0
0 1 1 1 1 2 0
0 0 0 0 0 0 0
x[:,:,2]
0 0 0 0 0 0 0
0 0 0 2 2 2 0
0 1 2 2 2 0 0
0 1 1 1 1 0 0
0 1 2 0 0 0 0
0 2 2 2 1 0 0
0 0 0 0 0 0 0
Filter W0 (3x3x3)
w0[:,:,0]
0 0 -1
-1 0 0
-1 -1 -1
w0[:,:,1]
1 1 -1
0 0 0
-1 1 1
w0[:,:,2]
-1 -1 -1
0 1 1
0 -1 0
Bias b0 (1x1x1)
b0[:,:,0]
1
Filter W1 (3x3x3)
w1[:,:,0]
-1 0 0
1 -1 -1
0 0 -1
w1[:,:,1]
-1 0 1
1 -1 1
-1 0 1
w1[:,:,2]
-1 -1 -1
-1 1 -1
0 1 -1
Bias b1 (1x1x1)
b1[:,:,0]
0
Output Volume (3x3x2)
o[:,:,0]
-3 -1 4
-2 -7 -4
1 -1 1
o[:,:,1]
-7 3 1
-7 -11 -1
-4 -2 -4

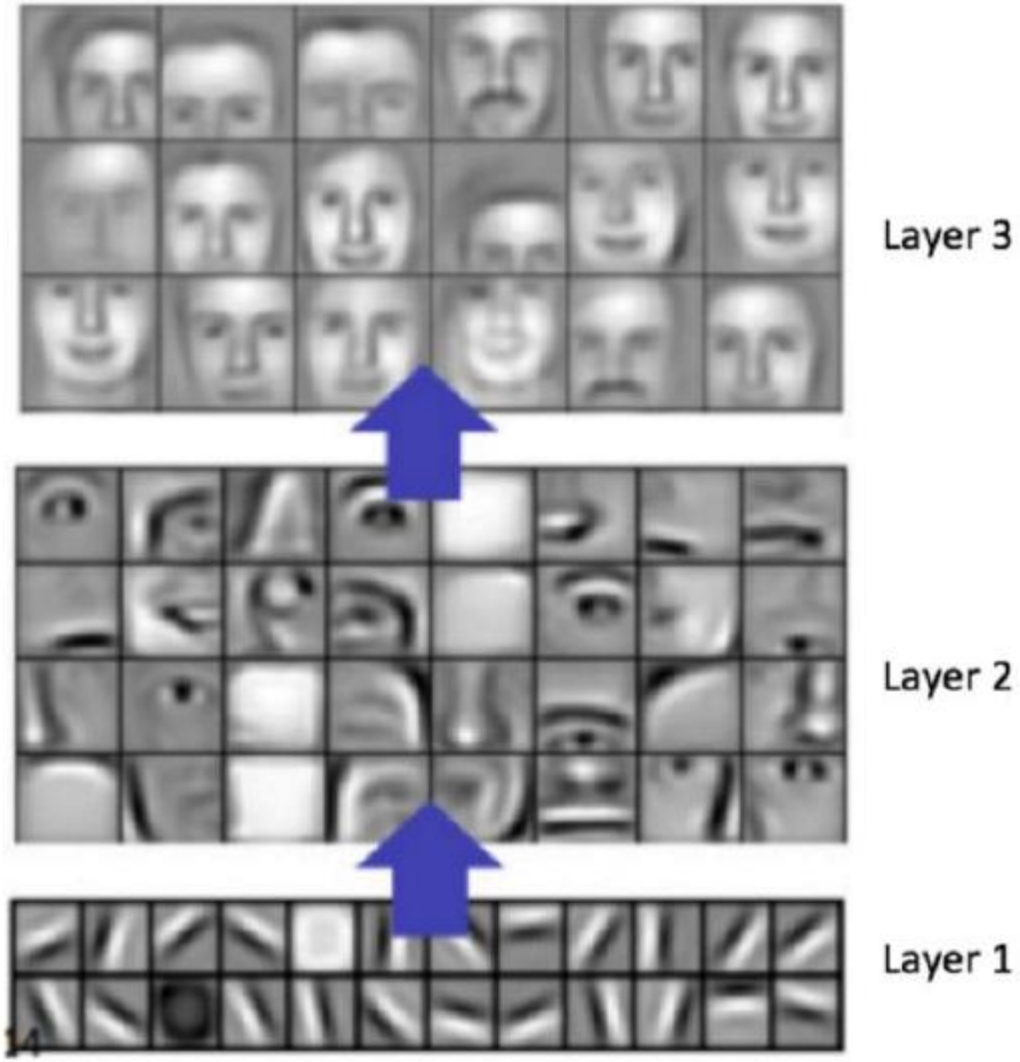
Convolution

Operation	Filter	Convolved Image
Identity	$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \end{bmatrix}$	
Edge detection	$\begin{bmatrix} 1 & 0 & -1 \\ 0 & 0 & 0 \\ -1 & 0 & 1 \end{bmatrix}$	
	$\begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix}$	
	$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$	

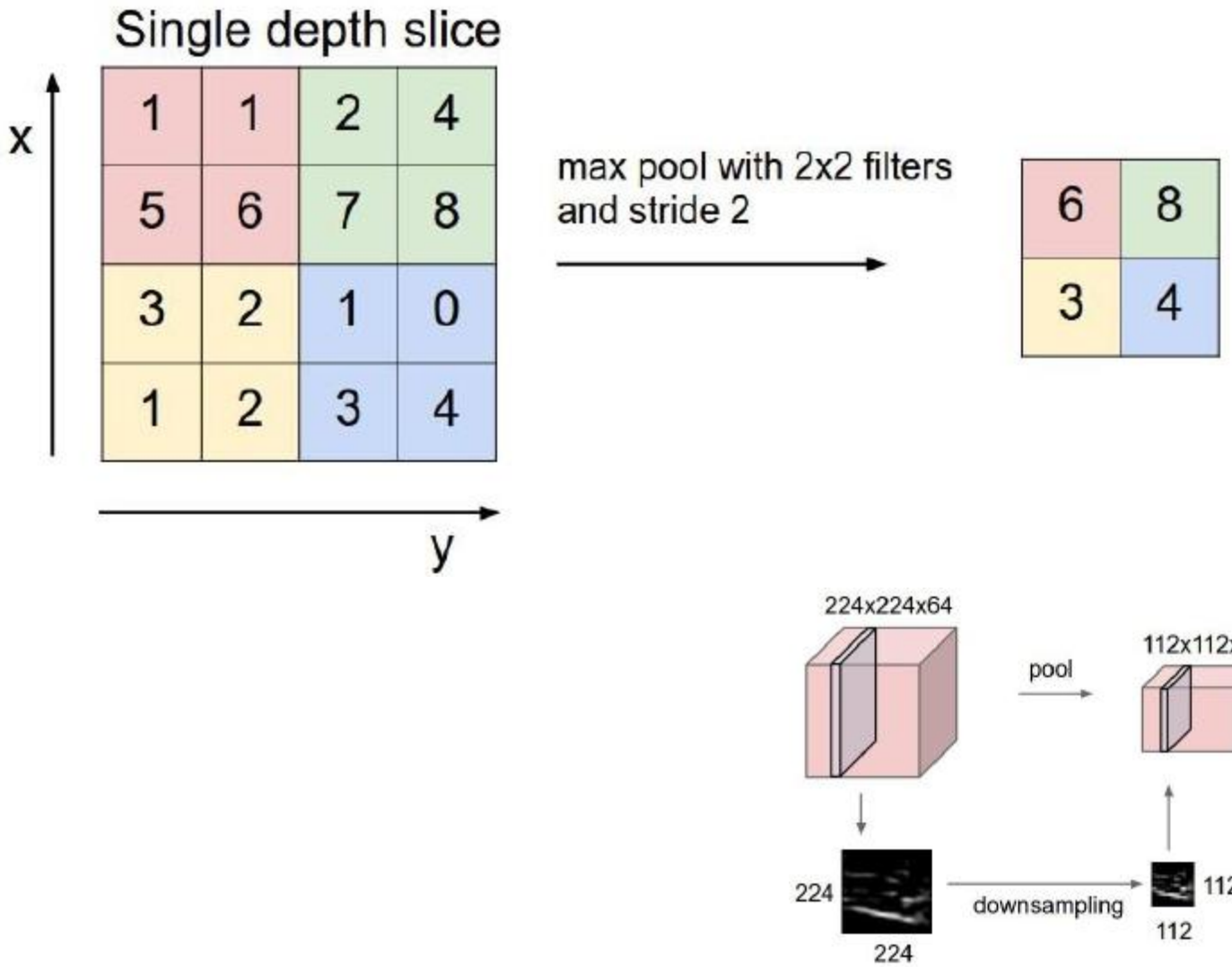
Convolution



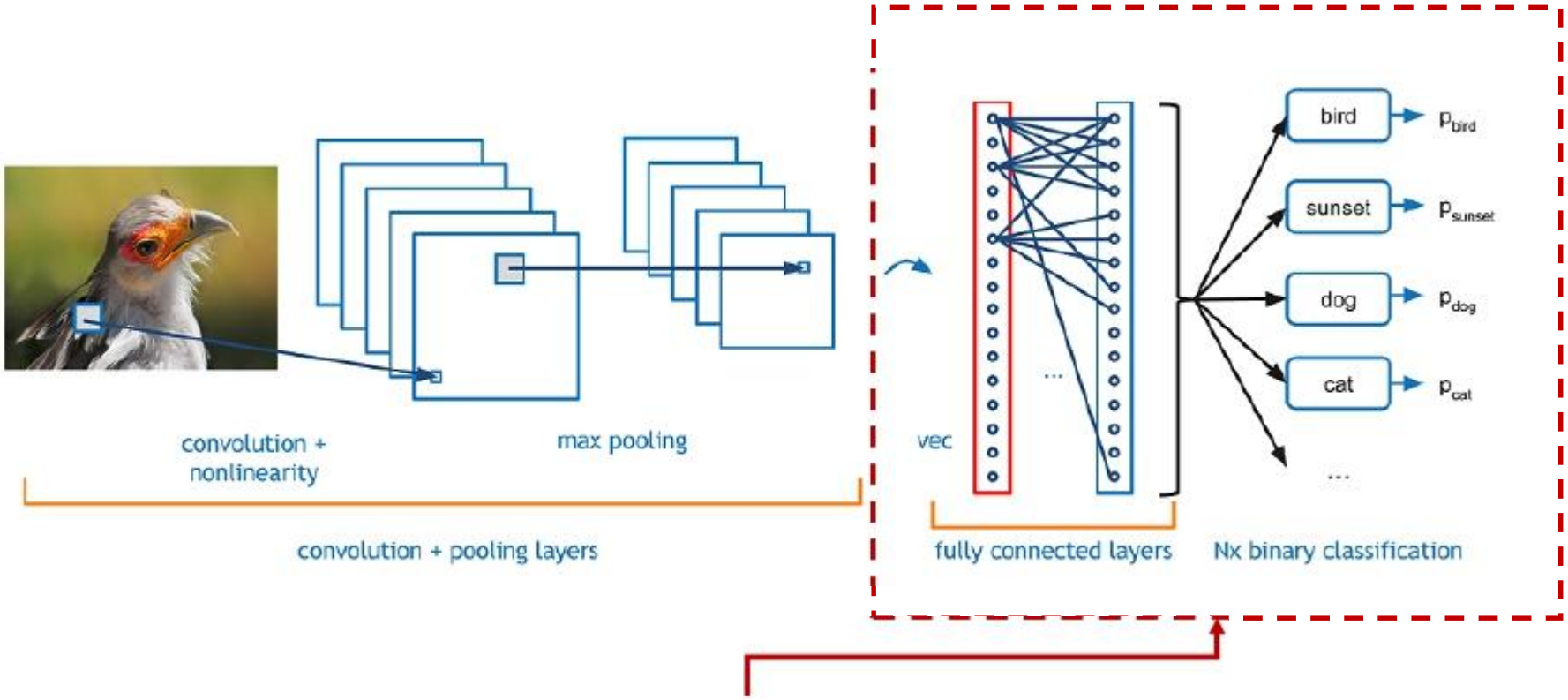
Convolution: Representation Learning



ConvNets: Pooling



Same Architecture, Many Applications



This part might look different for:

- Different image classification domains
- Image captioning with recurrent neural networks
- Image object localization with bounding box
- Image segmentation with fully convolutional networks
- Image segmentation with deconvolution layers

Case Study: **ImageNet**

What is ImageNet?

