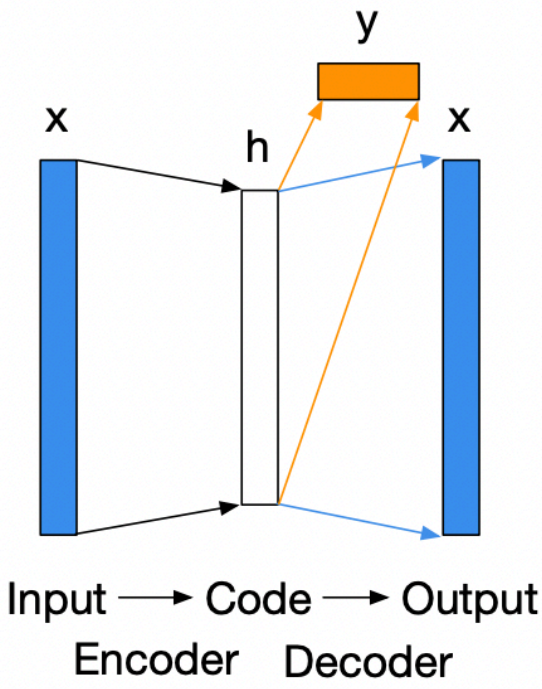
Deep Learning

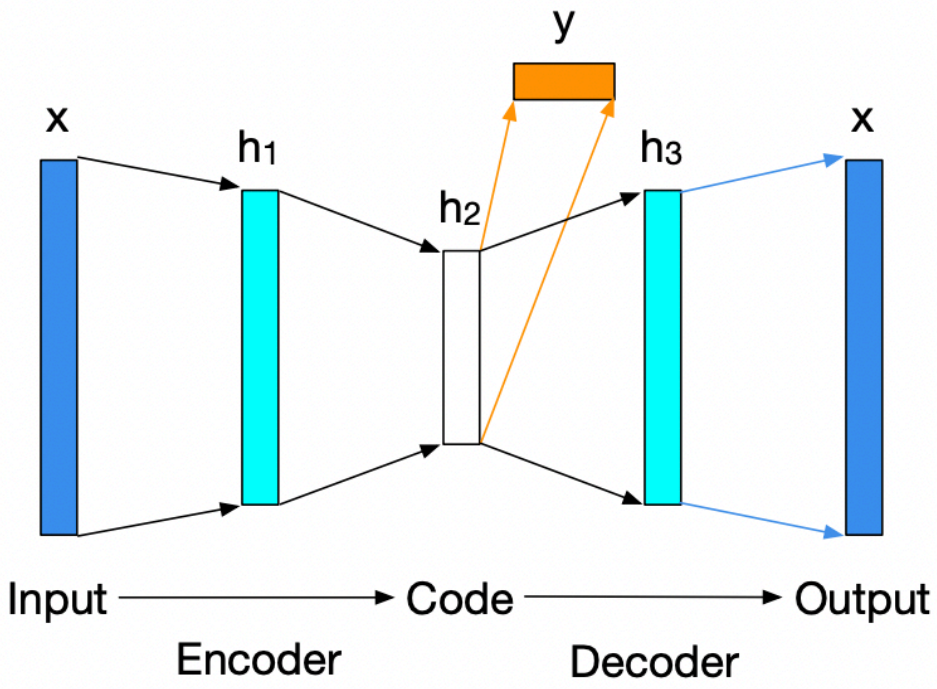
Mayank Vatsa

Supervised AE

- How to make AE Supervised?

Supervised AE





What would be the generic optimization function?

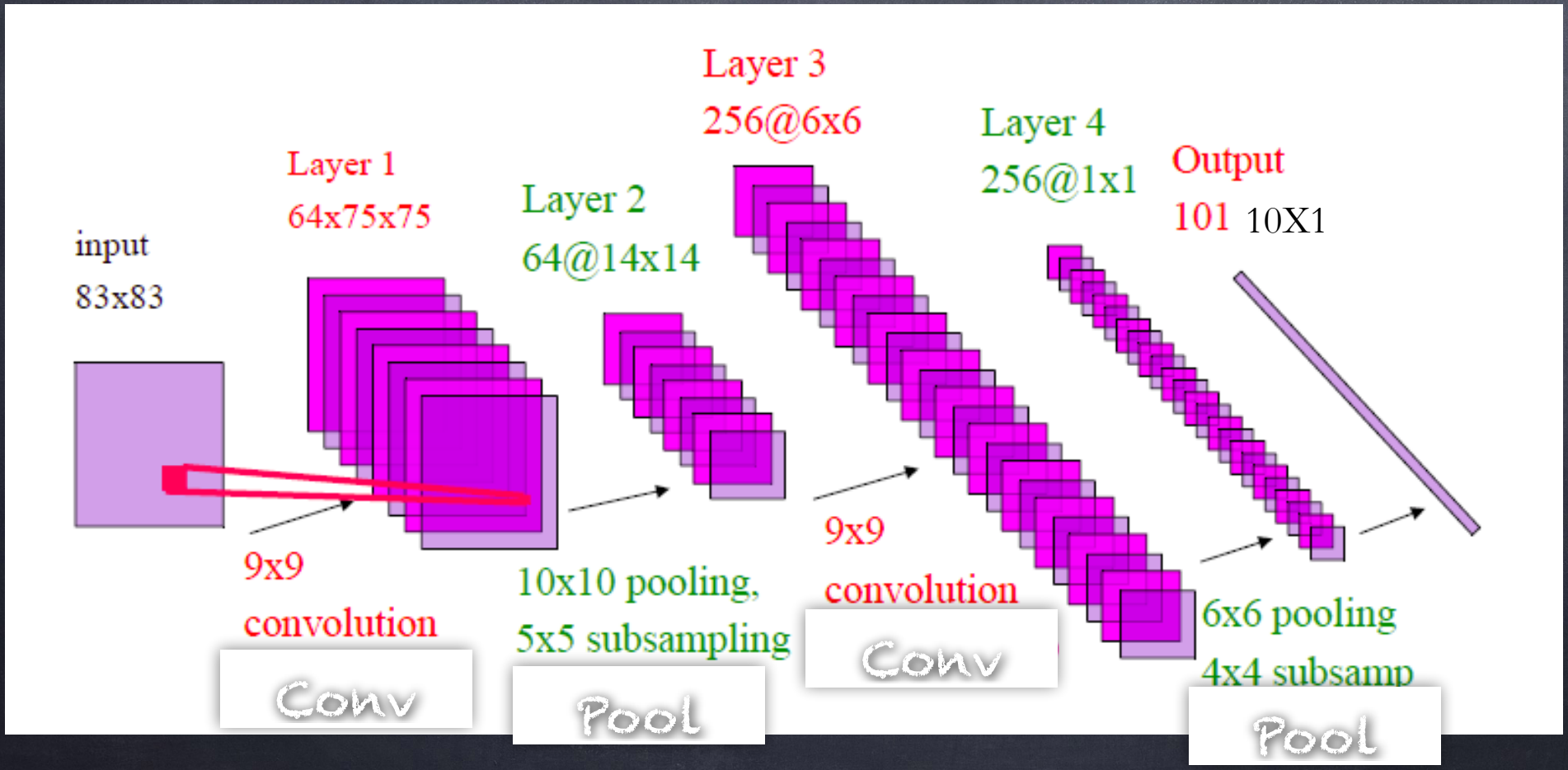
- Recall AE function

Moving to CNN

- Convolutional neural network

CNN

Convolutional Neural Network

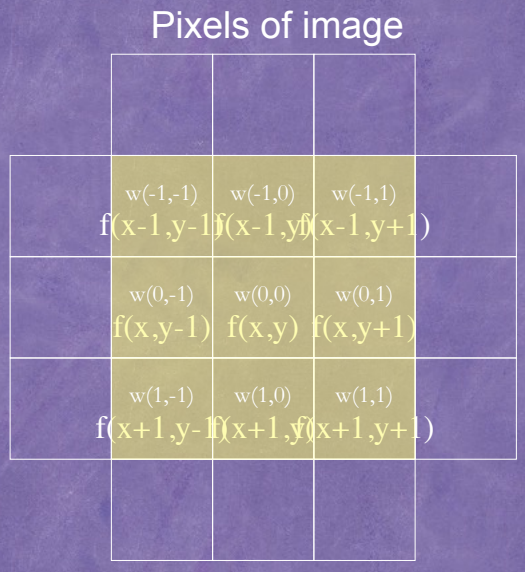


What do you mean by convolution?

What do you mean by convolution?

- In maths: a = conv(b,c)
- process of transforming an image/signal by applying a kernel over each pixel and its local neighbors across the entire image. The kernel is a matrix of values whose size and values determine the transformation effect of the convolution process.

What do you mean by convolution?

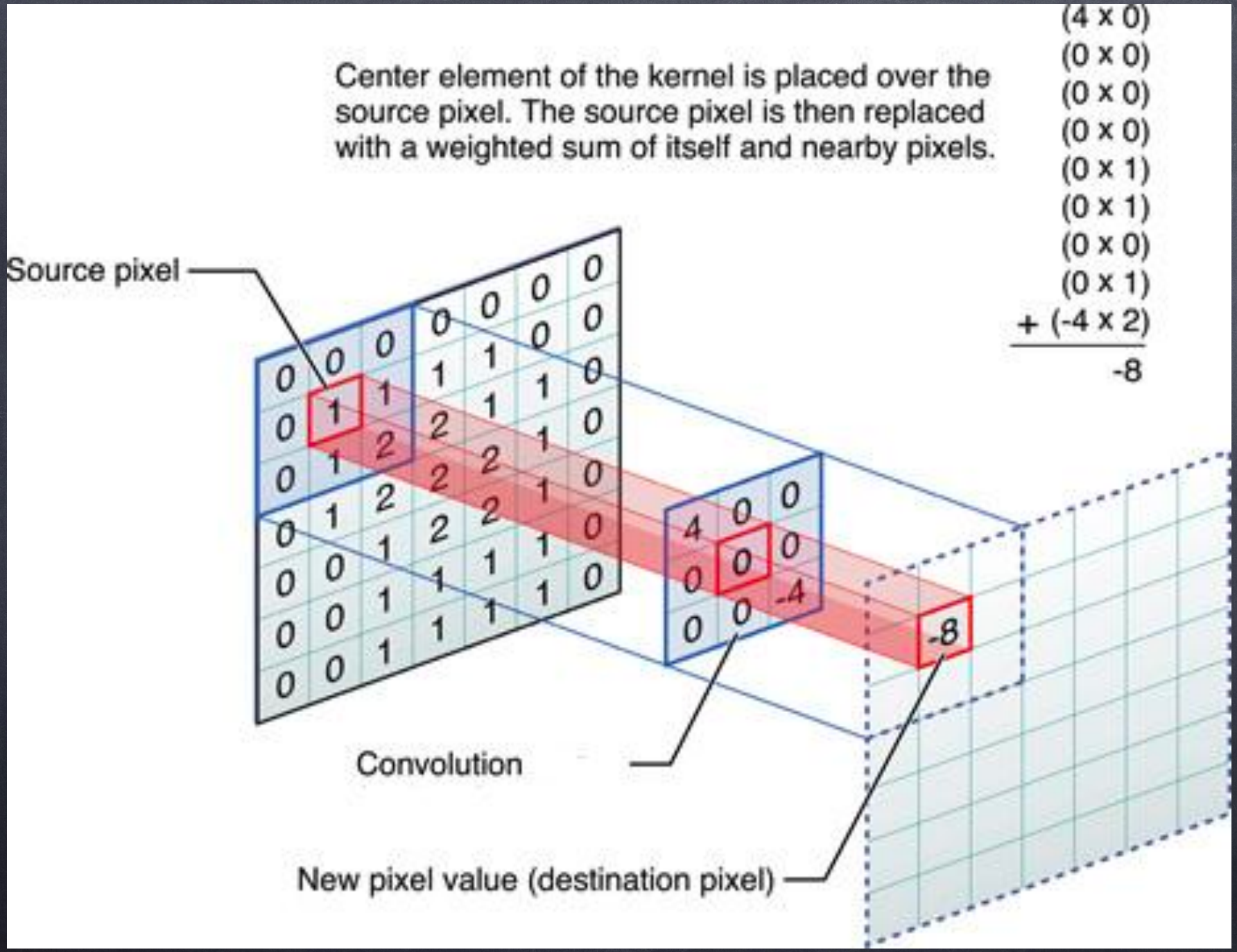


The result is the sum of products of the mask coefficients with the corresponding pixels directly under the mask

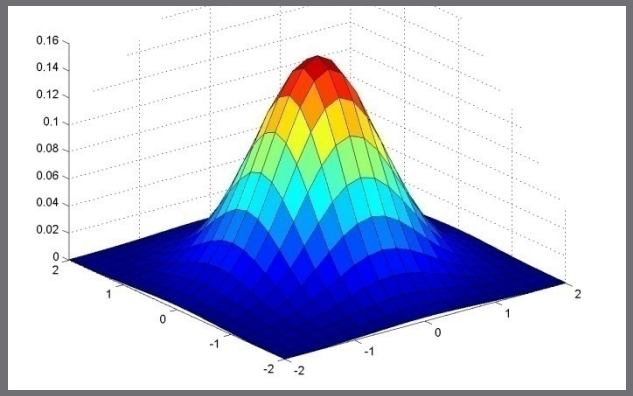
Mask coefficients

w(-1,-1)	w(-1,0)	w(-1,1)
w(0,-1)	w(0,0)	w(0,1)

$$f(x,y) = w(-1,-1)f(x-1,y-1) + w(-1,0)f(x-1,y) + w(-1,1)f(x-1,y+1) + w(0,-1)f(x,y-1) + w(0,0)f(x,y) + w(0,1)f(x,y+1) + w(1,-1)f(x+1,y-1) + w(1,0)f(x+1,y) + w(1,1)f(x+1,y+1)$$


Center element of the kernel is placed over the source pixel. The source pixel is then replaced with a weighted sum of itself and nearby pixels.
(4 x 0)
(0 x 0)
(0 x 0)
(0 x 0)
(0 x 1)
(0 x 1)
(0 x 0)
(0 x 1)
+ (-4 x 2)
-8
Source pixel
Convolution
New pixel value (destination pixel)

Gaussian Filter



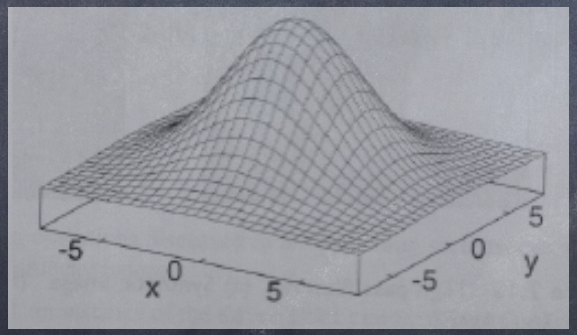
$$G_\sigma = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

Source: C. Rasmussen

Gaussian Filter

- The weights are samples of the Gaussian function

$$G_\sigma(x, y) = \frac{1}{2\pi\sigma^2} \exp^{-\frac{x^2 + y^2}{2\sigma^2}}$$



7×7 Gaussian mask

1	1	2	2	2	1	1
1	2	2	4	2	2	1
2	2	4	8	4	2	2
2	4	8	16	8	4	2
2	2	4	8	4	2	2
1	2	2	4	2	2	1
1	1	2	2	2	1	1

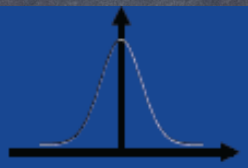
Gaussian Filter

- As σ increases, more samples must be obtained to represent the Gaussian function accurately.
- Therefore, σ controls the amount of smoothing

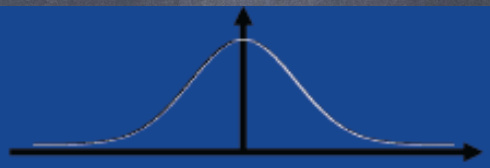
15 × 15 Gaussian mask

2	2	3	4	5	5	6	6	6	5	5	4	3	2	2
2	3	4	5	7	7	8	8	8	7	7	5	4	3	2
3	4	6	7	9	10	10	11	10	10	9	7	6	4	3
4	5	7	9	10	12	13	13	13	12	10	9	7	5	4
5	7	9	11	13	14	15	16	15	14	13	11	9	7	5
5	7	10	12	14	16	17	18	17	16	14	12	10	7	5
6	8	10	13	15	17	19	19	19	17	15	13	10	8	6
6	8	11	13	16	18	19	20	19	18	16	13	11	8	6
6	8	10	13	15	17	19	19	19	17	15	13	10	8	6
5	7	10	12	14	16	17	18	17	16	14	12	10	7	5
5	7	9	11	13	14	15	16	15	14	13	11	9	7	5
4	5	7	9	10	12	13	13	13	12	10	9	7	5	4
3	4	6	7	9	10	10	11	10	10	9	7	6	4	3
2	3	4	5	7	7	8	8	8	7	7	5	4	3	2
2	2	3	4	5	5	6	6	6	5	5	4	3	2	2

Gaussian Filter



small σ

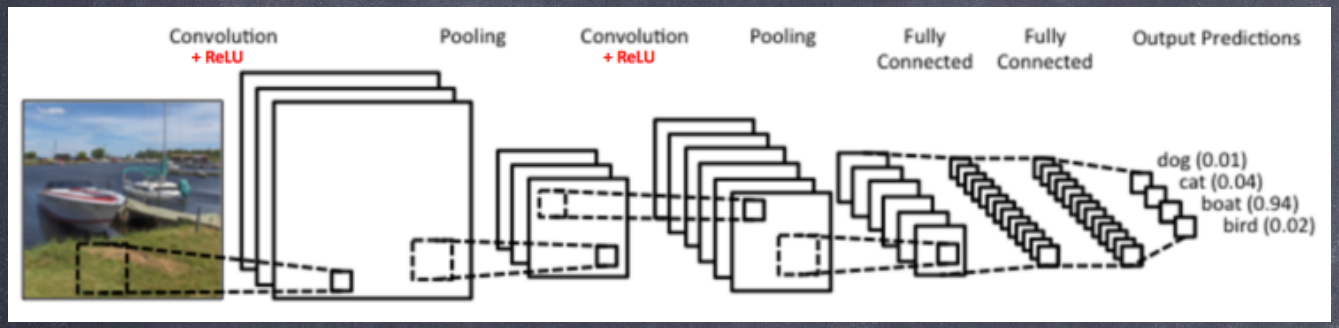


large σ

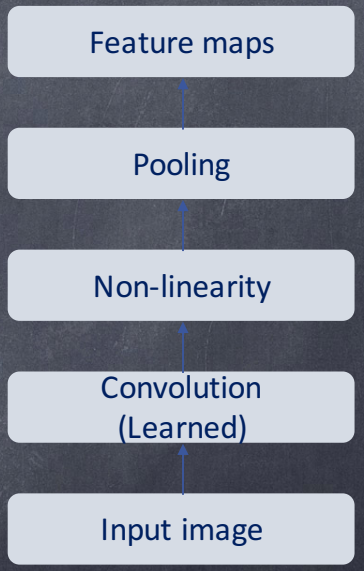
limited smoothing

strong smoothing

CNN

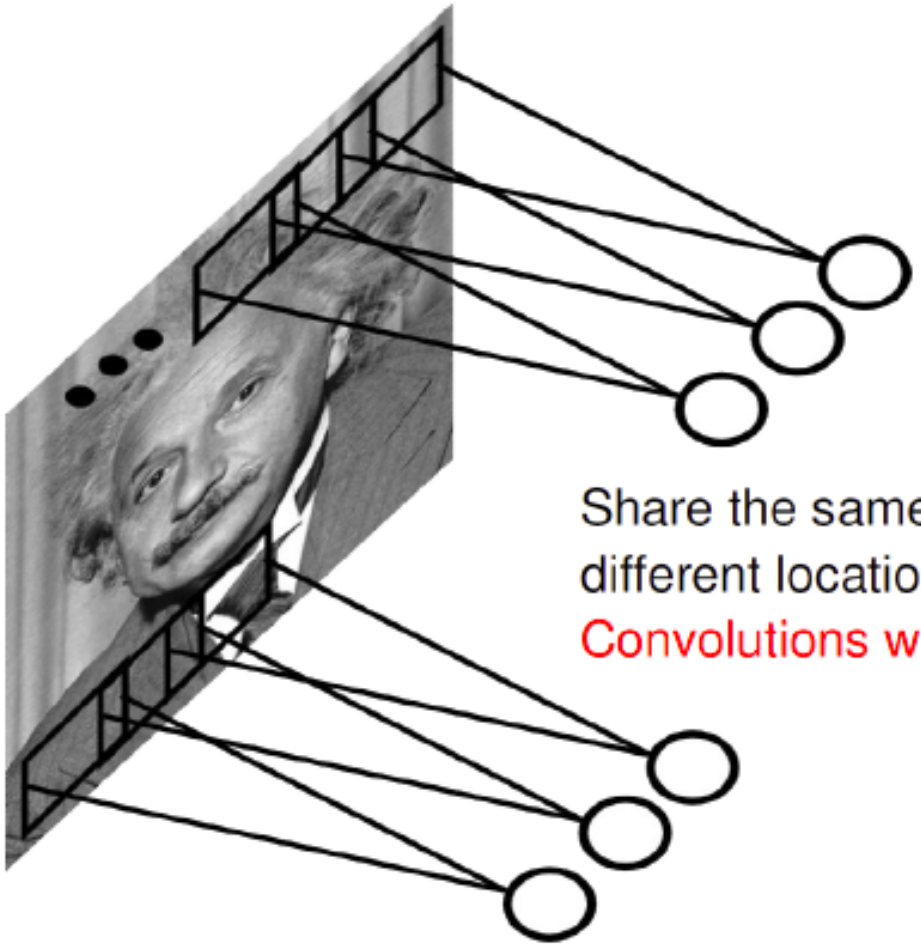


- Sample ConvNet Architecture

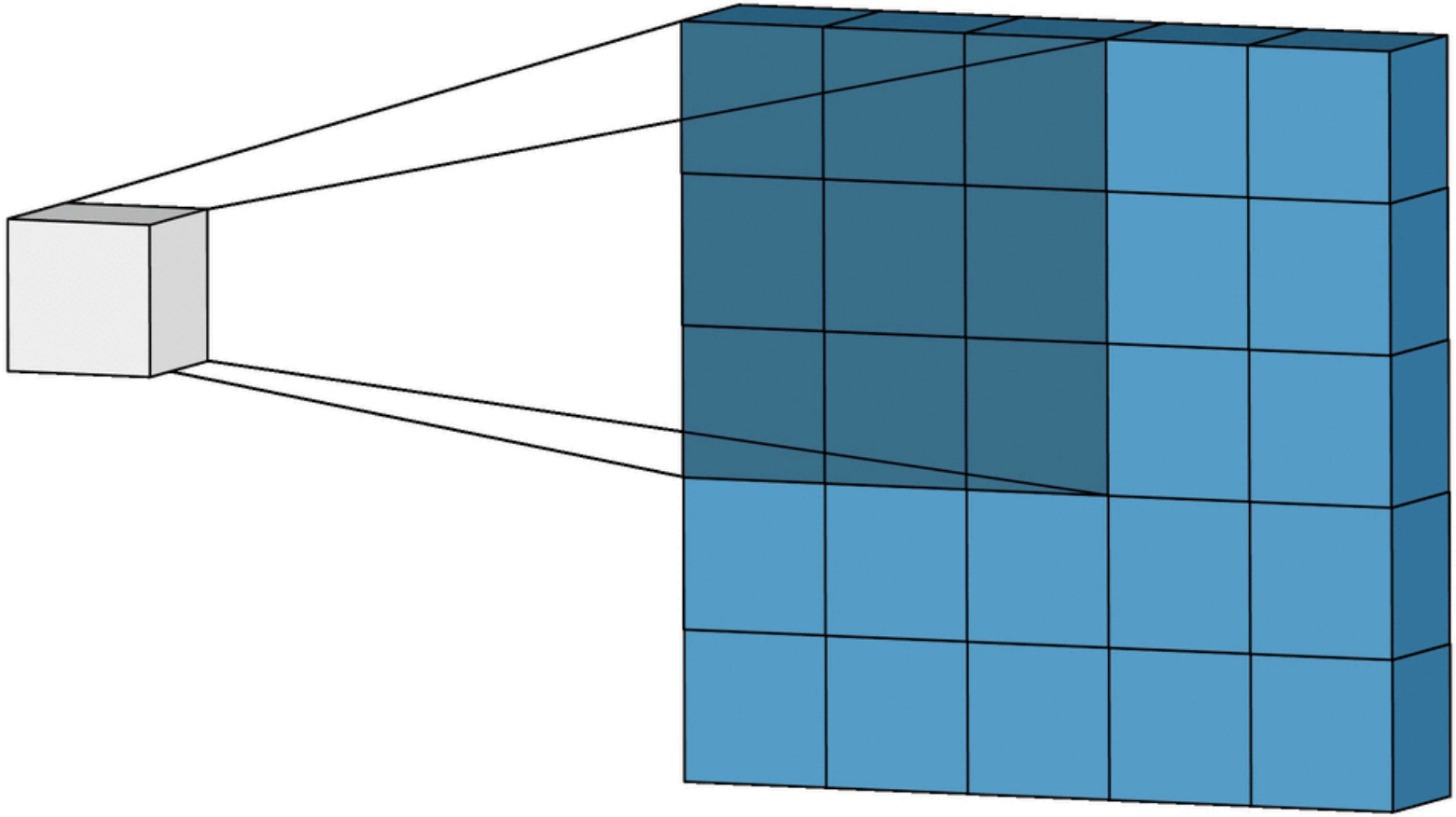


CNN

CONVOLUTIONAL NET

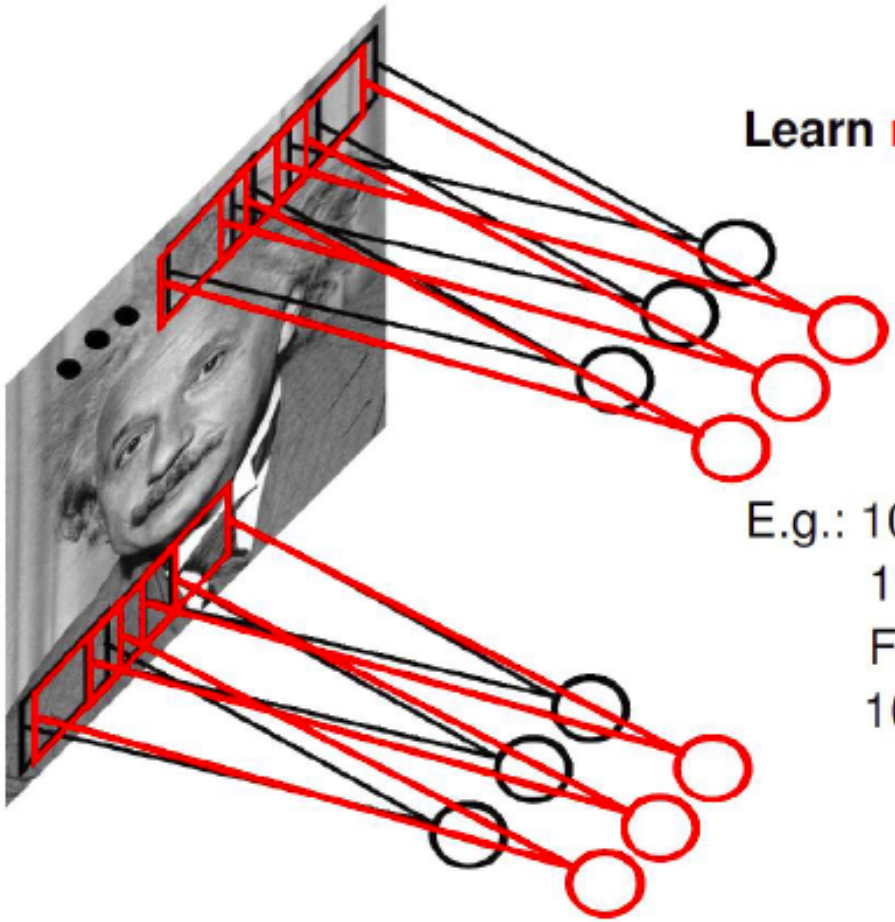


Share the same parameters across different locations:
Convolutions with learned kernels



