



Deep Learning for Vision & Language

Computer Vision II: Convolutional Neural Network Architectures

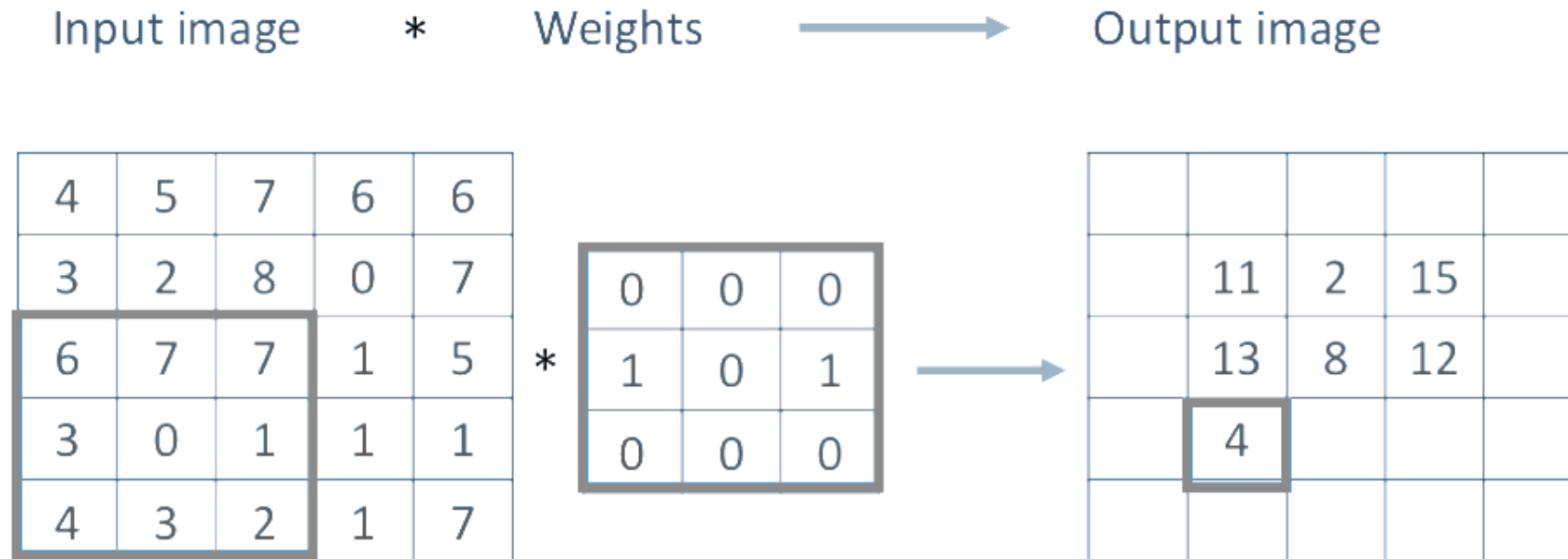


Assignment 1

- Due Monday at midnight (No extension for any reason but especially not due to errors/outages in Google Colab)
 - Everyone should have a version submitted on canvas by now.
Maybe you should have one or two questions left to complete.
- If that's not the case – then accept right now that you probably will not be able to complete the assignment and that's fine. You can make sure you finish the following assignments.

Most important operation for Computer Vision

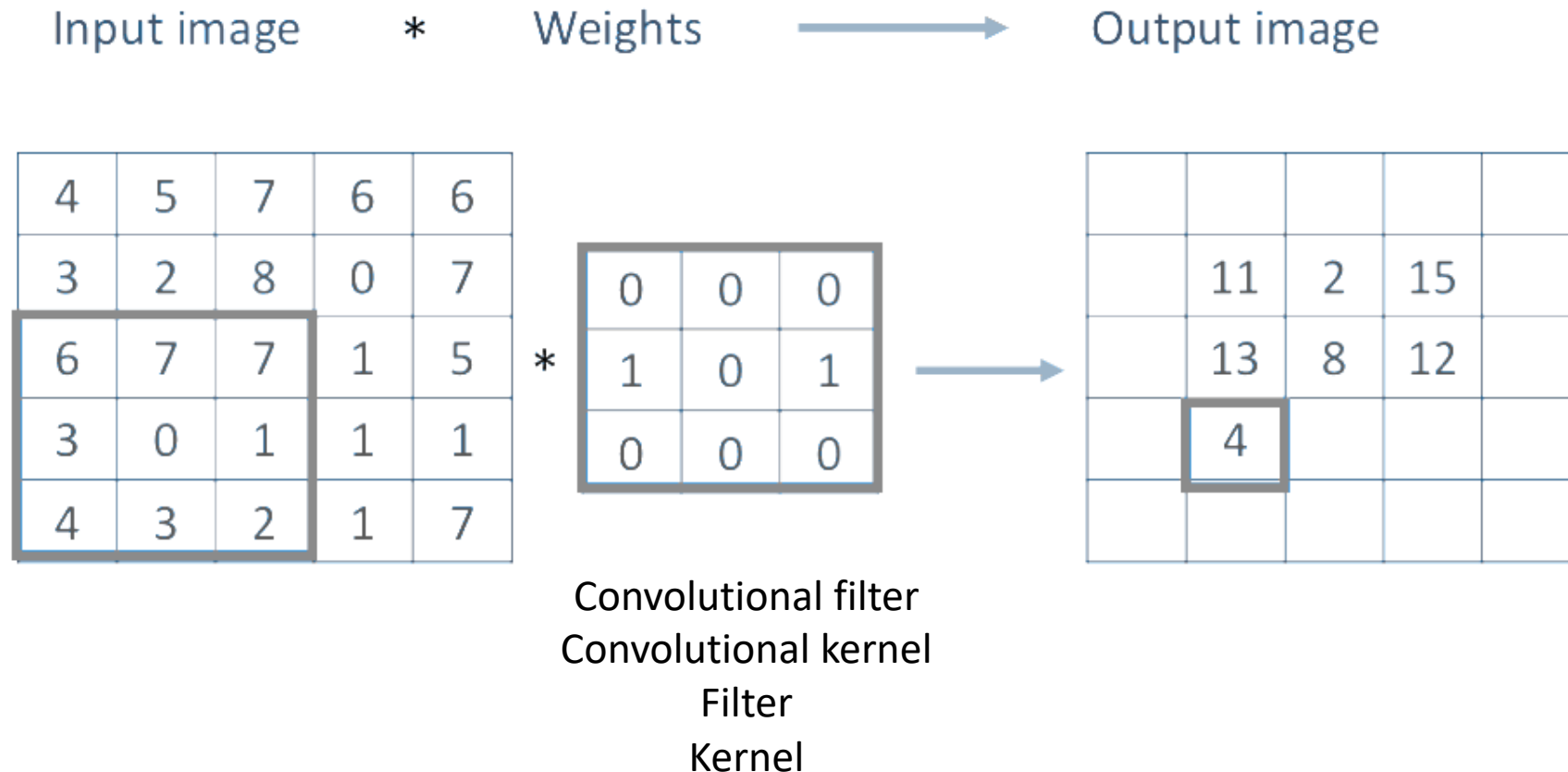
- The Convolution Operation



<https://www.cs.rice.edu/~vo9/recognition/animation.gif>

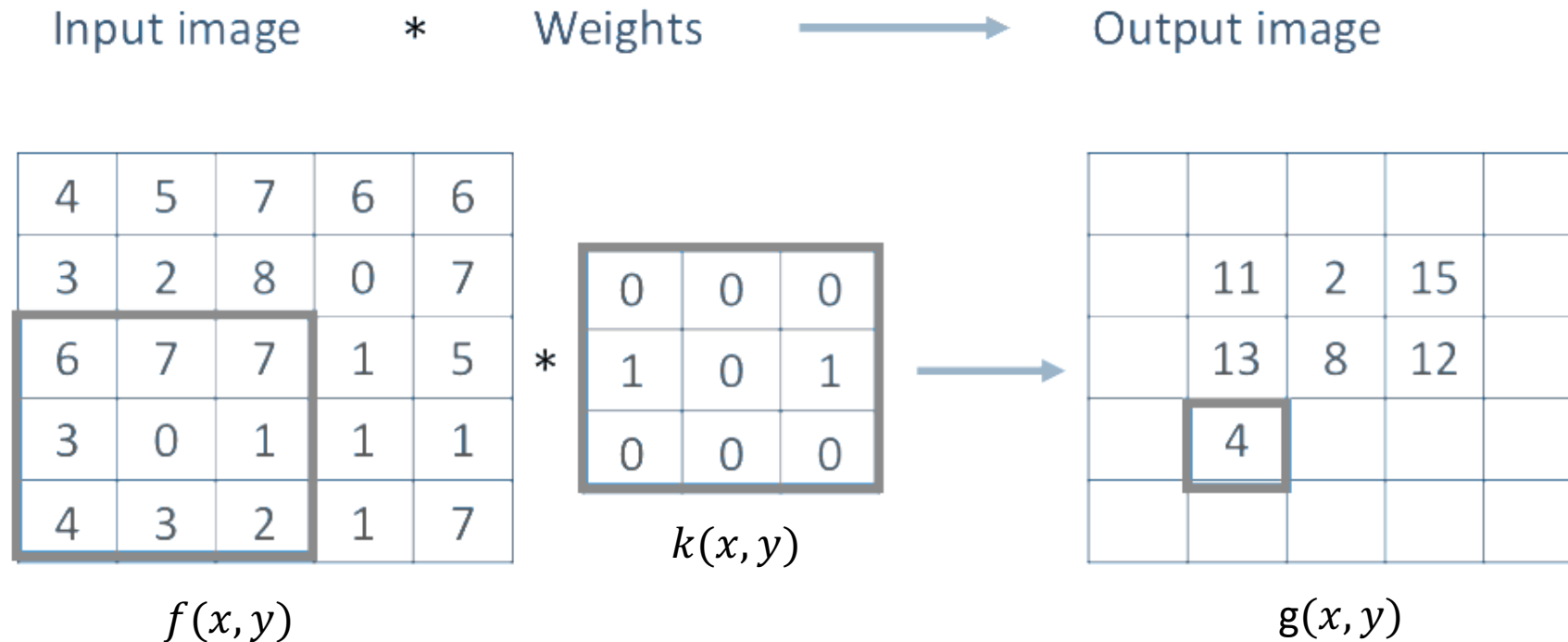
Most important operation for Computer Vision

- The Convolution Operation



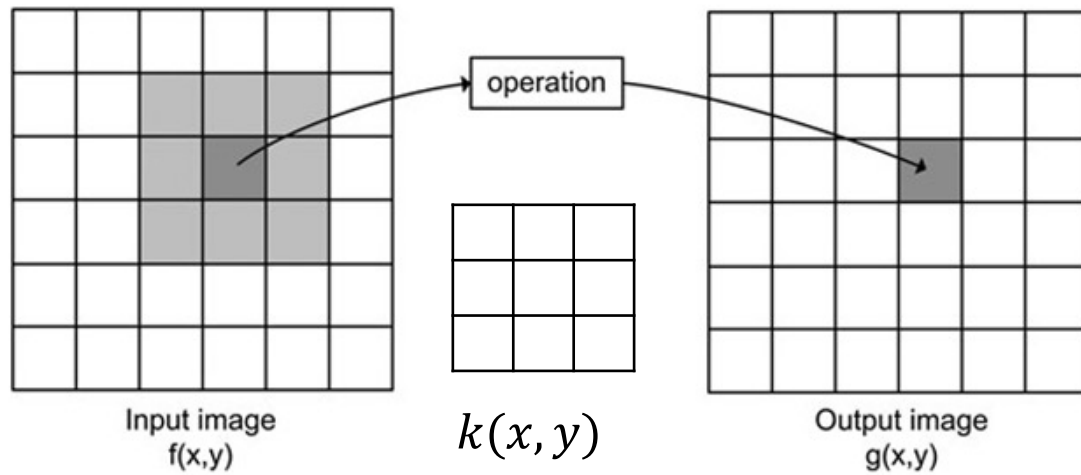
Most important operation for Computer Vision