- **ImageNet:** dataset of 14+ million images (21,841 categories)
- Let's take the high level category of **fruit** as an example:
  - Total 188,000 images of fruit
  - There are 1206 Granny Smith apples:

Networks:

- AlexNet (2012)
- ZFNet (2013)
- VGGNet (2014)
- GoogLeNet (2014)
- ResNet (2015)
- CUImage (2016)
- SENet (2017)

Competition: **ILSVRC:** ImageNet Large Scale Visual Recognition Challenge

- Human error: 5.1%
  - Surpassed in 2015

AlexNet (2012): First CNN (15.4%)

- 8 layers
- 61 million parameters

ZFNet (2013): 15.4% to 11.2%

- 8 layers
- More filters. Denser stride.

VGGNet (2014): 11.2% to 7.3%

- Beautifully uniform:
  3x3 conv, stride 1, pad 1, 2x2 max pool
- 16 layers
- 138 million parameters

GoogLeNet (2014): 11.2% to 6.7%

- Inception modules
- 22 layers
- 5 million parameters
  (throw away fully connected layers)

ResNet (2015): 6.7% to 3.57%

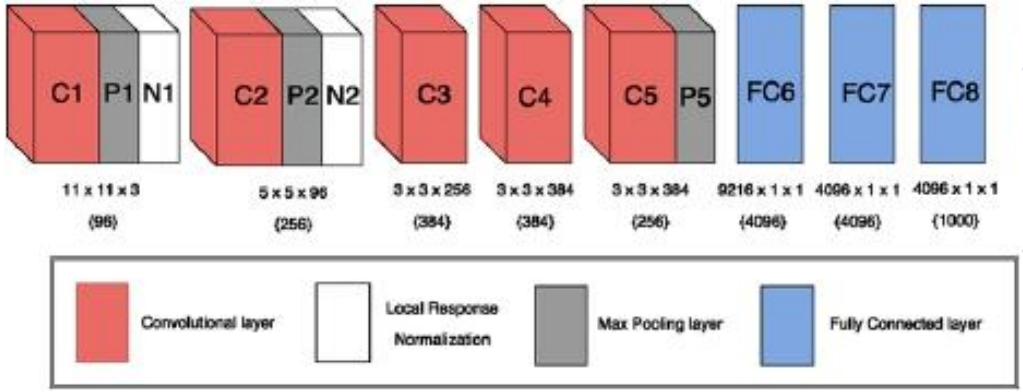
- More layers = better performance
- 152 layers

CUImage (2016): 3.57% to 2.99%

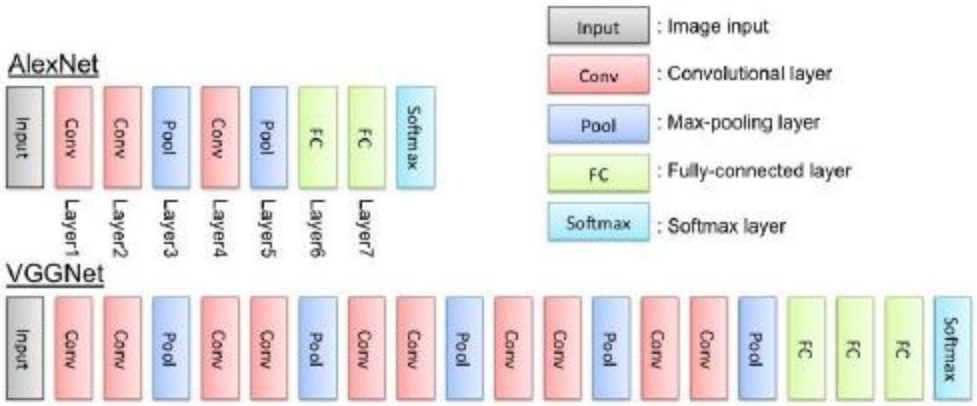
- Ensemble of 6 models

SENet (2017): 2.99% to 2.251%

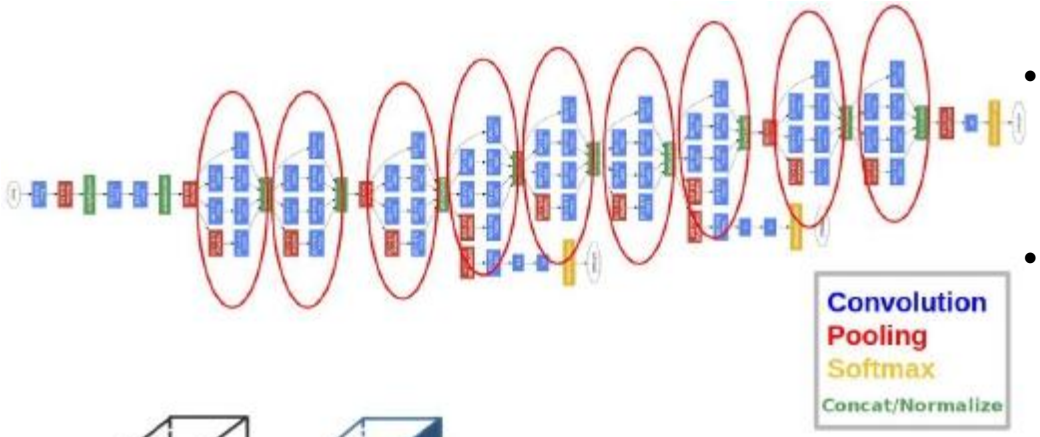
- Squeeze and excitation block: network is allowed to adaptively adjust the weighting of each feature map in the convolutional block.

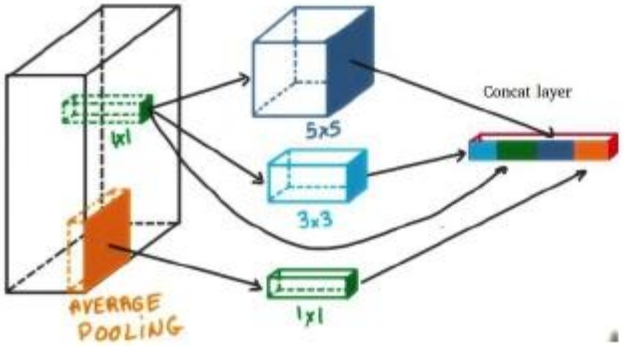


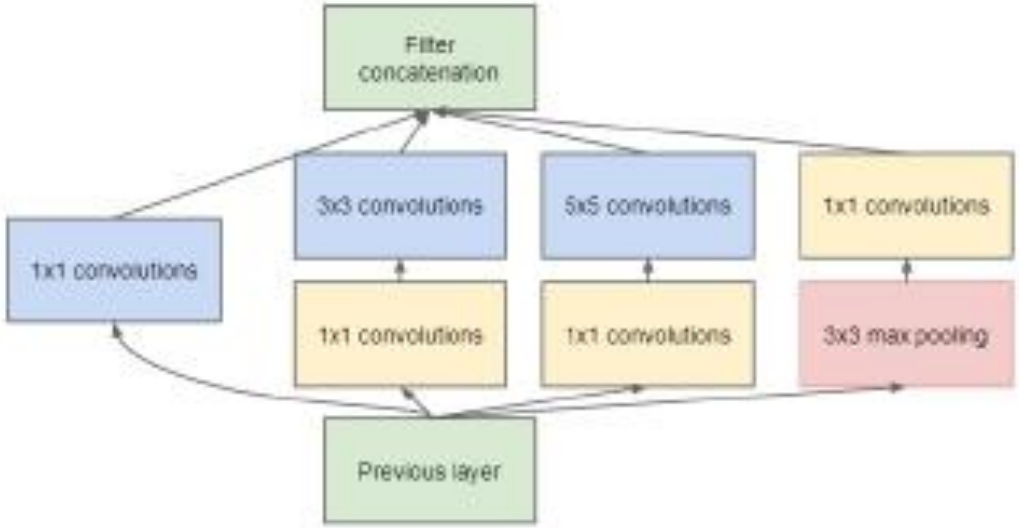
- **AlexNet (2012): First CNN (15.4%)**
  - 8 layers
  - 61 million parameters
- **ZFNet (2013): 15.4% to 11.2%**
  - 8 layers
  - More filters. Denser stride.
- **VGGNet (2014): 11.2% to 7.3%**
  - Beautifully uniform:
    3x3 conv, stride 1, pad 1, 2x2 max pool
  - 16 layers
  - 138 million parameters
- **GoogLeNet (2014): 11.2% to 6.7%**
  - Inception modules
  - 22 layers
  - 5 million parameters
    (throw away fully connected layers)
- **ResNet (2015): 6.7% to 3.57%**
  - More layers = better performance
  - 152 layers
- **CUImage (2016): 3.57% to 2.99%**
  - Ensemble of 6 models



- **AlexNet (2012): First CNN (15.4%)**
  - 8 layers
  - 61 million parameters
- **ZFNet (2013): 15.4% to 11.2%**
  - 8 layers
  - More filters. Denser stride.
- **VGGNet (2014): 11.2% to 7.3%**
  - Beautifully uniform:
    3x3 conv, stride 1, pad 1, 2x2 max pool
  - 16 layers
  - 138 million parameters
- **GoogLeNet (2014): 11.2% to 6.7%**
  - Inception modules
  - 22 layers
  - 5 million parameters
    (throw away fully connected layers)
- **ResNet (2015): 6.7% to 3.57%**
  - More layers = better performance
  - 152 layers
- **CUImage (2016): 3.57% to 2.99%**
  - Ensemble of 6 models