CNN

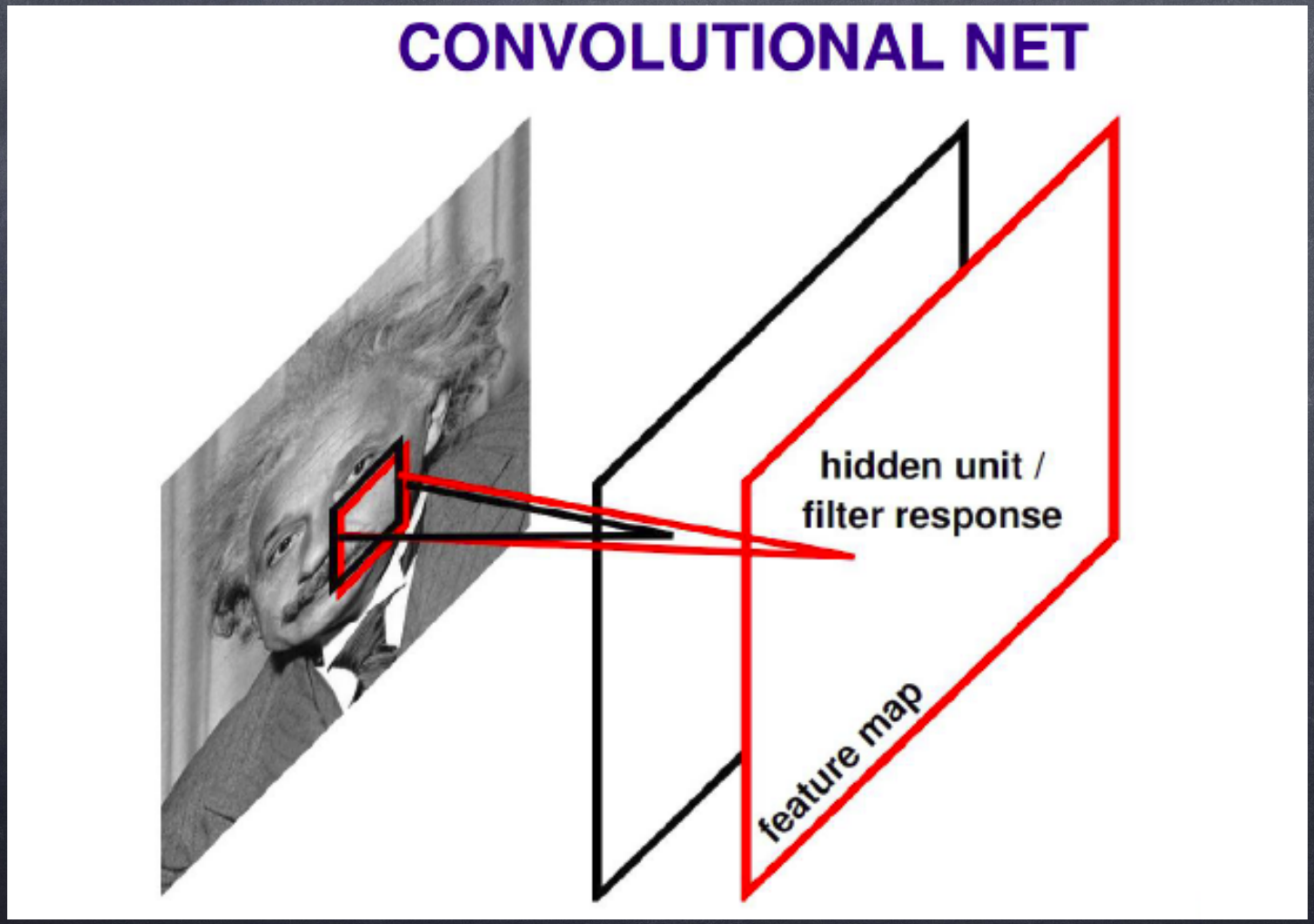
CONVOLUTIONAL NET

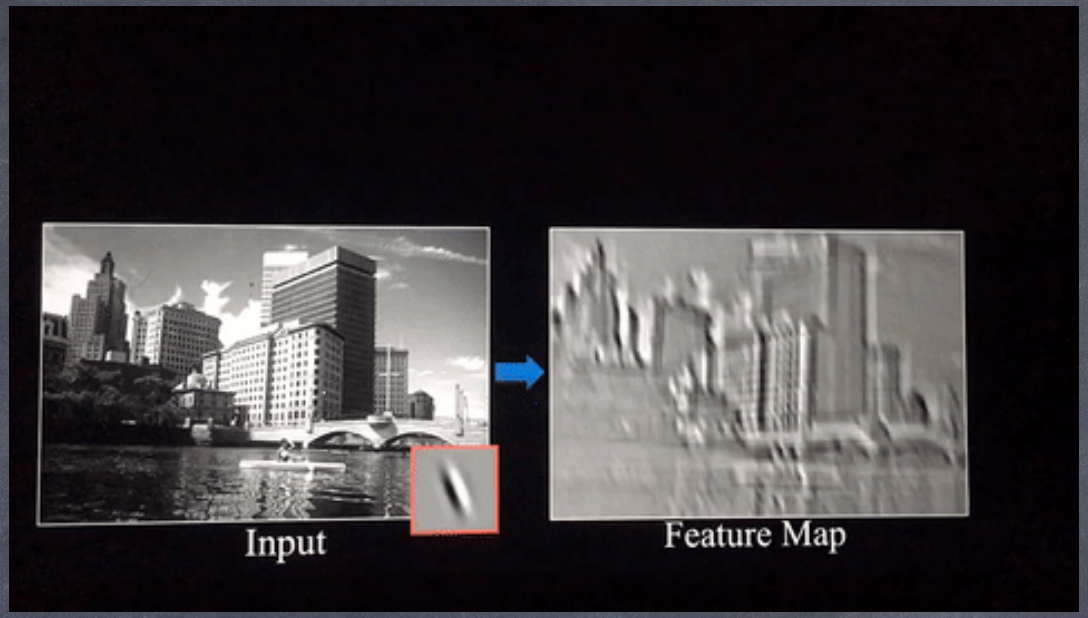


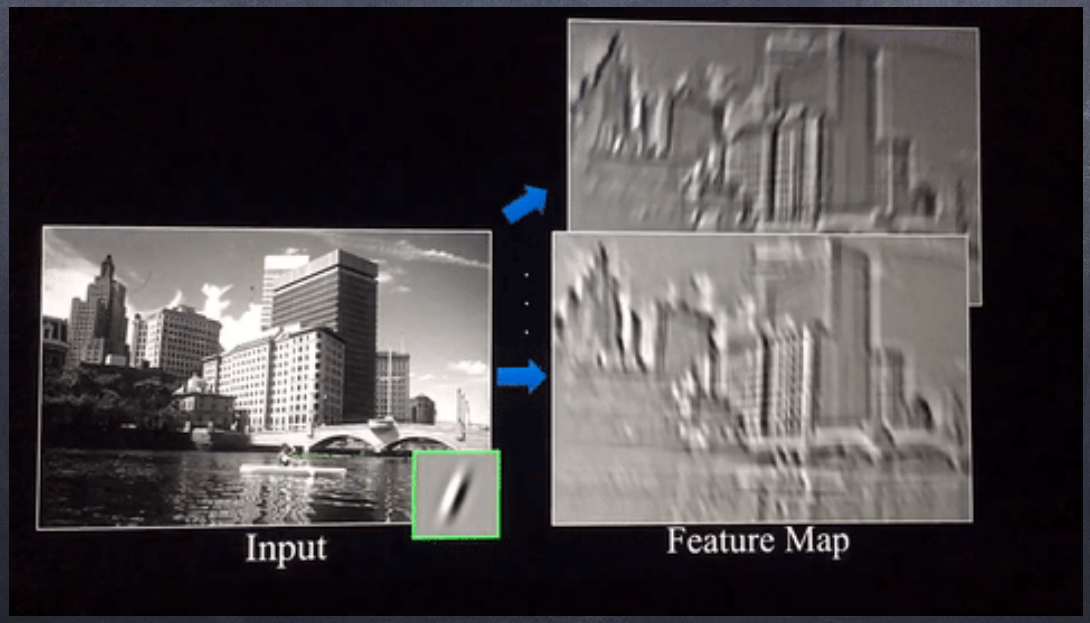
Learn multiple filters.

E.g.: 1000x1000 image
100 Filters
Filter size: 10x10
10K parameters

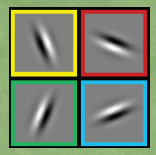
CNN


CONVOLUTIONAL NET
hidden unit /
filter response
feature map


Input
Feature Map

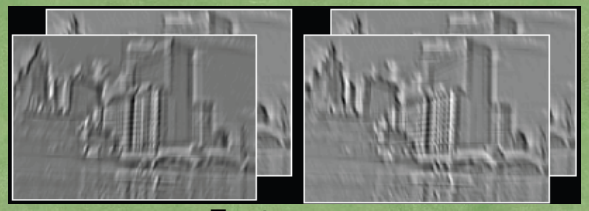

Input
Feature Map

An Example of Filtering

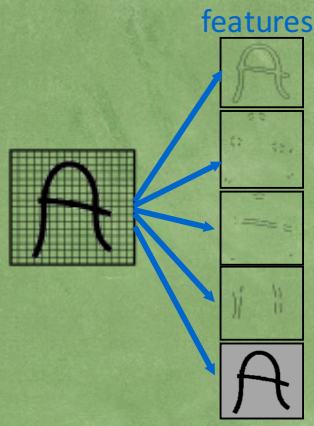


Filter
(kernel)

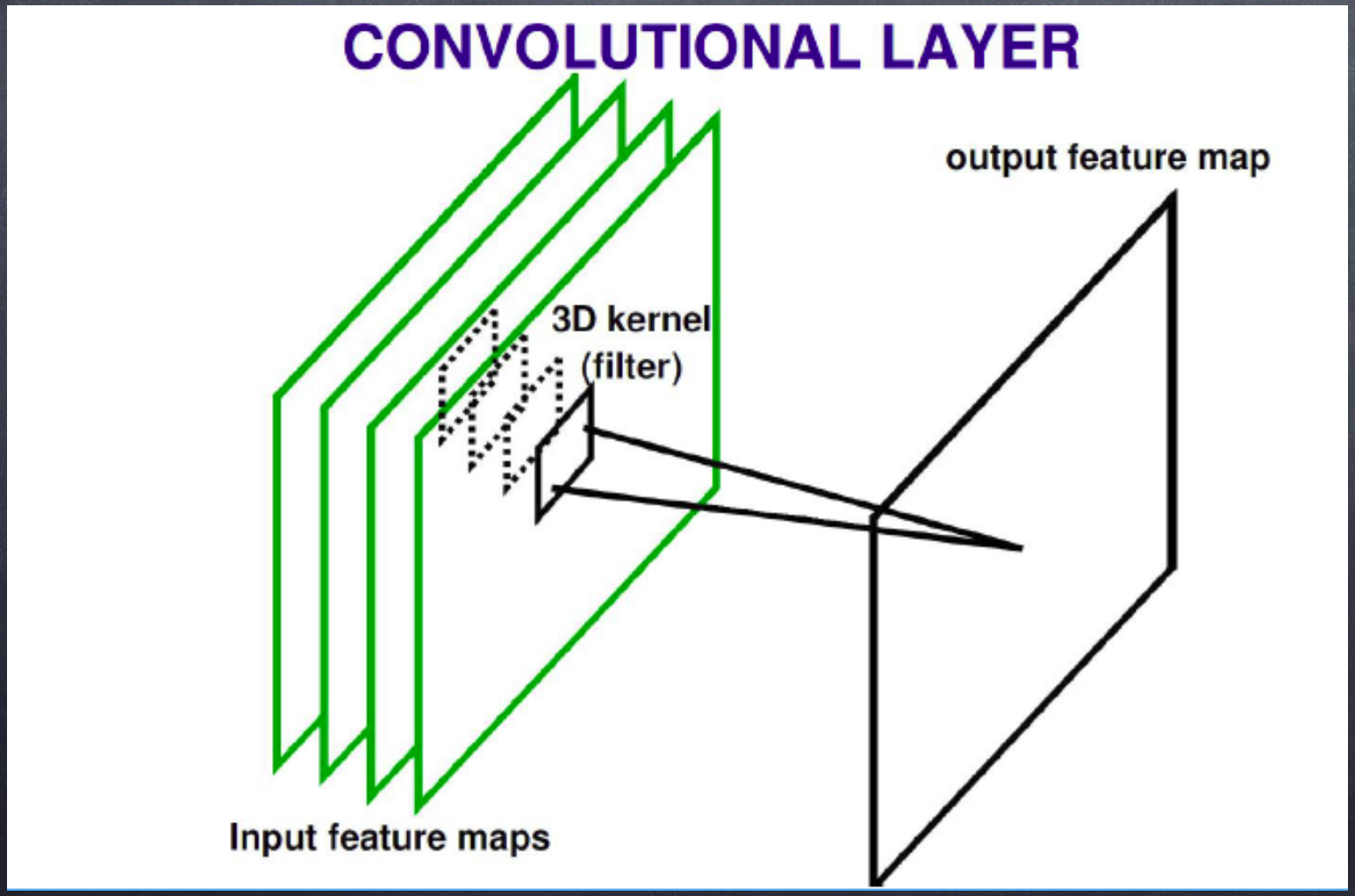
Input



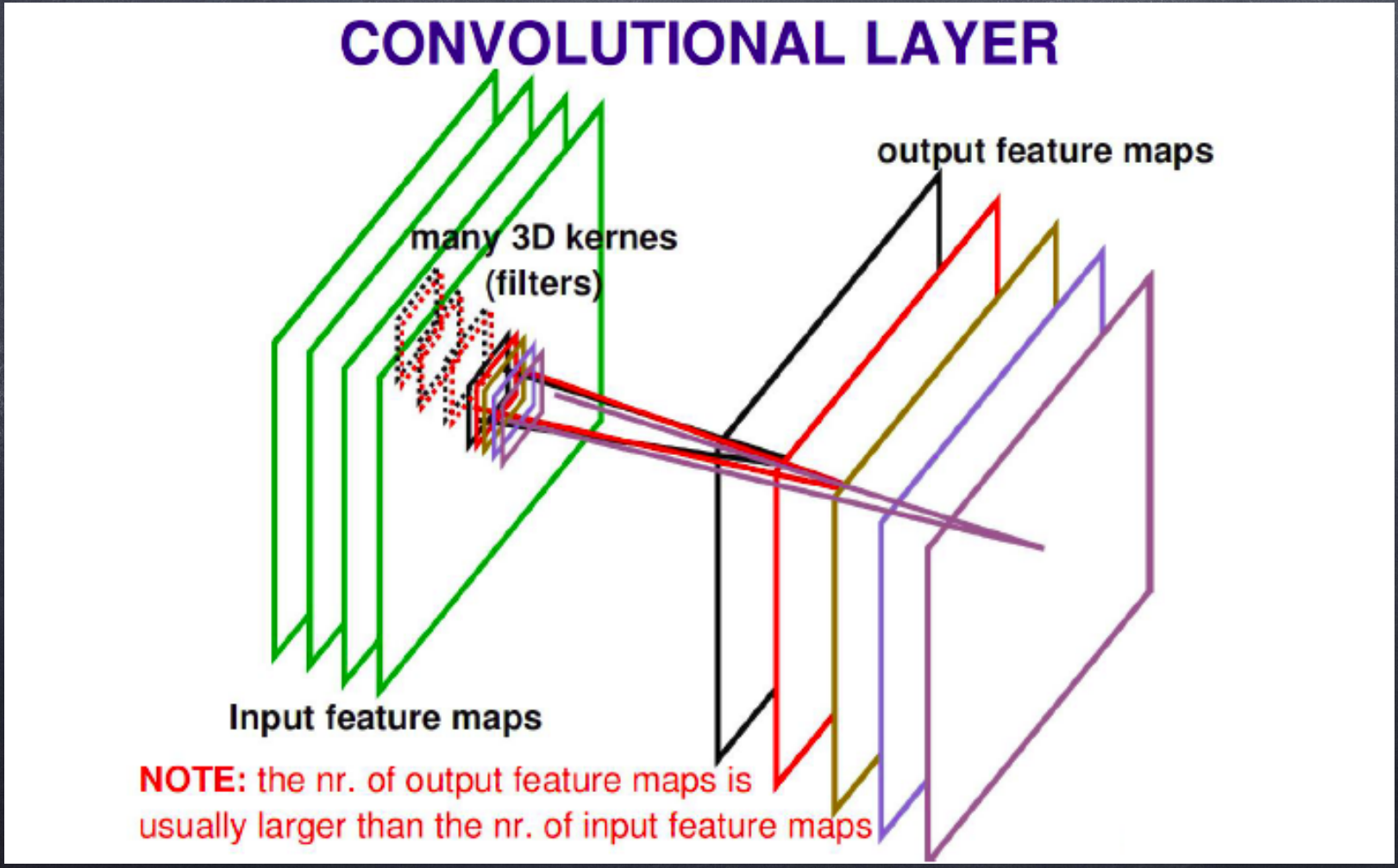
Feature map



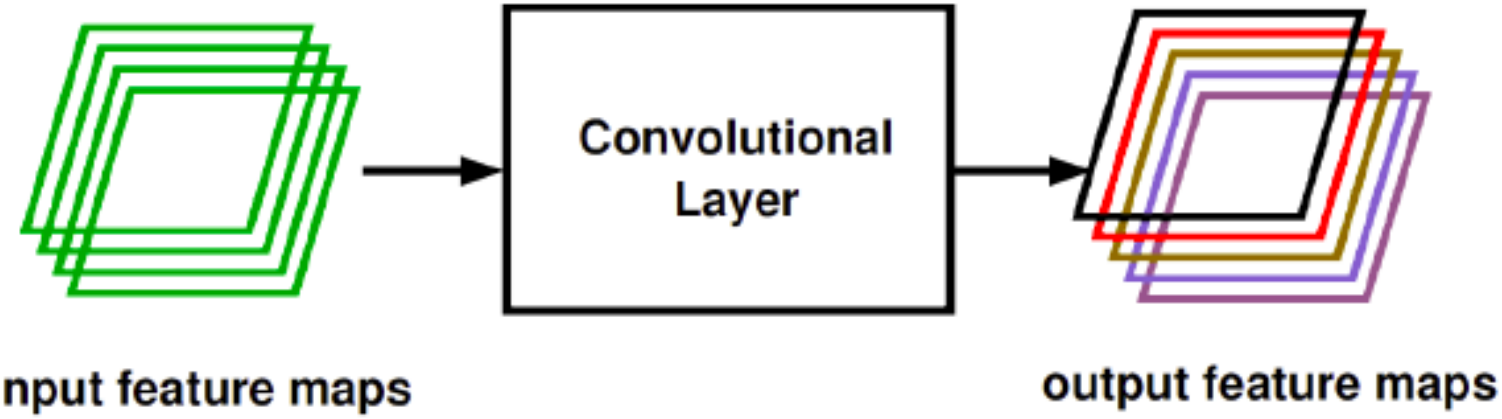
CNN


CONVOLUTIONAL LAYER
output feature map
3D kernel
(filter)
Input feature maps

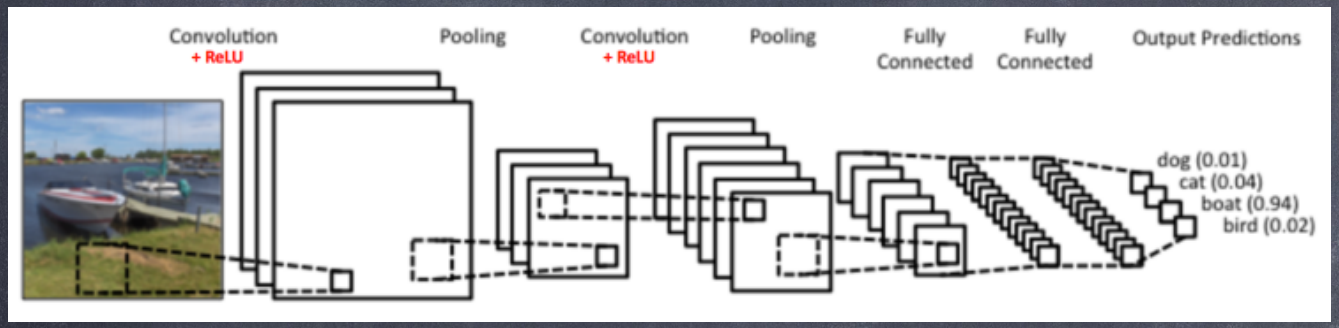
CNN


CONVOLUTIONAL LAYER
output feature maps
many 3D kernes
(filters)
Input feature maps
NOTE: the nr. of output feature maps is
usually larger than the nr. of input feature maps

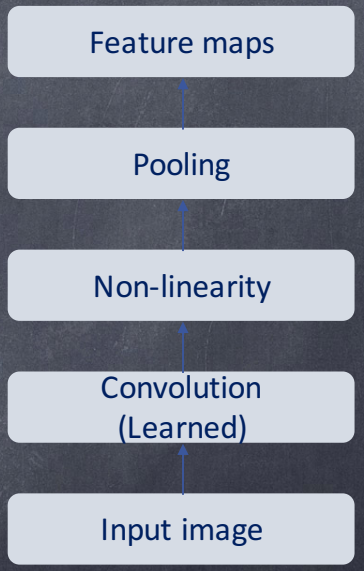
CNN


Convolutional Layer
input feature maps
output feature maps

Recall this image ...



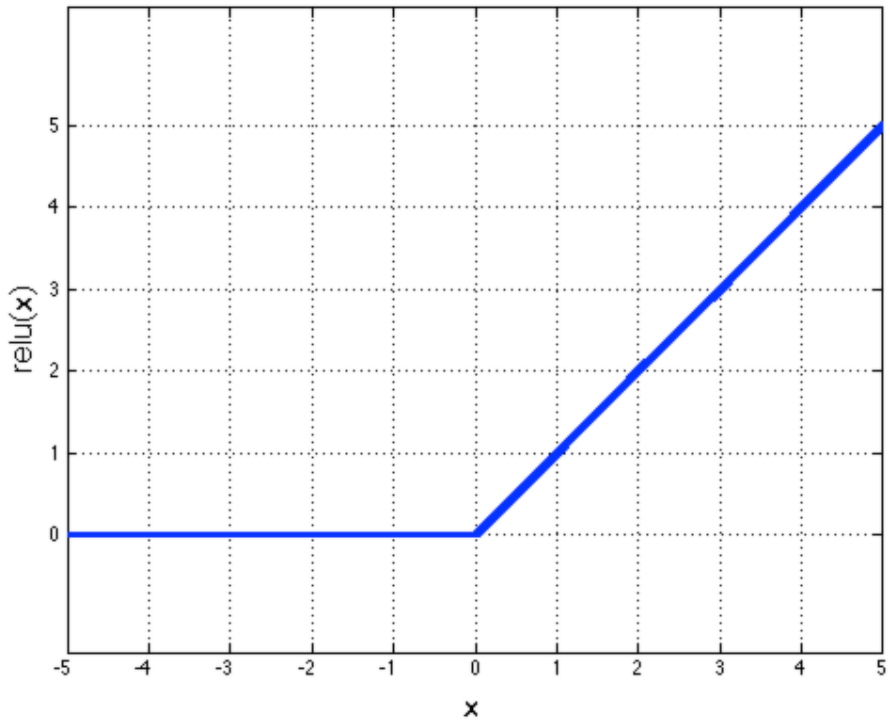
- Sample ConvNet Architecture



ReLU

- Rectified linear (ReLU) : max(0,x)

$$h^{(i)} = \max(w^{(i)T}x, 0) = \begin{cases} w^{(i)T}x & w^{(i)T}x > 0 \\ 0 & \text{else} \end{cases}$$



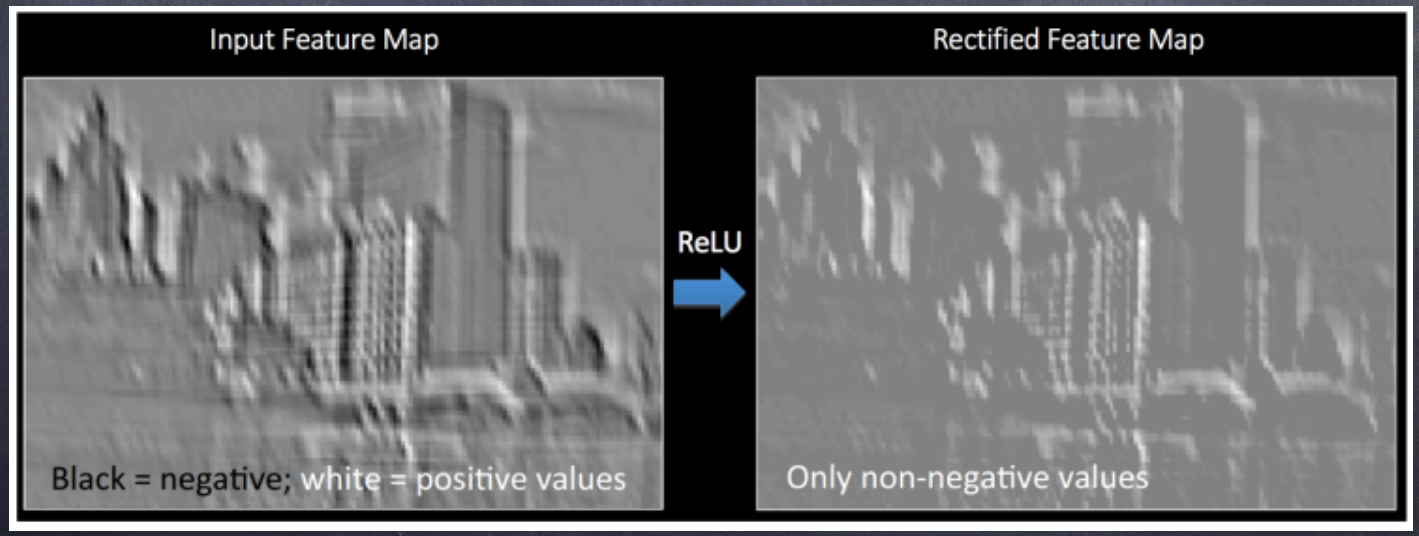

```
1 if input > 0:
2       return input
3 else:
4       return 0
```

Leaky ReLU

$$h^{(i)} = \max(w^{(i)T}x, 0) = \begin{cases} w^{(i)T}x & w^{(i)T}x > 0 \\ 0.01w^{(i)T}x & \text{else} \end{cases}$$