- The Convolution Operation



$$g(x, y) = \sum_v \sum_u k(u, v) f(x - u, y - v)$$

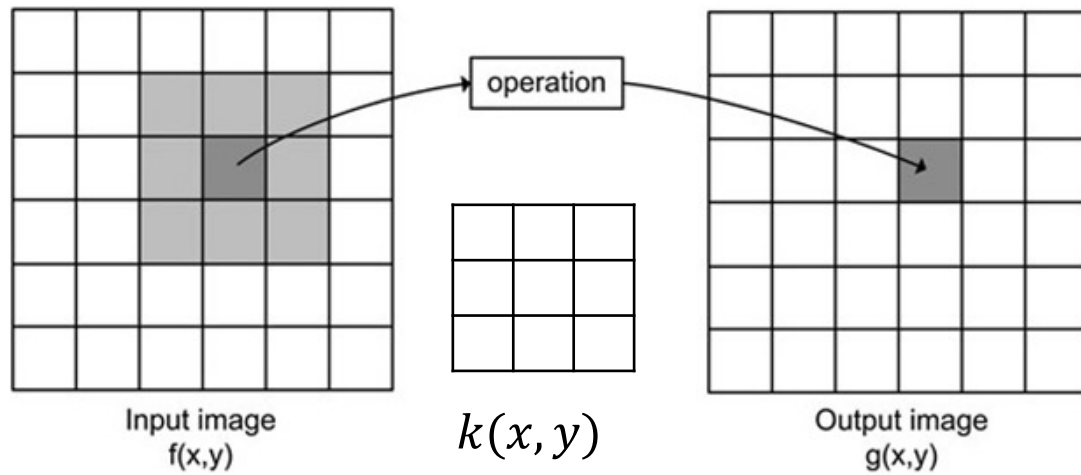
Image filtering: Convolution operator e.g. mean filter



$$k(x,y) =$$

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Image filtering: Convolution operator e.g. mean filter



$$k(x,y) =$$

1/9	1/9	1/9
1/9	1/9	1/9
1/9	1/9	1/9

Example: box filter

$g[\cdot, \cdot]$

	1	1	1
1	1	1	1
9	1	1	1

Image filtering

$$g[\cdot, \cdot] \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

$f[\cdot, \cdot]$

0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0	0
0	0	0	90	90	90	90	90	0	0	0
0	0	0	90	90	90	90	90	0	0	0
0	0	0	90	0	90	90	90	0	0	0
0	0	0	90	90	90	90	90	0	0	0
0	0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0

$h[\cdot, \cdot]$

$$h[m, n] = \sum_{k, l} g[k, l] f[m + k, n + l]$$

Image filtering

$$g[\cdot, \cdot] \frac{1}{9}$$

1	1	1
1	1	1
1	1	1

$f[\cdot, \cdot]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$h[\cdot, \cdot]$

	0	10							

$$h[m, n] = \sum_{k, l} g[k, l] f[m + k, n + l]$$

Image filtering

$$g[\cdot, \cdot] \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

$f[\cdot, \cdot]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$h[\cdot, \cdot]$

	0	10	20						

$$h[m, n] = \sum_{k, l} g[k, l] f[m + k, n + l]$$

Image filtering

$$g[\cdot, \cdot] \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

$f[\cdot, \cdot]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$h[\cdot, \cdot]$

	0	10	20	30					

$$h[m, n] = \sum_{k, l} g[k, l] f[m + k, n + l]$$

Image filtering

$$g[\cdot, \cdot] \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

$f[\cdot, \cdot]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$h[\cdot, \cdot]$

	0	10	20	30	30				

$$h[m, n] = \sum_{k, l} g[k, l] f[m + k, n + l]$$

Image filtering

$$g[\cdot, \cdot] \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

$f[\cdot, \cdot]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$h[\cdot, \cdot]$

	0	10	20	30	30				

$$h[m, n] = \sum_{k, l} g[k, l] f[m + k, n + l]$$

Image filtering

$$g[\cdot, \cdot] = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

$f[\cdot, \cdot]$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$h[\cdot, \cdot]$

	0	10	20	30	30				
							?		
				50					

$$h[m, n] = \sum_{k, l} g[k, l] f[m+k, n+l]$$

Image filtering

$$g[\cdot, \cdot] = \frac{1}{9} \begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

$$f[\cdot, \cdot]$$

0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	90	0	90	90	90	0	0
0	0	0	90	90	90	90	90	0	0
0	0	0	0	0	0	0	0	0	0
0	0	90	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0

$$h[\cdot, \cdot]$$

	0	10	20	30	30	30	20	10	
	0	20	40	60	60	60	40	20	
	0	30	60	90	90	90	60	30	
	0	30	50	80	80	90	60	30	
	0	30	50	80	80	90	60	30	
	0	20	30	50	50	60	40	20	
	10	20	30	30	30	30	20	10	
	10	10	10	0	0	0	0	0	

$$h[m, n] = \sum_{k, l} g[k, l] f[m + k, n + l]$$

Box Filter

What does it do?

- Replaces each pixel with an average of its neighborhood
- Achieve smoothing effect (remove sharp features)

$$\frac{1}{9} \begin{matrix} & g[\cdot, \cdot] \\ \begin{matrix} 1 \\ 1 \\ 1 \end{matrix} & \begin{matrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{matrix} \end{matrix}$$

Image filtering: e.g. Mean Filter

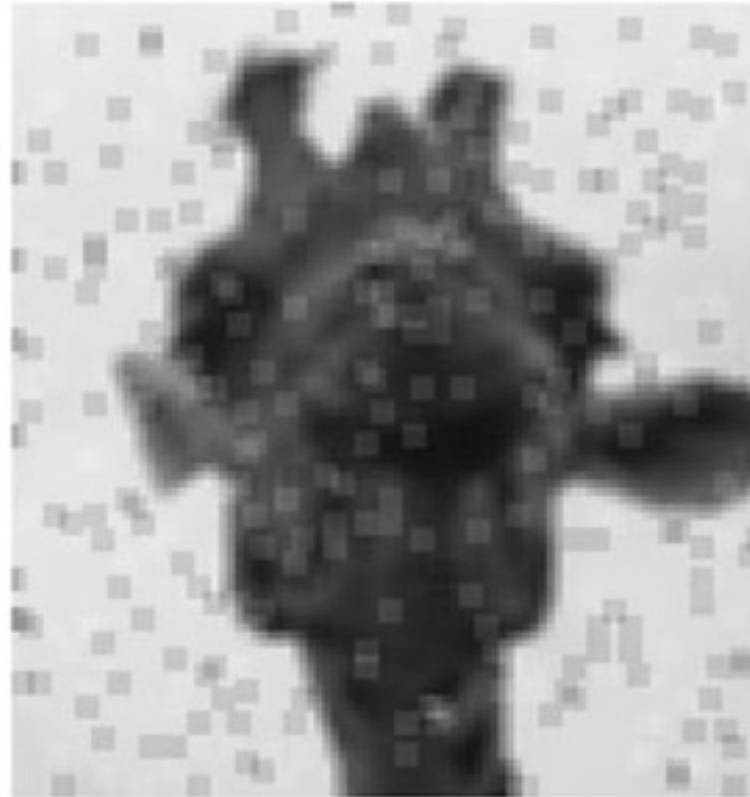
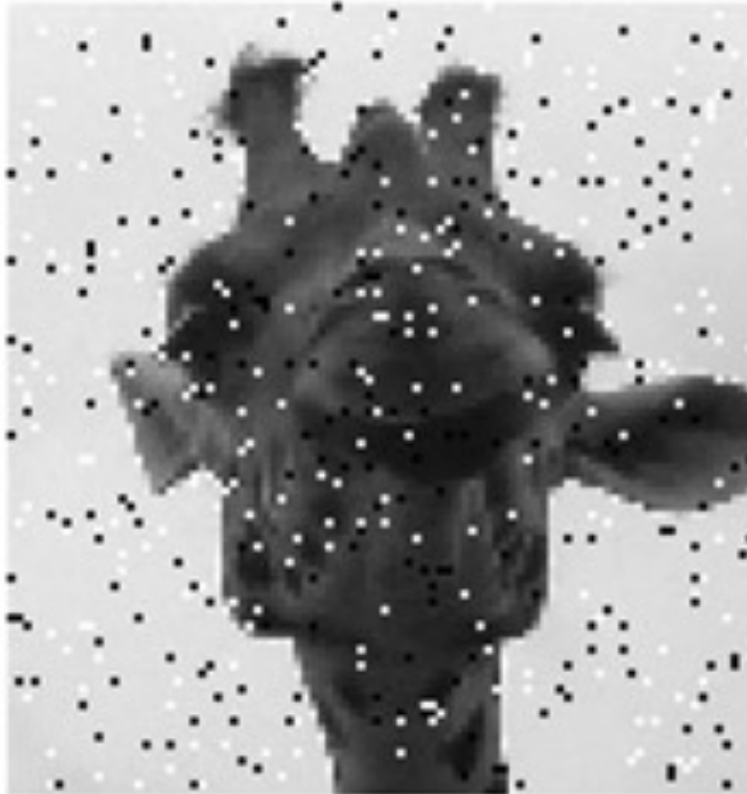
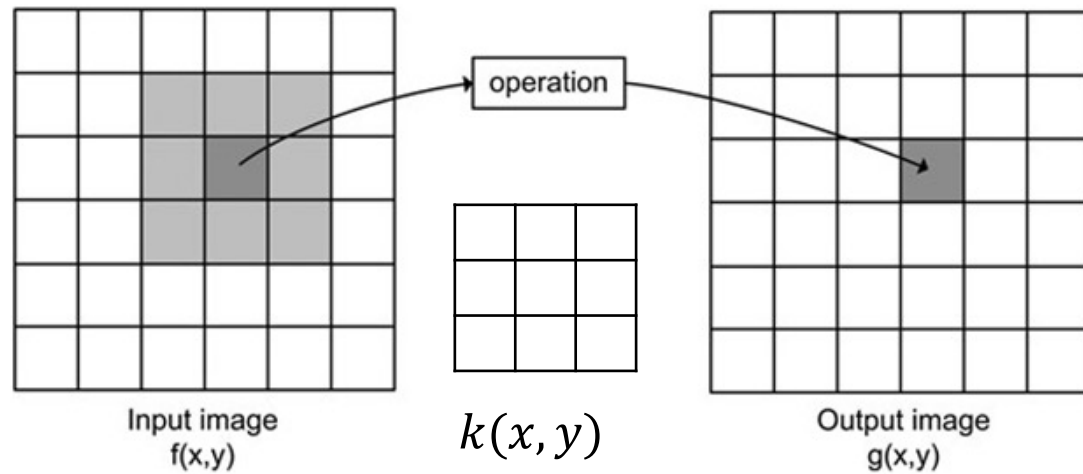


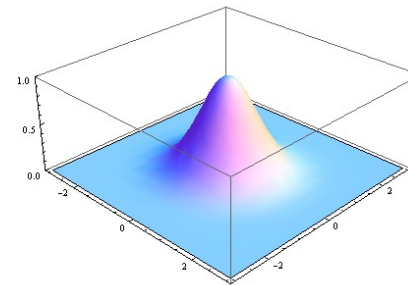
Image filtering: Convolution operator

Important filter: gaussian filter (gaussian blur)



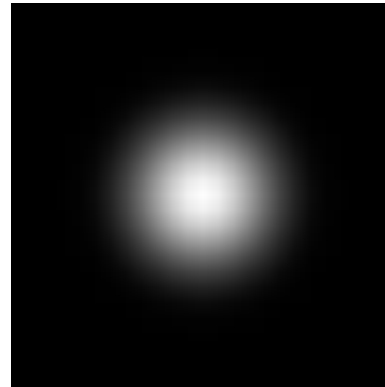
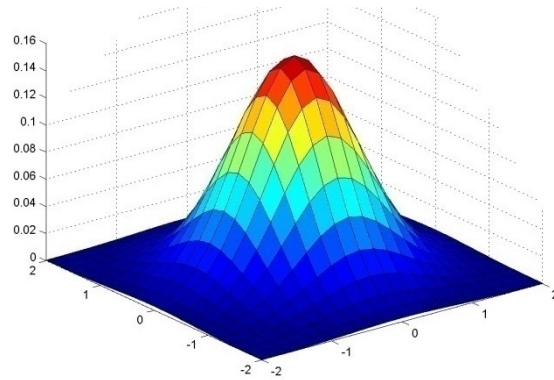
$$k(x, y) =$$

1/16	1/8	1/16
1/8	1/4	1/8
1/16	1/8	1/16



Important filter: Gaussian

- Weight contributions of neighboring pixels by nearness



0.003	0.013	0.022	0.013	0.003
0.013	0.059	0.097	0.059	0.013
0.022	0.097	0.159	0.097	0.022
0.013	0.059	0.097	0.059	0.013
0.003	0.013	0.022	0.013	0.003