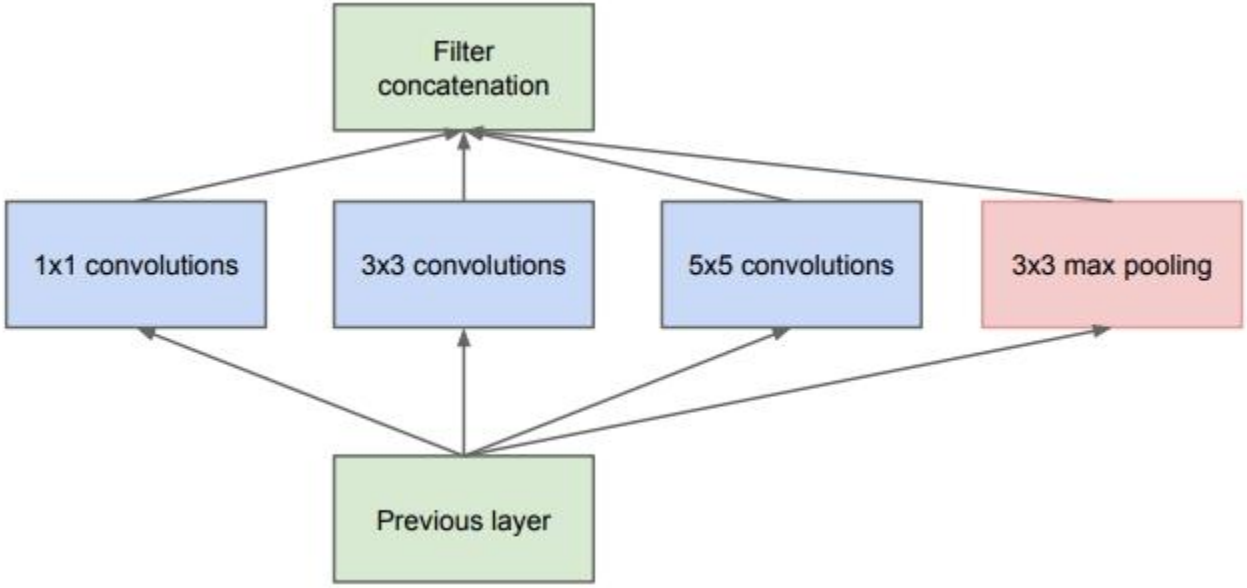




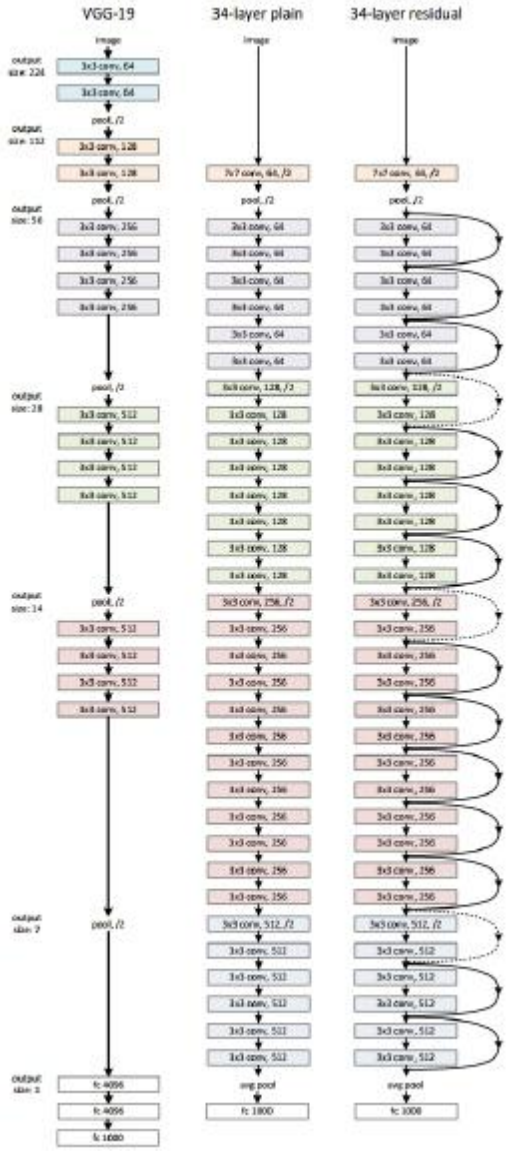


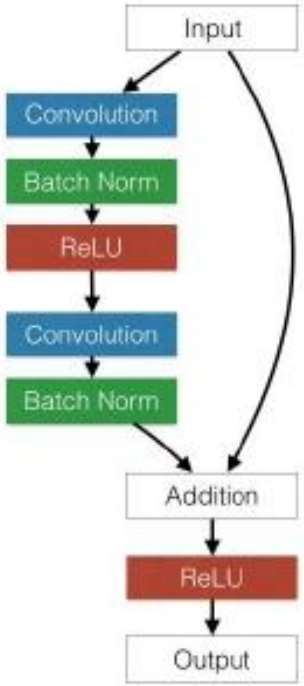
- **AlexNet (2012): First CNN (15.4%)**
  - 8 layers
  - 61 million parameters
- **ZFNet (2013): 15.4% to 11.2%**
  - 8 layers
  - More filters. Denser stride.
- **VGGNet (2014): 11.2% to 7.3%**
  - Beautifully uniform:
    3x3 conv, stride 1, pad 1, 2x2 max pool
  - 16 layers
  - 138 million parameters
- **GoogLeNet (2014): 11.2% to 6.7%**
  - Inception modules
  - 22 layers
  - 5 million parameters
    (throw away fully connected layers)
- **ResNet (2015): 6.7% to 3.57%**
  - More layers = better performance
  - 152 layers
- **CUImage (2016): 3.57% to 2.99%**
  - Ensemble of 6 models

Inception Module



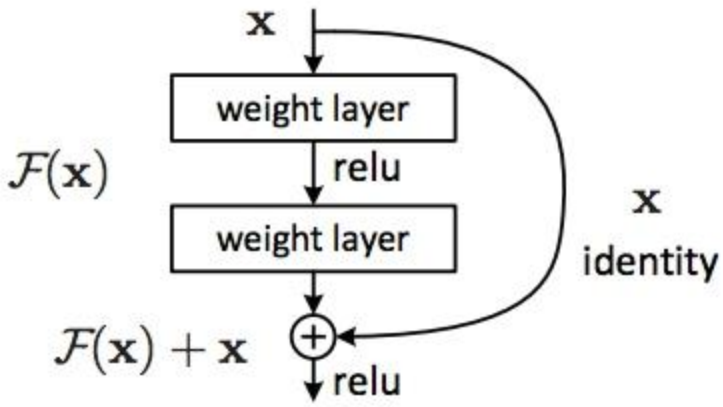
- **Process:** do different size convolutions, and concatenate
- Convolution sizes:
  - Smaller convolutions: local features with high resolution
  - Larger convolutions: high-abstracted features
- **Result**: Fewer parameters and better performance



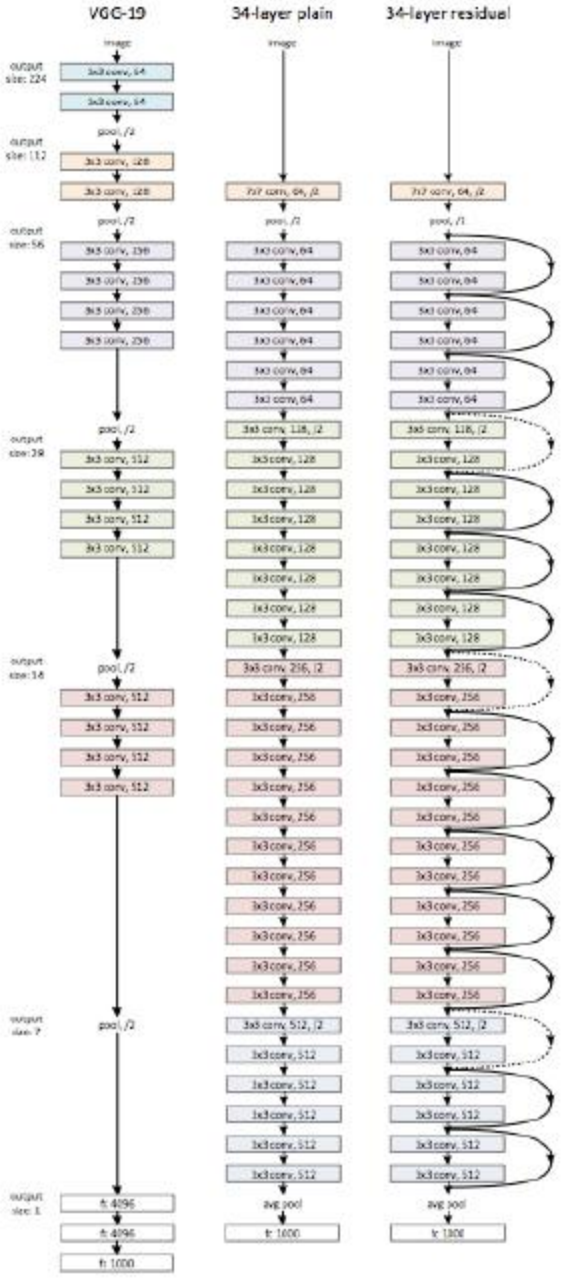


- **AlexNet (2012): First CNN (15.4%)**
  - 8 layers
  - 61 million parameters
- **ZFNet (2013): 15.4% to 11.2%**
  - 8 layers
  - More filters. Denser stride.
- **VGGNet (2014): 11.2% to 7.3%**
  - Beautifully uniform:
    3x3 conv, stride 1, pad 1, 2x2 max pool
  - 16 layers
  - 138 million parameters
- **GoogLeNet (2014): 11.2% to 6.7%**
  - Inception modules
  - 22 layers
  - 5 million parameters
    (throw away fully connected layers)
- **ResNet (2015): 6.7% to 3.57%**
  - More layers = better performance
  - 152 layers
- **CUImage (2016): 3.57% to 2.99%**
  - Ensemble of 6 models

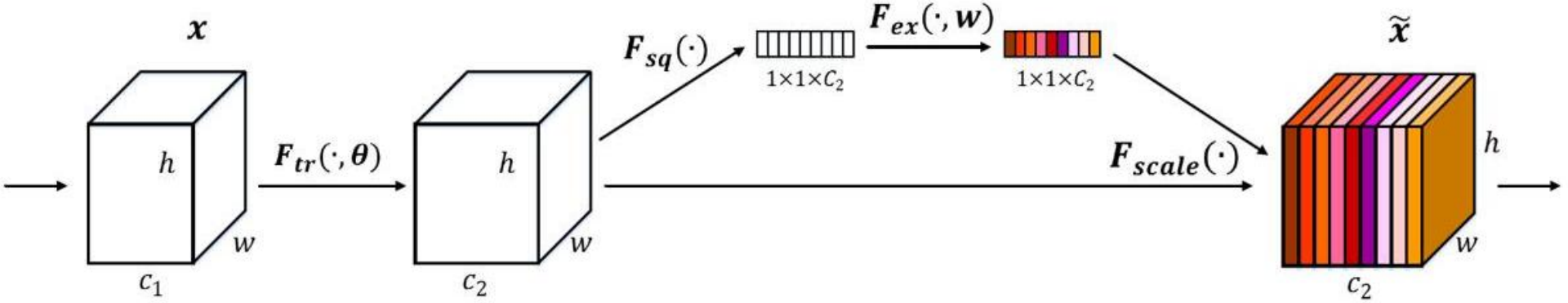
Residual Block



- **Initial Observation:**
  - Network depth often increases representation power, but is harder to train.
- Residual Block:
  - Repeat a simple network block (think: RNN)
  - Pass input along without transformation: help ensure that each layer learns something new



SENet: Squeeze-and-Excitation Networks



Squeeze	Excitation	Scale
• Shrinking feature maps $\in \mathbb{R}^{w\times h\times c_2}$ through spatial dimensions $(w \times h)$ • Global distribution of channel-wise responses	• Learning $W \in \mathbb{R}^{c_2\times c_2}$ to explicitly model channel-association • Gating mechanism to produce channel-wise weights	• Reweighting the feature maps $\in \mathbb{R}^{w\times h\times c_2}$

- **Content-aware channel weighting:** Add parameters to each channel of a convolutional block so that the network can adaptively adjust the weighting of each feature map
- This approach is simple and can be added to any model
  - **Takeaway for thought:** Parameterize everything (that's cost-effective) including higher-order hyper-parameters.