ReLU

- Easy to implement and backprop

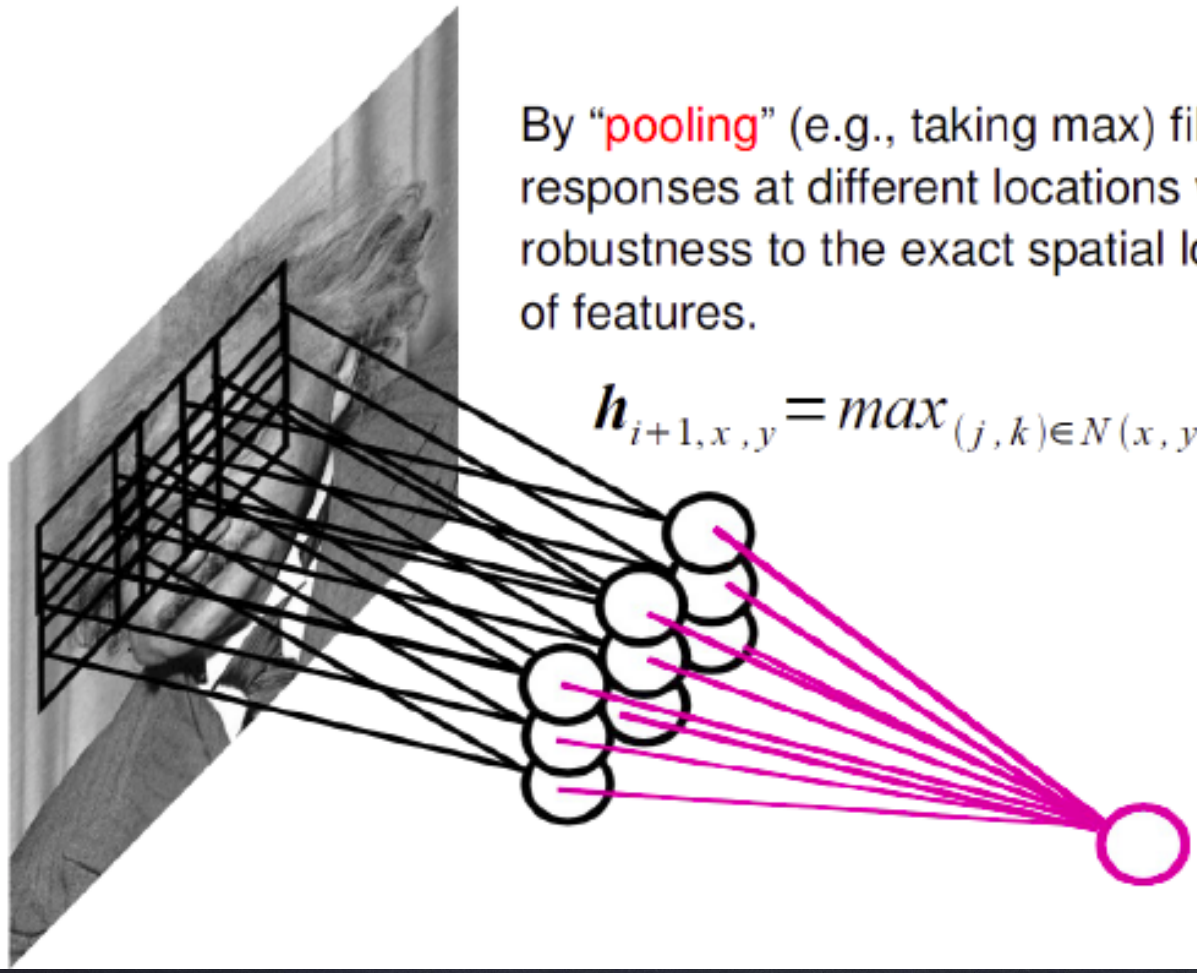


CNN

POOLING

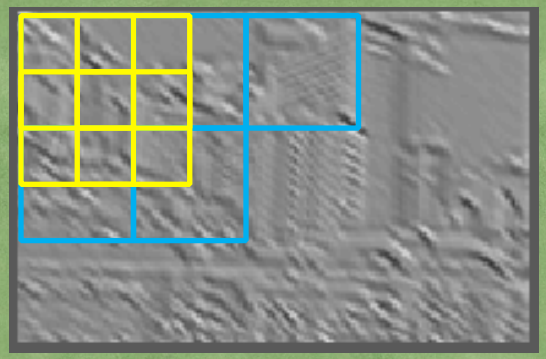
By "pooling" (e.g., taking max) filter responses at different locations we gain robustness to the exact spatial location of features.

$$\boldsymbol{h}_{i+1,x,y} = max_{(j,k)\in N(x,y)} \boldsymbol{h}_{i,j,k}$$

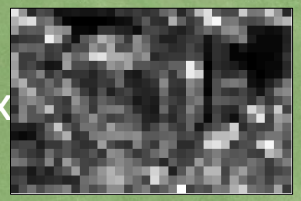


CNN

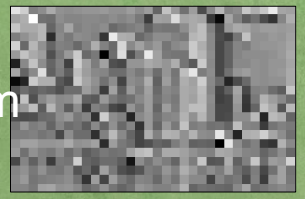
- Spatial Pooling
  - Non-overlapping / overlapping regions
  - Sum or max
  - Invariance to small transformations
  - Larger receptive fields
    (see more of input)



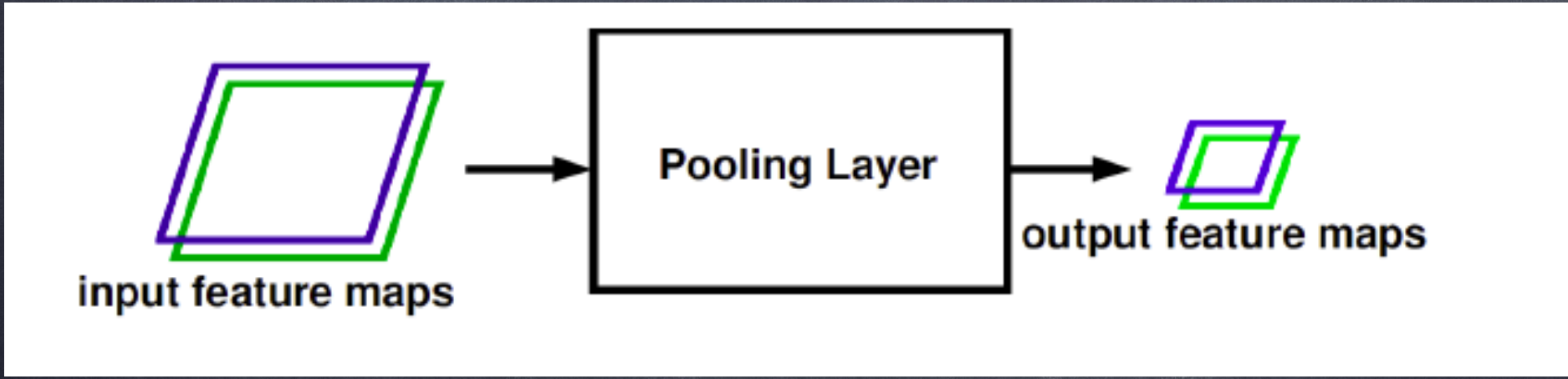
Max



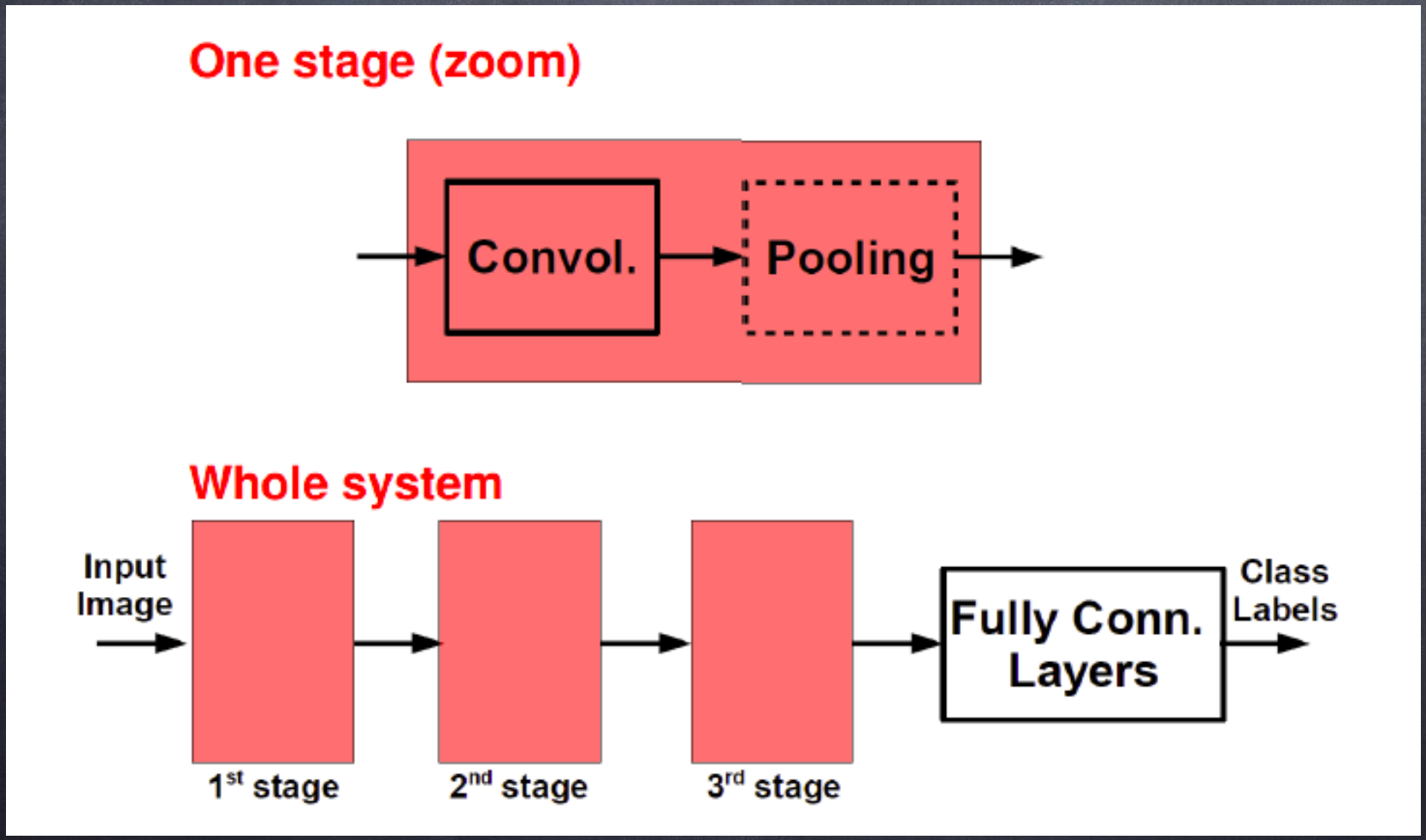
Sum



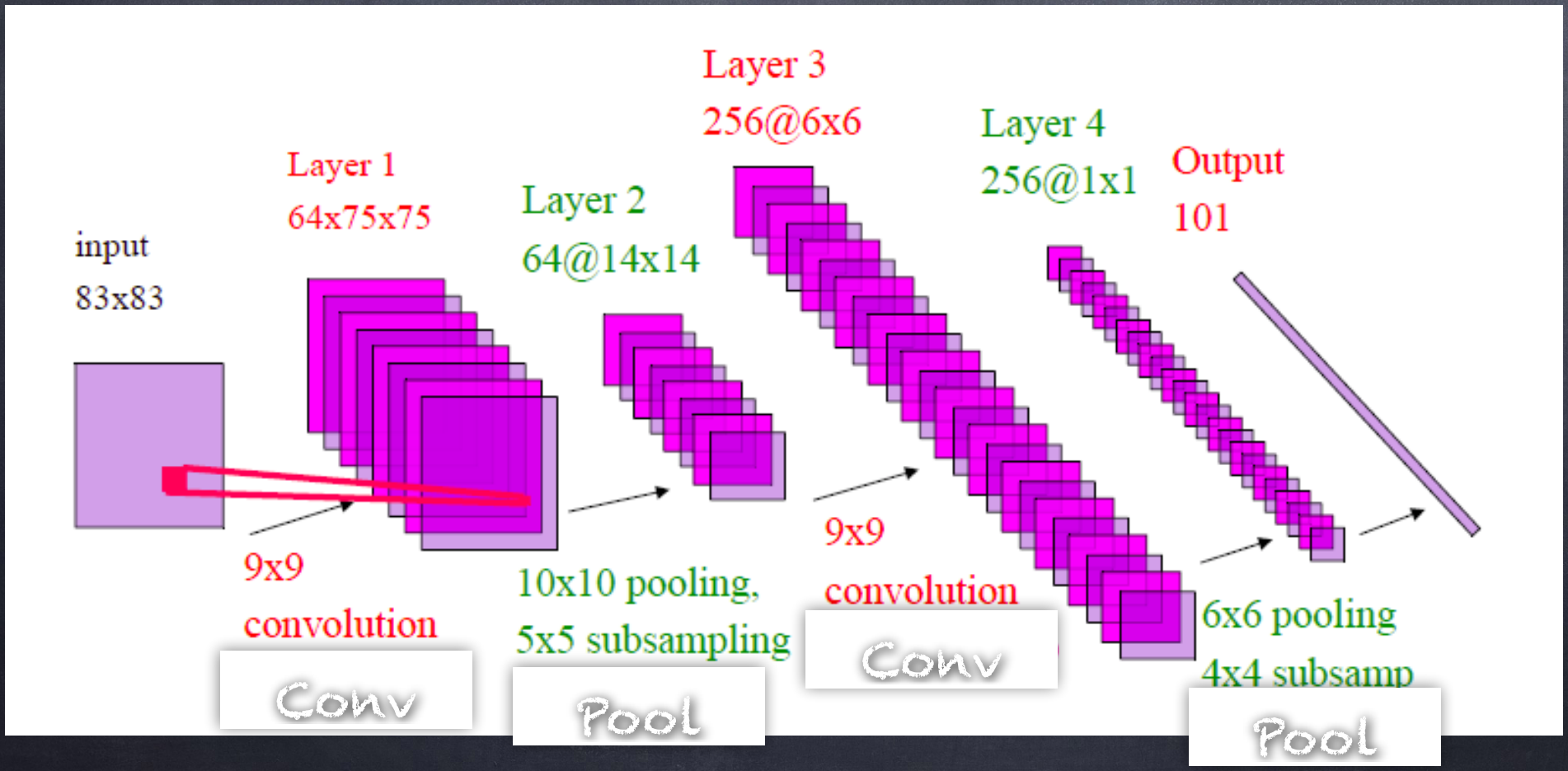
CNN



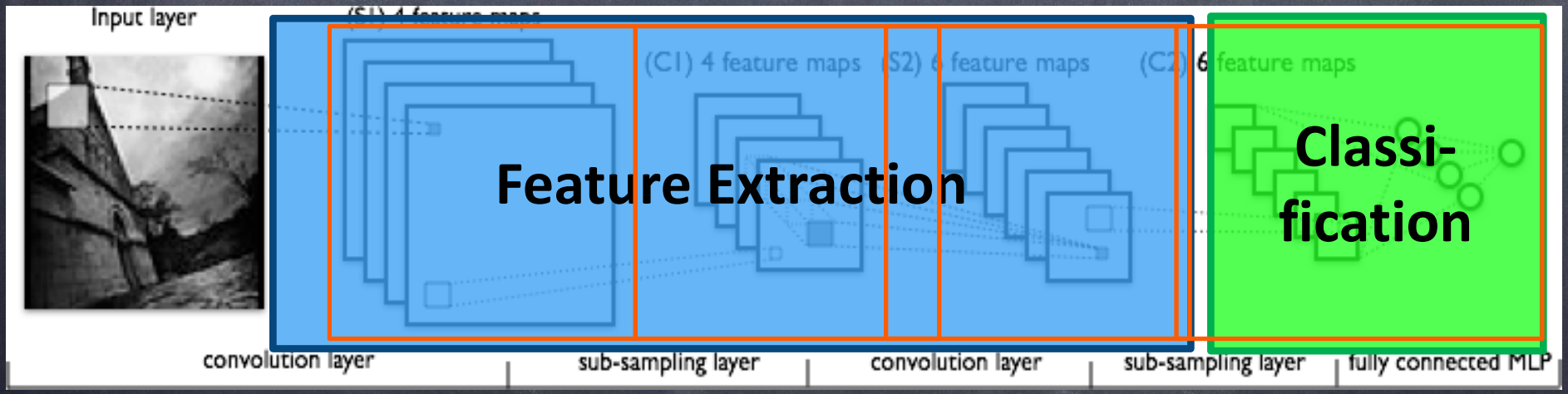
CNN


One stage (zoom)
Convol.
Pooling
Whole system
Input Image
1st stage
2nd stage
3rd stage
Fully Conn. Layers
Class Labels

LeNet Architecture



CNN - as feature extractor and classifier



Training CNN

Training set $\mathcal{T} = \left\{ \left(\mathbf{x}^{(k)}, y^{(k)} \right) \right\}_{k=1}^{p}$

Goal $y^{(k)} \approx f\left(\mathbf{x}^{(k)} \right)$ for all k

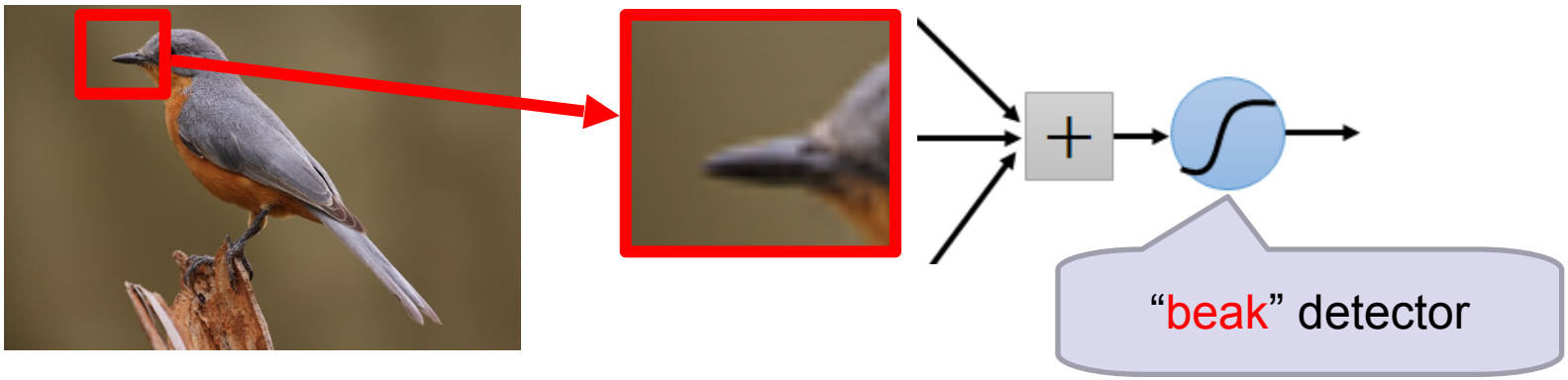
$$\min E = \frac{1}{2} \sum_{k=1}^{p} \left[y^{(k)} - f\left(\mathbf{x}^{(k)} \right) \right]^2$$

Error function may change

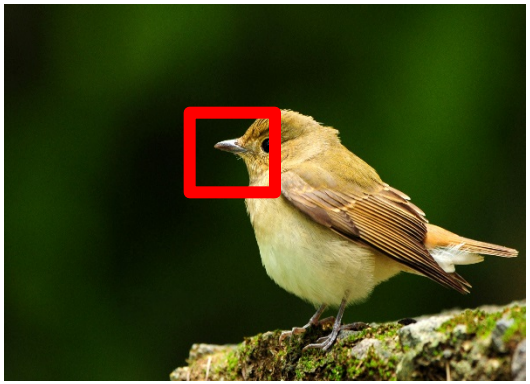
Consider learning an image:

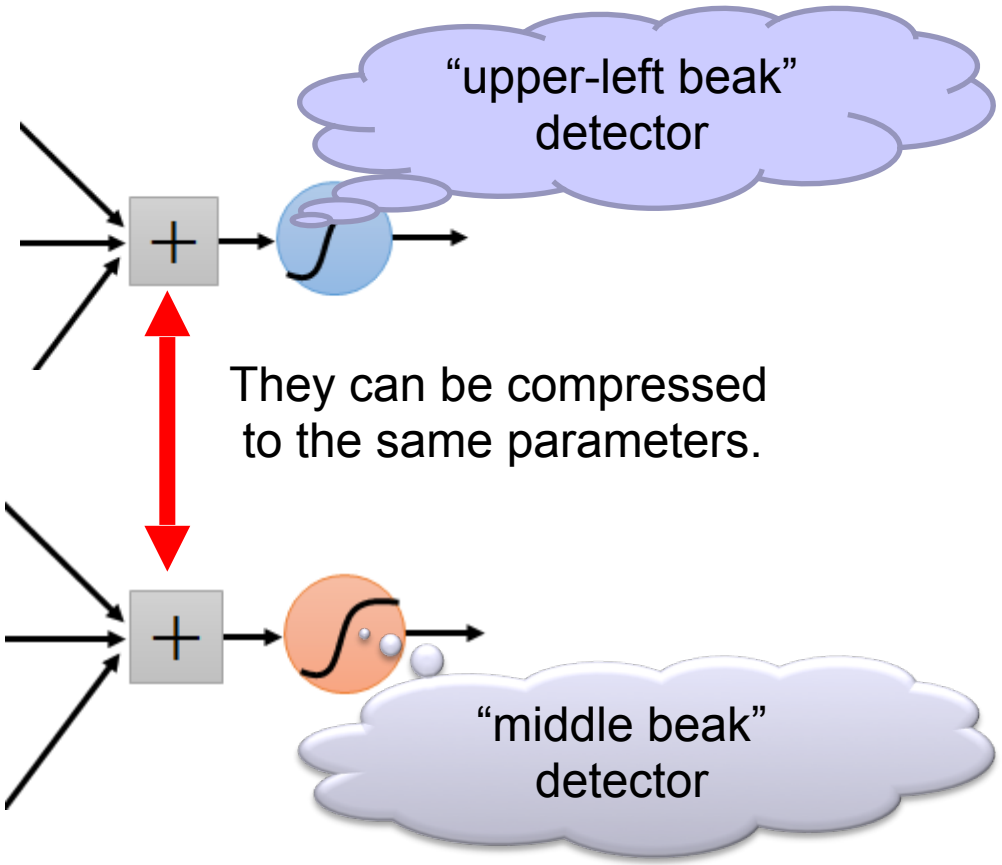
- Some patterns are much smaller than the whole image

Can represent a small region with fewer parameters



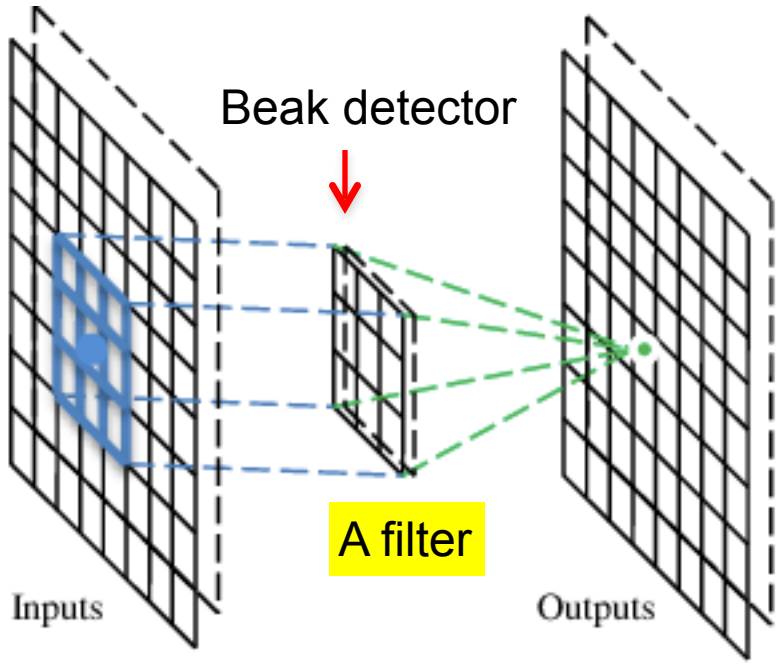
Same pattern appears in different places:
They can be compressed!
What about training a lot of such “small” detectors
and each detector must “move around”.





A convolutional layer

A CNN is a neural network with some convolutional layers (and some other layers). A convolutional layer has a number of filters that does convolutional operation.



Convolution

These are the network parameters to be learned.

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

⋮ ⋮

Each filter detects a small pattern (3 x 3).

Convolutio

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

Dot product

6 x 6 image

Convolutio

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

If stride=2

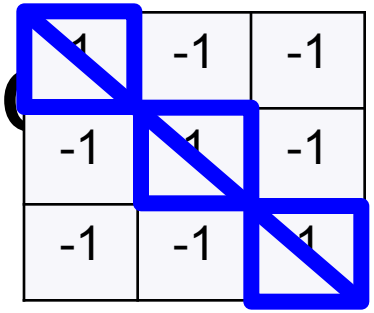
1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