5 x 5, $\sigma = 1$

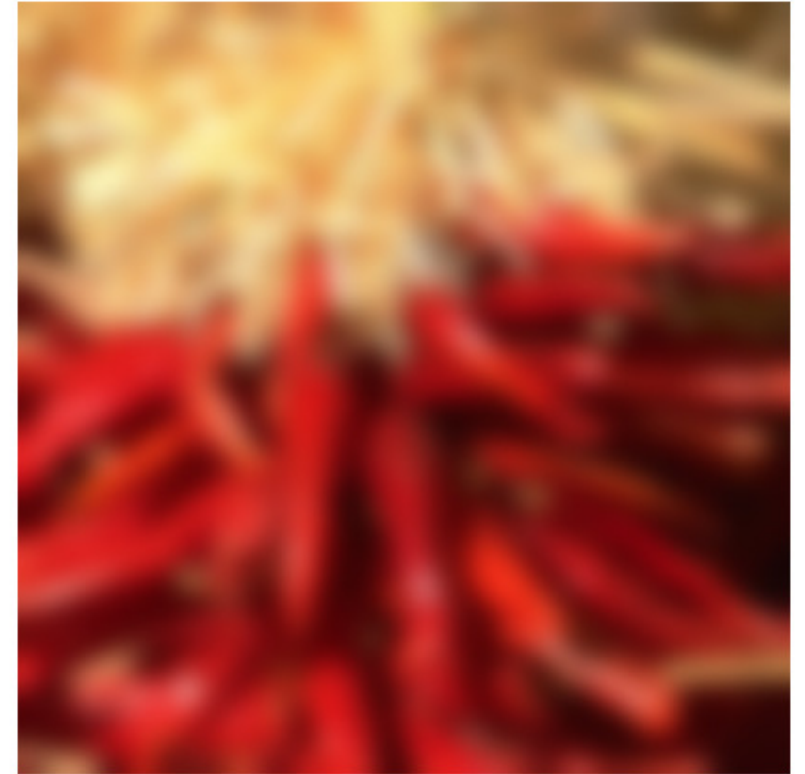
$$G_{\sigma} = \frac{1}{2\pi\sigma^2} e^{-\frac{(x^2+y^2)}{2\sigma^2}}$$

Image filtering: Convolution operator e.g. gaussian filter (gaussian blur)



Practical matters

- What about near the edge?
 - the filter window falls off the edge of the image
 - need to extrapolate
 - methods:
 - clip filter (black)
 - wrap around
 - copy edge
 - reflect across edge

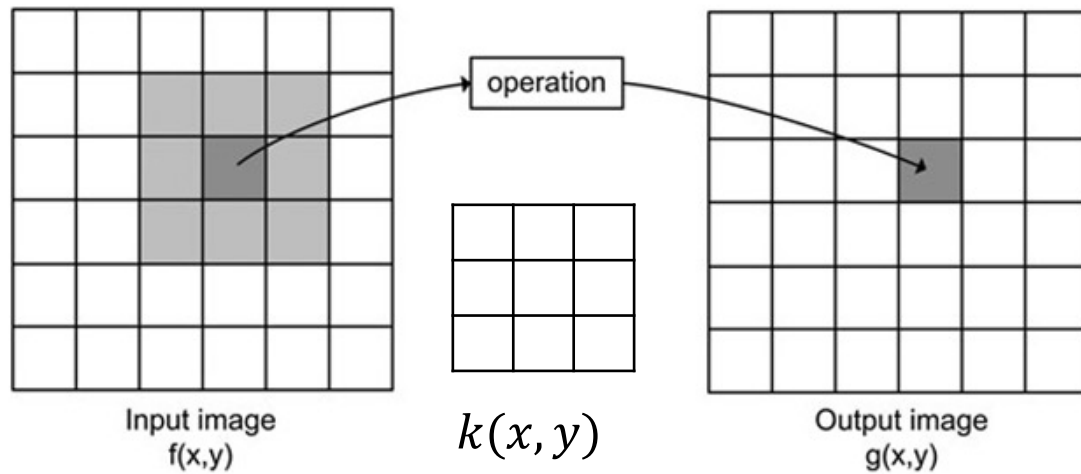


Convolution: Useful Operator for Image Processing

- Not all image filtering – region neighborhood operators can be expressed as convolutions.
- They also can be used to extract information about edges and shapes .e.g. for image recognition
- Convolutional operations are at the basis of convolutional neural networks.

Image filtering: Convolution operator

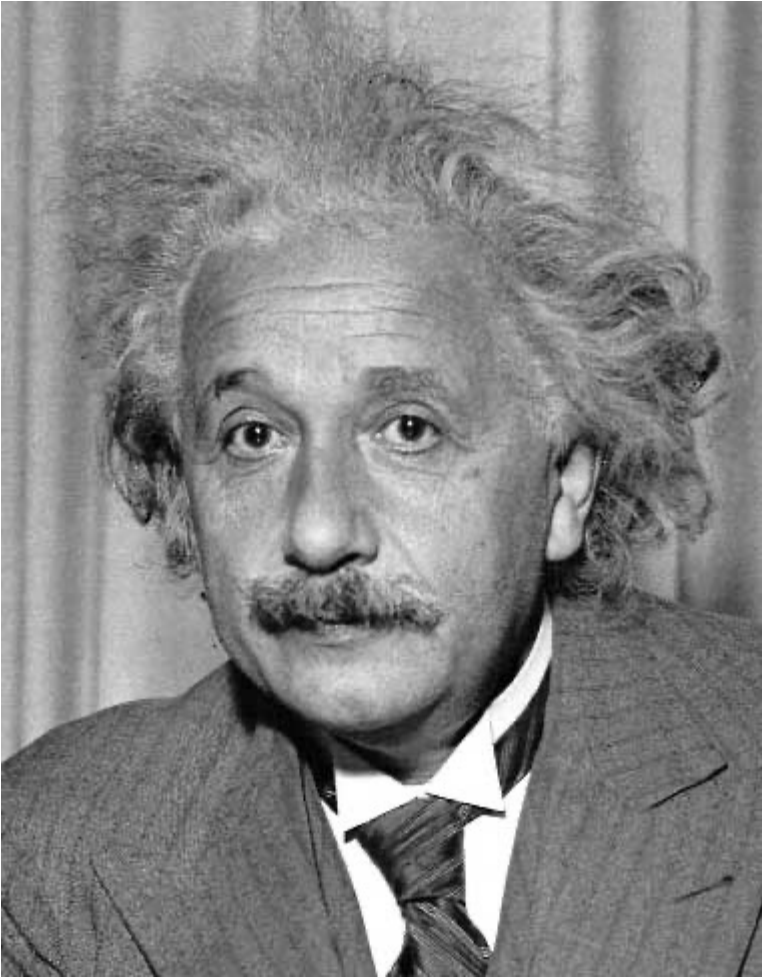
Important Filter: Sobel operator



$$k(x,y) =$$

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

Other filters



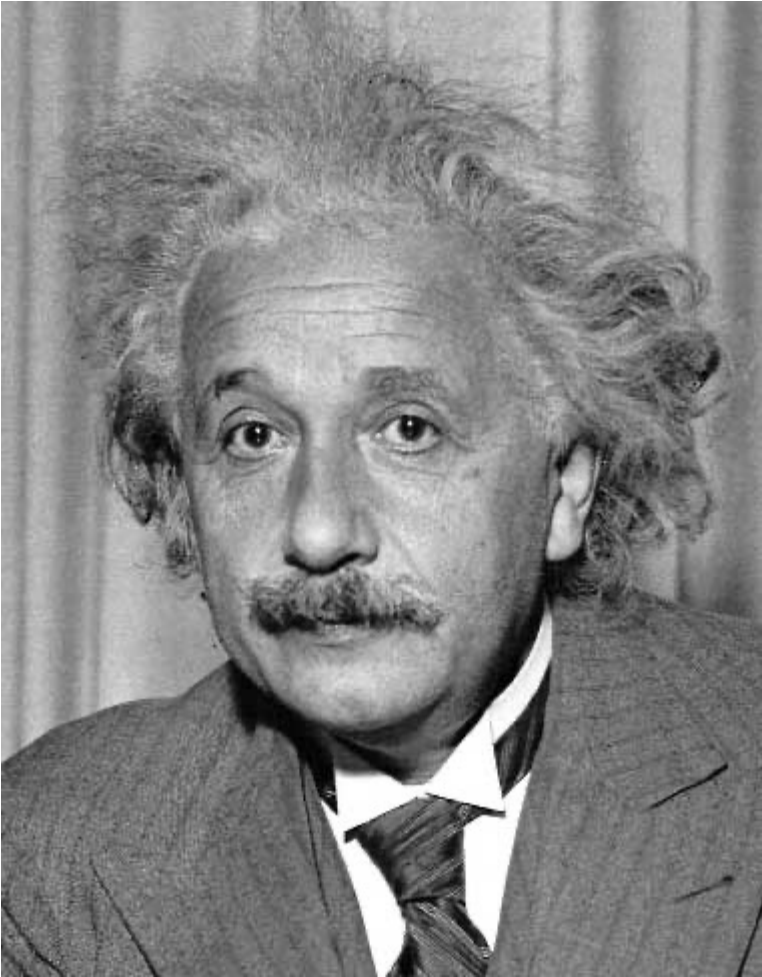
1	0	-1
2	0	-2
1	0	-1

Sobel



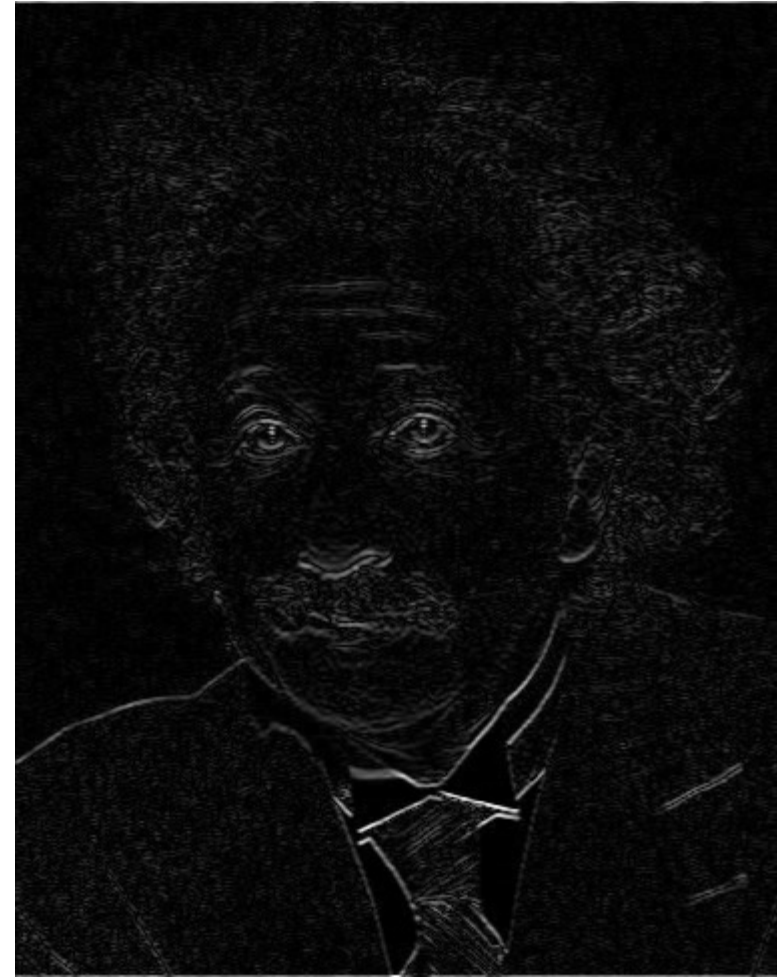
Vertical Edge
(absolute value)

Other filters



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

Sobel

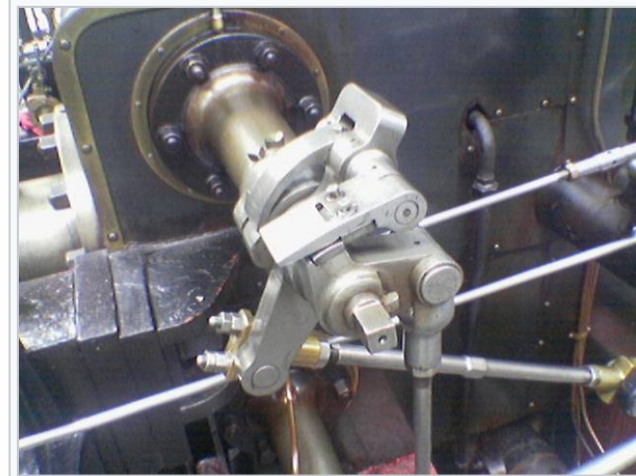


Horizontal Edge
(absolute value)

Sobel operators are equivalent to 2D partial derivatives of the image

- Vertical sobel operator – Partial derivative in X (width)
- Horizontal sobel operator – Partial derivative in Y (height)
- Can compute magnitude and phase at each location.
- Useful for detecting edges

https://en.wikipedia.org/wiki/Sobel_operator



A color picture of a steam engine



The Sobel operator applied to that image



Sobel filters are (approximate) partial derivatives of the image

Let $f(x, y)$ be your input image, then the partial derivative is:

$$\frac{\partial f(x, y)}{\partial x} = \lim_{h \rightarrow 0} \frac{f(x + h, y) - f(x, y)}{h}$$

Also:

$$\frac{\partial f(x, y)}{\partial x} = \lim_{h \rightarrow 0} \frac{f(x + h, y) - f(x - h, y)}{2h}$$