Fully Convolutional Networks

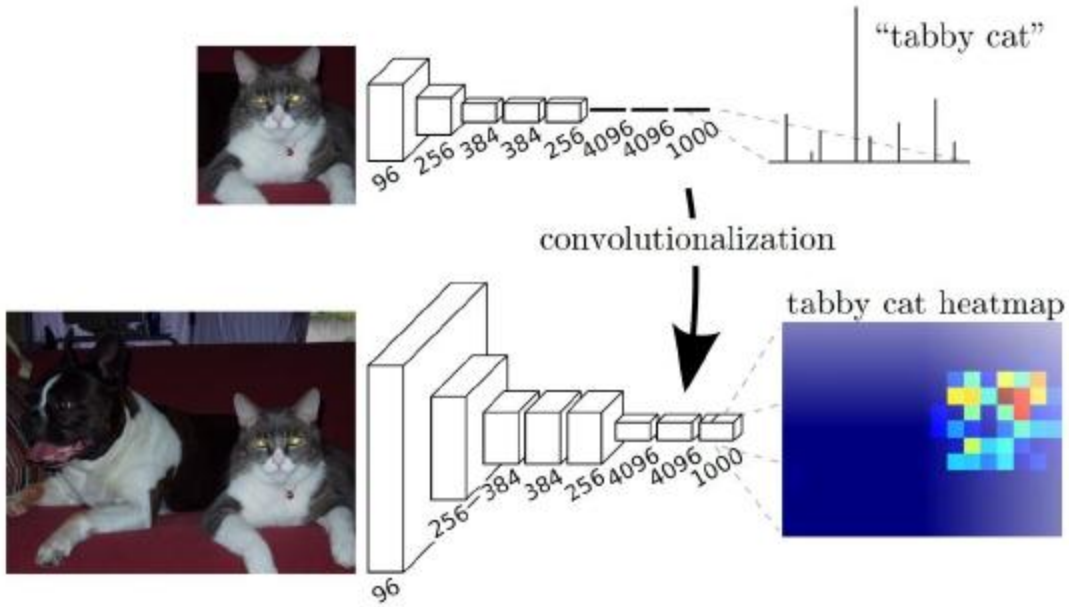
- **Goal:** Classify every pixel in an image.
- **Difficulty:** Hard
- Why?
  - When precise boundaries of objects matter (medical, driving)
  - Useful for fusing with other sensors (LIDAR)

FCN (Nov 2014)

Paper: “Fully Convolutional Networks for Semantic Segmentation”

- Repurpose ImageNet pretrained nets
- Upsample using deconvolution
- Skip connections to improve coarseness of upsampling

Computer Vision: Segmentation



Original

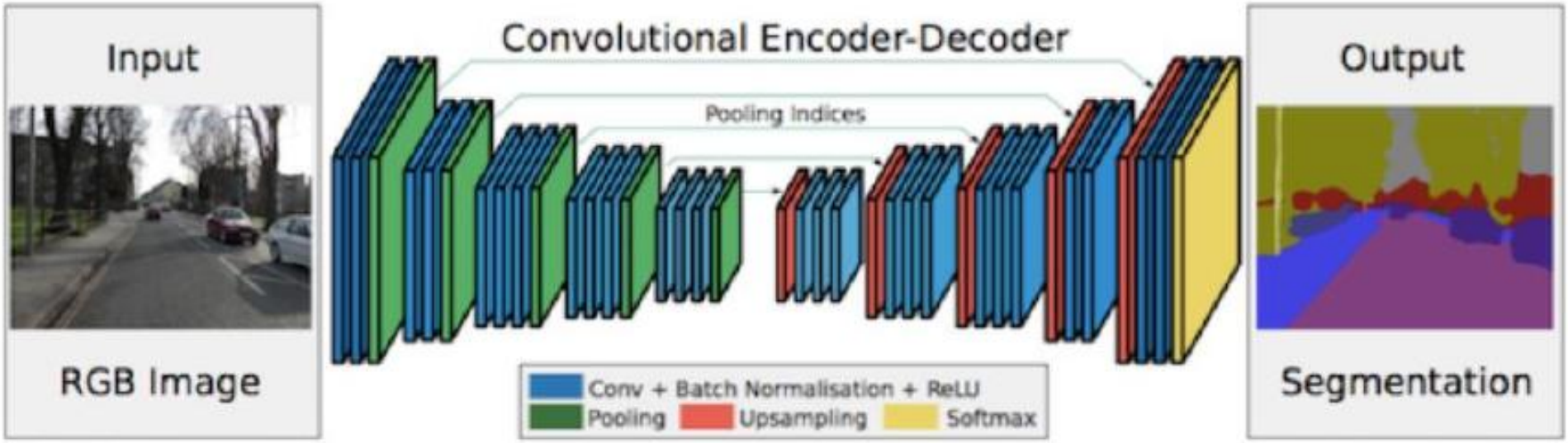
Ground Truth

FCN-8

SegNet (Nov 2015)

Paper: “SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation”

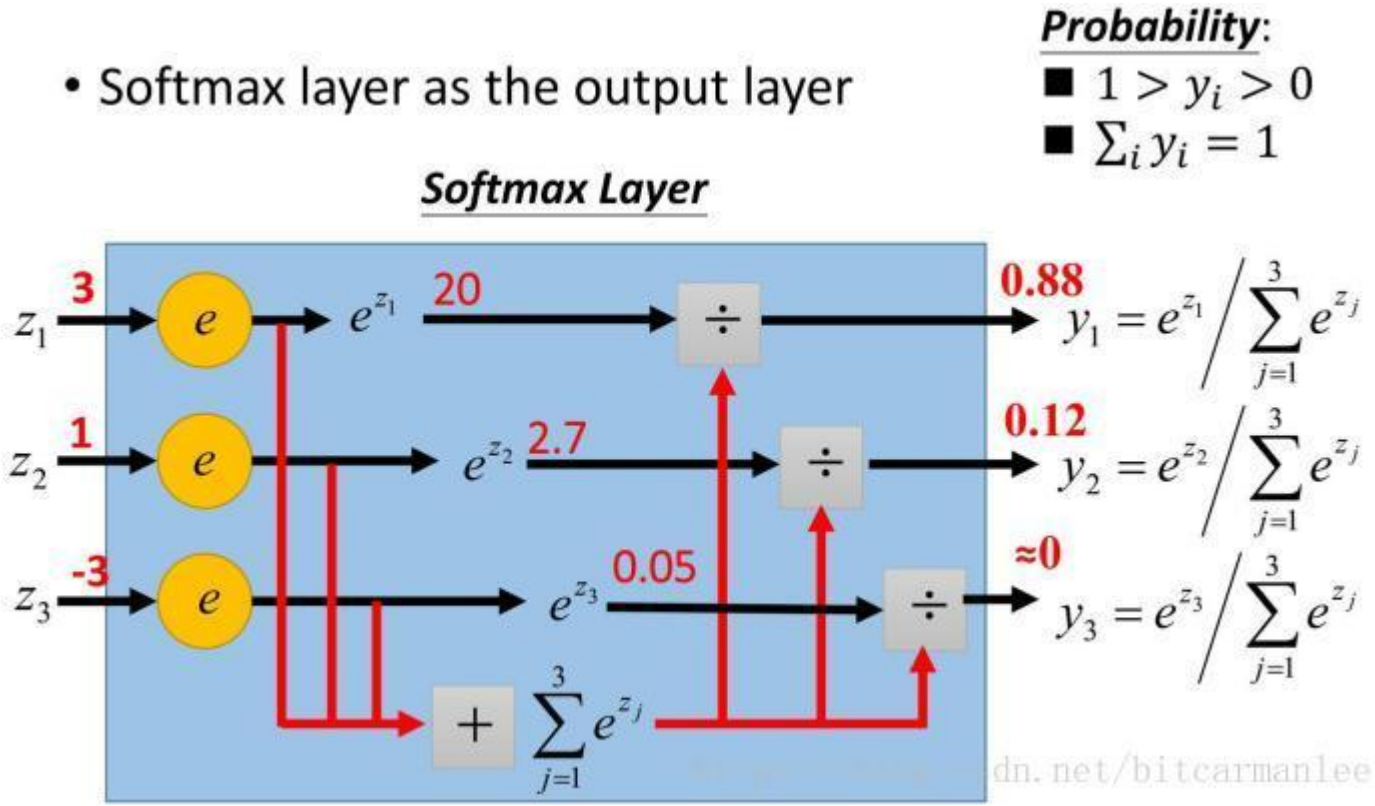
- Maxpooling indices transferred to decoder to improve the segmentation resolution.



假设有一个数组V，V_i表示V中的第i个元素，那么这个元素的softmax值为:

$$S_i = \frac{e^i}{\sum_j e^j}$$

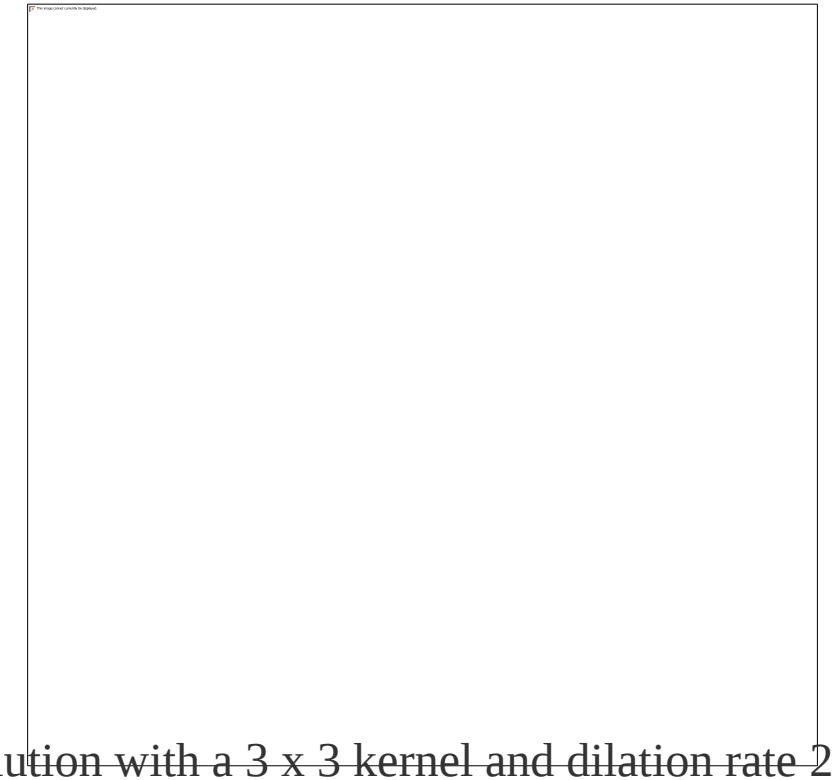
该元素的softmax值，就是该元素的指数与所有元素指数和的比值。



Dilated Convolutions (Nov 2015)

Paper: “Multi-Scale Context Aggregation by Dilated Convolutions”

- Since pooling decreases resolution:
  - Added “dilated convolution layer”
- Still interpolate up from 1/8 of original image size



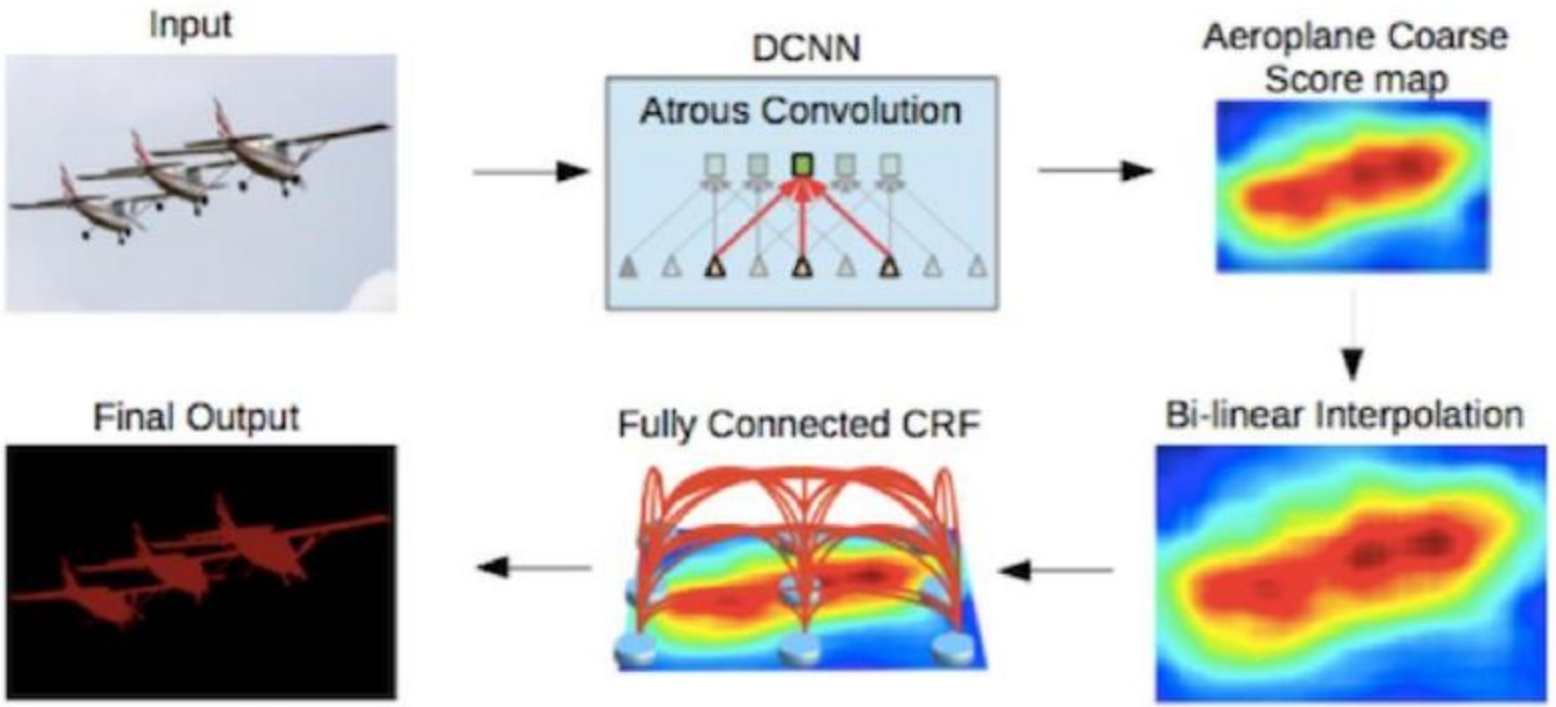
Dilated Convolution with a 3 x 3 kernel and dilation rate 2