3

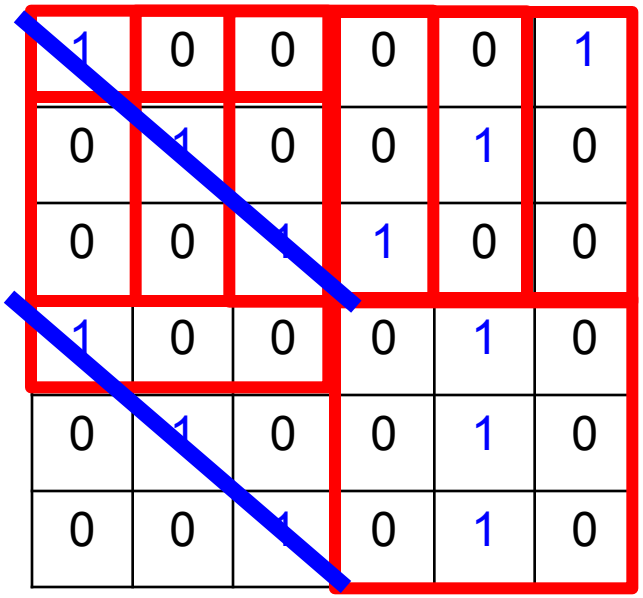
-3

Convolutio

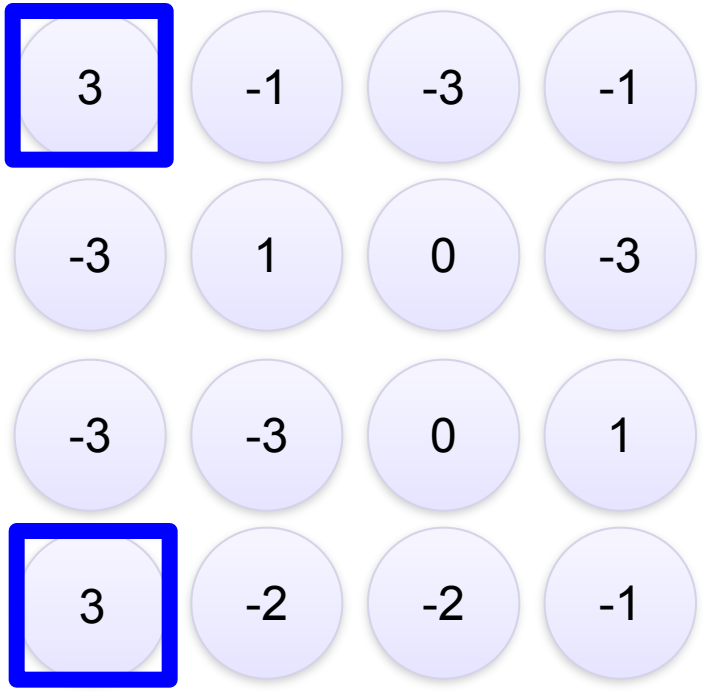


Filter 1

stride=1



6 x 6 image



Convolutio

-1	1	-1
-1	1	-1
-1	1	-1

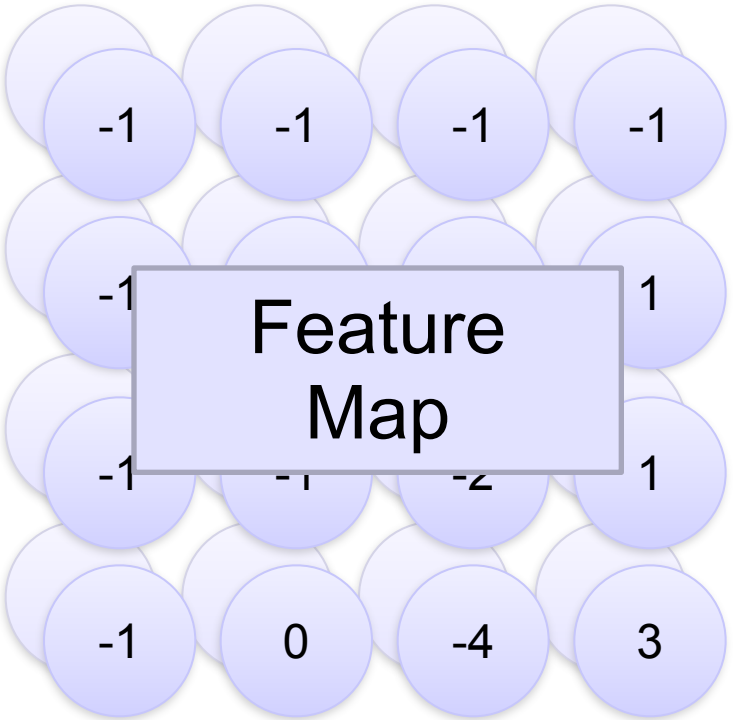
Filter 2

stride=1

1	0	0	0	0	1
0	1	0	0	1	0
0	0	1	1	0	0
1	0	0	0	1	0
0	1	0	0	1	0
0	0	1	0	1	0

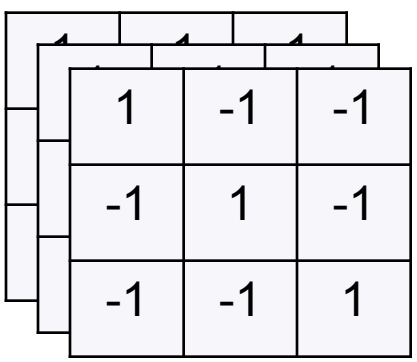
6 x 6 image

Repeat this for each filter

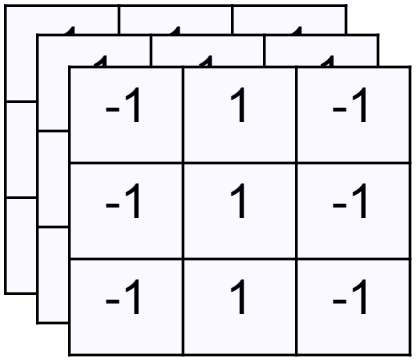


Two 4 x 4 images
Forming 2 x 4 x 4 matrix

Color image: RGB 3 channels

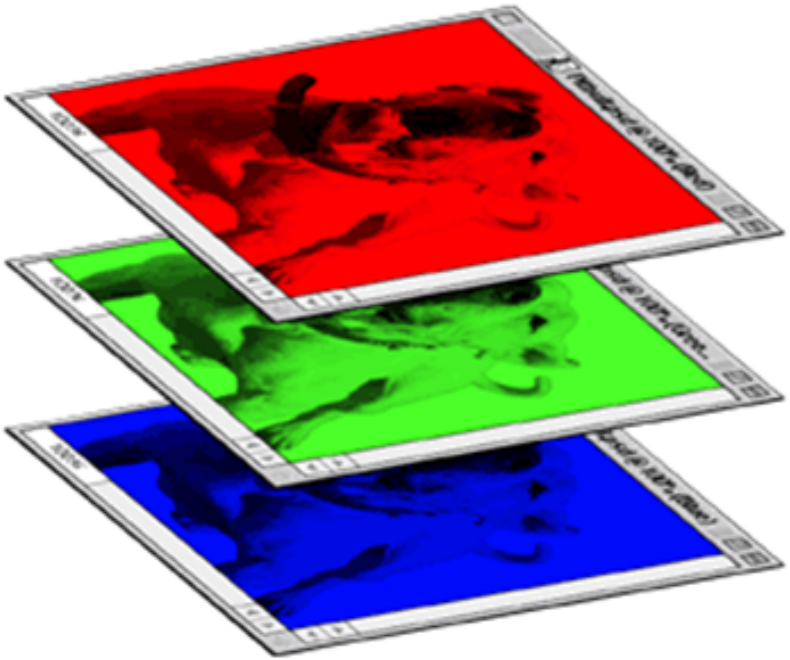


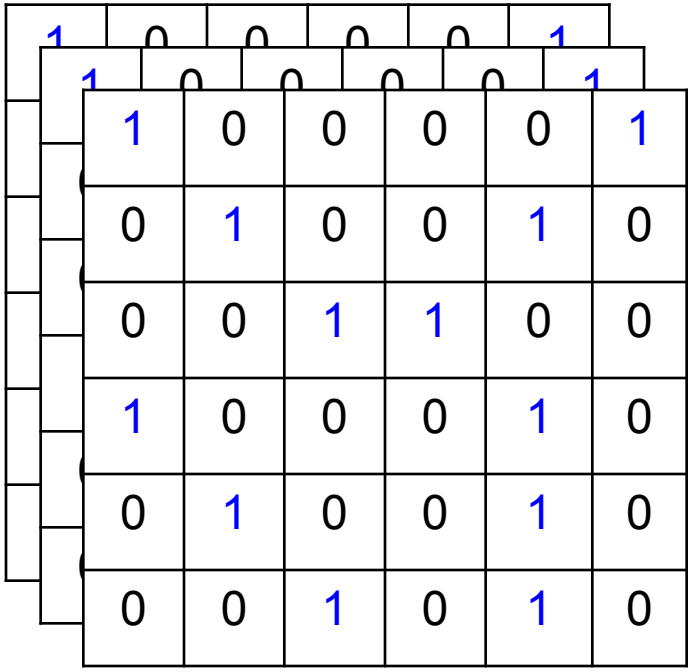
Filter 1



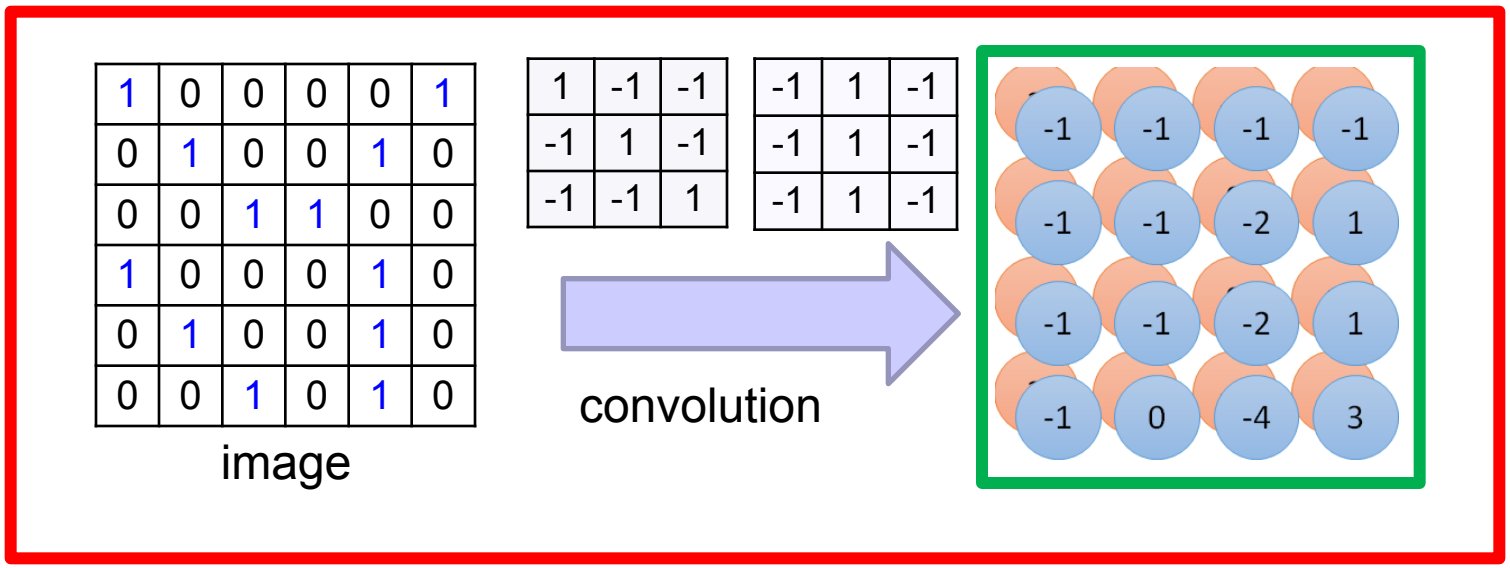
Filter 2

Color image

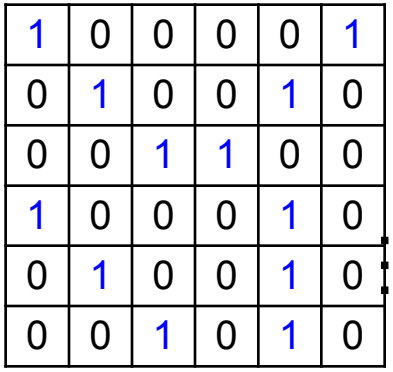


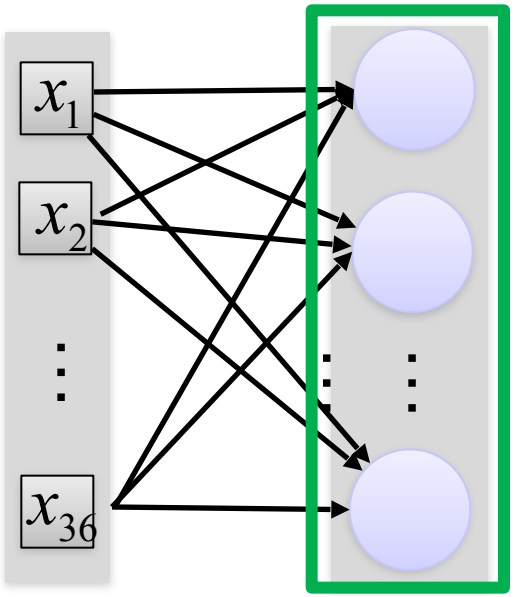


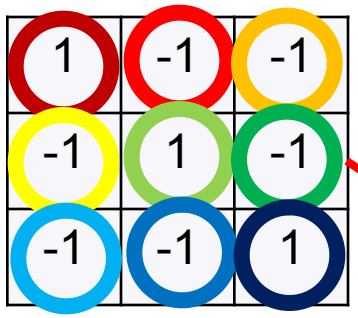
Convolution v.s. Fully Connected



Fully-connected







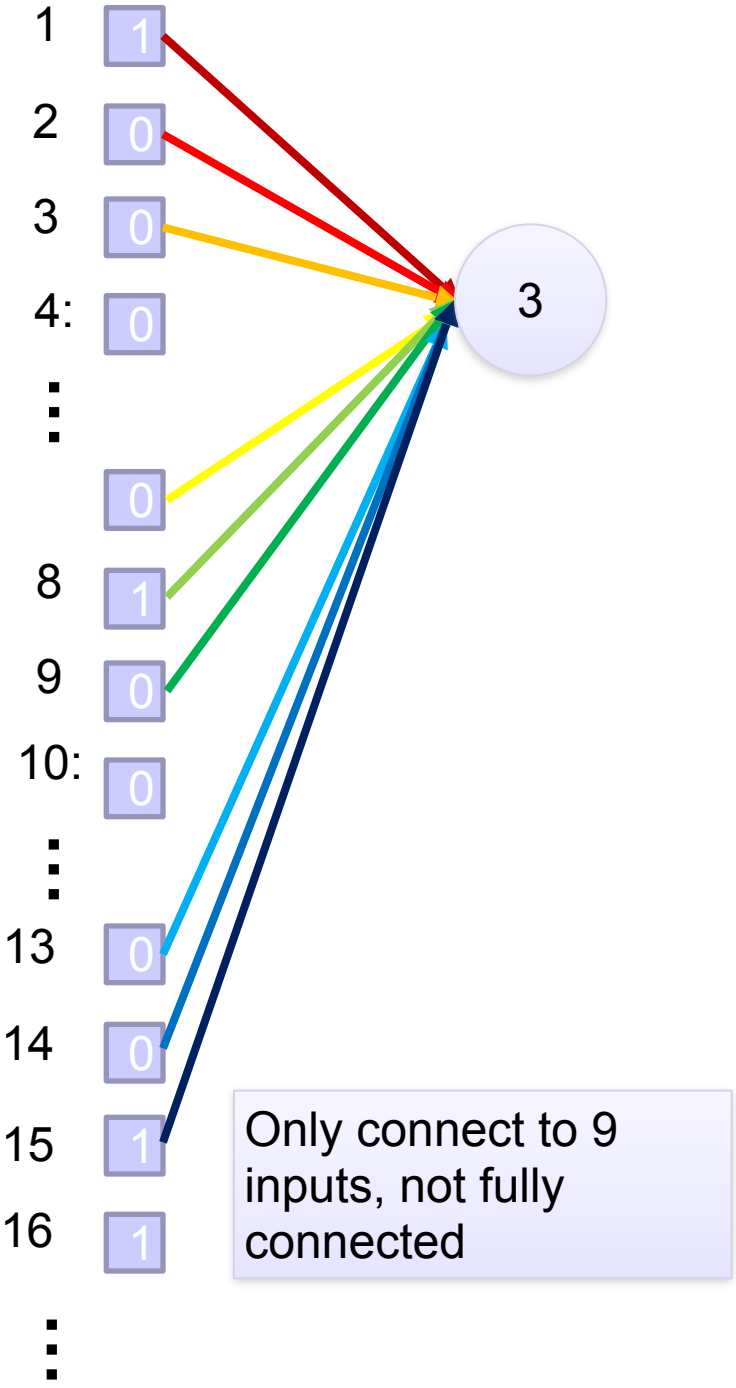
Filter 1

1	0	0	0	0	
0	1	0	0	1	
0	0	1	1	0	
1	0	0	0	1	
0	1	0	0	1	0
0	0	1	0	1	0

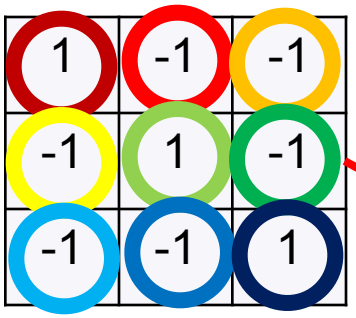
6 x 6 image

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

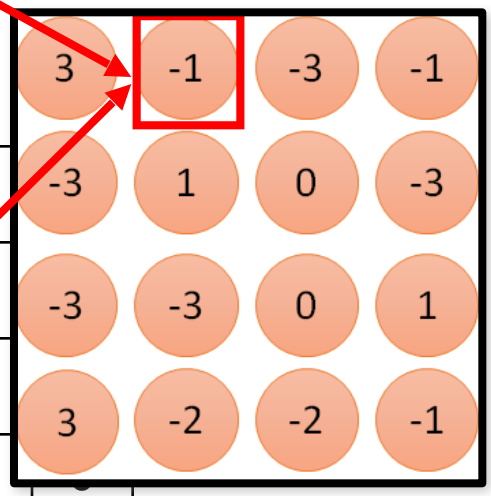
fewer parameters!



Only connect to 9 inputs, not fully connected



Filter 1

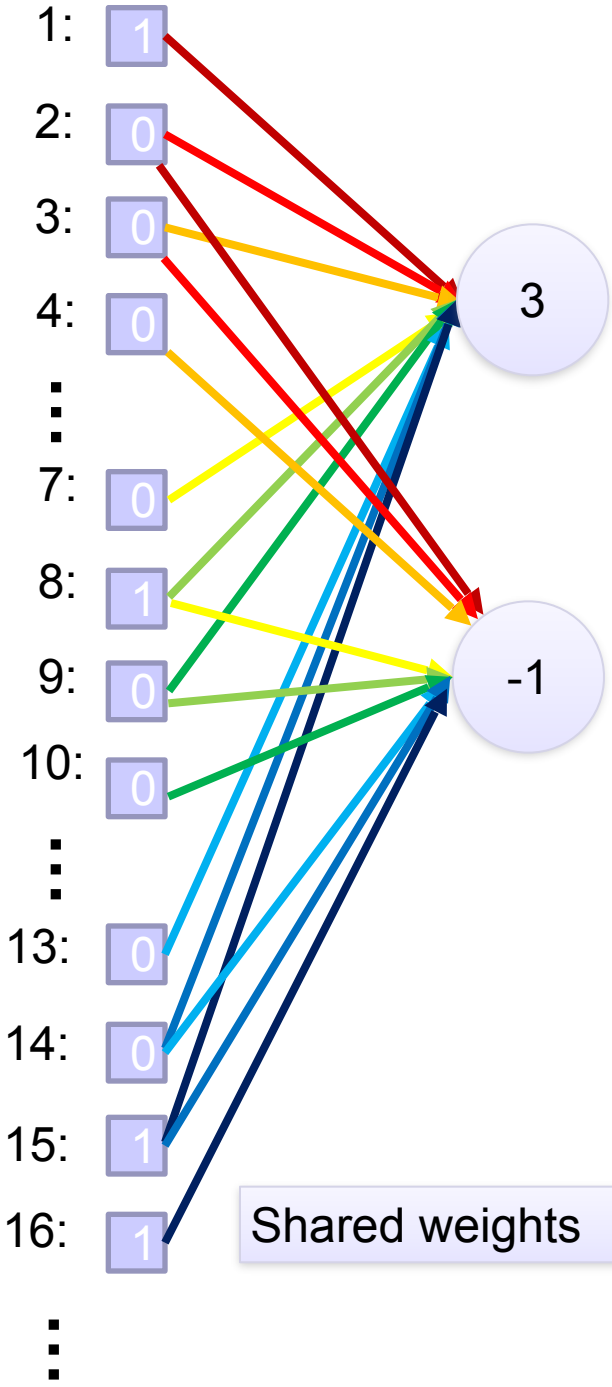


1	0	0	0	0	
0	1	0	0	1	
0	0	1	1	0	
1	0	0	0	1	
0	1	0	0	1	0
0	0	1	0	1	0

6 x 6 image

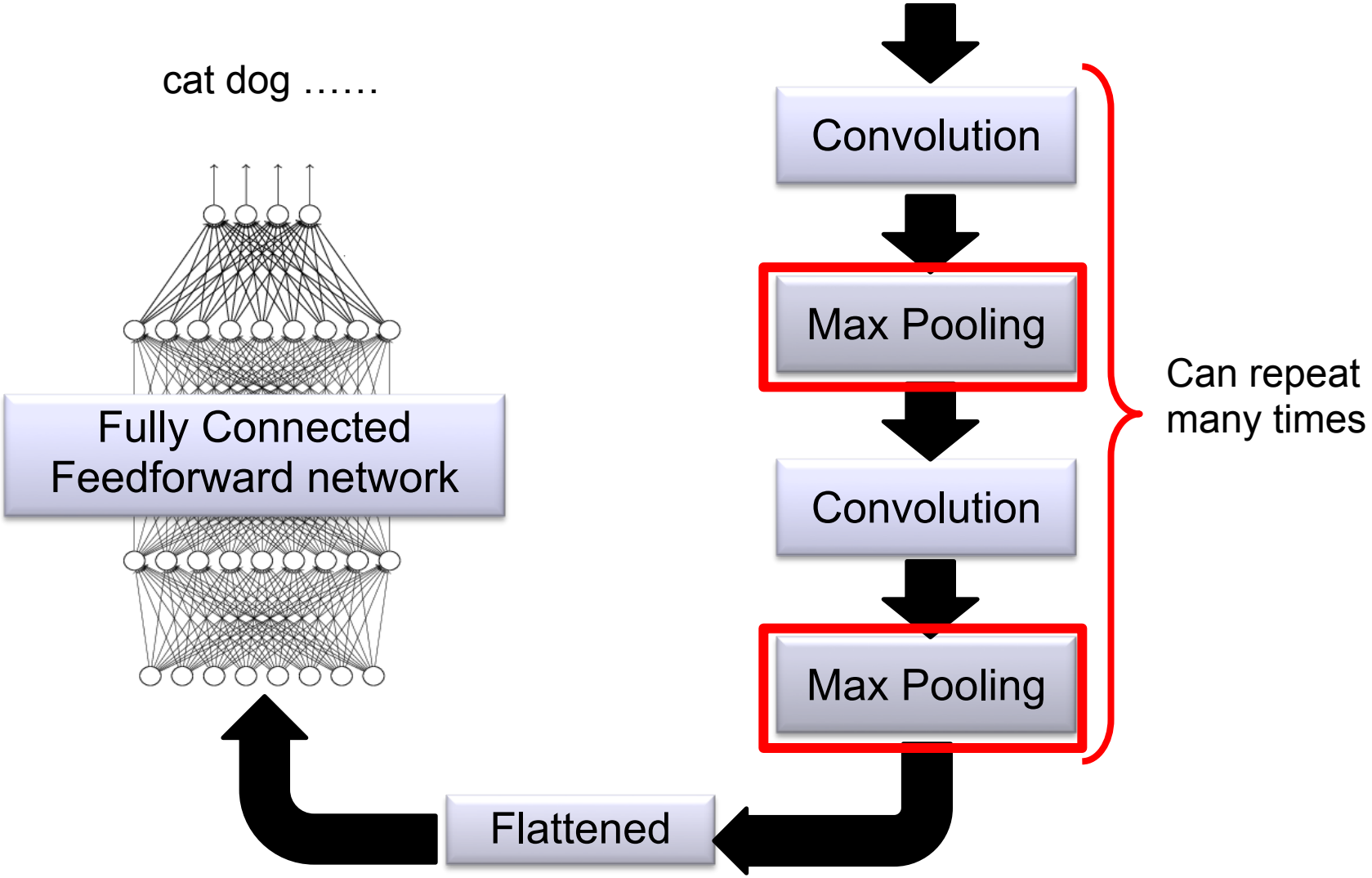
Fewer parameters

Even fewer parameters



Shared weights

The whole


cat dog ……
Convolution
Max Pooling
Convolution
Max Pooling
Can repeat many times
Flattened
Fully Connected Feedforward network