But digital images are not continuous, they are discrete

Let $f[x, y]$ be your input image, then the partial derivative is:

$$\Delta_x f[x, y] = f[x + 1, y] - f[x, y]$$

Also:
$$\Delta_x f[x, y] = f[x + 1, y] - f[x - 1, y]$$

But digital images are not continuous, they are discrete

Let $f[x, y]$ be your input image, then the partial derivative is:

$$\Delta_x f[x, y] = f[x + 1, y] - f[x, y]$$

$$k(x, y) =$$

-1	1
----	---

Also:

$$\Delta_x f[x, y] = f[x + 1, y] - f[x - 1, y]$$

$$k(x, y) =$$

-1	0	1
----	---	---

Sobel Operators Smooth in Y and then Differentiate in X

$$k(x, y) = \begin{array}{|c|} \hline 1 \\ \hline 2 \\ \hline 1 \\ \hline \end{array} * \begin{array}{|c|c|c|} \hline 1 & 0 & -1 \\ \hline \end{array} = \begin{array}{|c|c|c|} \hline 1 & 0 & -1 \\ \hline 2 & 0 & -2 \\ \hline 1 & 0 & -1 \\ \hline \end{array}$$

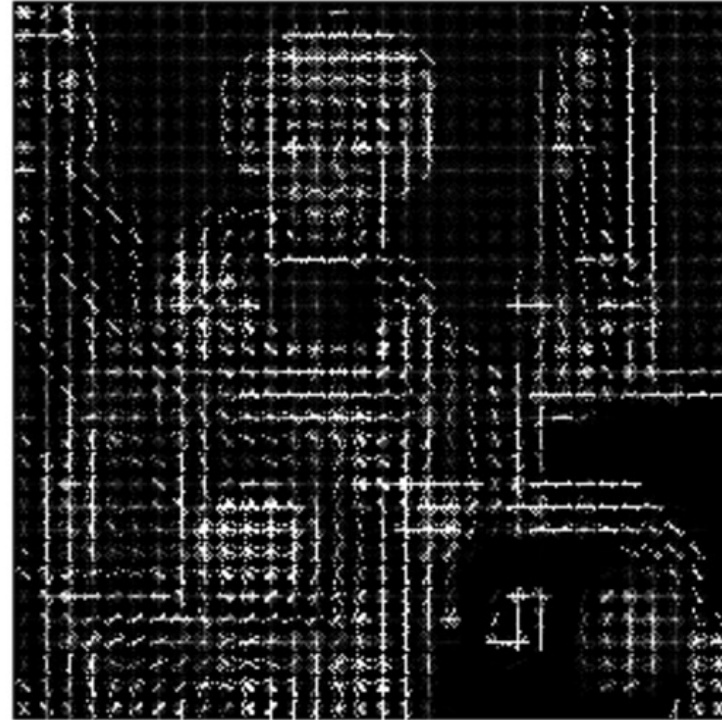
Similarly to differentiate in Y

Image Features: HoG

Input image



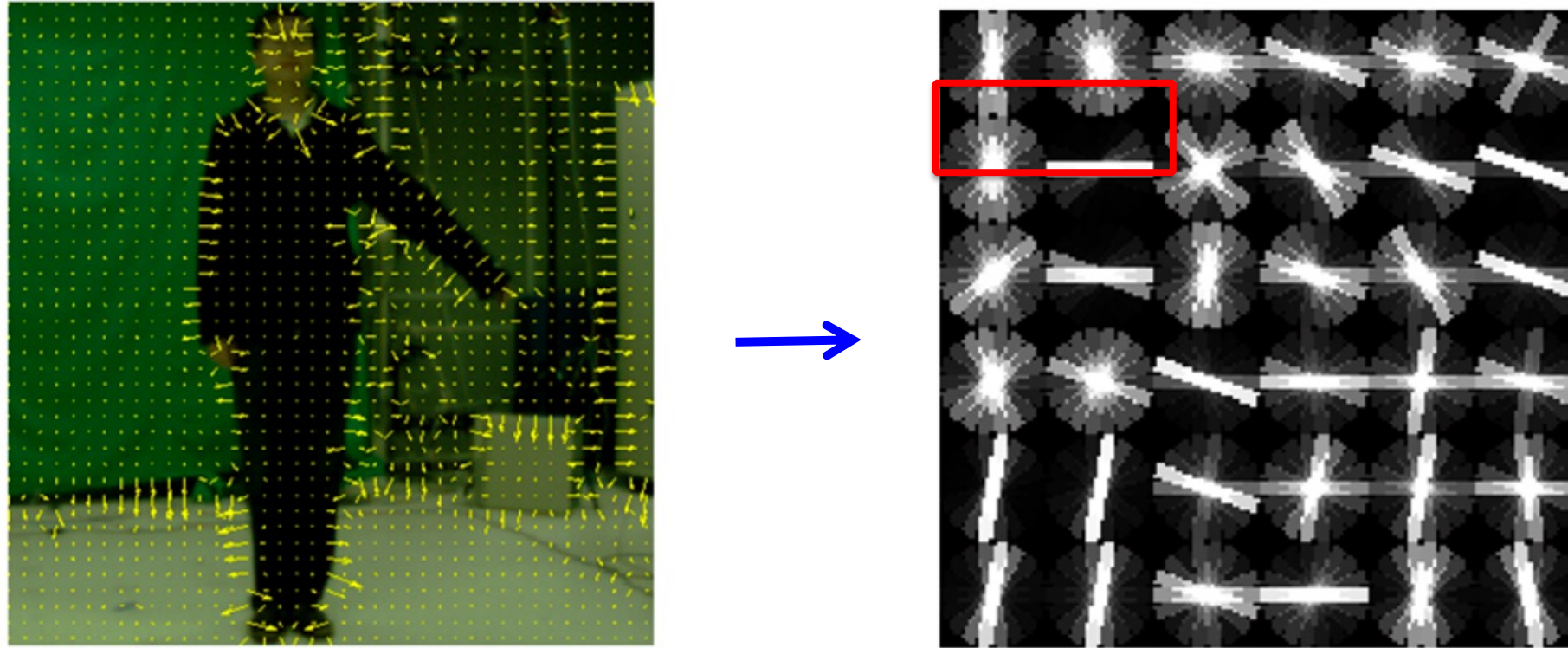
Histogram of Oriented Gradients



Paper by Navneet Dalal & Bill Triggs presented at CVPR 2005 for detecting people.

Scikit-image implementation

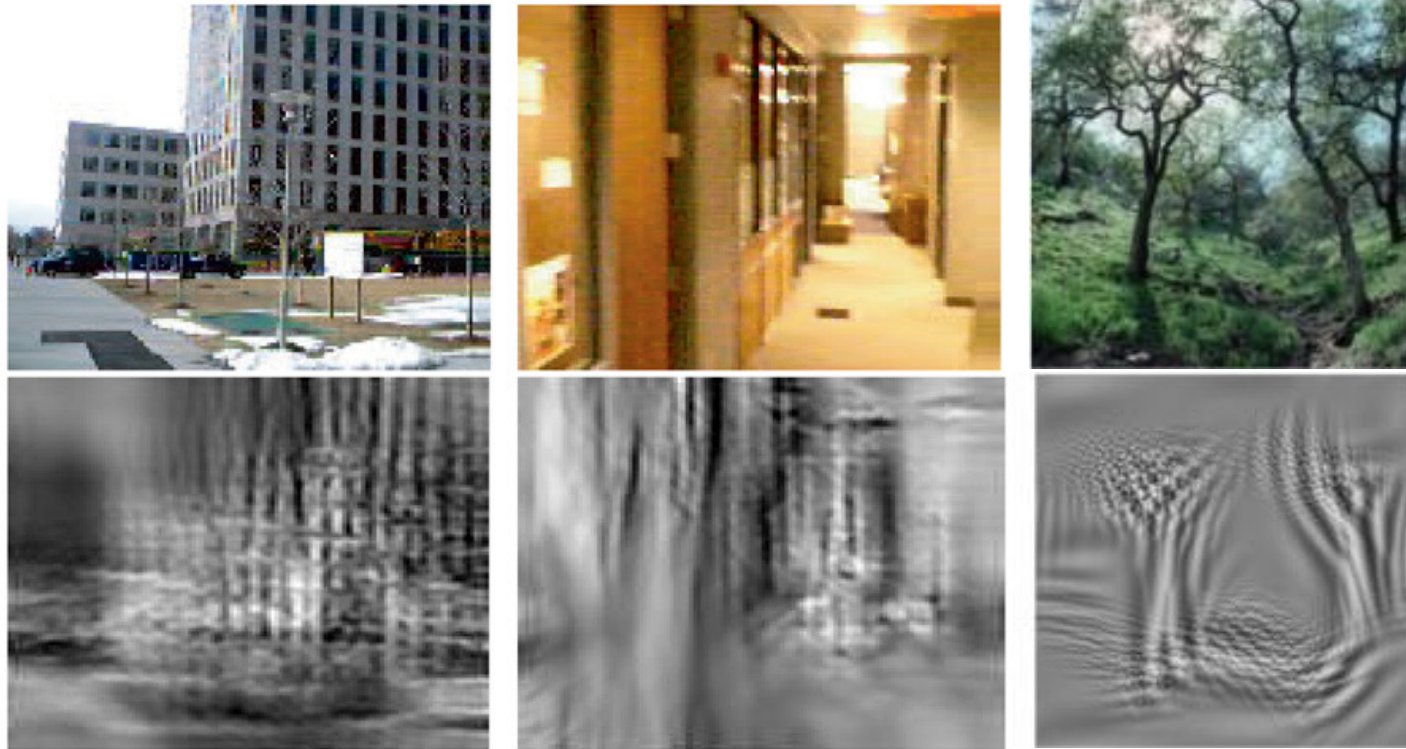
Image Features: HoG



+ Block Normalization

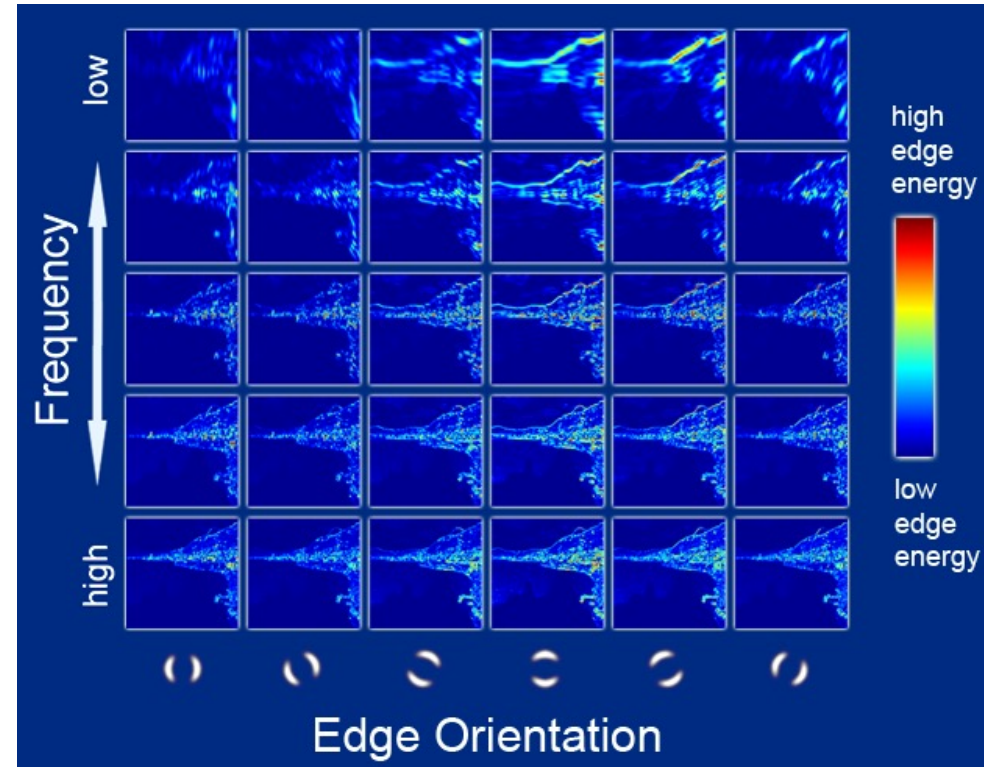
Paper by Navneet Dalal & Bill Triggs presented at CVPR 2005 for detecting people.
Figure from Zhuolin Jiang, Zhe Lin, Larry S. Davis, ICCV 2009 for human action recognition.