Caffe

DeepLap v1, v2 (Jun 2016)

Paper: “DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs”

- Added fully-connected Conditional Random Fields (CRFs) – as a post-processing step
  - Smooth segmentation based on the underlying image intensities



Key Aspects of Segmentation

- **Fully convolutional networks (FCNs)** - replace fully-connected layers with convolutional layers(deconvolutional layers)
  - Deeper, updated models (now ResNet) consistent with ImageNet Challenge object classification tasks.

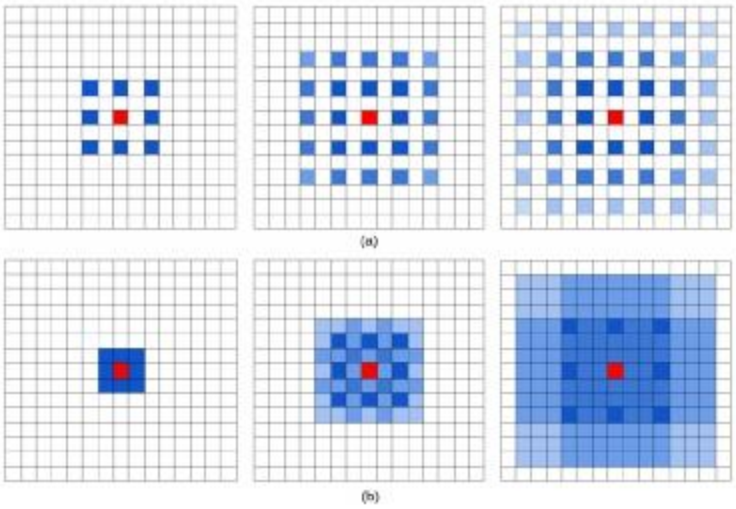
- **Conditional Random Fields (CRFs)** to capture both local and long-range dependencies within an image to refine the prediction map.

- **Dilated convolution** (aka Atrous convolution) – maintain computational cost, increase resolution of intermediate feature maps

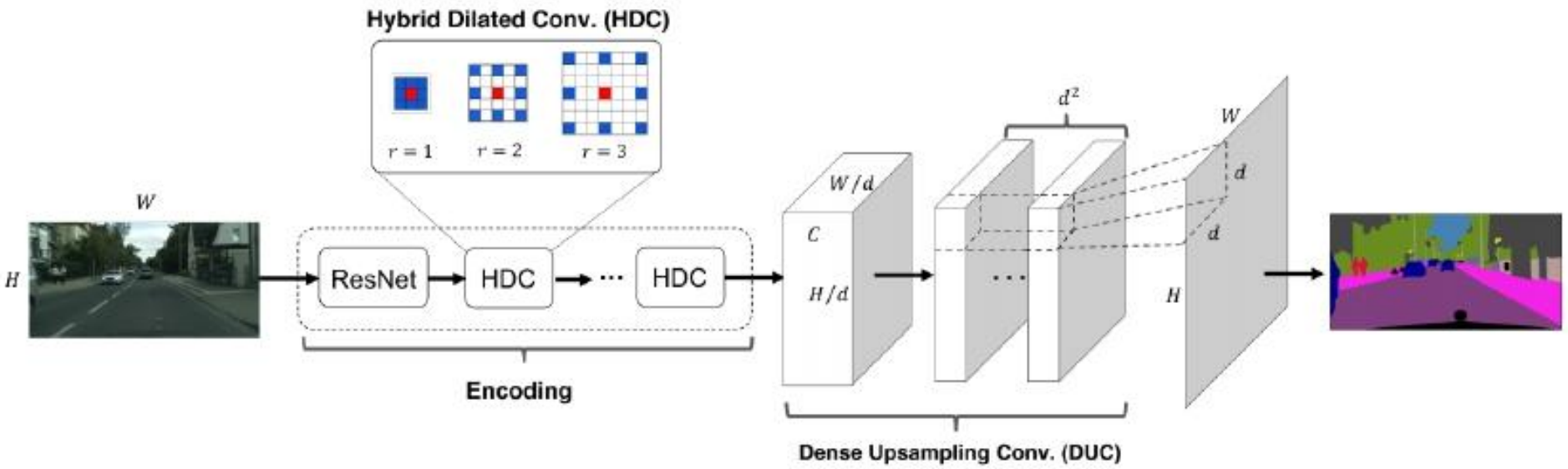
ResNet-DUC (Nov 2017)

Paper: “Understanding Convolution for Semantic Segmentation”

- Dense upsampling convolution (DUC) instead of bilinear upsampling
  - **Learnable:** Learn the upscaling filters



- Hybrid dilated convolution (HDC)
  - Use a different dilation rate



FlowNet 1.0 (May 2015)

Paper: “FlowNet: Learning Optical Flow with Convolutional Networks ”

- Learn flow from image-pair, end to end.
  - FlowNetS – stacks two images as input
  - FlowNetC – convolute separately, combine with correlation layer

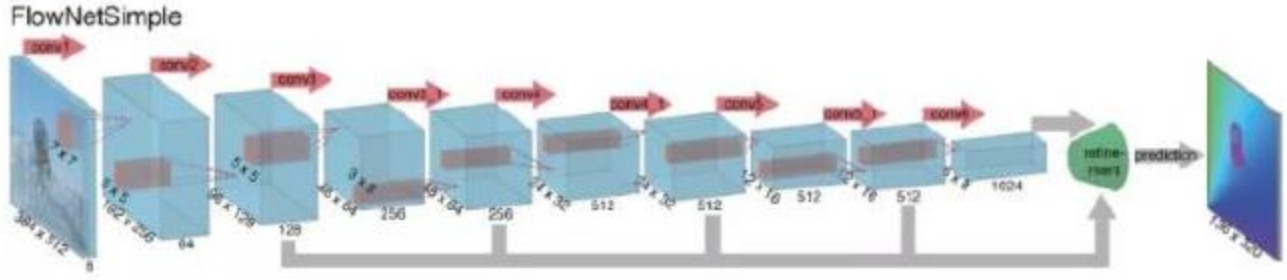
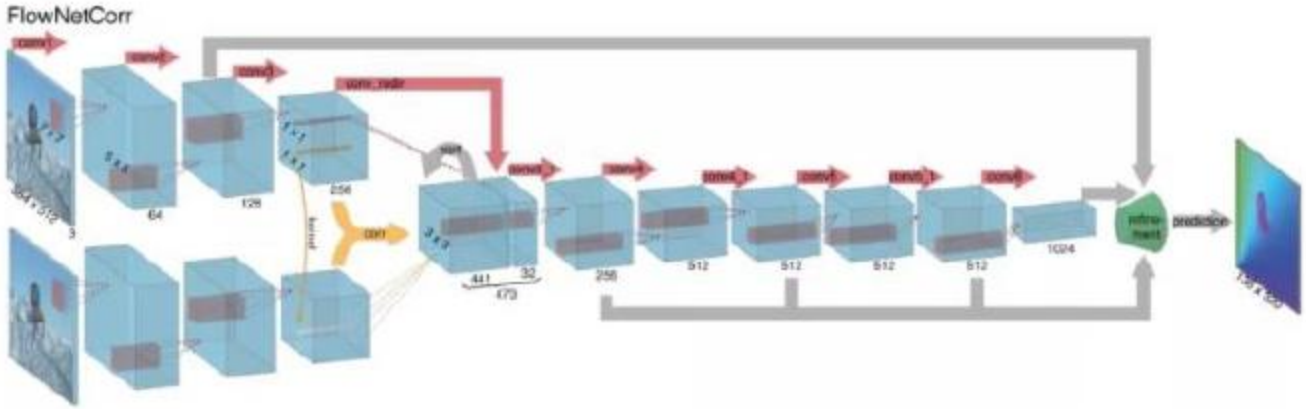


Fig. 1



FlowNet 2.0 (Dec 2016)

Paper: “FlowNet 2.0: Evolution of Optical Flow Estimation with Deep Networks”

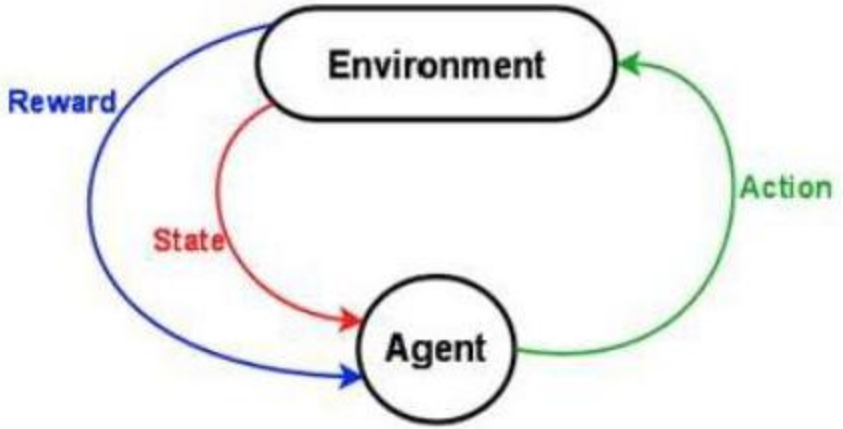
- Stack FlowNetS and FlowNetC
- Improvement over FlowNet
  - Smooth flow fields
  - Preserves fine-motion detail on the edge of object
  - Runs at 8-140fps
- Observations:
  - Stacking networks as an approach
  - Order of training dataset matters

Reinforcement Learning -Deep Q-Learning

Agent and Environment

- At each step the agent:
  - Executes action
  - Receives observation (new state)
  - Receives reward

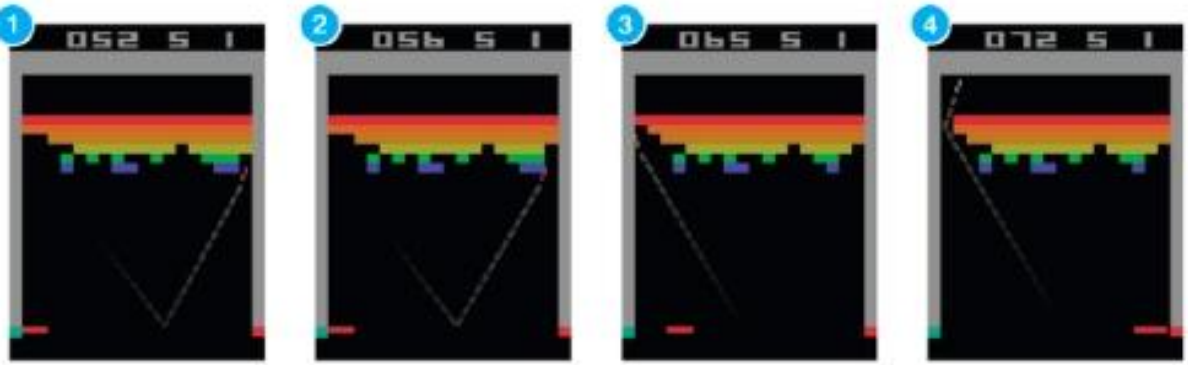
- The environment:
  - Receives action
  - Emits observation (new state)
  - Emits reward