Max Pooling

1	-1	-1
-1	1	-1
-1	-1	1

Filter 1

-1	1	-1
-1	1	-1
-1	1	-1

Filter 2

3	-1
-3	1

-3	-1
0	-3

-3	-3
3	-2

0	1
-2	-1

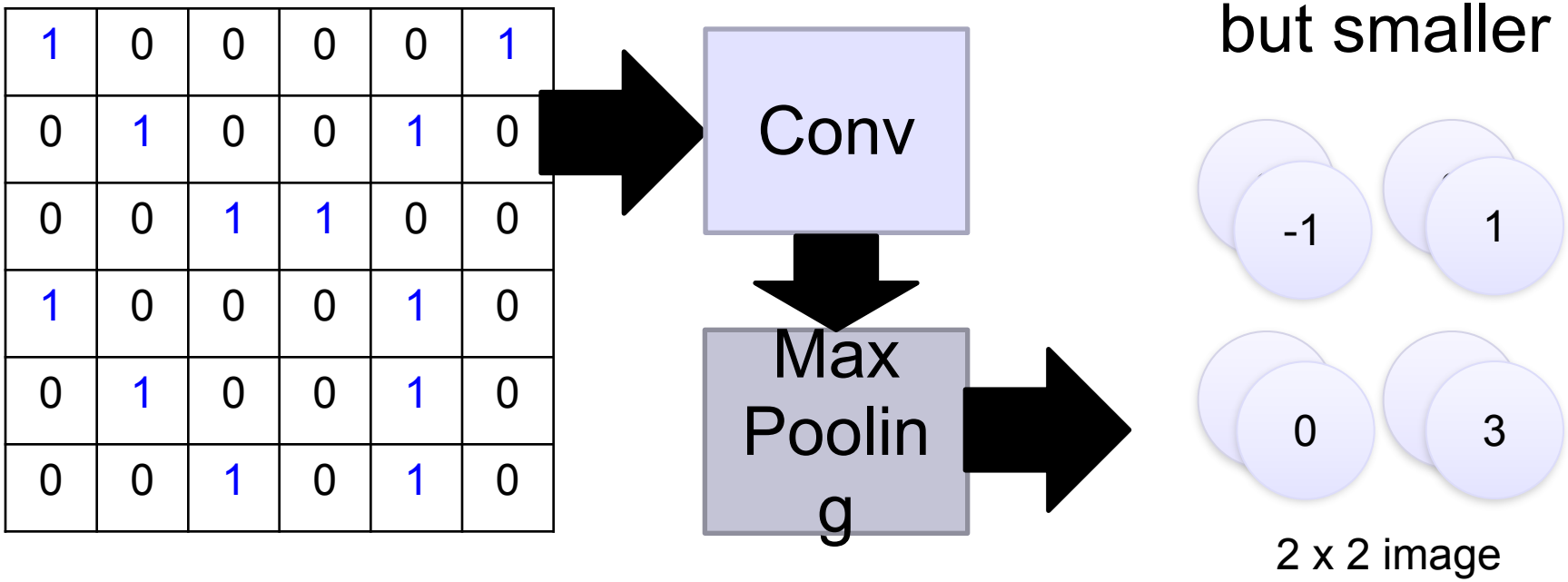
-1	-1
-1	-1

-1	-1
-2	1

-1	-1
-1	0

-2	1
-4	3

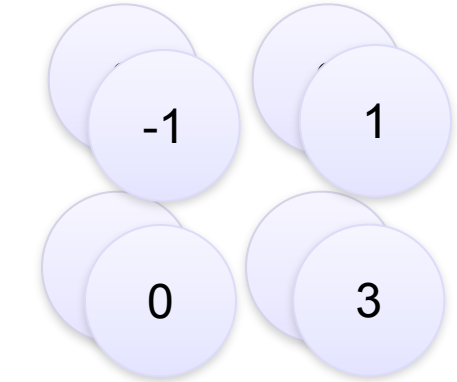
Max Pooling

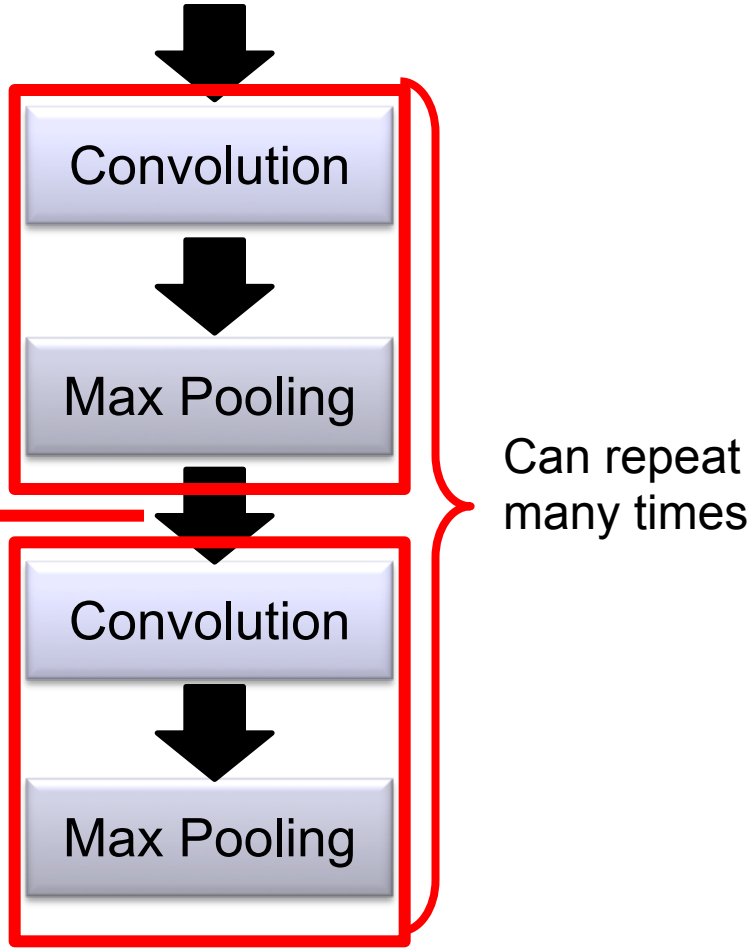


6 x 6 image

Each filter is a channel

The whole





Smaller than the original image

The number of channels is the number of filters

The whole

cat dog ……

Convolution

Max Pooling

A new image

Convolution

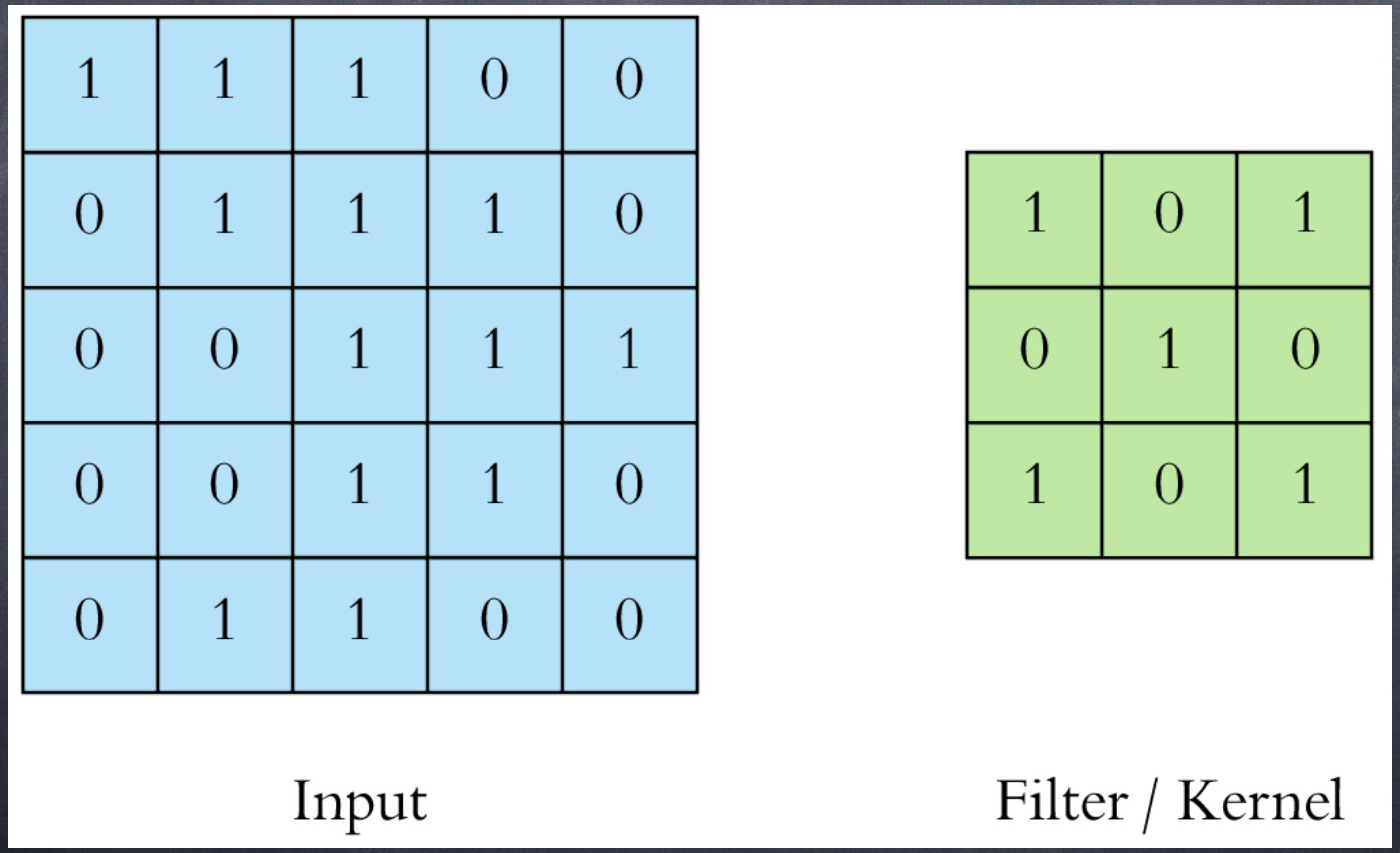
Max Pooling

A new image

Flattened

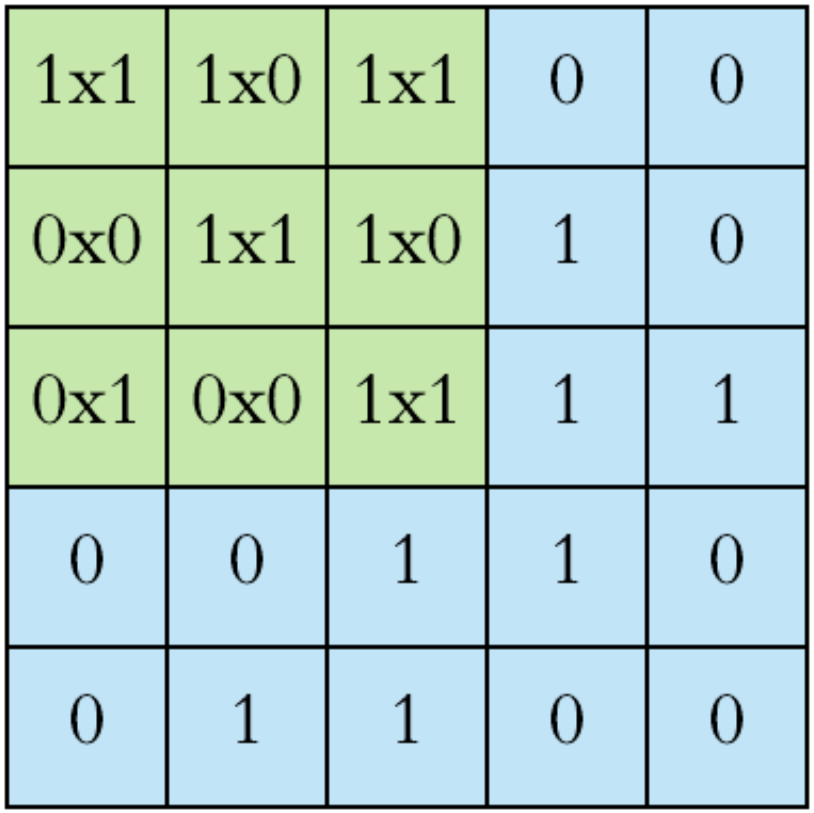
Fully Connected Feedforward network

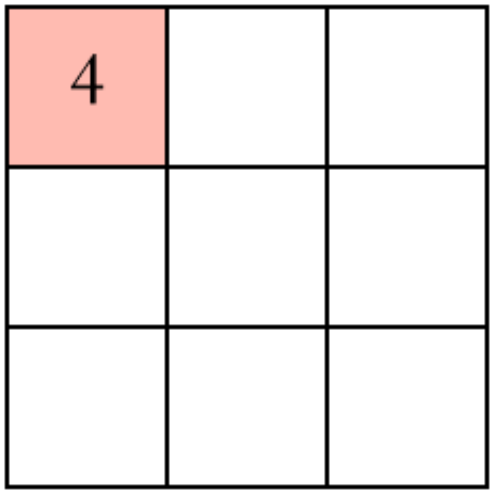
Let us try ourselves



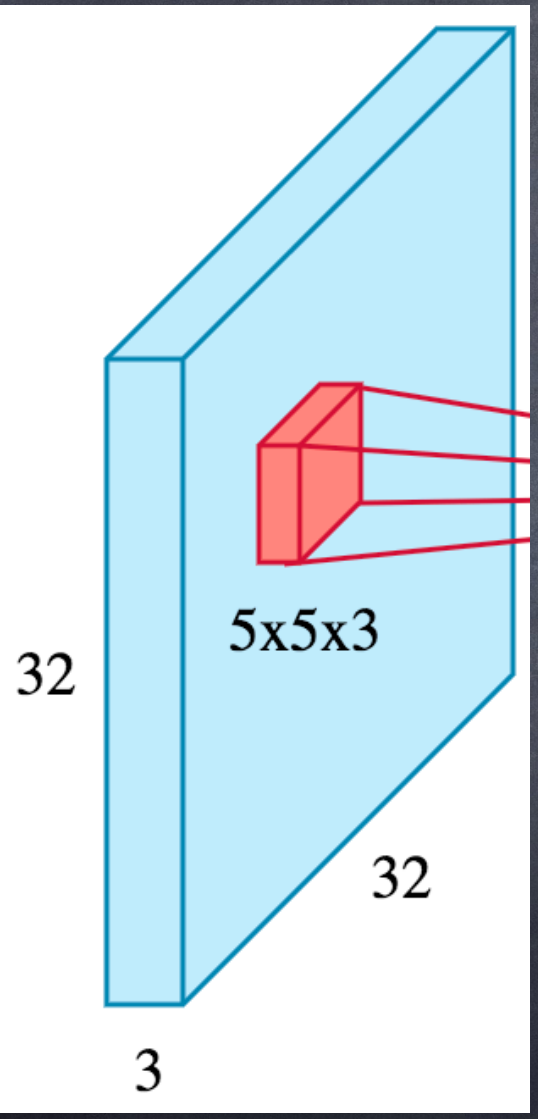
1x1	1x0	1x1	0	0
0x0	1x1	1x0	1	0
0x1	0x0	1x1	1	1
0	0	1	1	0
0	1	1	0	0

4		

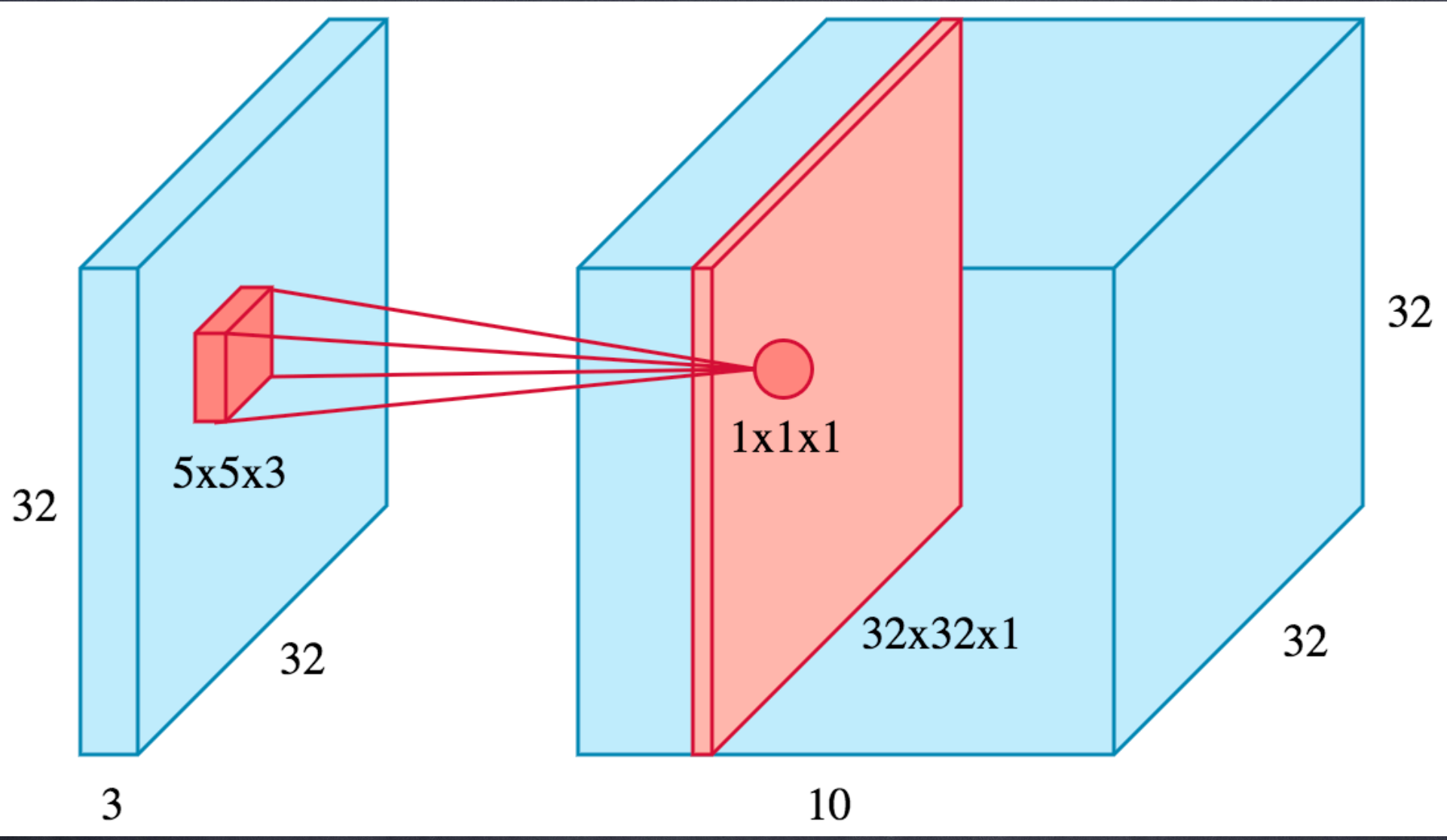

1x1 1x0 1x1 0 0
0x0 1x1 1x0 1 0
0x1 0x0 1x1 1 1
0 0 1 1 0
0 1 1 0 0


4

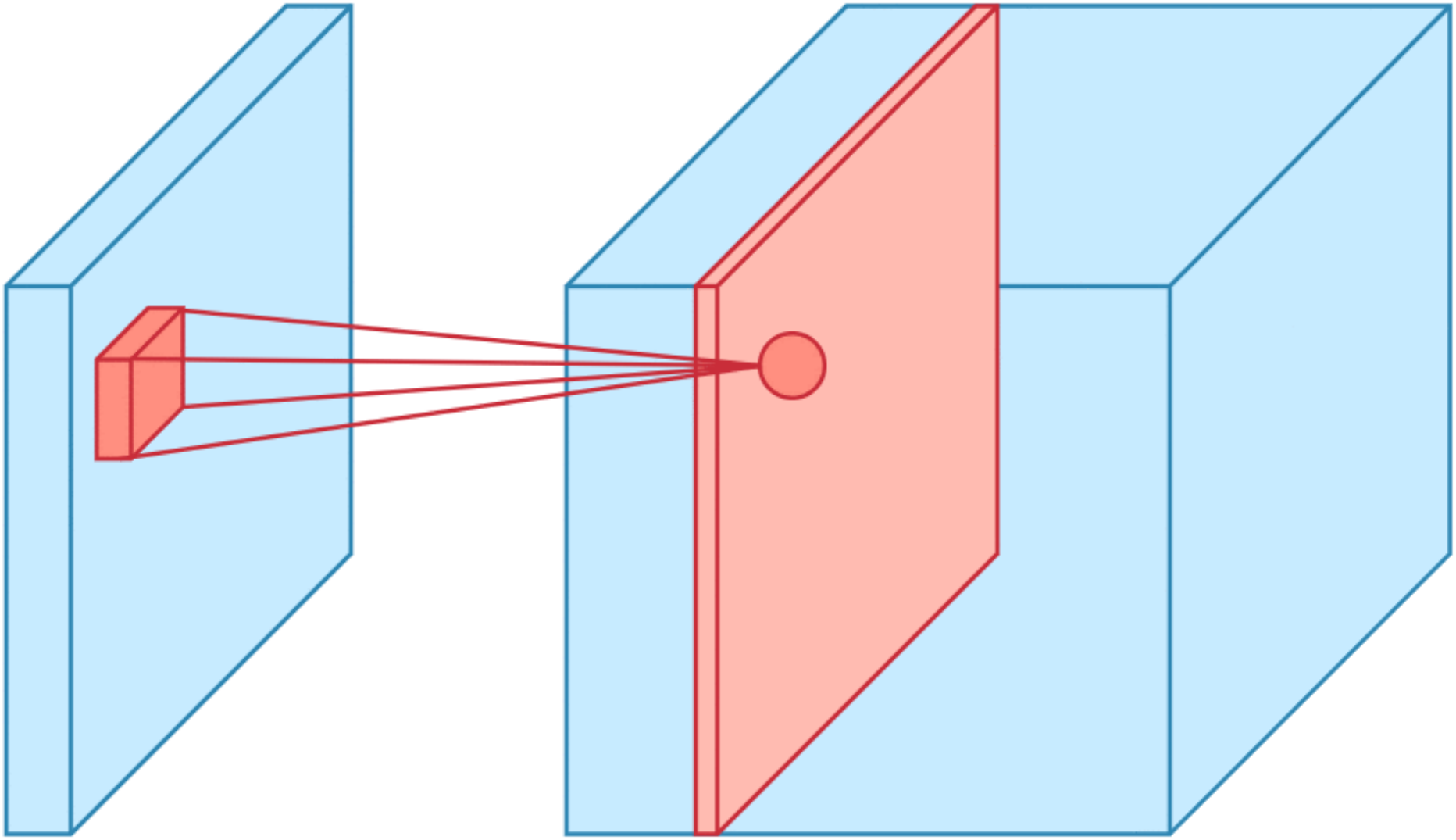
Convolving in 3D - 10 filters

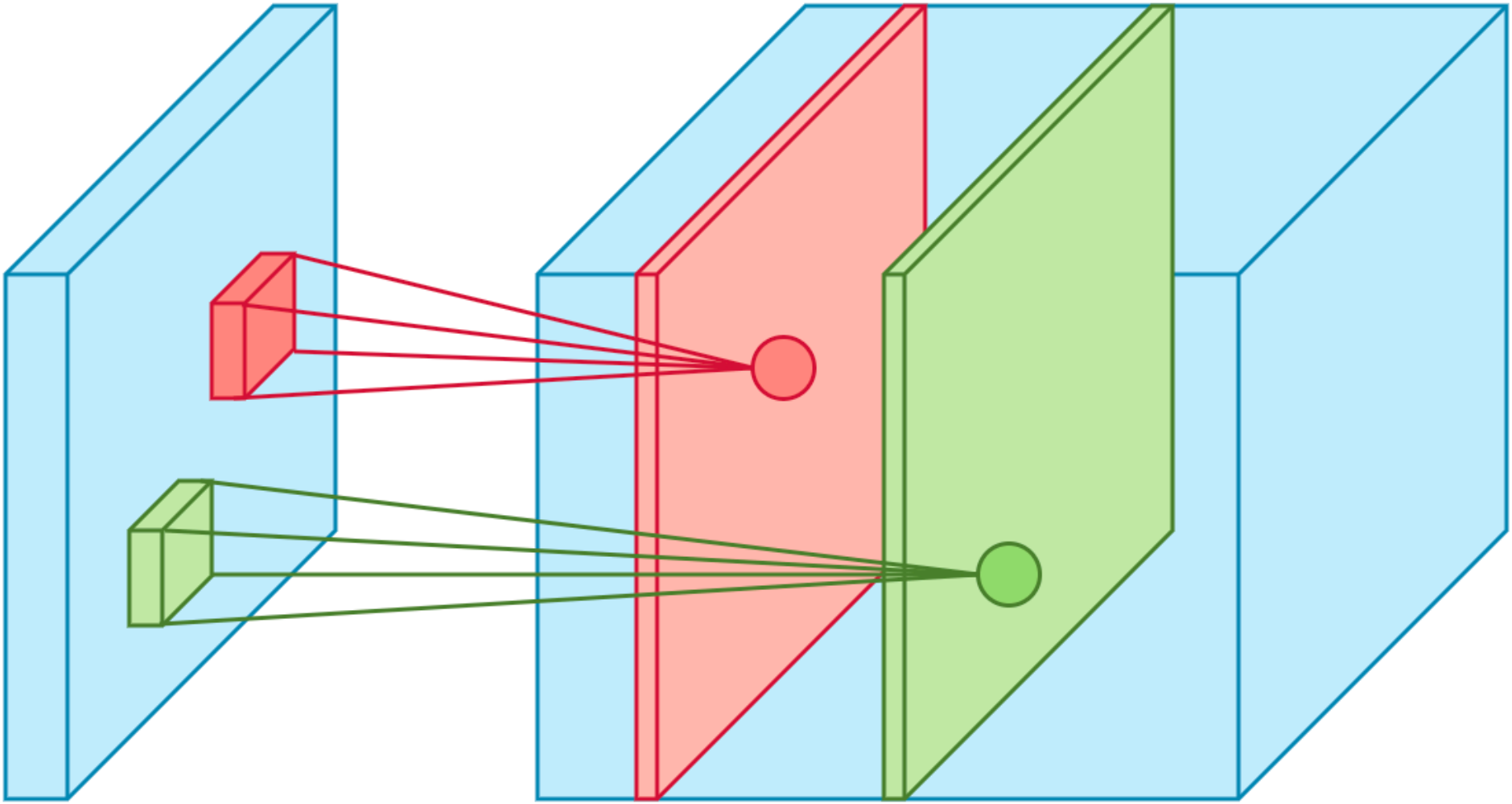


- Zero Pad
- Stride of 1
- 10 filters
- What would be size of output?

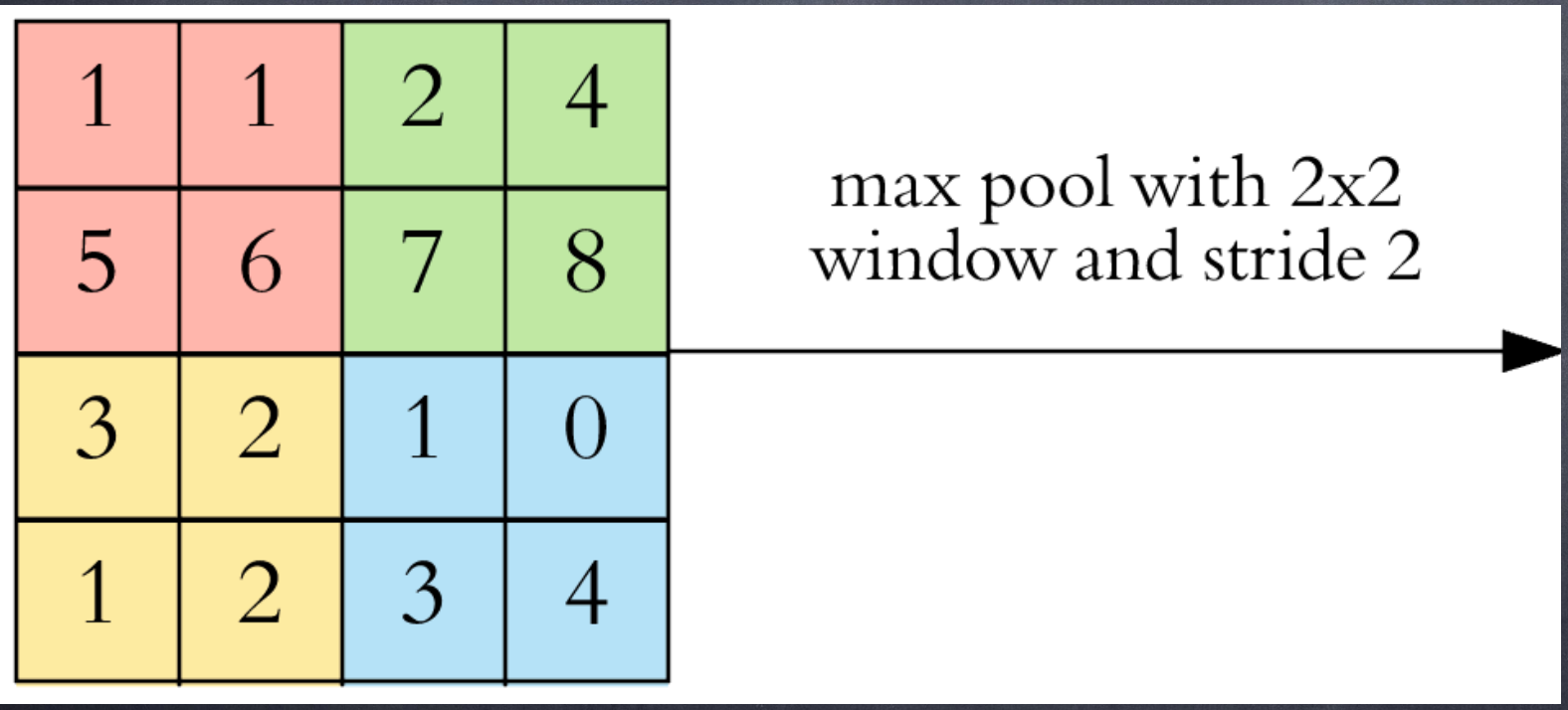


10 filters - each of size (3*3*5) - no zero padding

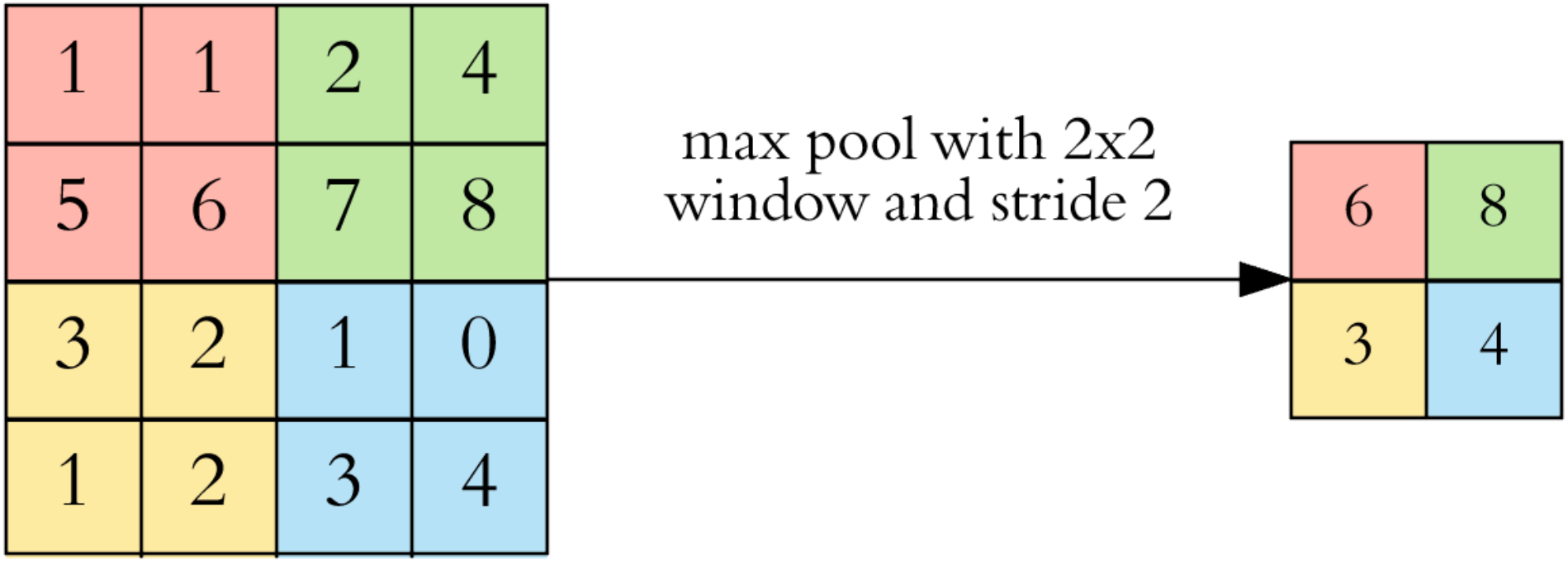


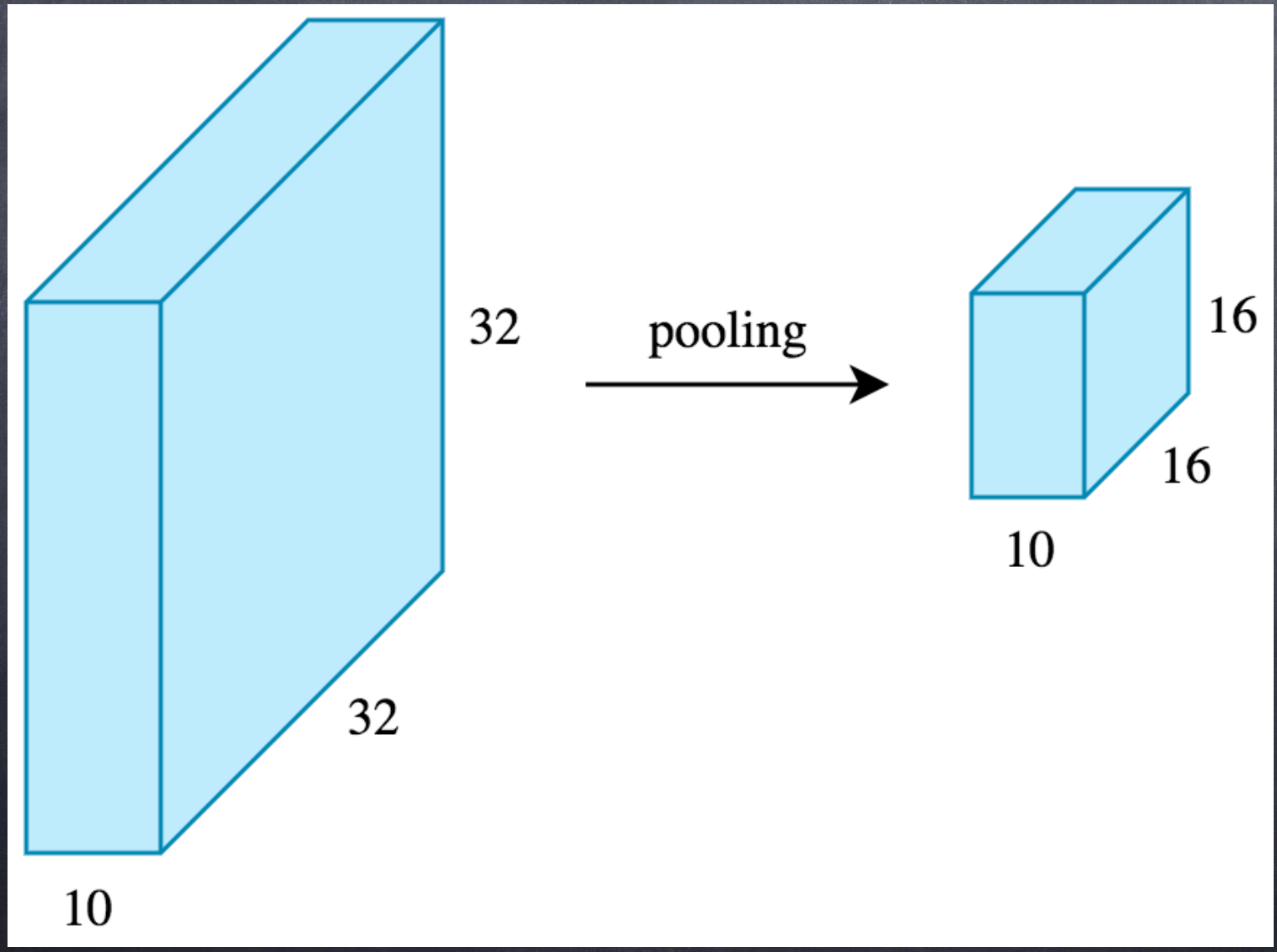


What will be the output?