Image Features: GIST



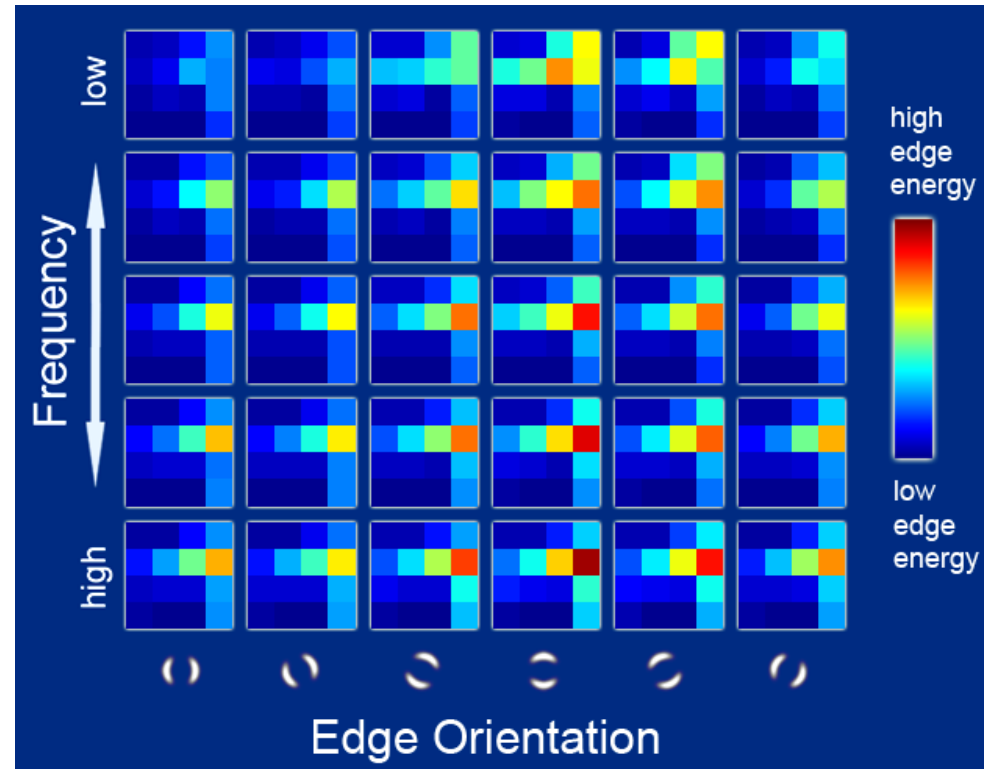
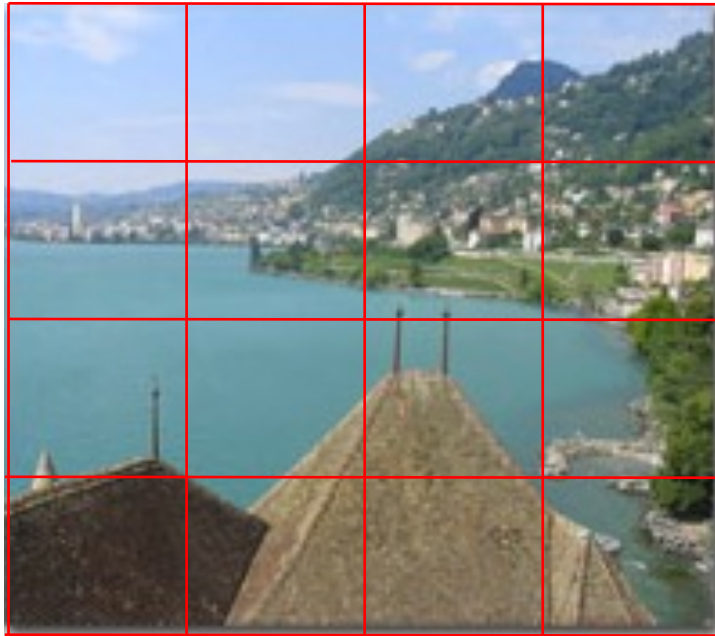
The “gist” of a scene: Oliva & Torralba, 2001

Image Features: GIST



Oriented edge response at multiple scales (5 spatial scales, 6 edge orientations)