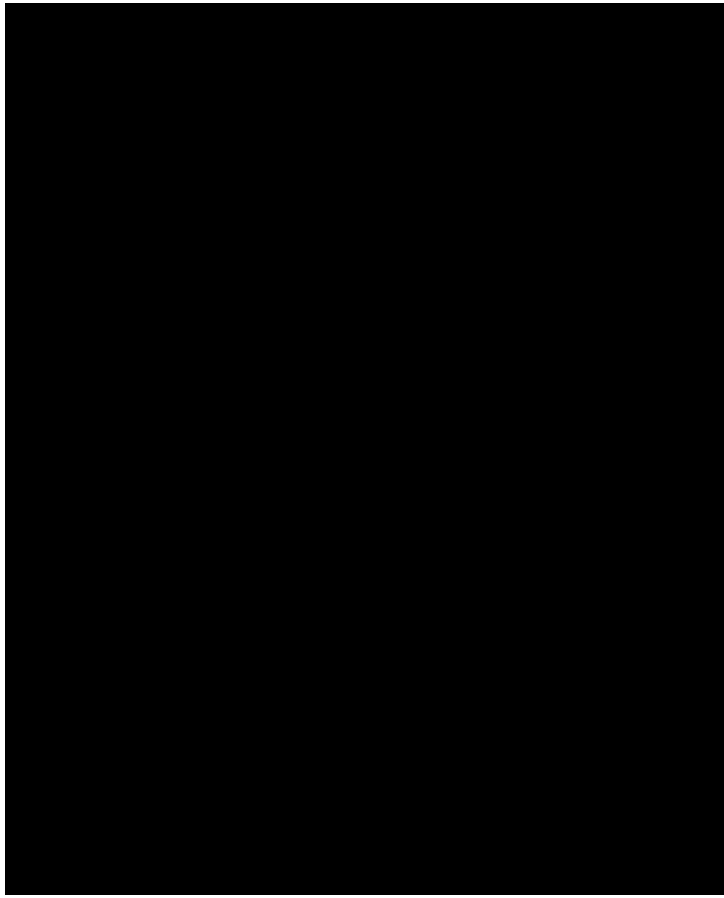


Reinforcement Learning

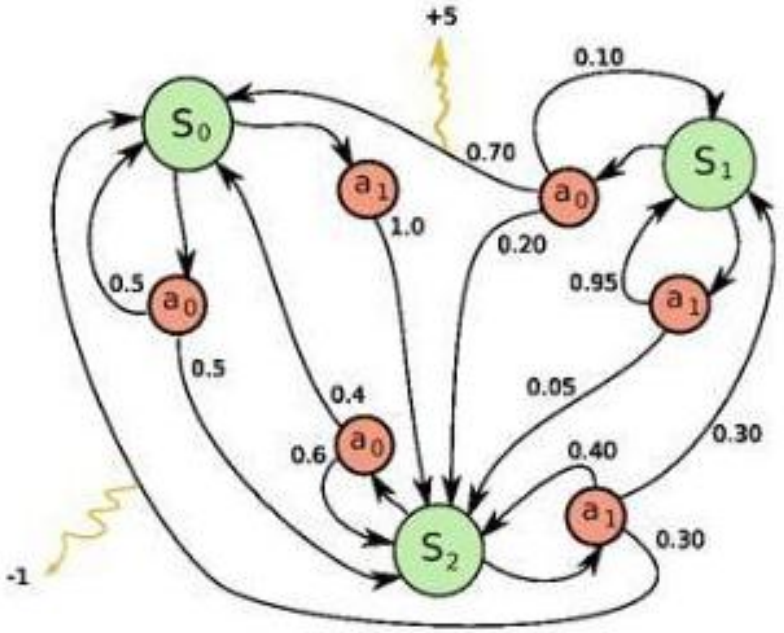
Reinforcement learning is a general-purpose framework for decision-making:

- An agent operates in an environment: **Atari Breakout**
- An agent has the capacity to **act**
- Each action influences the agent's **future state**
- Success is measured by a **reward** signal
- **Goal** is to select actions to maximize future reward





Markov Decision Process



$s0, a0, r1, s1, a1, r2, \ldots, sn-1, an-1, rn, sn$

state

action

reward

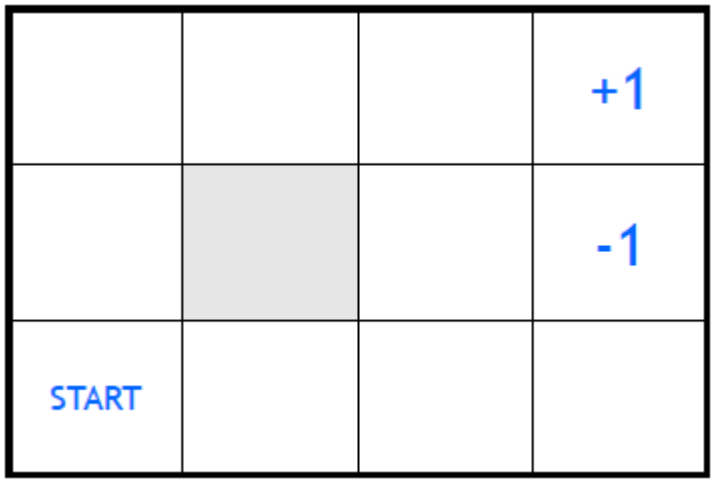
Terminalstate

Major Components of an RL Agent

An RL agent may include one or more of these components:

- **Policy:** agent's behavior function
- **Value function:** how good is each state and/or action
- **Model:** agent's representation of the environment

Robot in a Room



actions: UP, DOWN, LEFT, RIGHT

(Stochastic) model of the world:

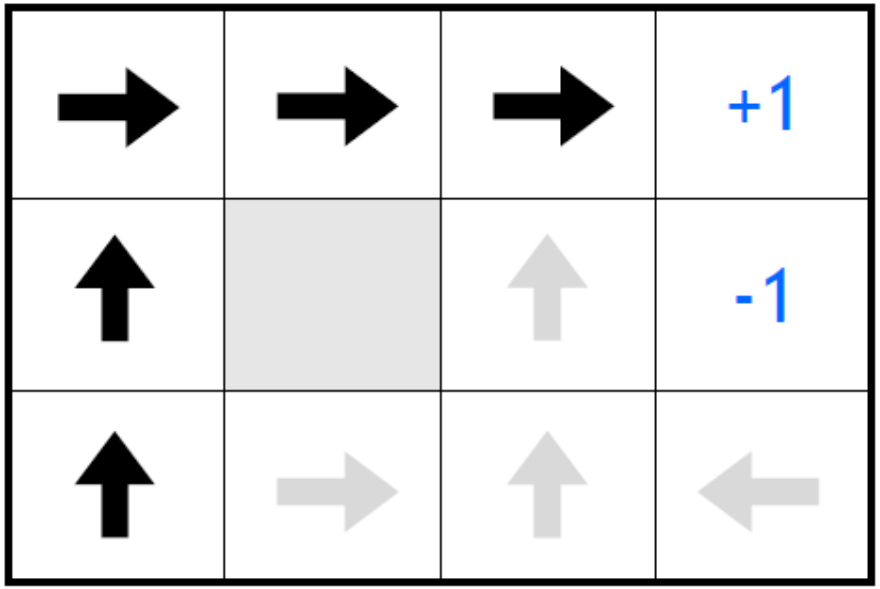
UP

80% move UP
10% move LEFT
10% move RIGHT

- reward +1 at [4,3], -1 at [4,2]
- reward -0.04 for each step
- what's the strategy to achieve max reward?
  - We can learn the model and plan
  - We can learn the value of (action, state) pairs and act greed/non-greedy
  - We can learn the policy directly while sampling from it
- what if the actions were deterministic?

Optimal Policy for a Deterministic World

Reward: **-0.04** for each step



actions: UP, DOWN, LEFT, RIGHT

When actions are deterministic:

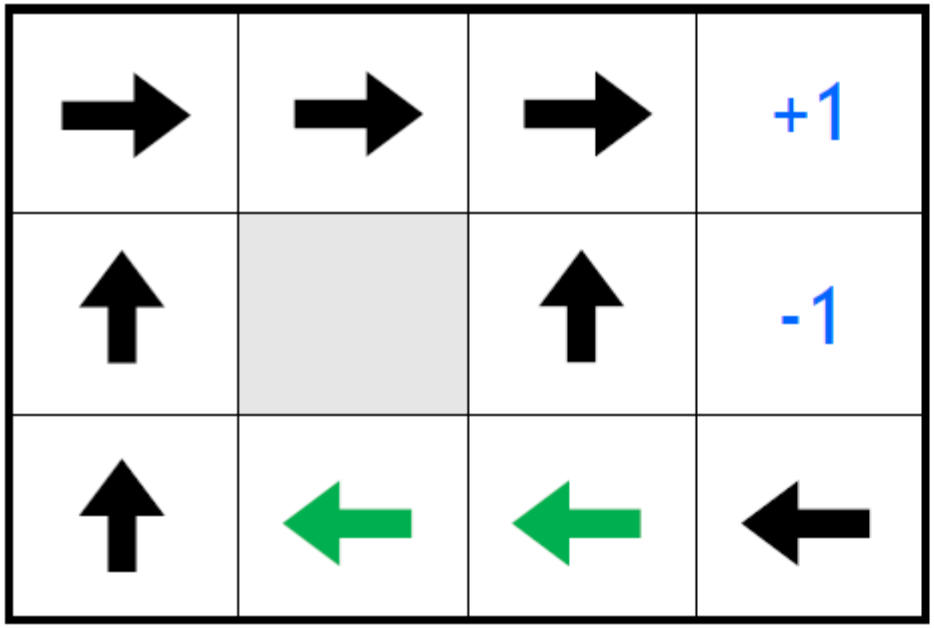
UP

100%	move UP
0%	move LEFT
0%	move RIGHT

Policy: Shortest path.

Optimal Policy for a Stochastic World

Reward: **-0.04** for each step



actions: UP, DOWN, LEFT, RIGHT

When actions are stochastic:

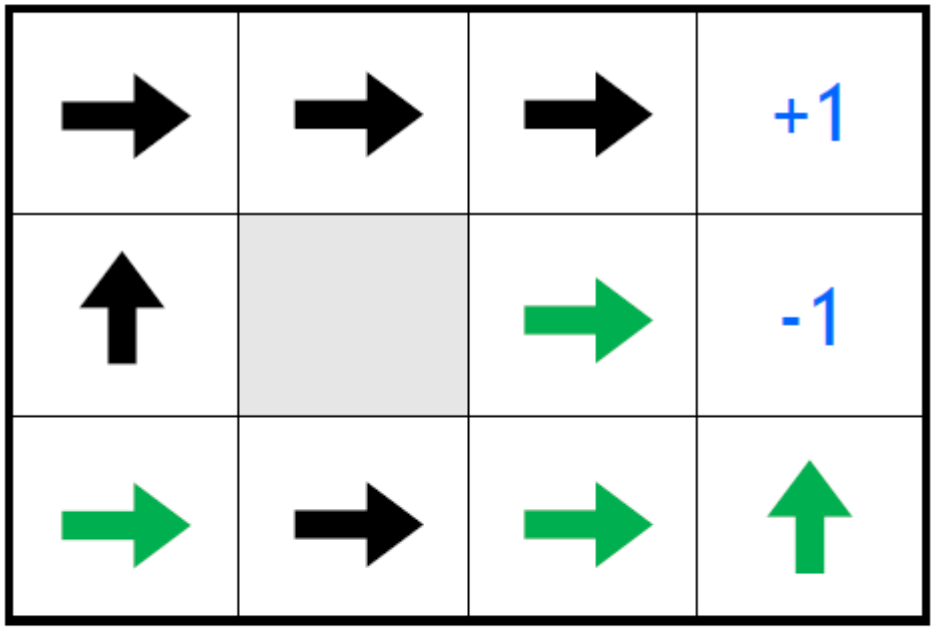
UP

80%	move UP
10%	move LEFT
10%	move RIGHT

Policy: Shortest path. Avoid -UP around -1 square.