What will be the output?



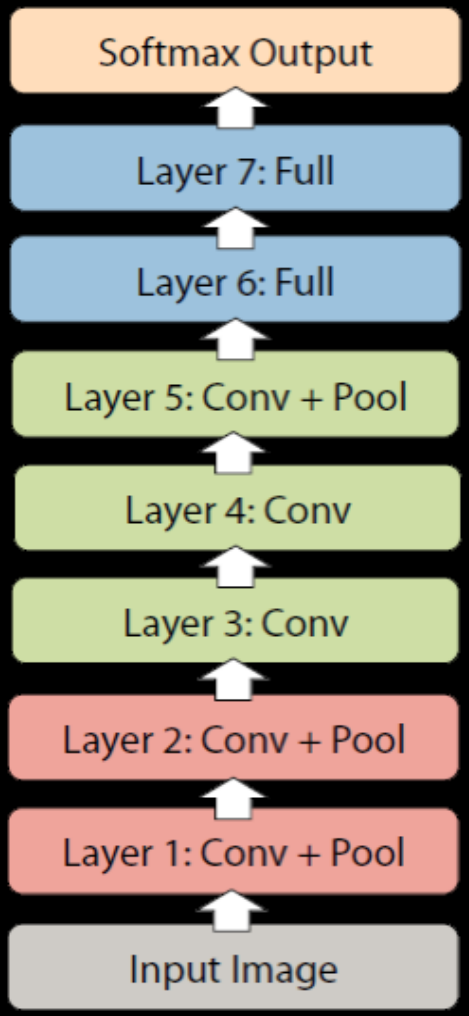

32
32
10
pooling
16
16
10

Network Depth

- What do you mean by Network Depth?

Why ConvNet should be Deep?

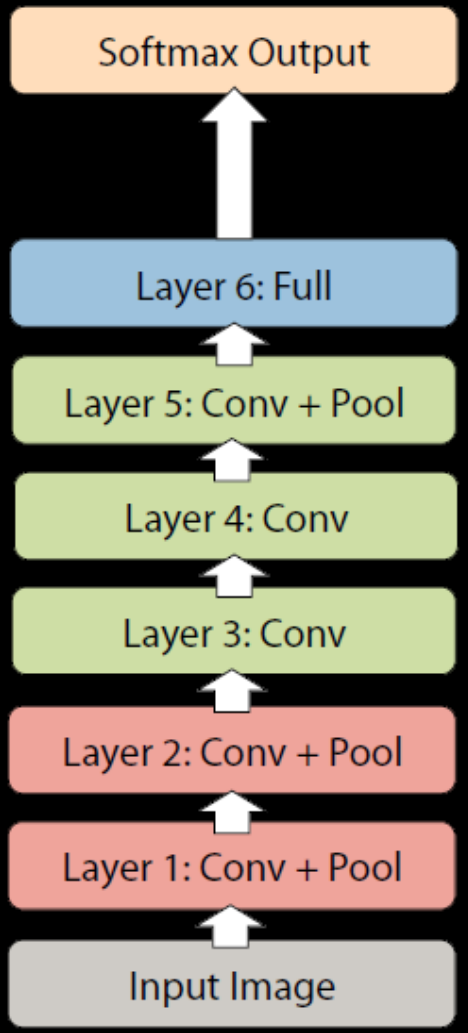
- 8 layers total
- Trained on Imagenet dataset [Deng et al. CVPR'09]
- 18.2% top-5 error
- Our reimplementation:
  18.1% top-5 error



Rob Fergus, NIPS 2013

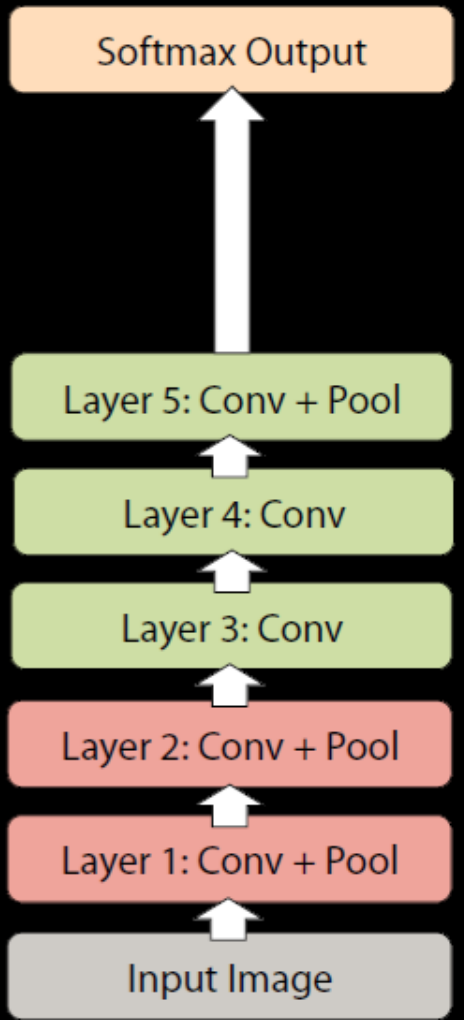
Why ConvNet should be Deep?

- Remove top fully connected layer
  - Layer 7
- Drop 16 million parameters
- Only 1.1% drop in performance!



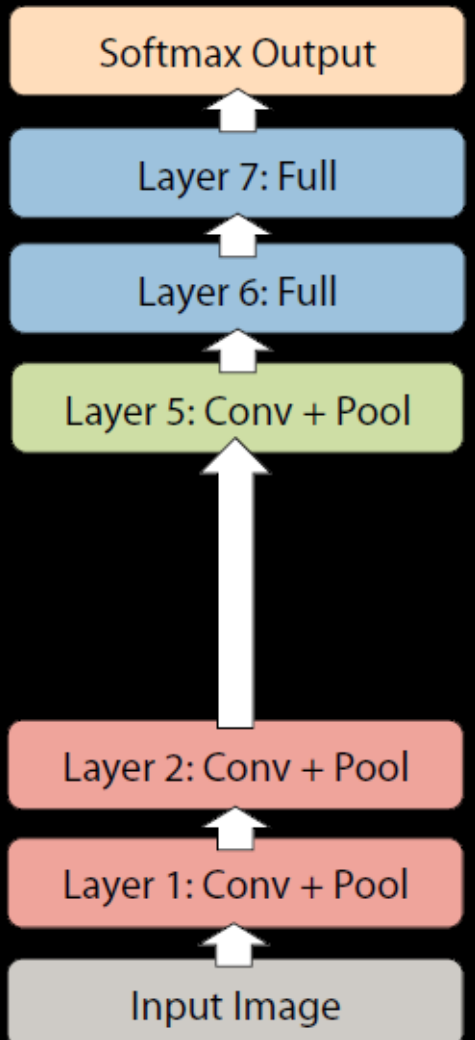
Why ConvNet should be Deep?

- Remove both fully connected layers
  - Layer 6 & 7
- Drop ~50 million parameters
- 5.7% drop in performance



Why ConvNet should be Deep?

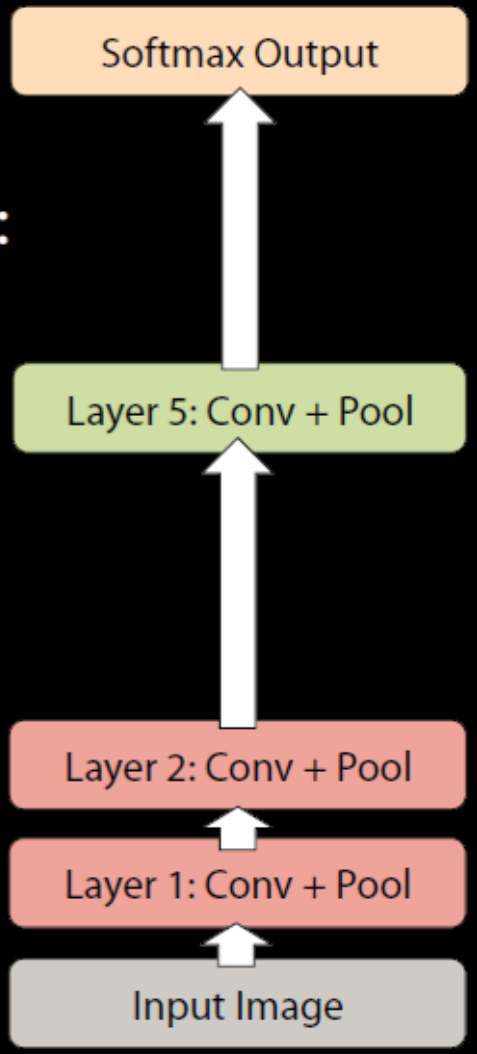
- Now try removing upper feature extractor layers:
  - Layers 3 & 4
- Drop ~1 million parameters
- 3.0% drop in performance



Why ConvNet should be Deep?

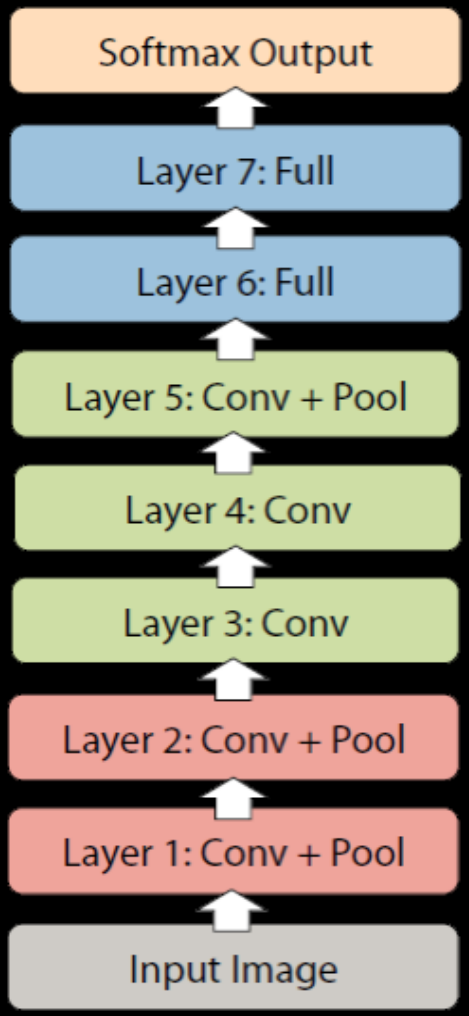
- Now try removing upper feature extractor layers & fully connected:
  - Layers 3, 4, 6 ,7
- Now only 4 layers
- 33.5% drop in performance

→Depth of network is key



Why ConvNet should be Deep?

- 8 layers total
- Trained on Imagenet dataset [Deng et al. CVPR'09]
- 18.2% top-5 error
- Our reimplementation:
  18.1% top-5 error



Rob Fergus, NIPS 2013

What is next important thing?

- How many filters?
- How to initialize weights?
  - Bias with Zero initialization
  - Weights with Random Initialization

Let us see in a bit more details

- What if w is initialized with large numbers?
- What if w is initialized with small numbers?

```
In [14]: x = torch.randn(512)
```

Let us see in a bit more details

- What if w is initialized with large numbers? - explosion
- What if w is initialized with small numbers? - vanishing

```
In [14]: x = torch.randn(512)
```

Let us see in a bit more details

- standard practice - when training neural networks to ensure that our inputs' values are scaled such that they fall inside such a normal distribution with a mean of 0 and a standard deviation of 1.

```
In [14]: x = torch.randn(512)
```

Extension: He initialization

- For ReLU activation, initialize weights as

$$\sqrt{\frac{2}{size^{[l-1]}}}$$

$$W^{[l]} = np.random.randn(size\_l, size\_l\text{-}1) * np.sqrt(2/size\_l\text{-}1)$$

Extension: Xavier initialization

- For tanh activation, initialize weights as

$$\sqrt{\frac{1}{size^{[l-1]}}}$$

$W^{[l]} = np.random.randn(size\_l, size\_l\text{-}1) * np.sqrt(1/size\_l\text{-}1)$

Benefits of He/Xavier Init

- They set the weights neither too much bigger than 1, nor too much less than 1. So, the gradients do not vanish or explode too quickly.
- They help avoid slow convergence, also ensuring that we do not keep oscillating off the minima.

Other tips and tricks

- Data level tricks
- Model level tricks
  - Network training
  - Model parameter tuning
  - Regularization
- Practice

Data Level Tricks

- Data augmentation
  - Original training data points: N
  - Augmented training data points: mN

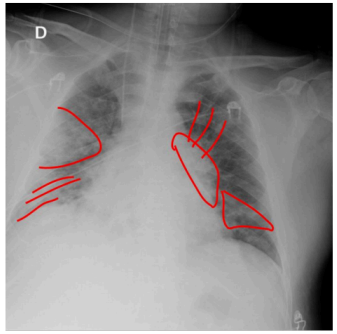
Data Level Tricks

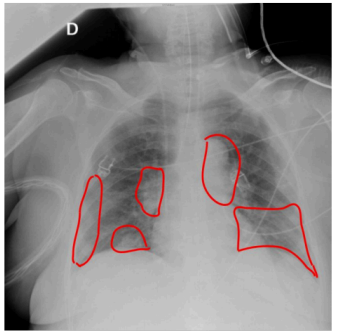
8N

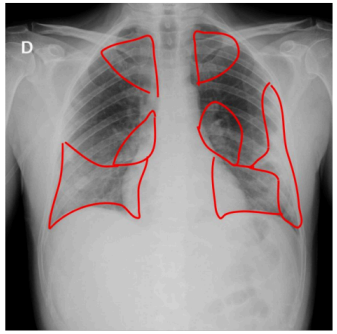
Original	Flip	Rotation	Random crop
			
• Image without any modification	• Flipped with respect to an axis for which the meaning of the image is preserved	• Rotation with a slight angle • Simulates incorrect horizon calibration	• Random focus on one part of the image • Several random crops can be done in a row

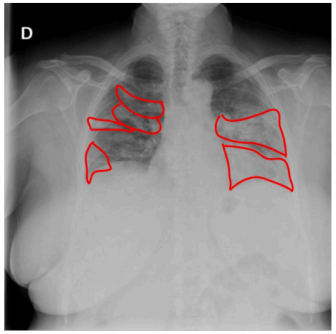
Color shift	Noise addition	Information loss	Contrast change
			
• Nuances of RGB is slightly changed • Captures noise that can occur with light exposure	• Addition of noise • More tolerance to quality variation of inputs	• Parts of image ignored • Mimics potential loss of parts of image	• Luminosity changes • Controls difference in exposition due to time of day

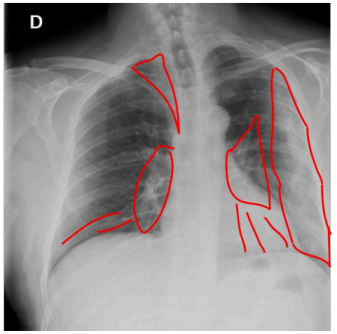
Medical images











Data Level Trick

- Batch size
  - Depending on the model size and memory, select a batch size
  - Batch size: total number of training examples present in a single batch.
  - Iterations is the number of batches needed to complete one epoch.
  - One Epoch is when an ENTIRE dataset is passed forward and backward through the neural network only ONCE.

Batch

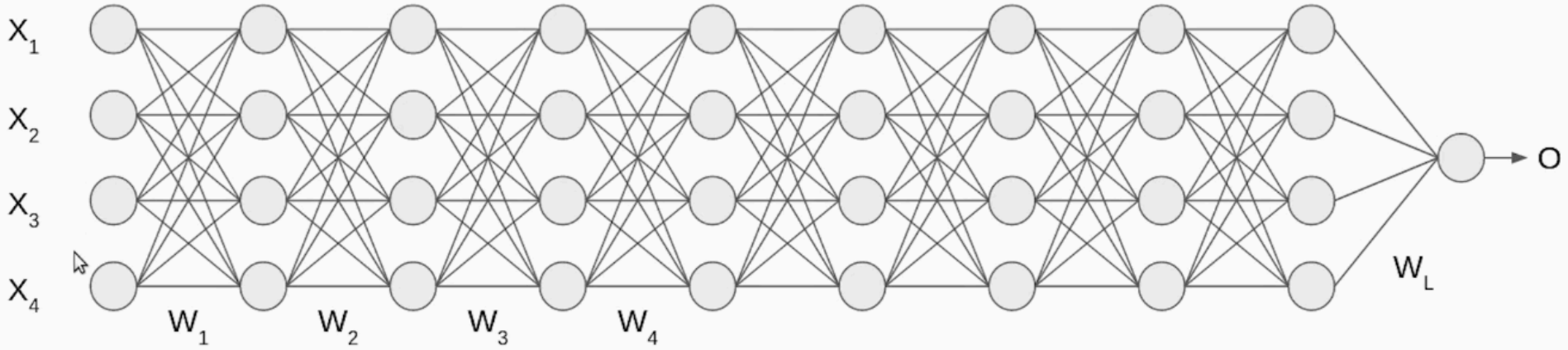
- We can divide the dataset of 2000 examples into batches of 500 then it will take 4 iterations to complete 1 epoch.

Mini Batch Gradient Descent

- During the training phase, updating weights is usually not based on the whole training set at once due to computation complexities or one data point due to noise issues.
- Instead, the update step is done on mini-batches, where the number of data points in a batch is a hyperparameter that we can tune.

Data Level Trick

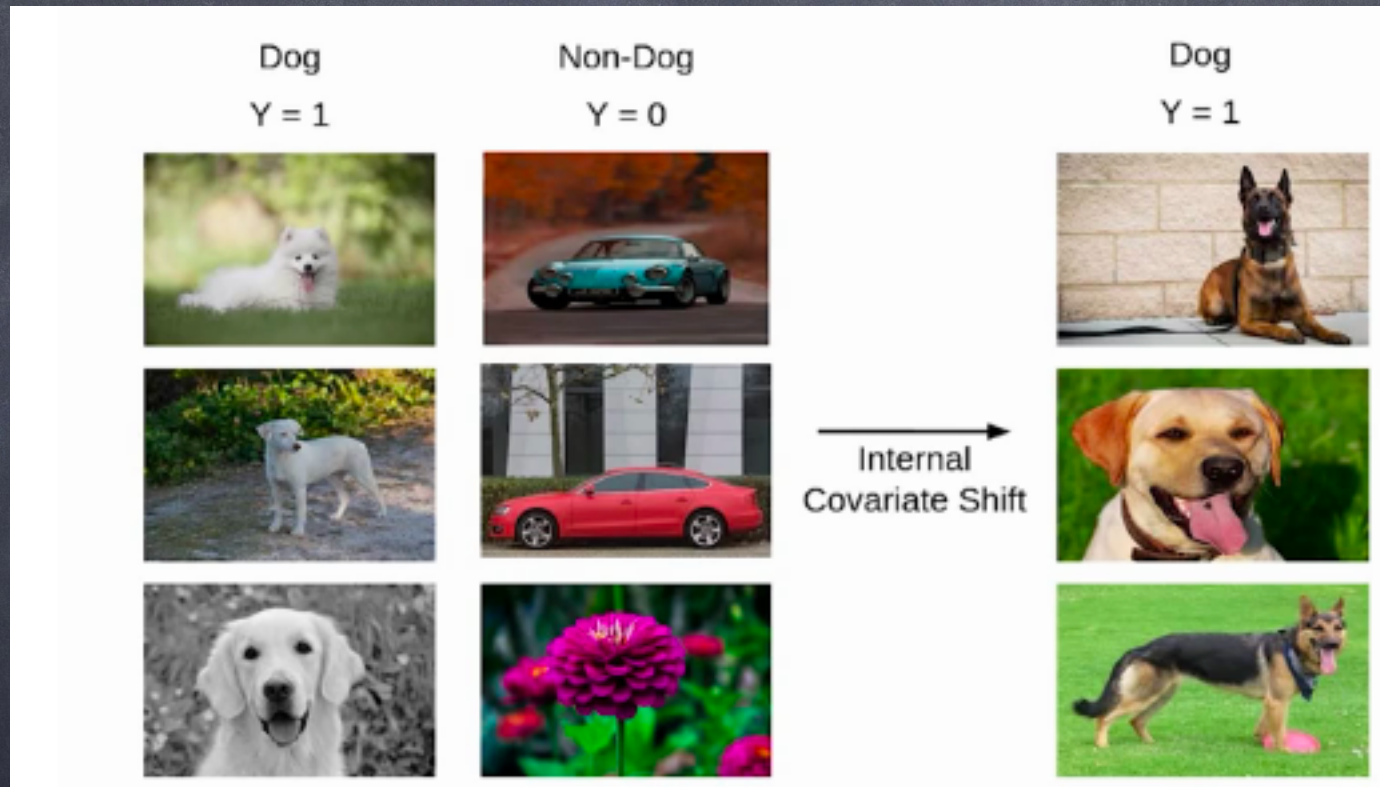
- Batch normalization
  - "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift" 2015



L = Number of layers

Bias = 0

Activation Function: Sigmoid


Dog
Y = 1
Non-Dog
Y = 0
Dog
Y = 1
Internal
Covariate Shift

Batch Norm

❒ **Batch normalization** — It is a step of hyperparameter γ, β that normalizes the batch $\{x_i\}$. By noting μ_B, σ_B^2 the mean and variance of that we want to correct to the batch, it is done as follows:

$$\boxed{x_i \longleftarrow \gamma \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} + \beta}$$

It is usually done after a fully connected/convolutional layer and before a non-linearity layer and aims at allowing higher learning rates and reducing the strong dependence on initialization.

Model Level Tricks

- Network training
- Model parameter tuning
- Regularization

Optimization Convergence

- Learning rate
  - The learning rate, α or η, indicates at which pace the weights get updated. It can be fixed or adaptively changed.
- Adaptive learning rate
  - Letting the learning rate vary when training a model can reduce the training time and improve the numerical optimal solution.

Gradient Descent

- Gradient descent is an iterative algorithm, that starts from a random point on a function and travels down its slope in steps until it reaches the lowest point of that function
  - New = old + gradient * learning rate

Stochastic Gradient Descent

- Motivation
  - 10,000 data points and 10 features
  - compute the derivative with respect to each of the features
    - 10000 * 10 = 100,000 computations
    - If 1000 epochs, then
    - 100,000 * 1000 = 100000000 computations

gradient descent is slow on huge data

Stochastic Gradient Descent

- "Stochastic", in plain terms means "random"
  - sample a small number of data points
  - "mini-batch" gradient descent

Adding Momentum

$$\Delta w_{ij} = (\eta * \frac{\partial E}{\partial w_{ij}})$$

weight increment

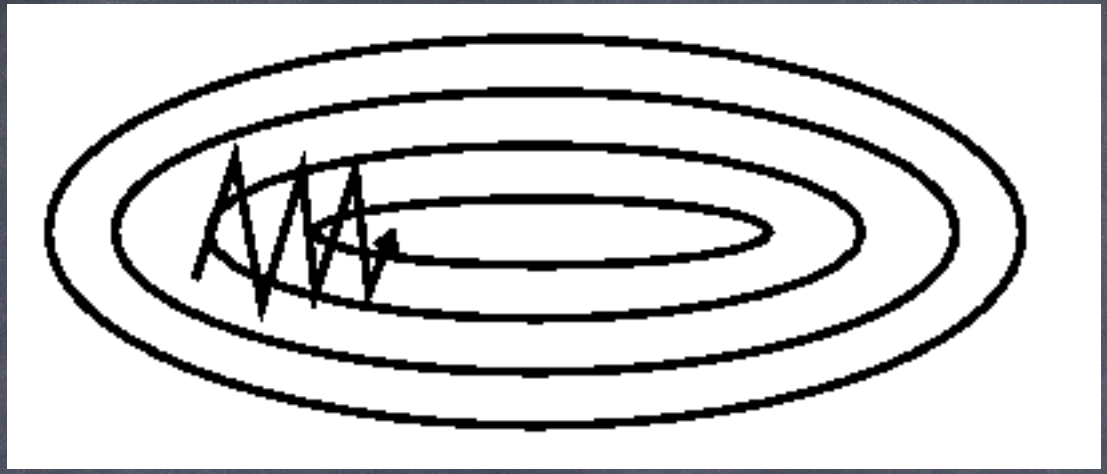
learning rate

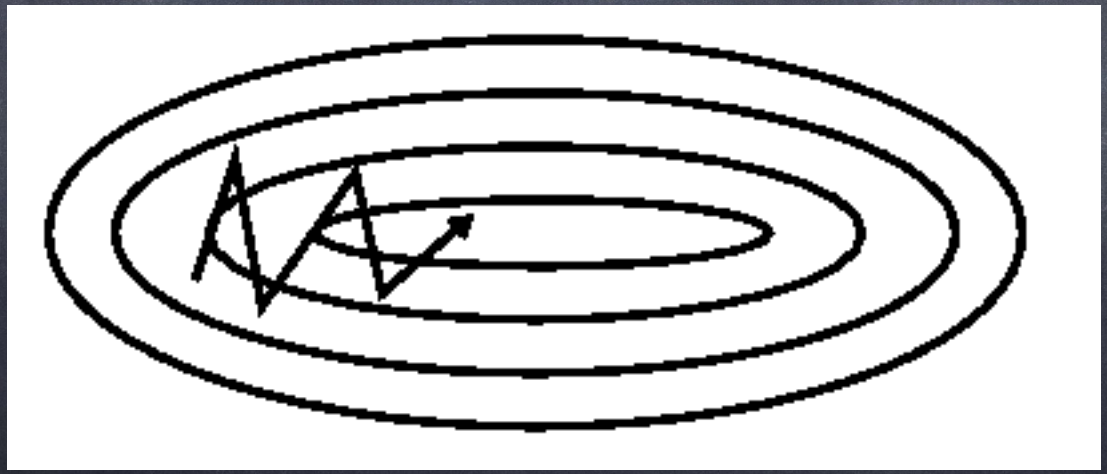
weight gradient

$$\Delta w_{ij} = (\eta * \frac{\partial E}{\partial w_{ij}}) + (\gamma * \Delta w_{ij}^{t-1})$$

momentum factor

weight increment, previous iteration





Why only one previous gradient?

$$g_{a,1} = \beta g_{a,1} + (1-\beta) g_{a,1} = g_{a,1}$$

$$g_{a,2} = \beta g_{a,1} + (1-\beta) g_{a,2}$$

$$g_{a,3} = \beta g_{a,2} + (1-\beta) g_{a,3} = \beta^2 g_{a,1} + \beta(1-\beta) g_{a,2} + (1-\beta) g_{a,3}$$

$$g_{a,4} = \beta g_{a,3} + (1-\beta) g_{a,4} = \beta^3 g_{a,1} + \beta^2(1-\beta) g_{a,2} + \beta(1-\beta) g_{a,3} + (1-\beta) g_{a,4}$$

RMSProp

For each Parameter w^j

(*j subscript dropped for clarity*)

$$\nu_t = \rho \nu_{t-1} + (1 - \rho) * g_t^2$$

$$\Delta \omega_t = -\frac{\eta}{\sqrt{\nu_t + \epsilon}} * g_t$$

$$\omega_{t+1} = \omega_t + \Delta \omega_t$$

η : *Initial Learning rate*

ν_t : *Exponential Average of squares of gradients*

g_t : *Gradient at time t along* ω^j

ADAM

- "... the name Adam is derived from adaptive moment estimation"

"Adam: A Method for Stochastic Optimization"

ADAM

For each Parameter w^j

(*j subscript dropped for clarity*)

$$\nu_t = \beta_1 * \nu_{t-1} + (1 - \beta_1) * g_t$$

$$s_t = \beta_2 * s_{t-1} + (1 - \beta_2) * g_t^2$$

$$\Delta\omega_t = -\eta \frac{\nu_t}{\sqrt{s_t + \epsilon}} * g_t$$

$$\omega_{t+1} = \omega_t + \Delta\omega_t$$

η : *Initial Learning rate*

g_t : *Gradient at time t along* ω^j

ν_t : *Exponential Average of gradients along* ω_j

s_t : *Exponential Average of squares of gradients along* ω_j

β_1, β_2 : *Hyperparameters*

- eta. Also referred to as the learning rate or step size. The proportion that weights are updated (e.g. 0.001). Larger values (e.g. 0.3) results in faster initial learning before the rate is updated. Smaller values (e.g. 1.0E-5) slow learning right down during training
- beta1. The exponential decay rate for the first moment estimates (e.g. 0.9).
- beta2. The exponential decay rate for the second-moment estimates (e.g. 0.999). This value should be set close to 1.0 on problems with a sparse gradient (e.g. NLP and computer vision problems).
- epsilon. Is a very small number to prevent any division by zero in the implementation (e.g. 10E-8).