Image Features: GIST



Aggregated edge responses over 4x4 windows

The 2D Convolutional Layer in a Neural Network

Input image

*

Weights



Output image

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

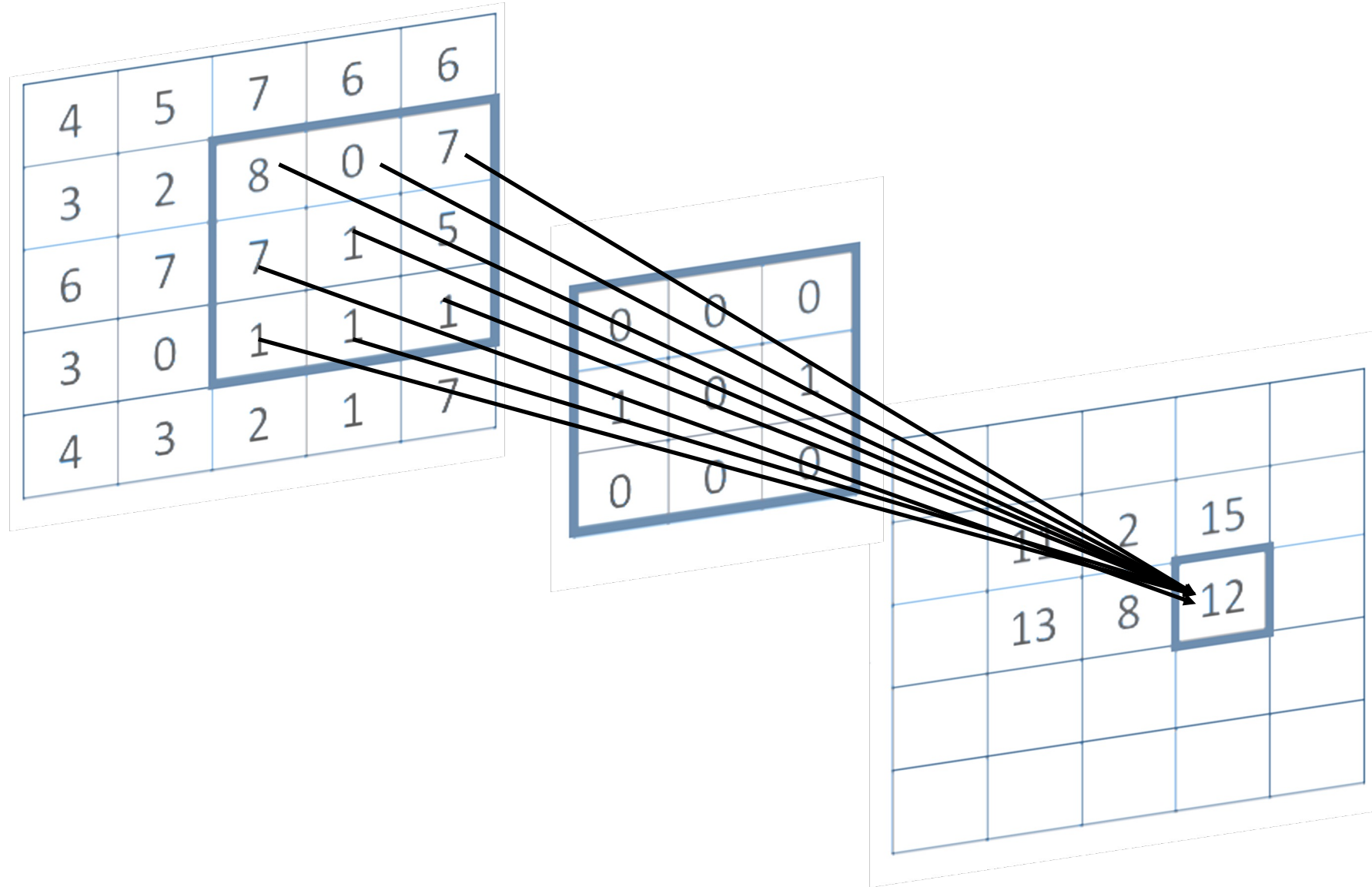
*

0	0	0
1	0	1
0	0	0

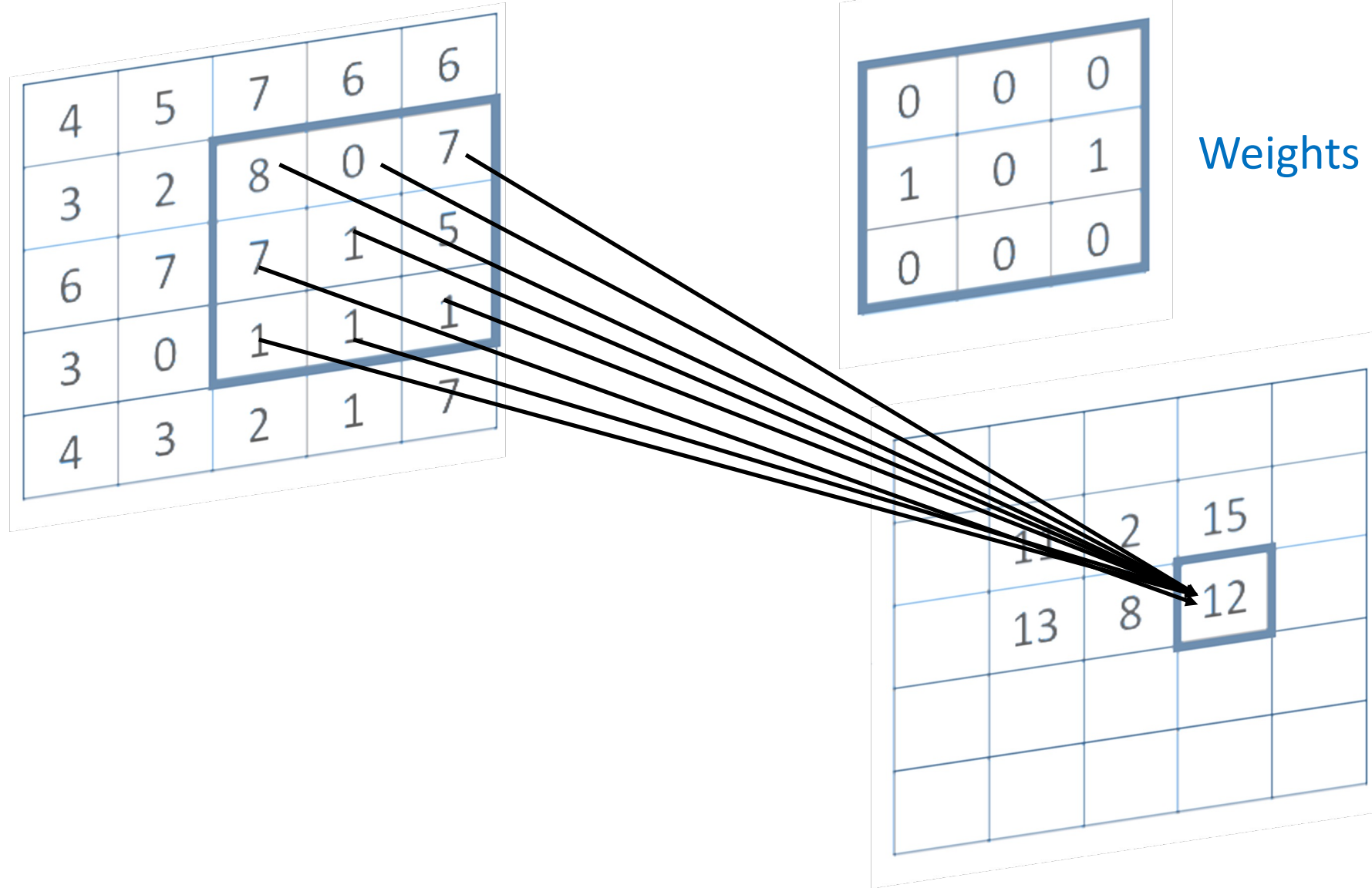


	11	2	15	
	13	8	12	

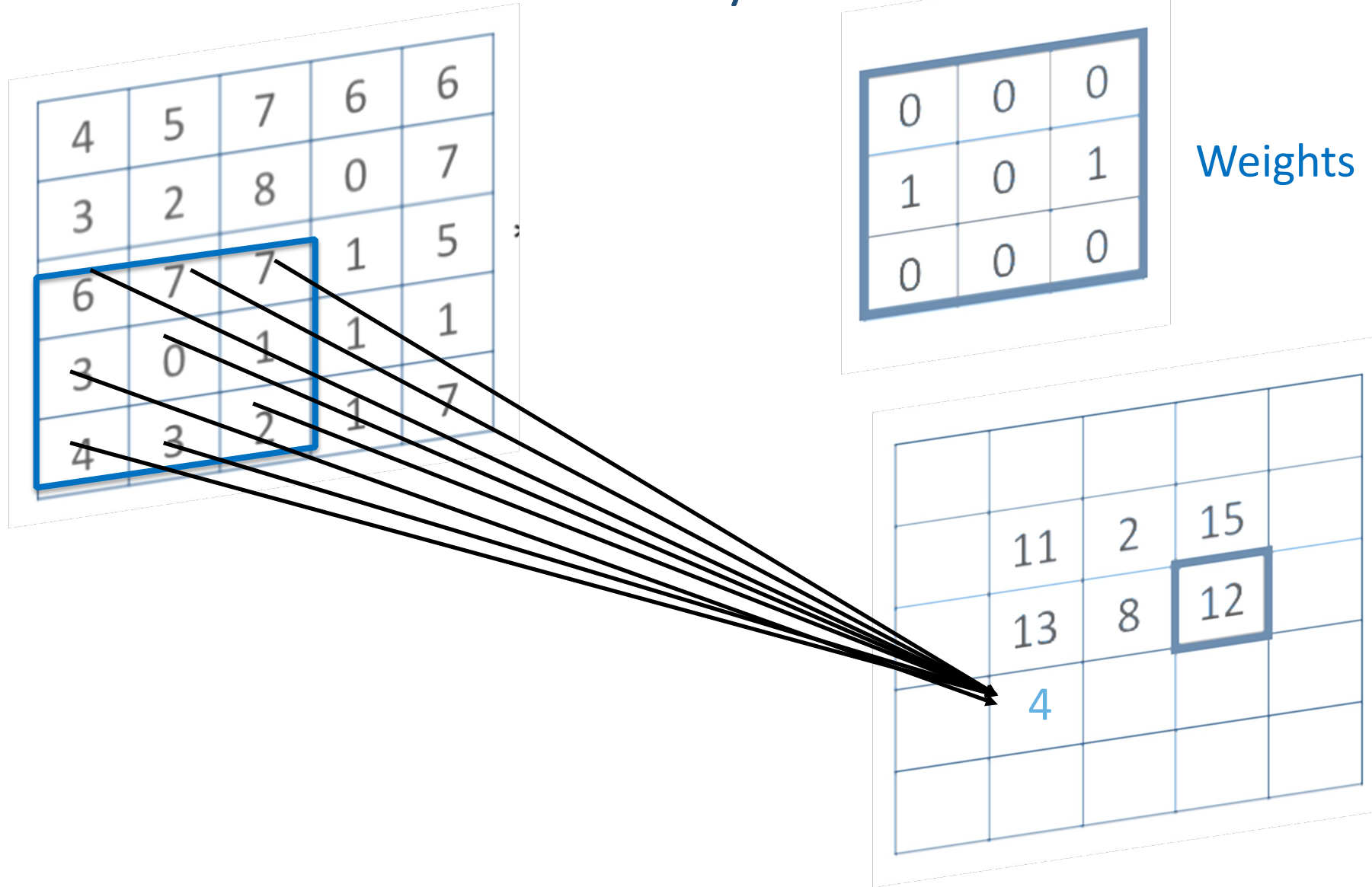
The 2D Convolutional Layer in a Neural Network



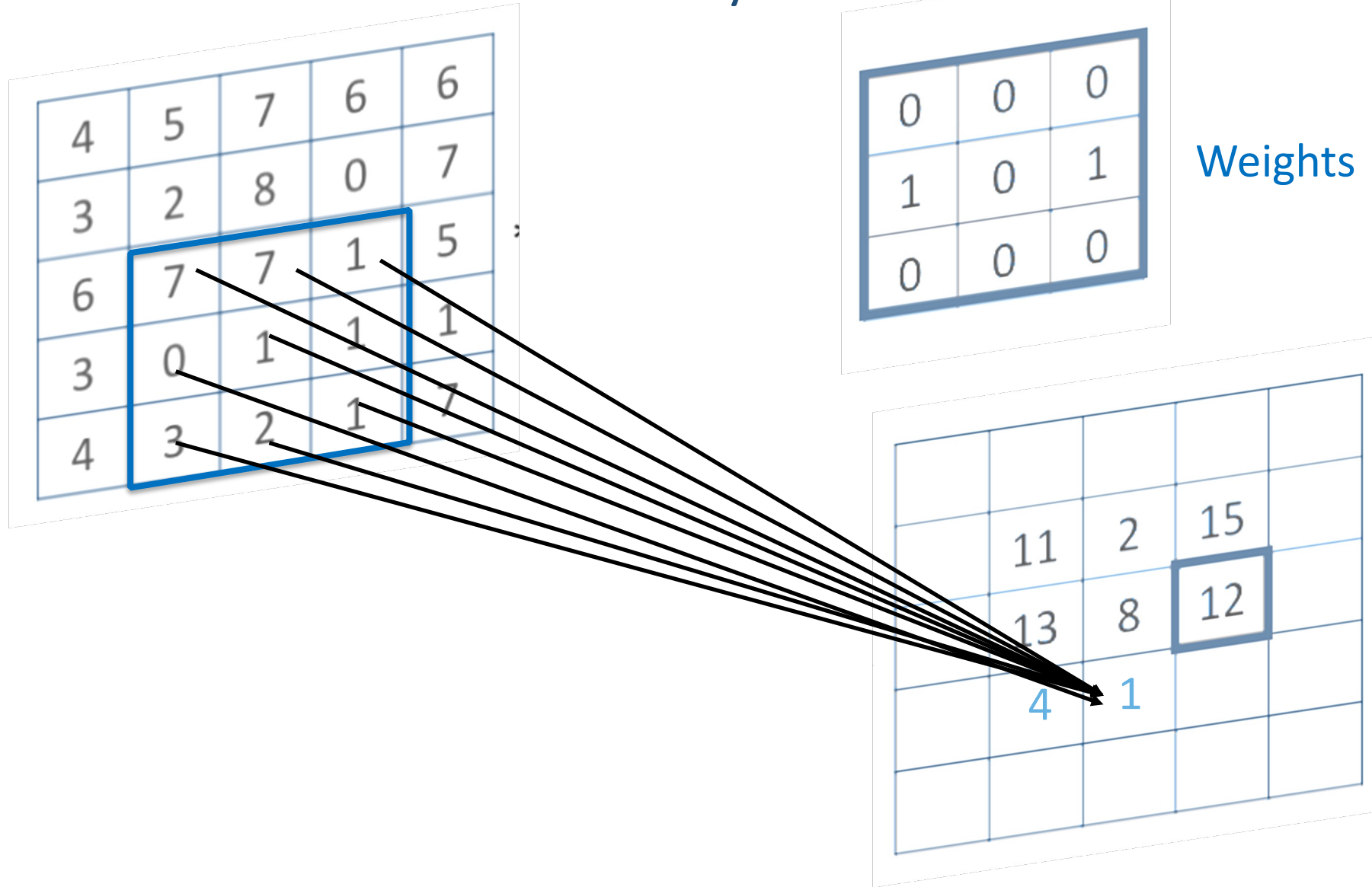
The 2D Convolutional Layer in a Neural Network



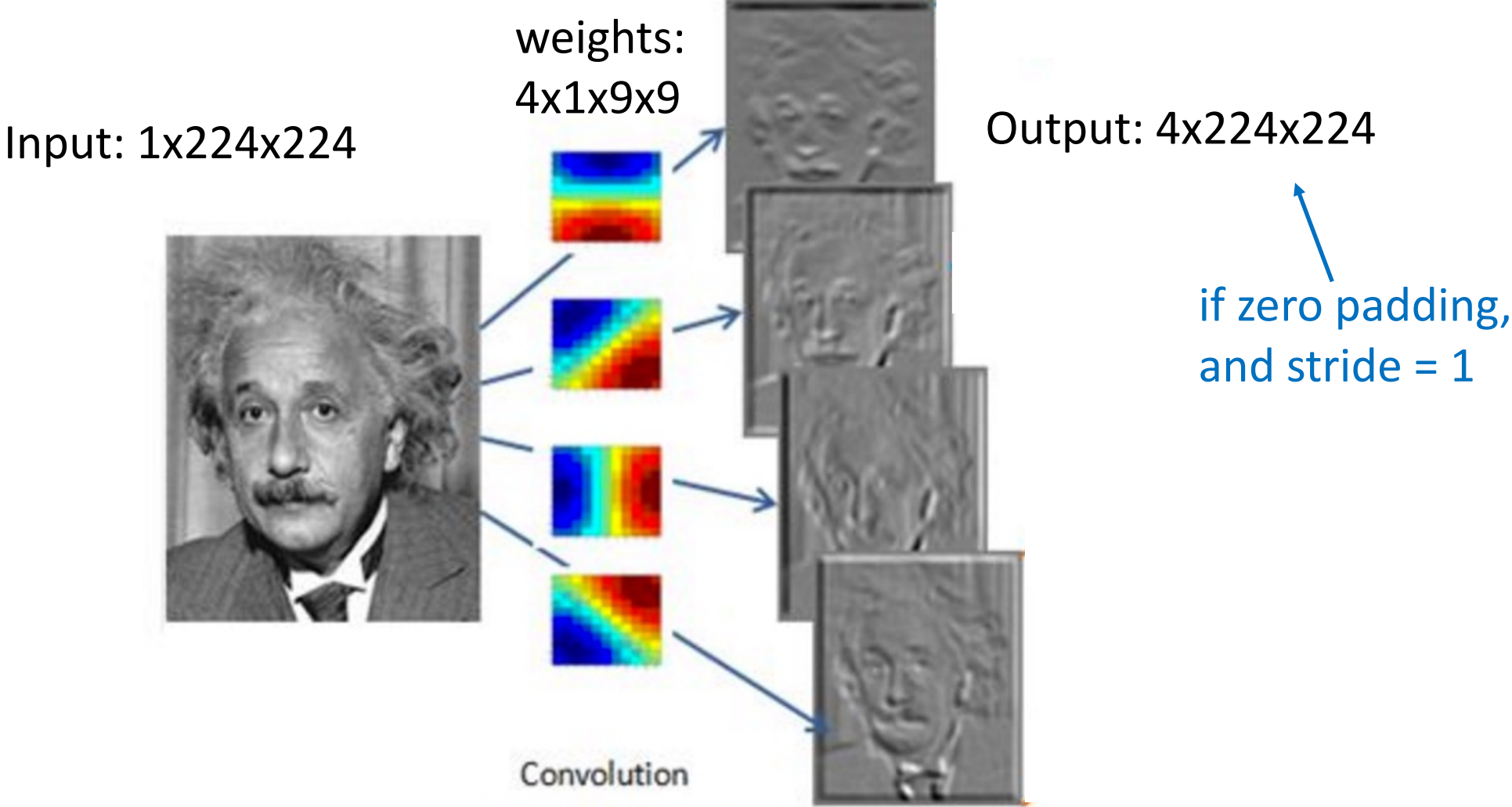
The 2D Convolutional Layer in a Neural Network



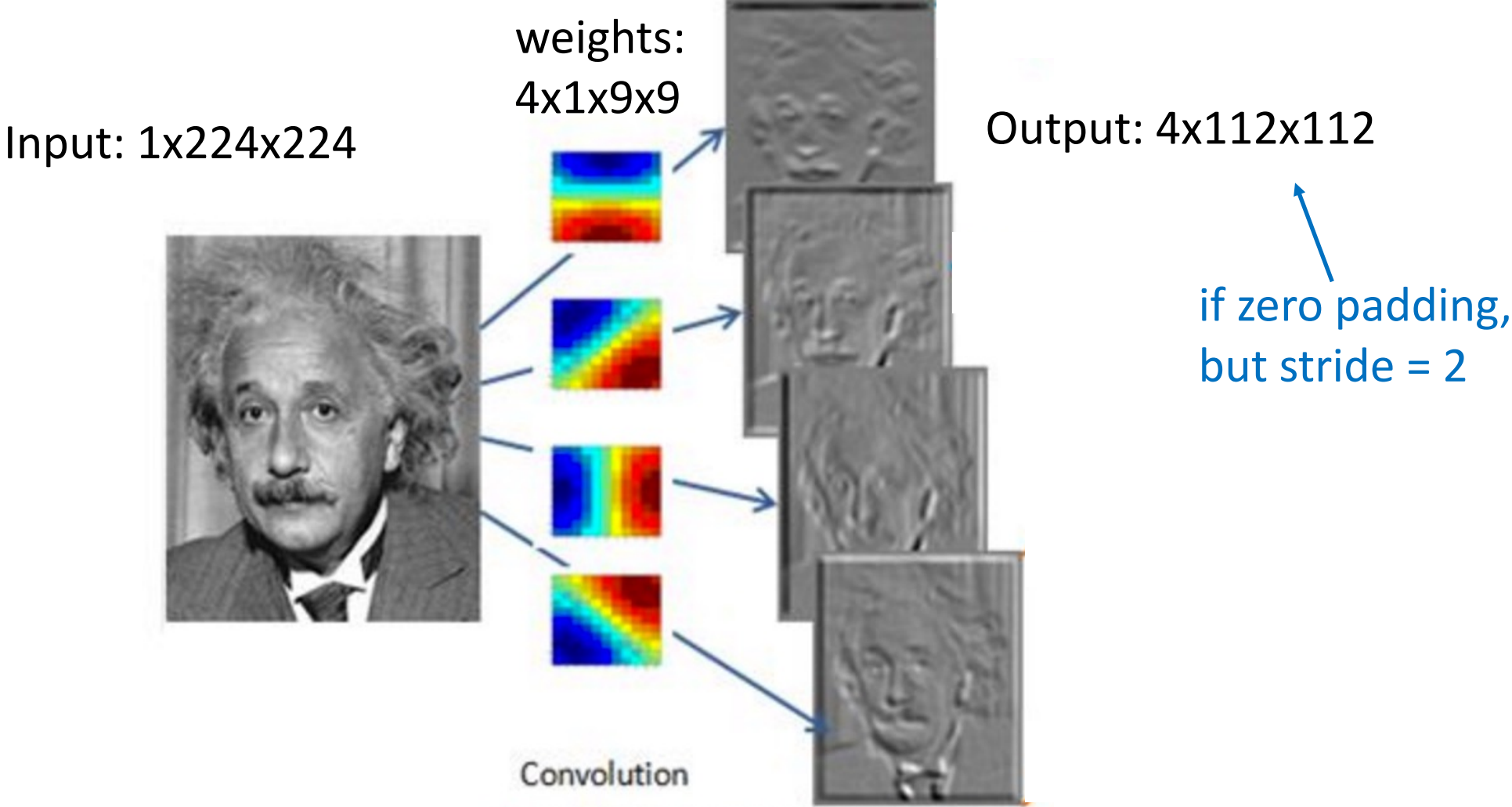
The 2D Convolutional Layer in a Neural Network