Optimal Policy for a Stochastic World

Reward: -2 for each step



actions: UP, DOWN, LEFT, RIGHT

When actions are stochastic:

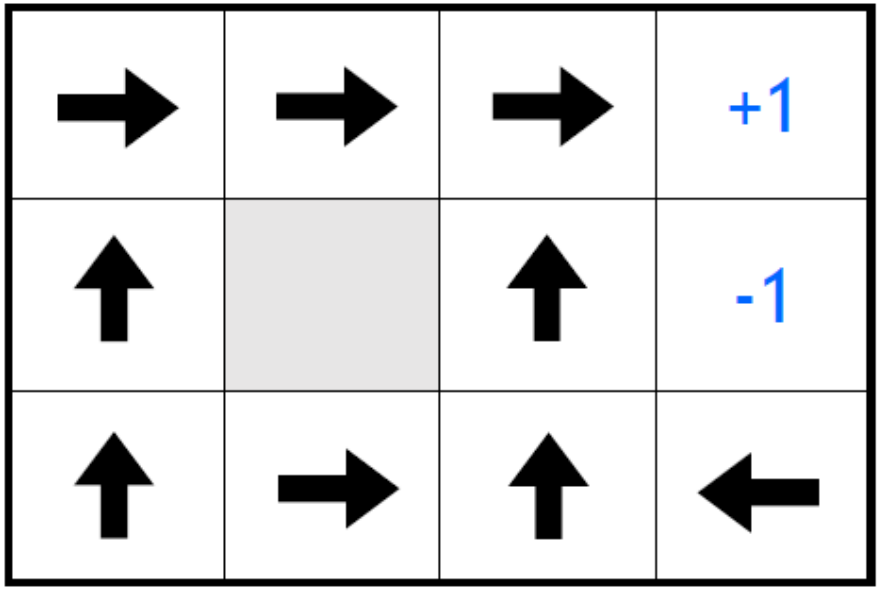
UP

80%	move UP
10%	move LEFT
10%	move RIGHT

Policy: Shortest path.

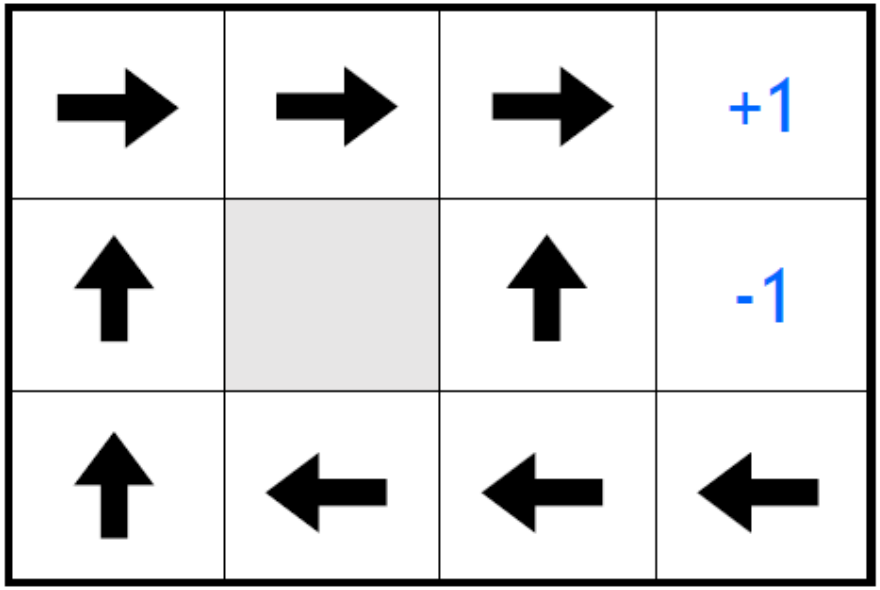
Optimal Policy for a Stochastic World

Reward: **-0.1** for each step



More urgent

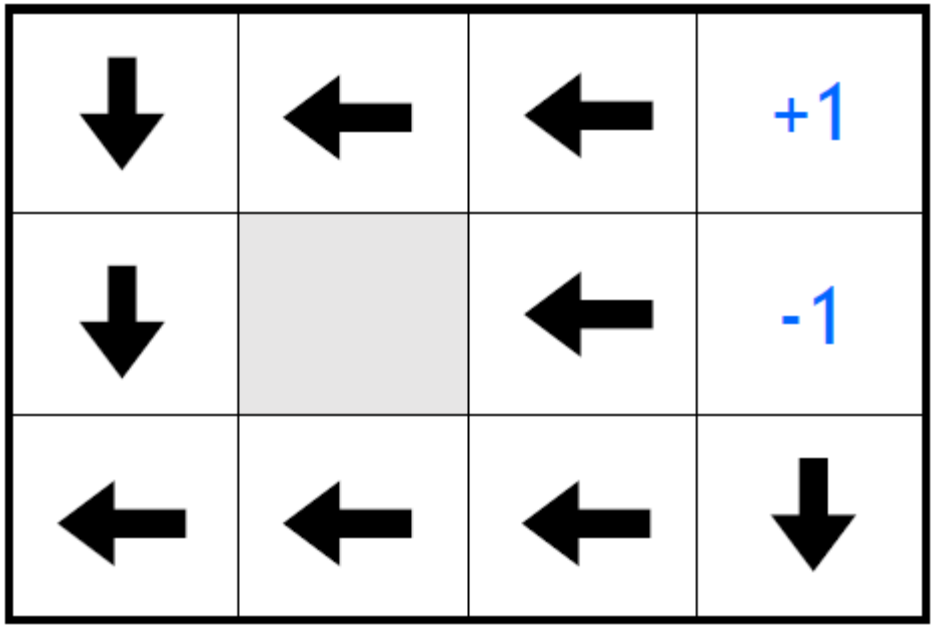
Reward: **-0.04** for each step



Less urgent

Optimal Policy for a Stochastic World

Reward: **+0.01** for each step



actions: UP, DOWN, LEFT, RIGHT

When actions are stochastic:

UP

80%	move UP
10%	move LEFT
10%	move RIGHT

Policy: Longest path.

Value Function

- Future reward $R = r_1 + r_2 + r_3 + \cdots + r_n$

$$R_t = r_t + r_{t+1} + r_{t+2} + \cdots + r_n$$

- Discounted future reward (environment is stochastic)

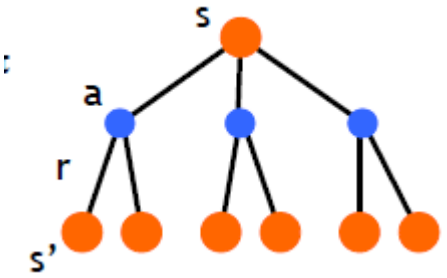
$$\begin{aligned} R_t &= r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \cdots + \gamma^{n-t} r_n \\ &= r_t + \gamma(r_{t+1} + \gamma(r_{t+2} + \cdots)) \\ &= r_t + \gamma R_{t+1} \end{aligned}$$

- A good strategy for an agent would be to always choose an action that maximizes the (discounted) future reward

- Why "discounted"?
  - Math trick to help analyze convergence
  - Uncertainty due to environment stochasticity, partial observability, or that life can end at any moment:

Q-Learning

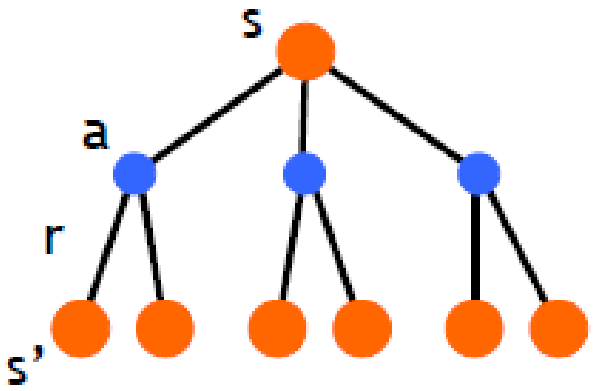
- State value function: $V_\pi(s)$
  - Expected return when starting in *s*, performing *a*, and following π
- State-action value function: $Q_\pi(s,a)$
  - Expected return when starting in *s*, performing *a,* and following π
- Useful for finding the optimal policy
  - Can estimate from experience (Monte Carlo)
  - Pick the best action using $Q_\pi(s,a)$



- Q-learning: off-policy
  - Use any policy to estimate Q that maximizes future reward: $Q(s_t, a_t) = max\ R_{t+1}$
  - Q directly approximates Q* (Bellman optimality equation)
  - Independent of the policy being followed
  - Only requirement: keep updating each (s,a) pair

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha \left(R_{t+1} + \gamma \max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t) \right)$$

Q-Learning



Learning Rate

Discount Factor

$$Q_{t+1}(s_t, a_t) = Q_t(s_t, a_t) + \alpha \left(R_{t+1} + \gamma \max_a Q_t(s_{t+1}, a) - Q_t(s_t, a_t) \right)$$

New State

Old State

Reward

Exploration vs Exploitation

- Key ingredient of Reinforcement Learning

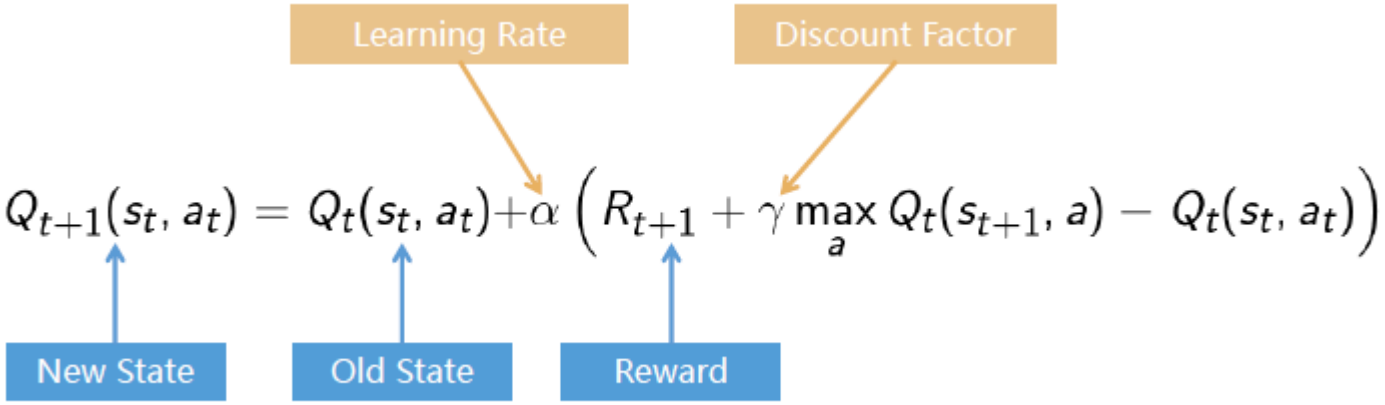
- Deterministic/greedy policy won't explore all actions
  - Don't know anything about the environment at the beginning
  - Need to try all actions to find the optimal one

- Maintain exploration
  - Use *soft* policies instead: $\pi(s,a)>0$ (for all s,a)

- ε-greedy policy
  - With probability 1-ε perform the optimal/greedy action
  - With probability ε perform a random action