Many other optimizers

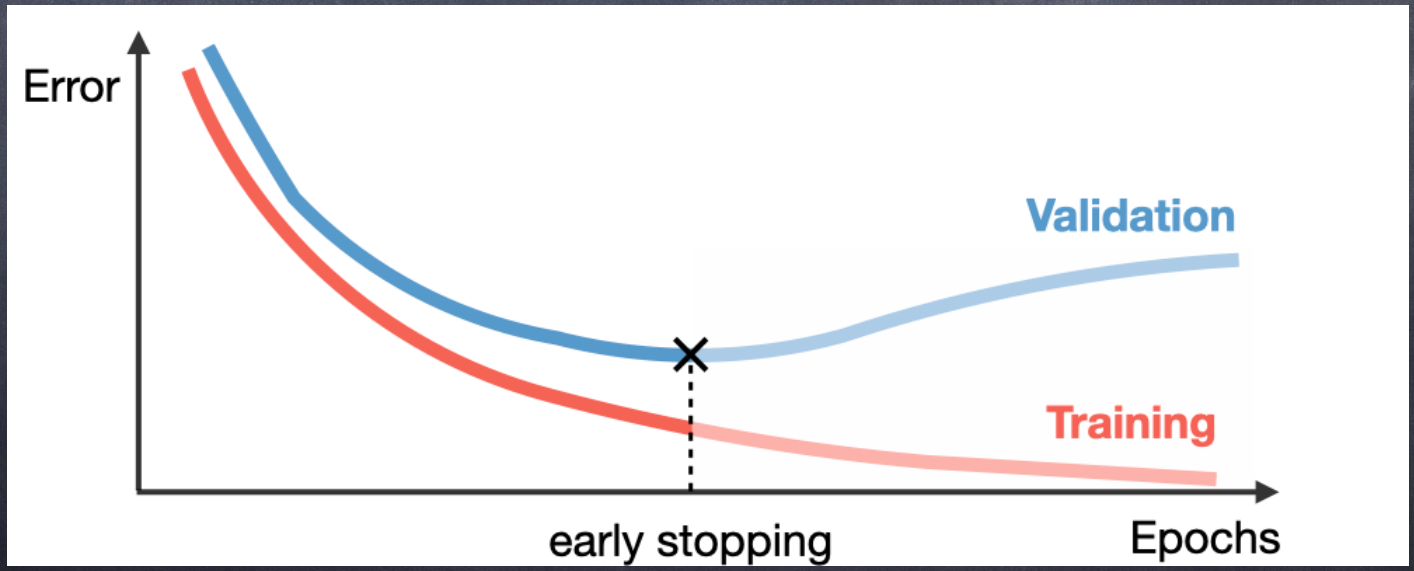
- Mostly Heuristics

Regularizations

- We have discussed
  - L1
  - L2
  - Dropout and Dropconnect

Early Stopping

Early stopping This regularization technique stops the training process as soon as the validation loss reaches a plateau or starts to increase.



Loss Functions

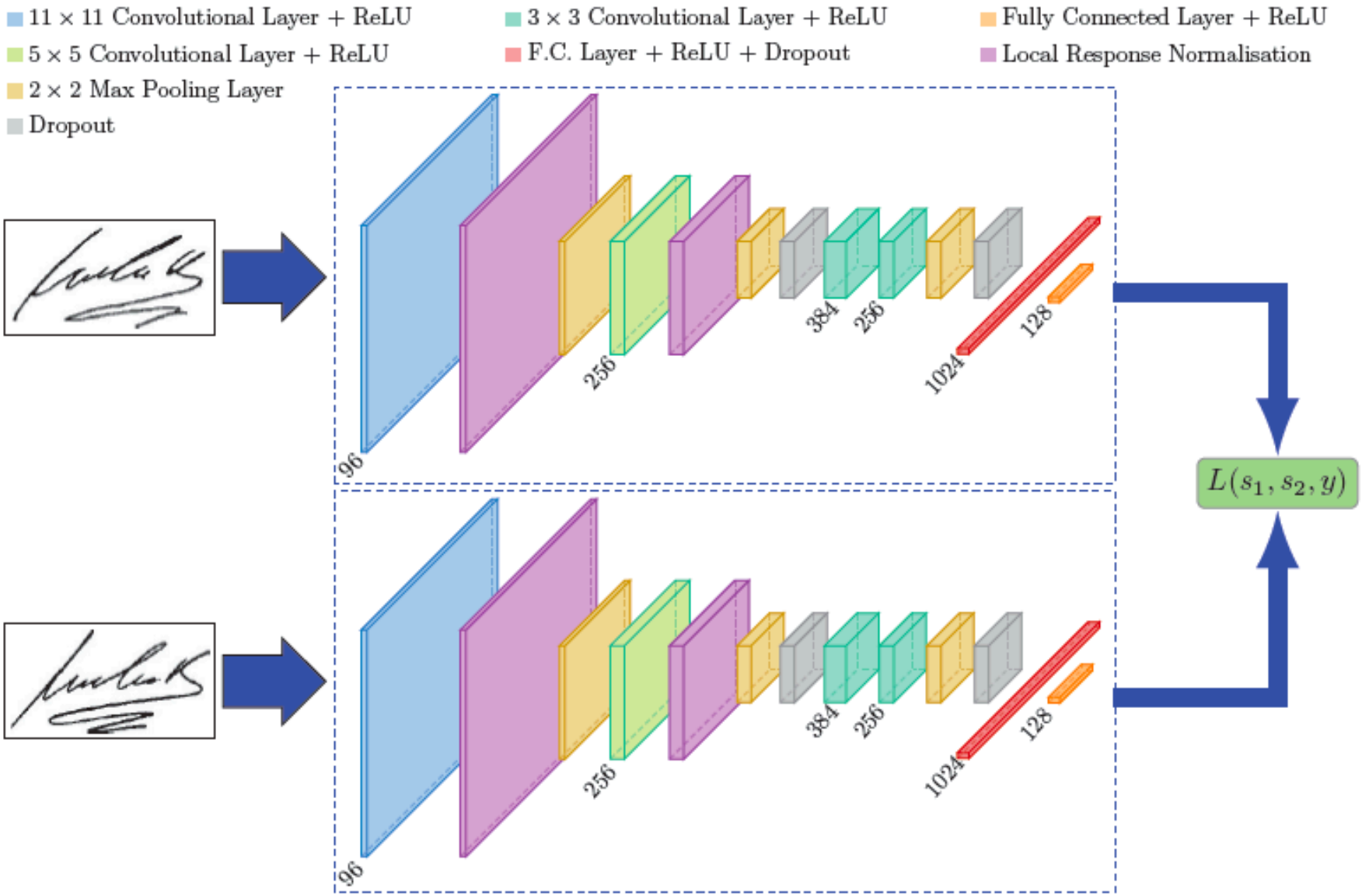
- There are different loss functions
  - We have studied a couple only
  - In the context of binary classification in neural networks, the cross-entropy loss L(z,y) is commonly used and is defined as follows:

$$L(z, y) = -\Big[y \log(z) + (1 - y) \log(1 - z)\Big]$$

What are other loss functions?

- What is the role of loss function?

Siamese Network



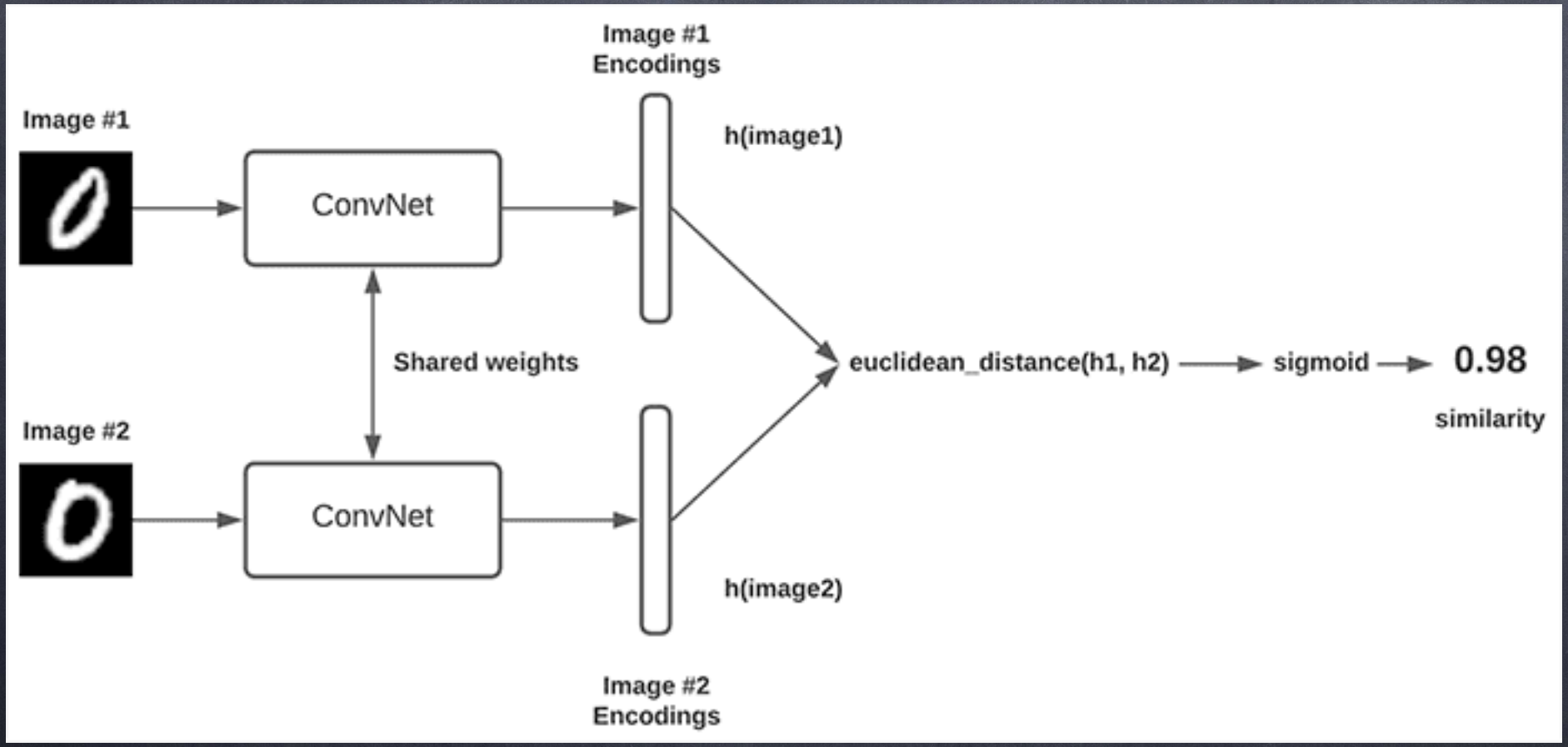
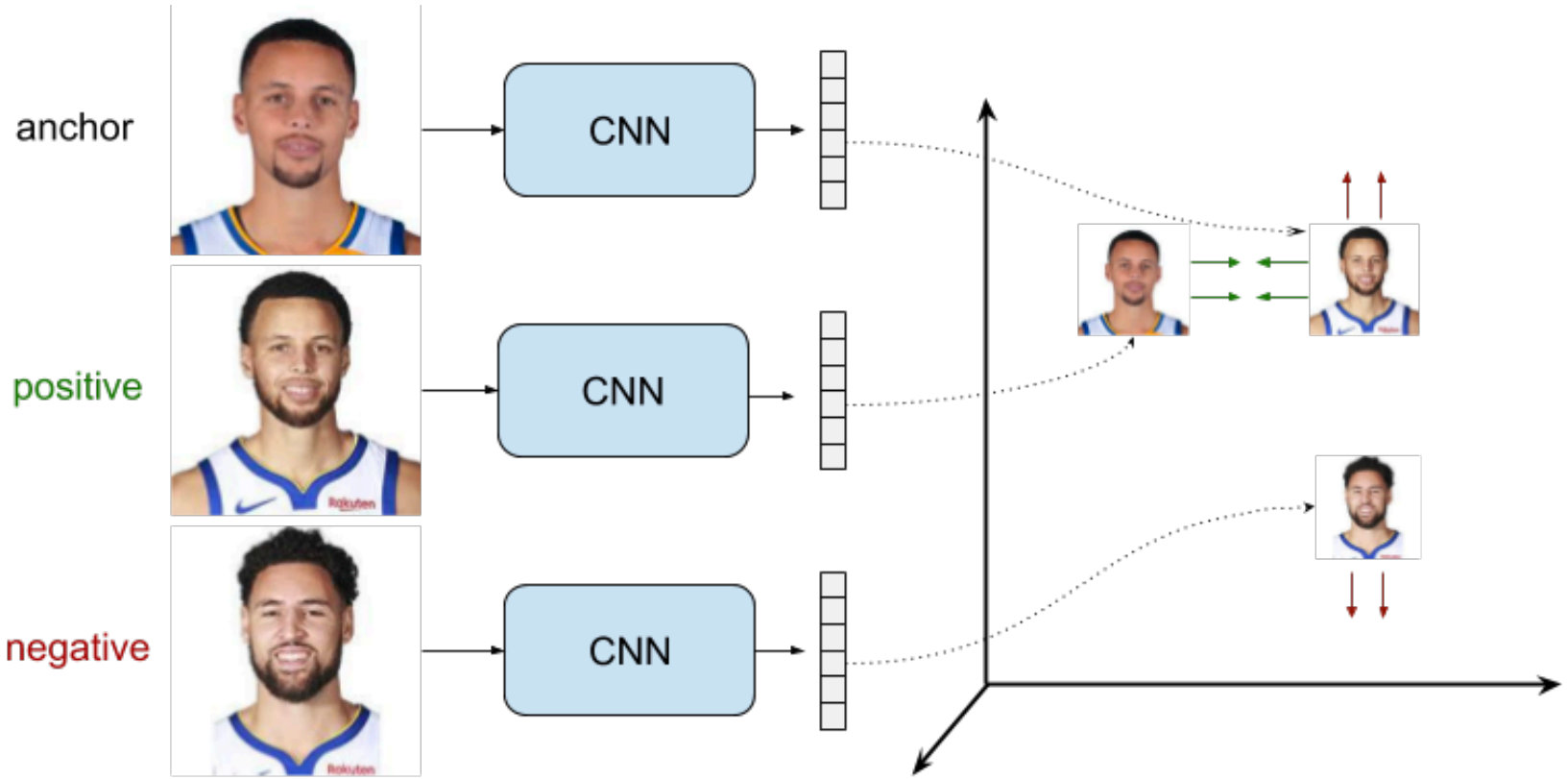
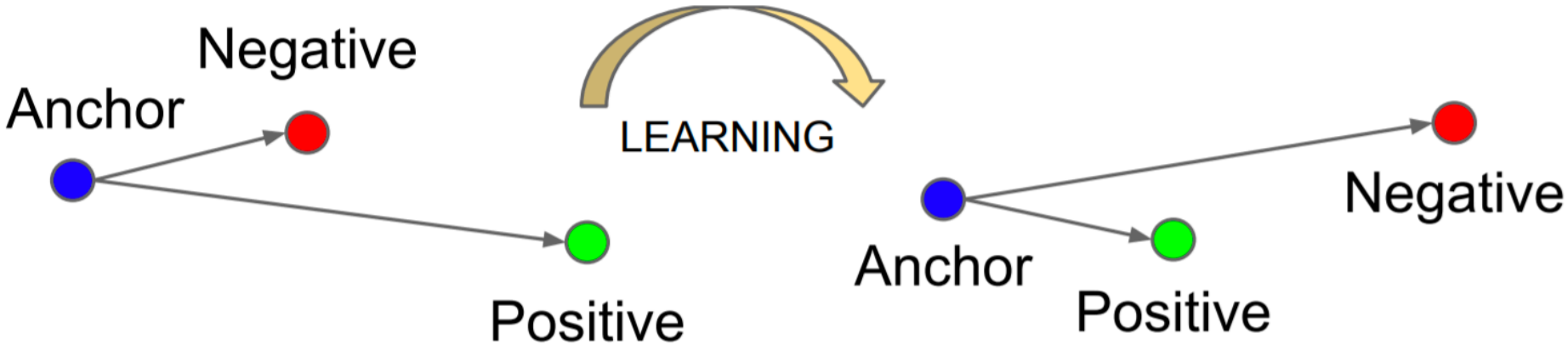


Image #1
Encodings
Image #1
h(image1)
ConvNet
Shared weights
euclidean_distance(h1, h2)
sigmoid
0.98
similarity
Image #2
ConvNet
h(image2)
Image #2
Encodings

Lets try to understand inter-class and intra-class concept



Triplet Loss

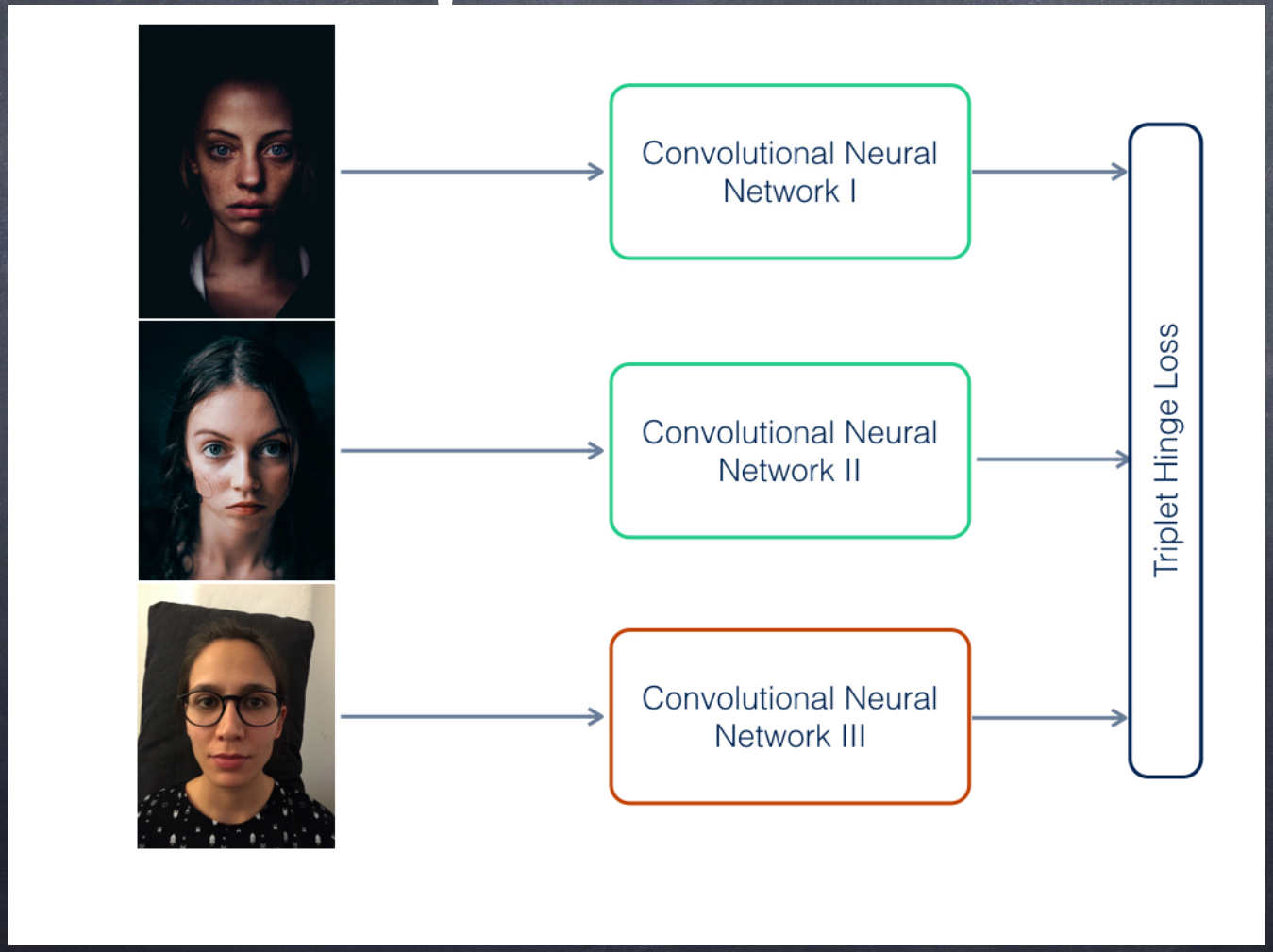


$$Loss = \sum_{i=1}^{N} \left[\| f_i^a - f_i^p \|_2^2 - \| f_i^a - f_i^n \|_2^2 + \alpha \right]_+$$