Convolutional Layer (with 4 filters)

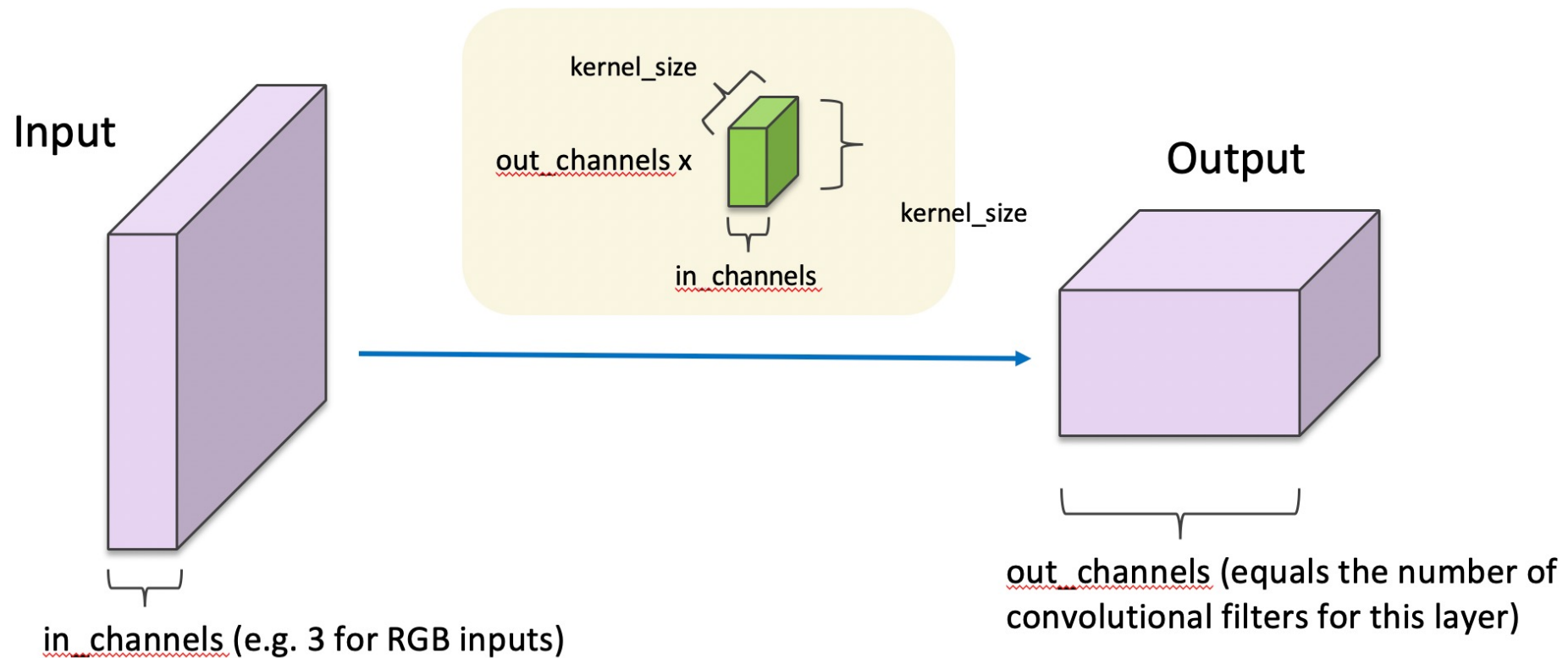


Convolutional Layer (with 4 filters)

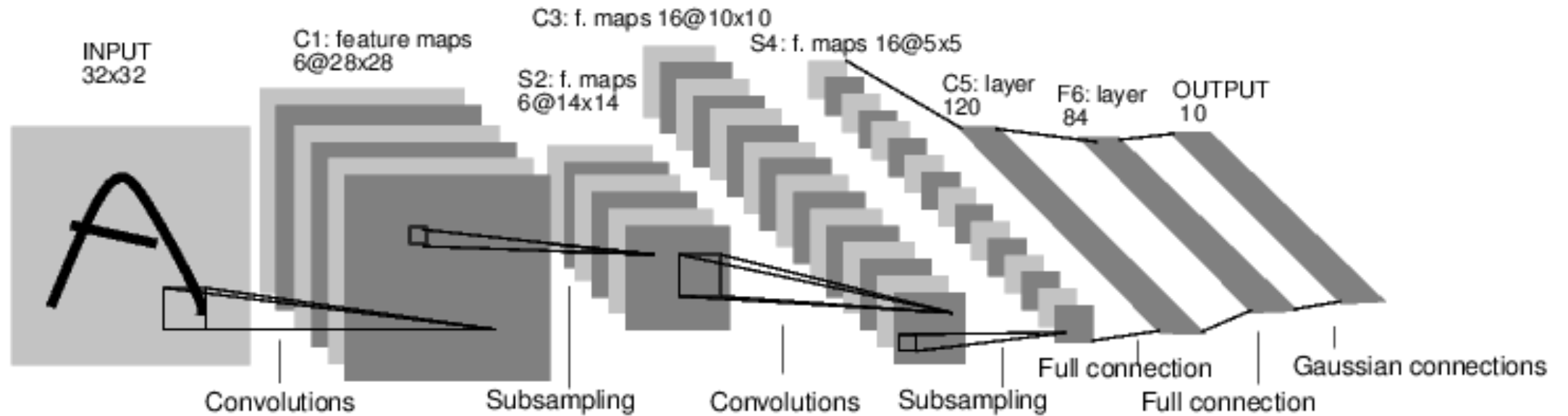


Convolutional Layer in pytorch

```
class torch.nn.Conv2d(in_channels, out_channels, kernel_size, stride=1, padding=0, dilation=1,  
groups=1, bias=True) \[source\]
```



Convolutional Network: LeNet



Yann LeCun

TITLE

CITED BY

YEAR

[Gradient-based learning applied to document recognition](#)

11736

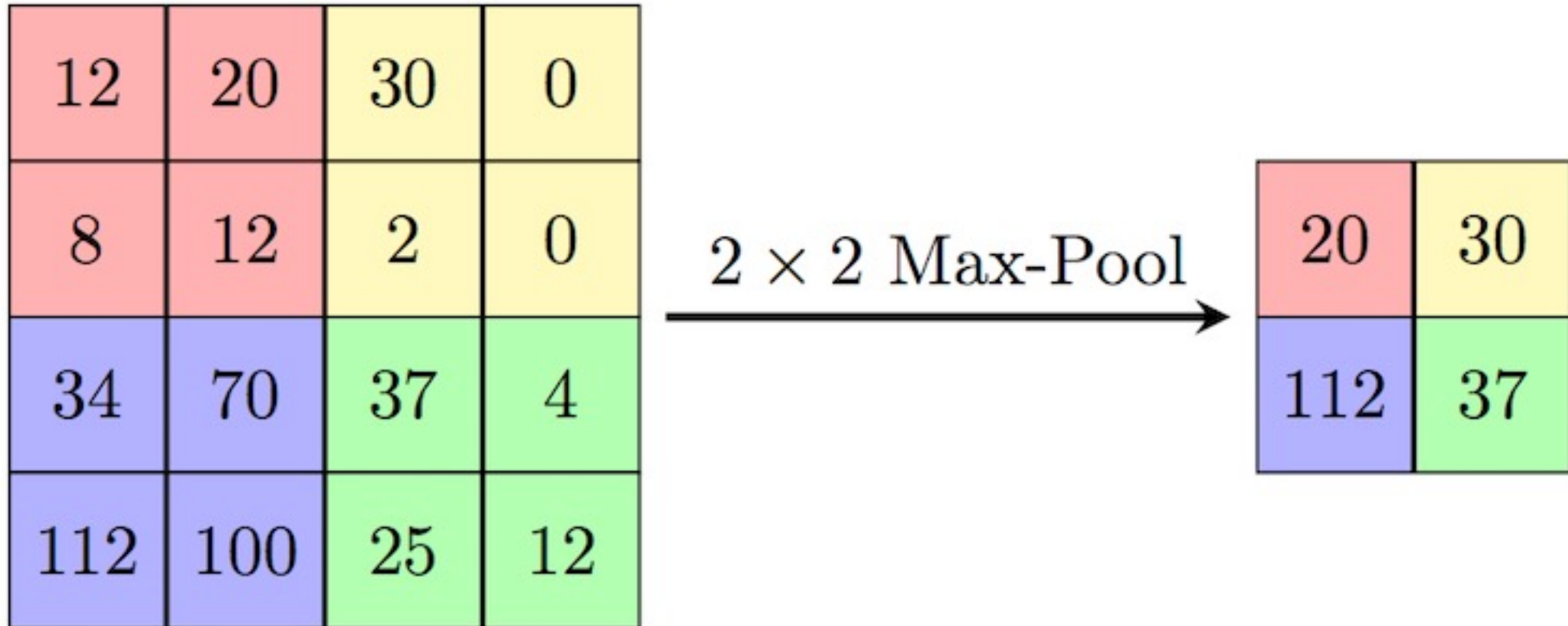
1998

Y LeCun, L Bottou, Y Bengio, P Haffner
Proceedings of the IEEE 86 (11), 2278-2324

LeNet in Pytorch

```
# LeNet is French for The Network, and is taken from Yann Lecun's 98 paper  
# on digit classification http://yann.lecun.com/exdb/lenet/  
# This was also a network with just two convolutional layers.  
class LeNet(nn.Module):  
    def __init__(self):  
        super(LeNet, self).__init__()  
        # Convolutional layers.  
        self.conv1 = nn.Conv2d(3, 6, 5)  
        self.conv2 = nn.Conv2d(6, 16, 5)  
  
        # Linear layers.  
        self.fc1 = nn.Linear(16*5*5, 120)  
        self.fc2 = nn.Linear(120, 84)  
        self.fc3 = nn.Linear(84, 10)  
  
    def forward(self, x):  
        out = F.relu(self.conv1(x))  
        out = F.max_pool2d(out, 2)  
        out = F.relu(self.conv2(out))  
        out = F.max_pool2d(out, 2)  
  
        # This flattens the output of the previous layer into a vector.  
        out = out.view(out.size(0), -1)  
        out = F.relu(self.fc1(out))  
        out = F.relu(self.fc2(out))  
        out = self.fc3(out)  
        return out
```