  - Will keep exploring the environment
  - Slowly move it towards greedy policy: ε -> 0

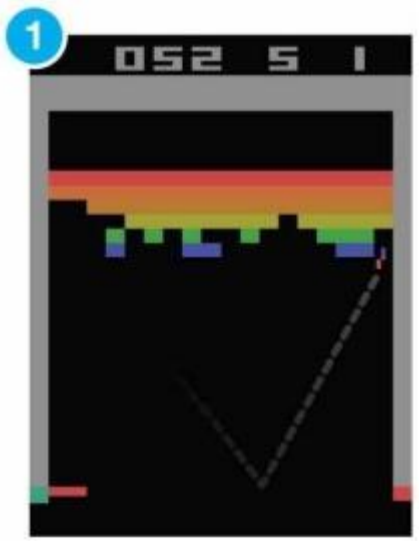
Q-Learning: Value Iteration




```
initialize Q[num_states,num_actions] arbitrarily
observe initial state s
repeat
      select and carry out an action a
      observe reward r and new state s'
      Q[s,a] = Q[s,a] + α(r + γ max_a' Q[s',a'] - Q[s,a])
      s = s'
until terminated
```

Q-Learning: Representation Matters

- In practice, Value Iteration is impractical
  - Very limited states/actions
  - Cannot generalize to unobserved states

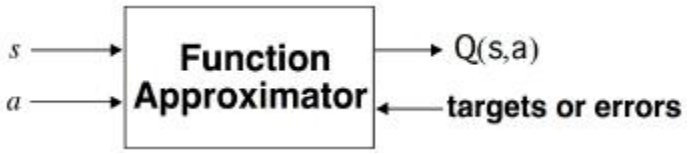


- Think about the **Breakout** game
  - State: screen pixels
    - Image size: 84×84 (resized)
    - Consecutive **4** images
    - Grayscale with **256** graylevels

$256^{84 \times 84 \times 4}$ rows in the Q-table!

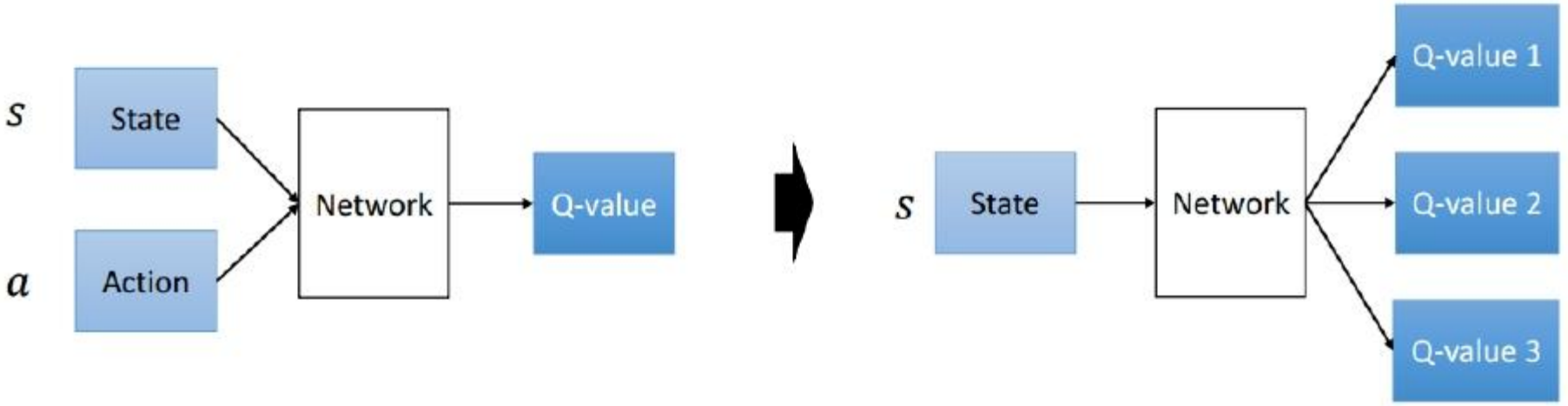
Deep Q-Learning

Use a function (with parameters) to approximate the Q-function

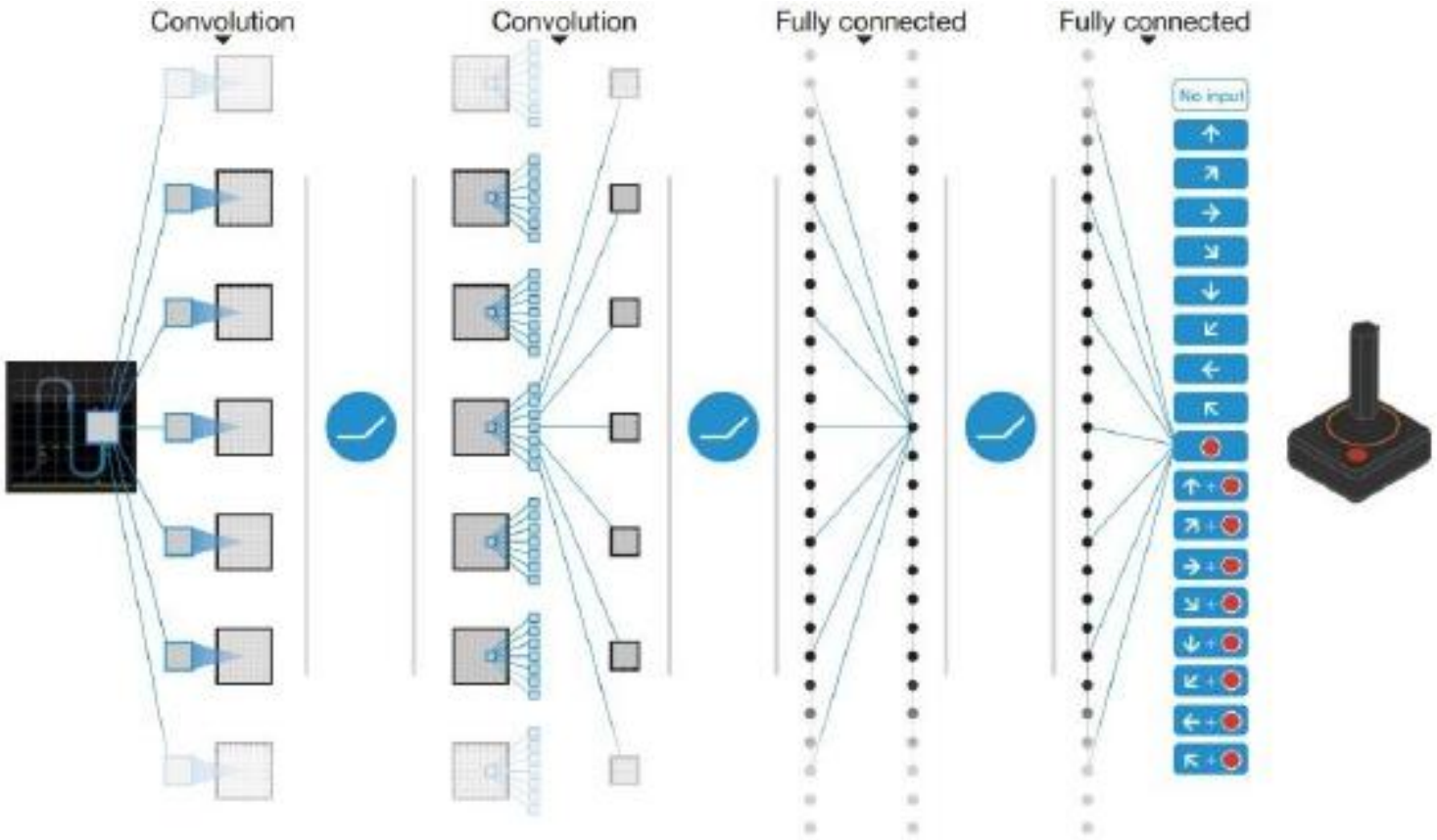


- Linear
- Non-linear:**Q-Network**

$$Q(s, a; \boldsymbol{\theta}) \approx Q^*(s, a)$$



Deep Q-Network: Atari



Layer	Input	Filter size	Stride	Num filters	Activation	Output
conv1	84x84x4	8x8	4	32	ReLU	20x20x32
conv2	20x20x32	4x4	2	64	ReLU	9x9x64
conv3	9x9x64	3x3	1	64	ReLU	7x7x64
fc4	7x7x64			512	ReLU	512
fc5	512			18	Linear	18

Deep Q-Network Training

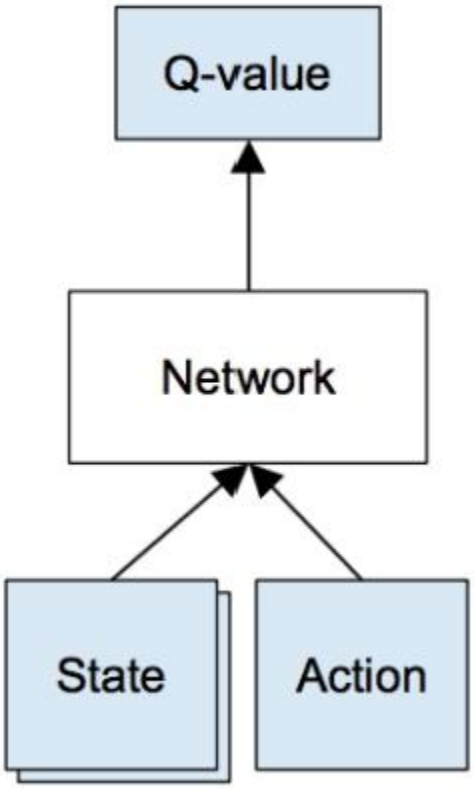
- Bellman Equation:

$$Q(s,a) = r + \gamma max_{a'} Q(s',a')$$

- Loss function (squared error):

$$L = \mathbb{E}[(\underbrace{\boldsymbol{r + \gamma max_{a'} Q(s',a')}}_{\textbf{target}} - Q(s,a))^2]$$

Deep Q-Network Training



Given a transition < *s, a, r, s'* >, the Q-table update rule in the previous algorithm must be replaced with the following:

- Do a feedforward pass for the current state *s* to get **predicted Q-values for all actions**
- Do a feedforward pass for the next state *s'* and calculate maximum overall network outputs $\max_{a'} Q(s', a')$
- Set Q-value target for action to $r + \gamma \max_{a'} Q(s', a')$ (use the max calculated in step 2).
  - For all other actions, set the Q-value target to the same as originally returned from step 1, making the error 0 for those outputs.
- Update the weights using backpropagation.

DQN Tricks

- Experience Replay
  - Stores experiences (actions, state transitions, and rewards) and creates mini-batches from them for the training process
- Fixed Target Network
  - Error calculation includes the target function depends on network parameters and thus changes quickly. Updating it only every 1,000 steps increases stability of training process.

$$Q(s_t, a) \leftarrow Q(s_t, a) + \alpha \left[r_{t+1} + \gamma \max_p Q(s_{t+1}, p) - Q(s_t, a) \right]$$

target Q function in the red rectangular is fixed

- Reward Clipping
  - To standardize rewards across games by setting all positive rewards to +1 and all negative to -1.
- Skipping Frames
  - Skip every 4 frames to take action

Deep Q-Learning Algorithm

initialize replay memory D
initialize action-value function Q with random weights
observe initial state s
repeat
    select an action a
        with probability ε select a random action
        otherwise select $a = \text{argmax}_{a'} Q(s, a')$
    carry out action a
    observe reward r and new state s'
    store experience $<s, a, r, s'>$ in replay memory D

    sample random transitions $<ss, aa, rr, ss'>$ from replay memory D
    calculate target for each minibatch transition
        if ss' is terminal state then $tt = rr$
        otherwise $tt = rr + \gamma \text{max}_{a'} Q(ss', aa')$
    train the Q network using $(tt - Q(ss, aa))^2$ as loss

    $s = s'$
until terminated