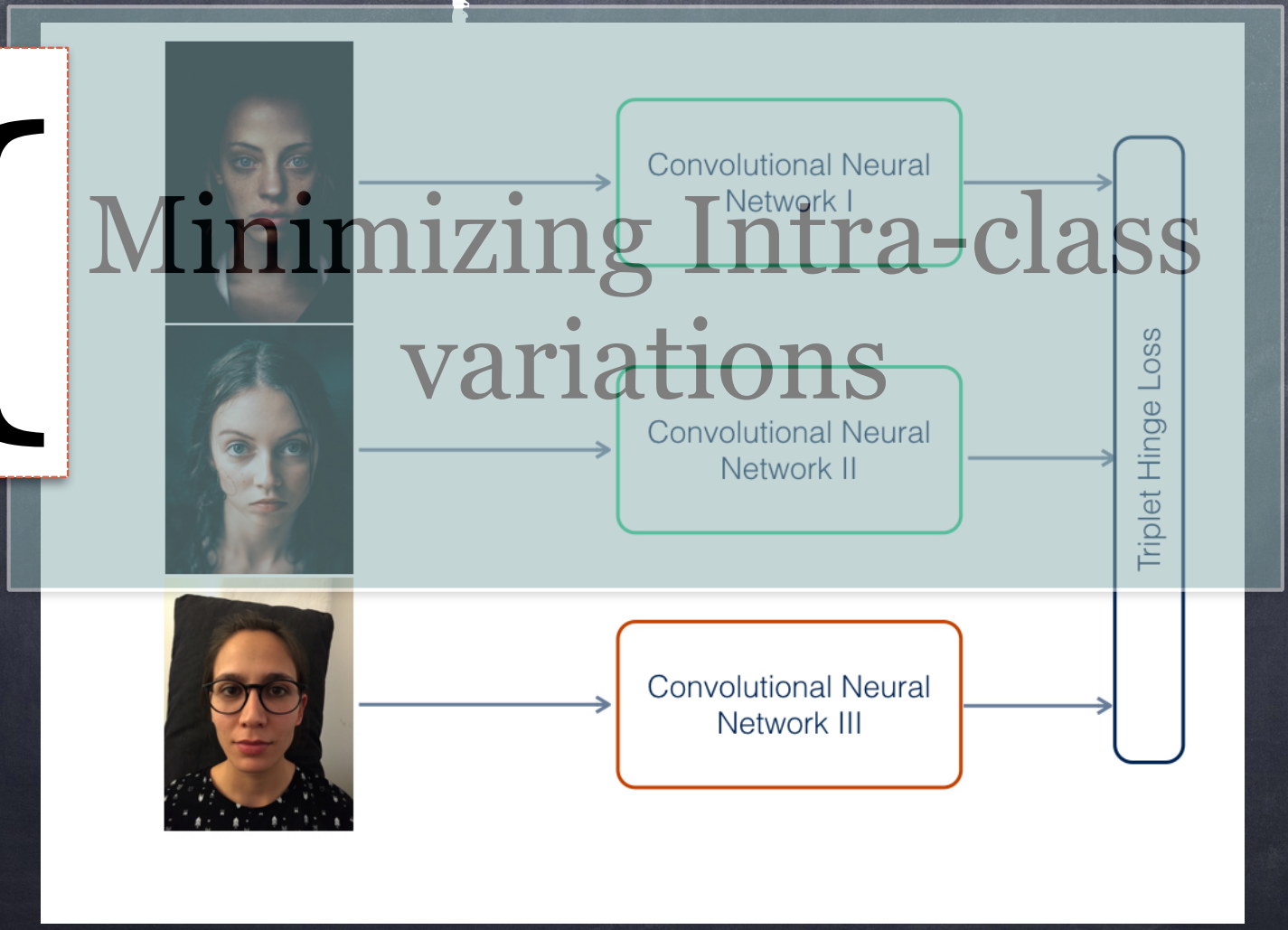
$$\| f_i^a - f_i^p \|_2^2$$

$$- \| f_i^a - f_i^n \|_2^2 \,.$$

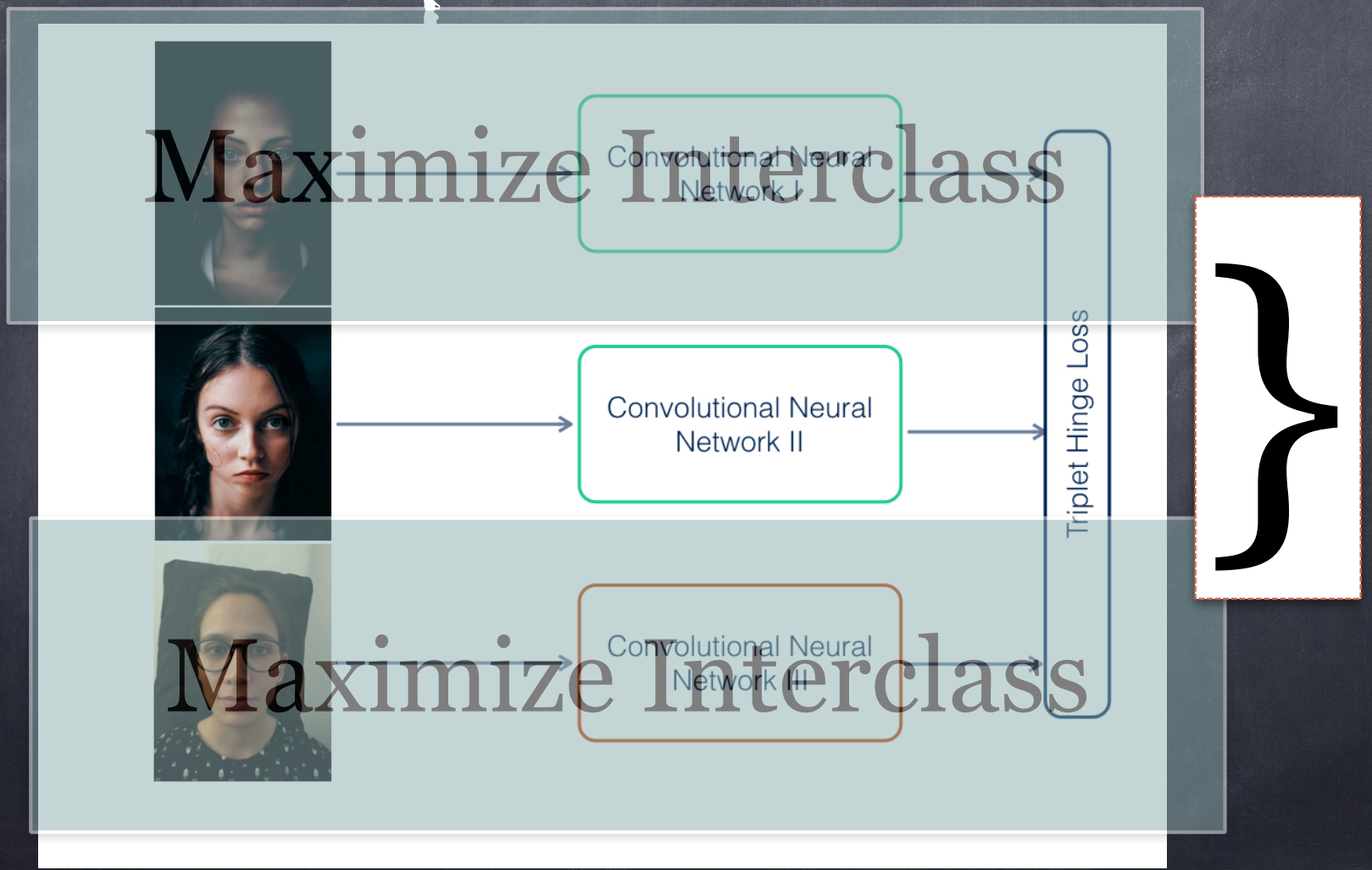
Let us understand Triplet Loss



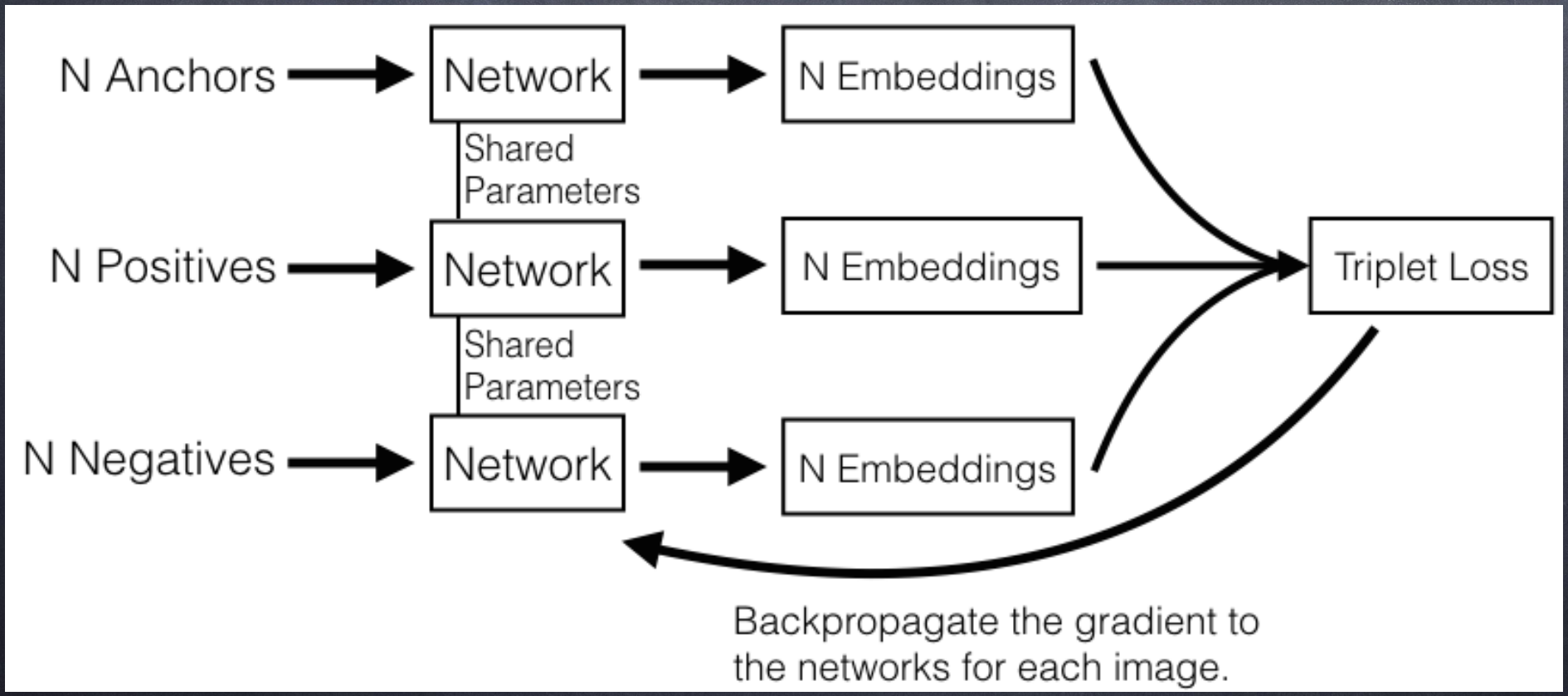
Let us understand Triplet Loss



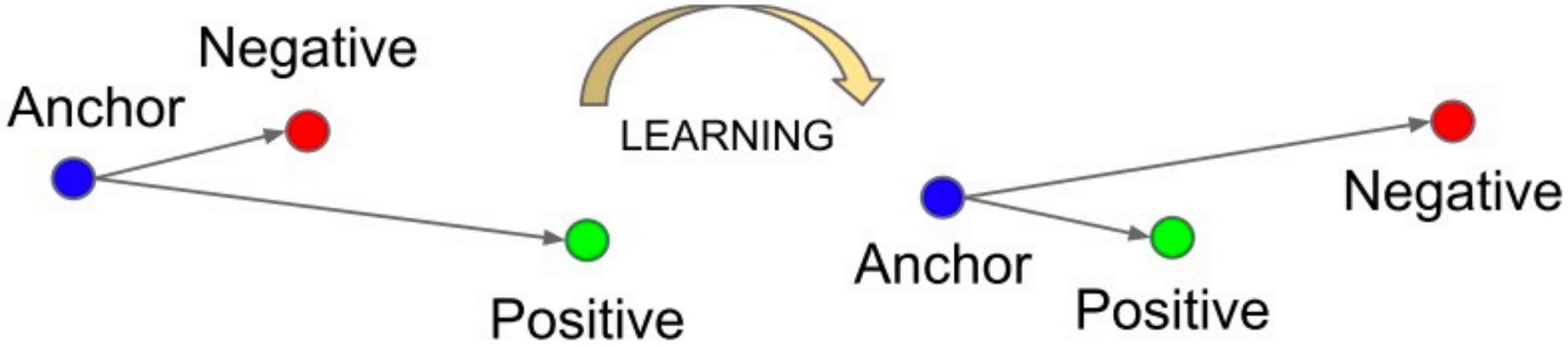
Let us understand Triplet Loss



Training Triplet Loss

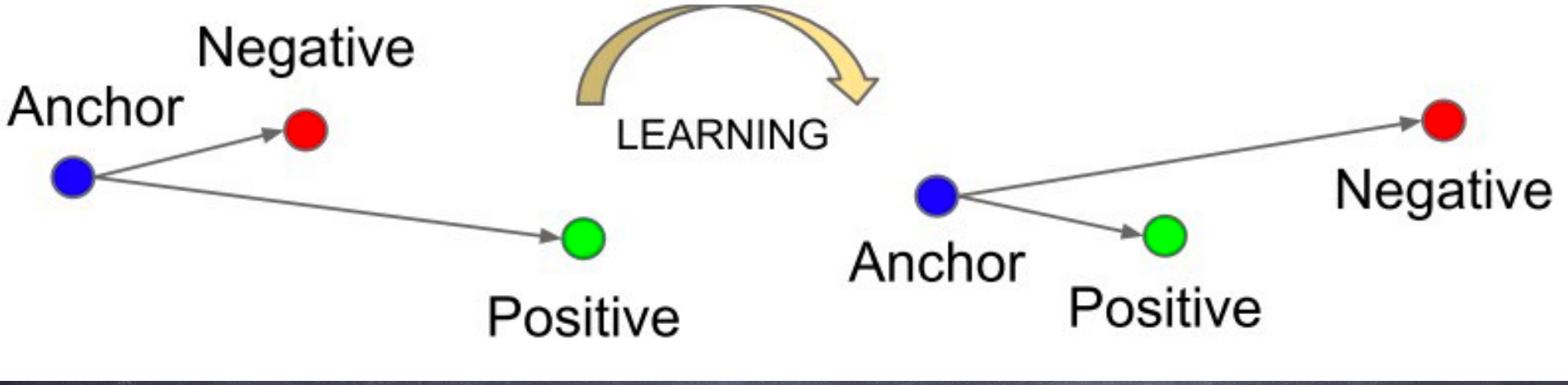


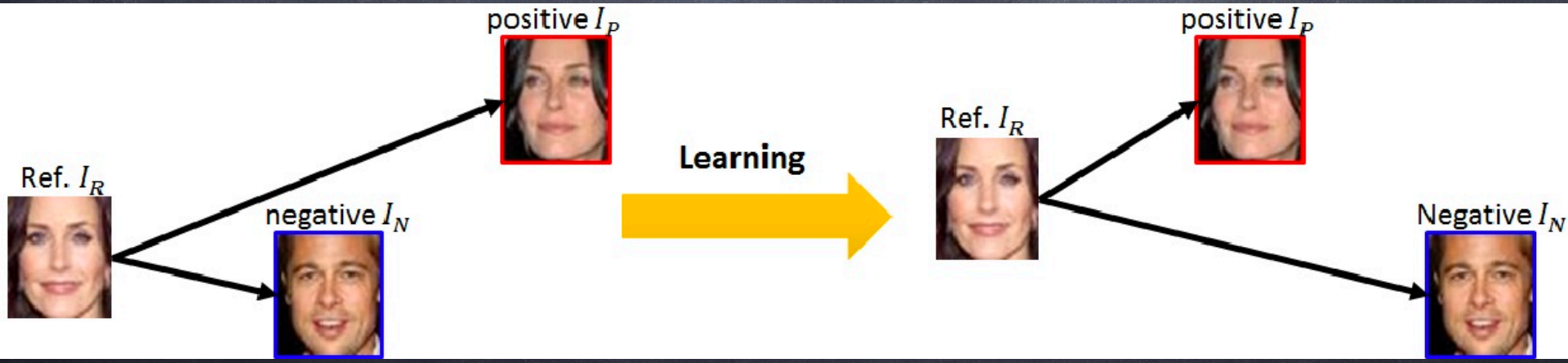
Triplet Loss



$$(||F(A)-F(P)||+\text{margin}) < ||F(A)-F(N)||$$

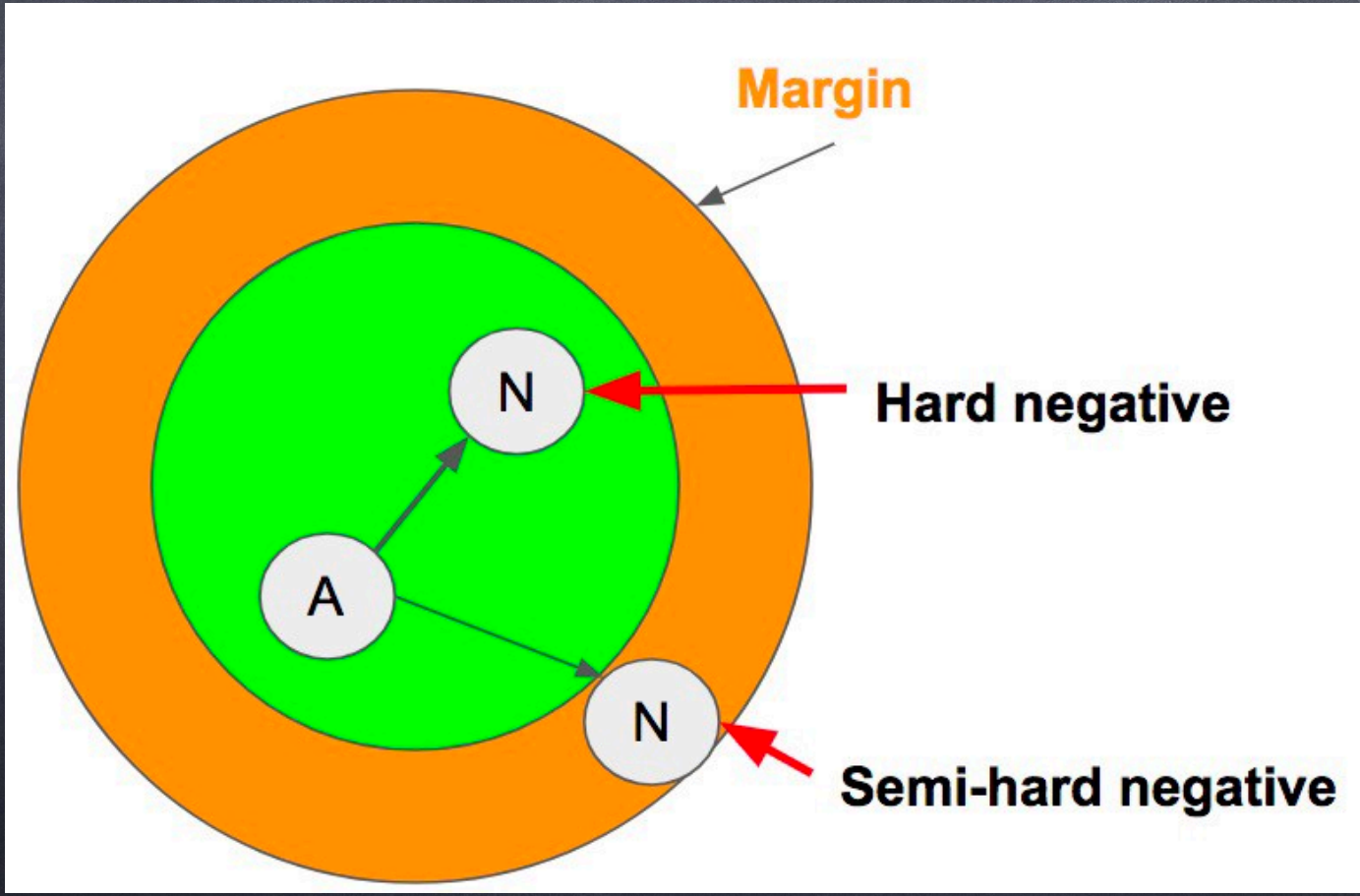
Triplet Loss





$$(||F(A)-F(P)||+\text{margin}) < ||F(A)-F(N)||$$

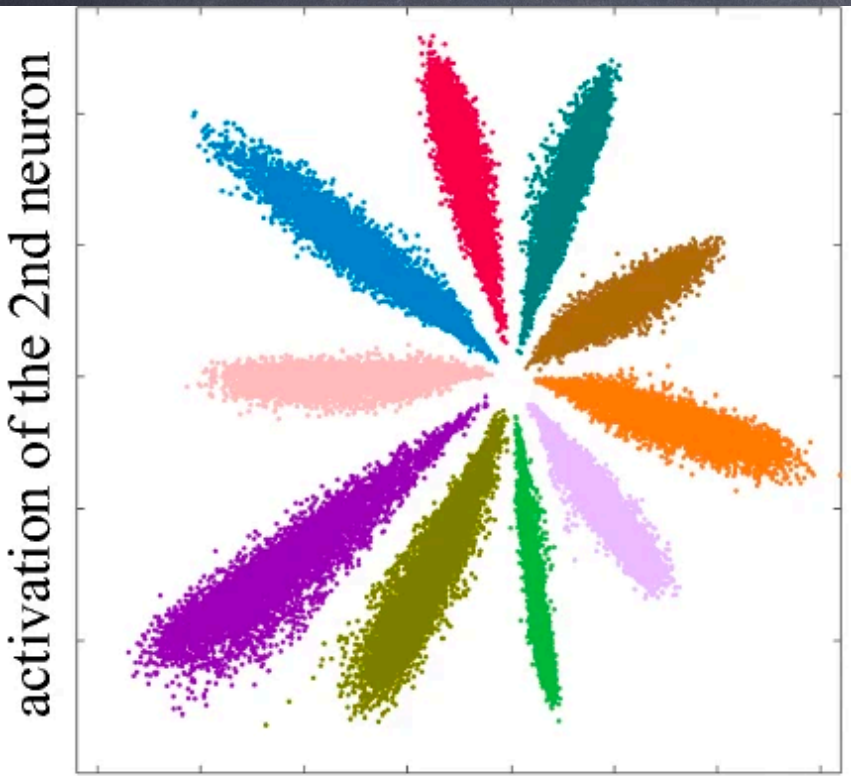
How to select triplet (A, P, N)?



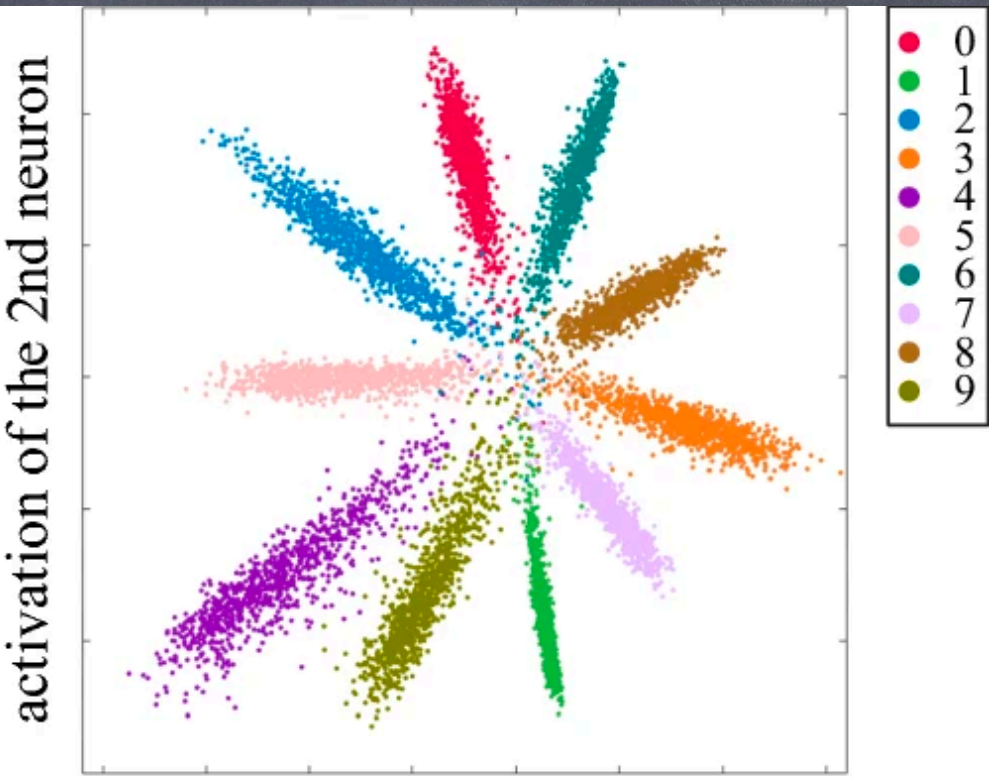
Is it all?

- Do you see any other possibility to further improve it?

Let us take a look at toy example



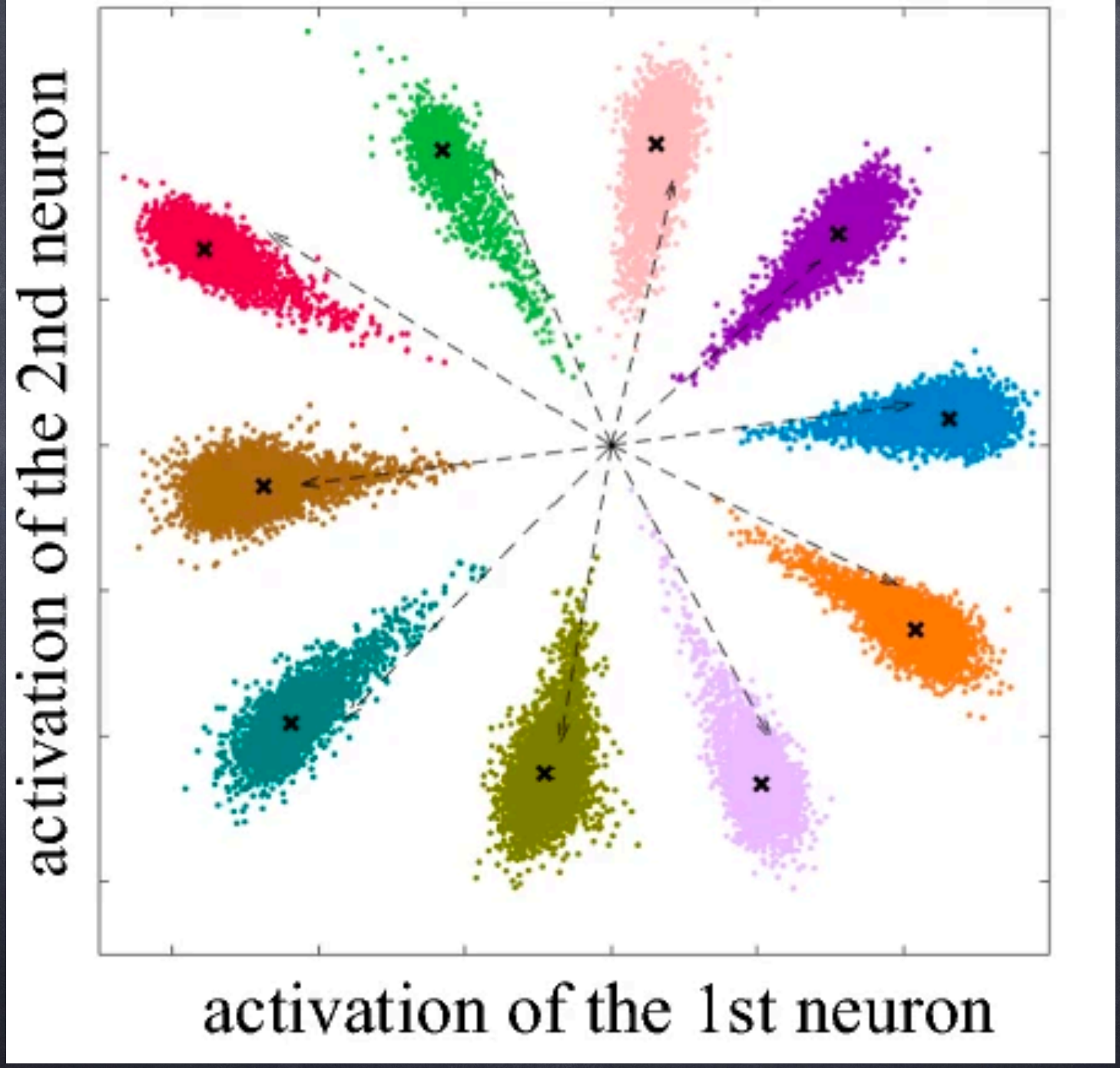
(a) training examples

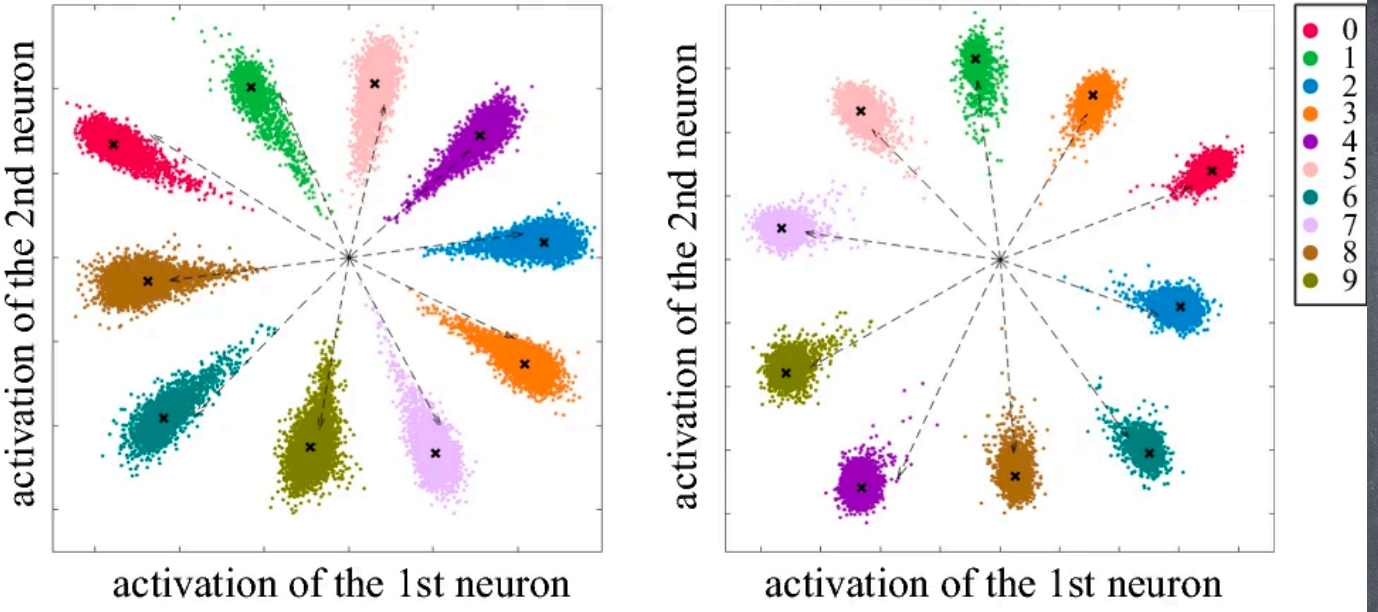


(b) testing examples

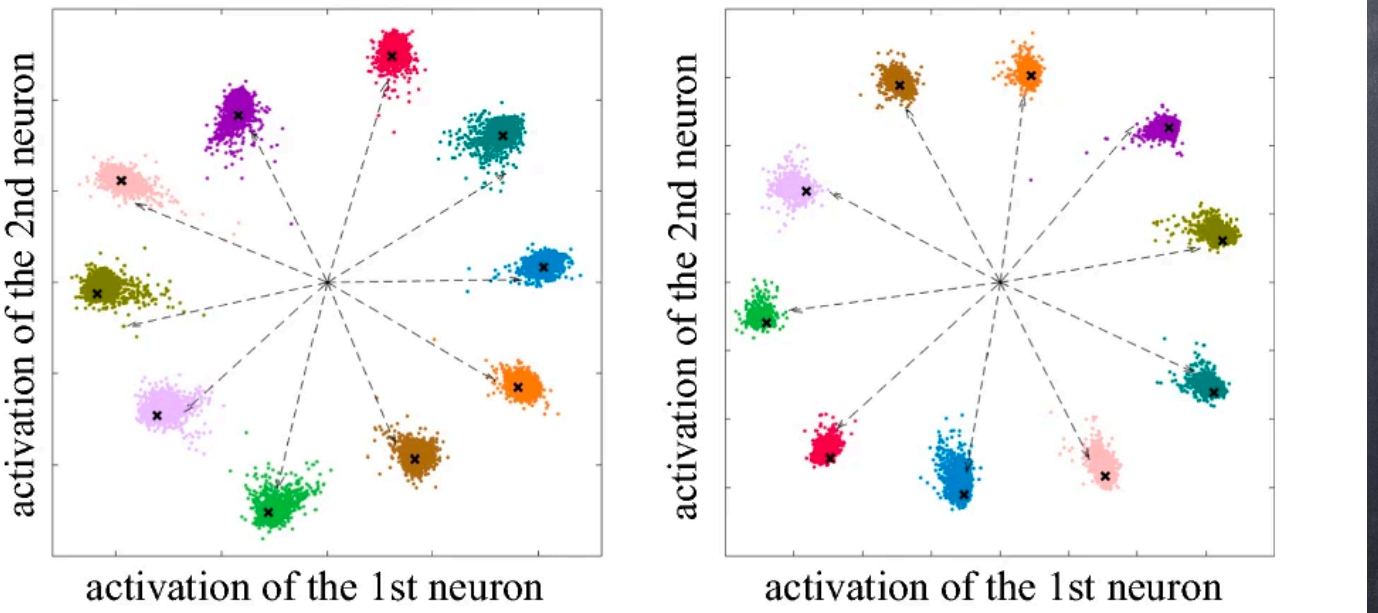
Center Loss

$$\mathcal{L}_C = \frac{1}{2m} \sum_{i=1}^{m} \| \boldsymbol{x}_i - \boldsymbol{c}_{y_i} \|_2^2,$$


activation of the 2nd neuron
activation of the 1st neuron



(a) $\lambda = 0.001$ **(b)** $\lambda = 0.01$



(c) $\lambda = 0.1$ **(d)** $\lambda = 1$