SpatialMaxPooling Layer



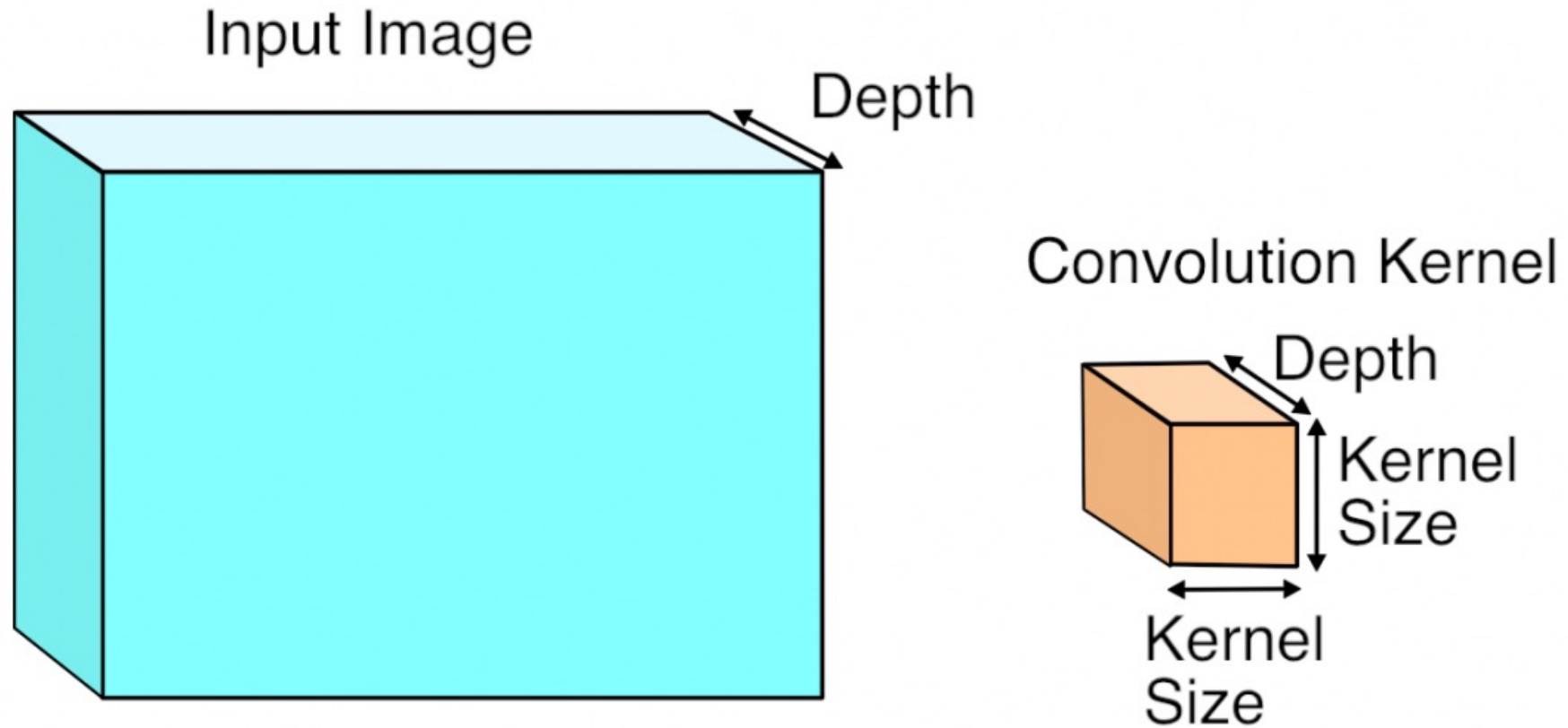
LeNet Summary

- 2 Convolutional Layers + 3 Linear Layers
- + Non-linear functions: ReLUs or Sigmoids
+ Max-pooling operations

New Architectures Proposed

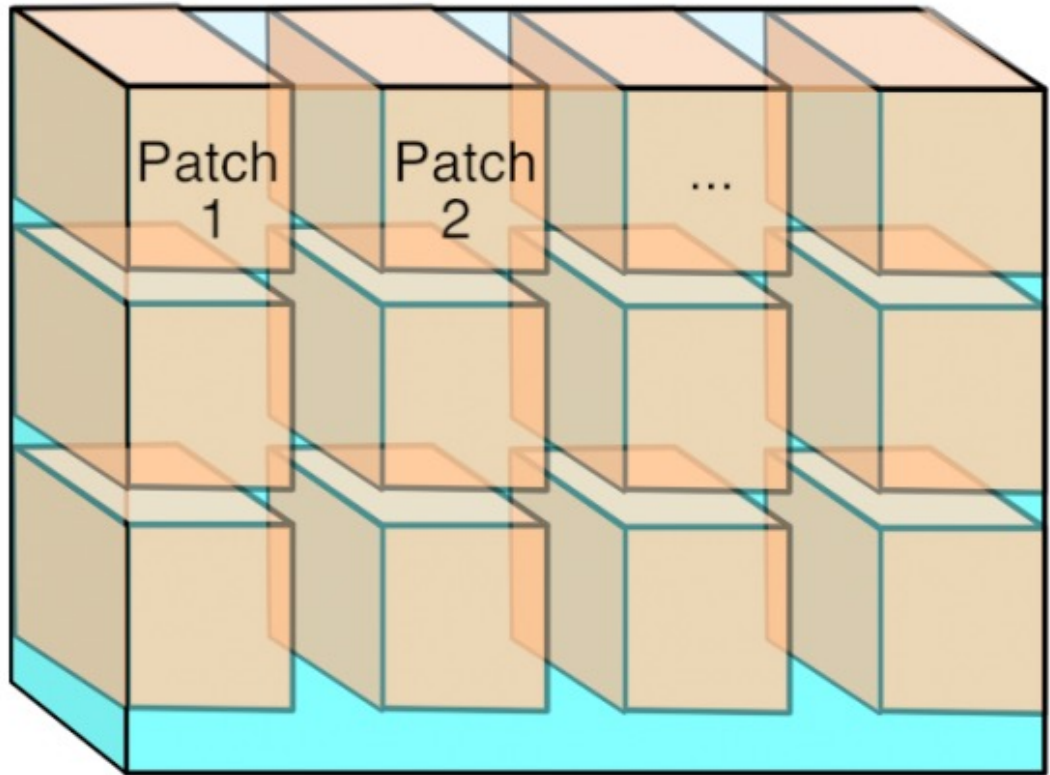
- Alexnet (Krizhevsky et al NIPS 2012) [**Required Reading**]
- VGG (Simonyan and Zisserman 2014)
- GoogLeNet (Szegedy et al CVPR 2015)
- ResNet (He et al CVPR 2016)
- DenseNet (Huang et al CVPR 2017)

Convolutional Layers as Matrix Multiplication

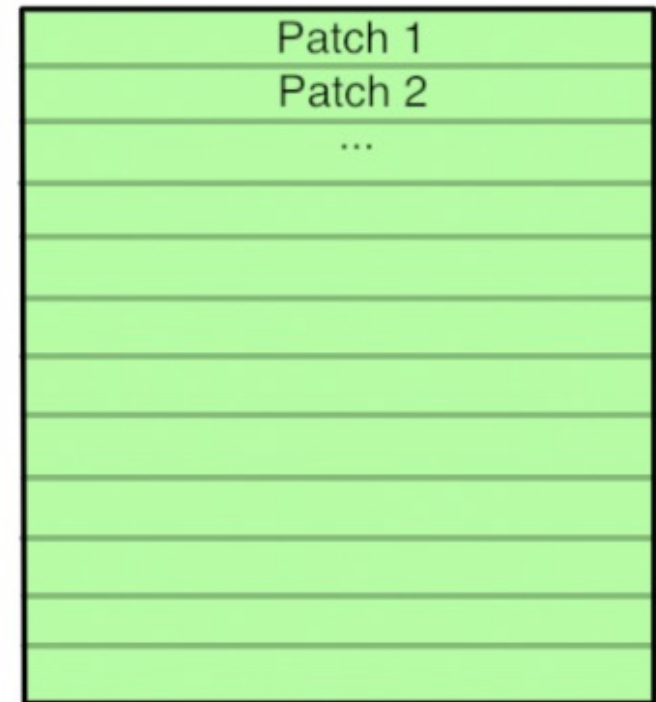


Convolutional Layers as Matrix Multiplication

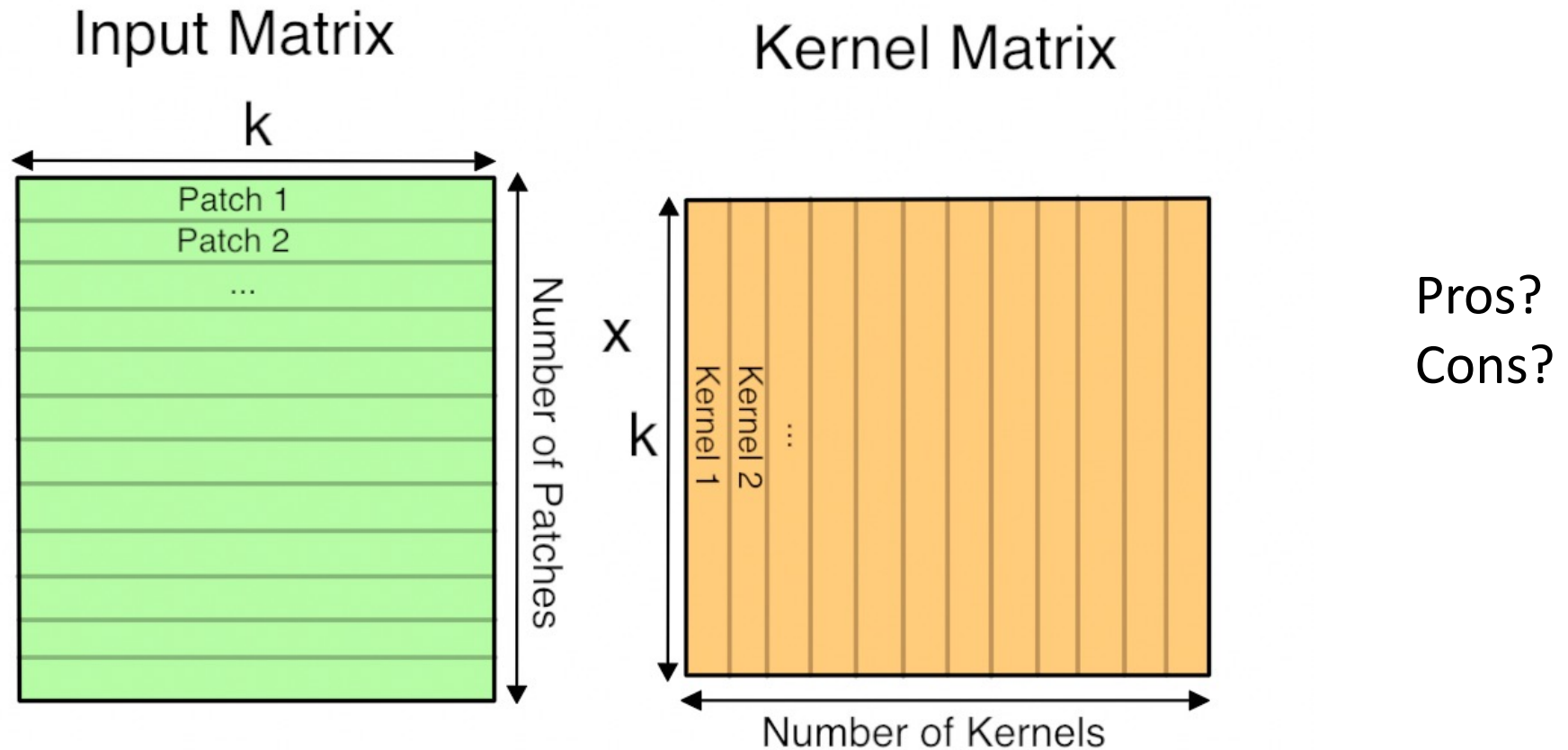
Input Image



im2col
=>



Convolutional Layers as Matrix Multiplication



CNN Computations are Computationally Expensive

- However highly parallelizable
- GPU Computing is used in practice
- CPU Computing in fact is prohibitive for training these models

The Alexnet network (Krizhevsky et al NIPS 2012)

ImageNet Classification with Deep Convolutional Neural Networks

Alex Krizhevsky
University of Toronto
kriz@cs.utoronto.ca

Ilya Sutskever
University of Toronto
ilya@cs.utoronto.ca

Geoffrey E. Hinton
University of Toronto
hinton@cs.utoronto.ca

The Problem: Classification

Classify an image into 1000 possible classes:

e.g. Abyssinian cat, Bulldog, French Terrier, Cormorant, Chickadee,
red fox, banjo, barbell, hourglass, knot, maze, viaduct, etc.



cat, tabby cat (0.71)

Egyptian cat (0.22)

red fox (0.11)

.....

The Data: ILSVRC

Imagenet Large Scale Visual Recognition Challenge (ILSVRC): Annual Competition

1000 Categories

~1000 training images per Category

~1 million images in total for training

~50k images for validation

Only images released for the test set but no annotations,
evaluation is performed centrally by the organizers (max 2 per week)

The Evaluation Metric: Top K-error

True label: Abyssinian cat

Top-1 error: 1.0

Top-1 accuracy: 0.0

Top-2 error: 1.0

Top-2 accuracy: 0.0

Top-3 error: 1.0

Top-3 accuracy: 0.0

Top-4 error: 0.0

Top-4 accuracy: 1.0

Top-5 error: 0.0

Top-5 accuracy: 1.0



cat, tabby cat (0.61)

Egyptian cat (0.22)

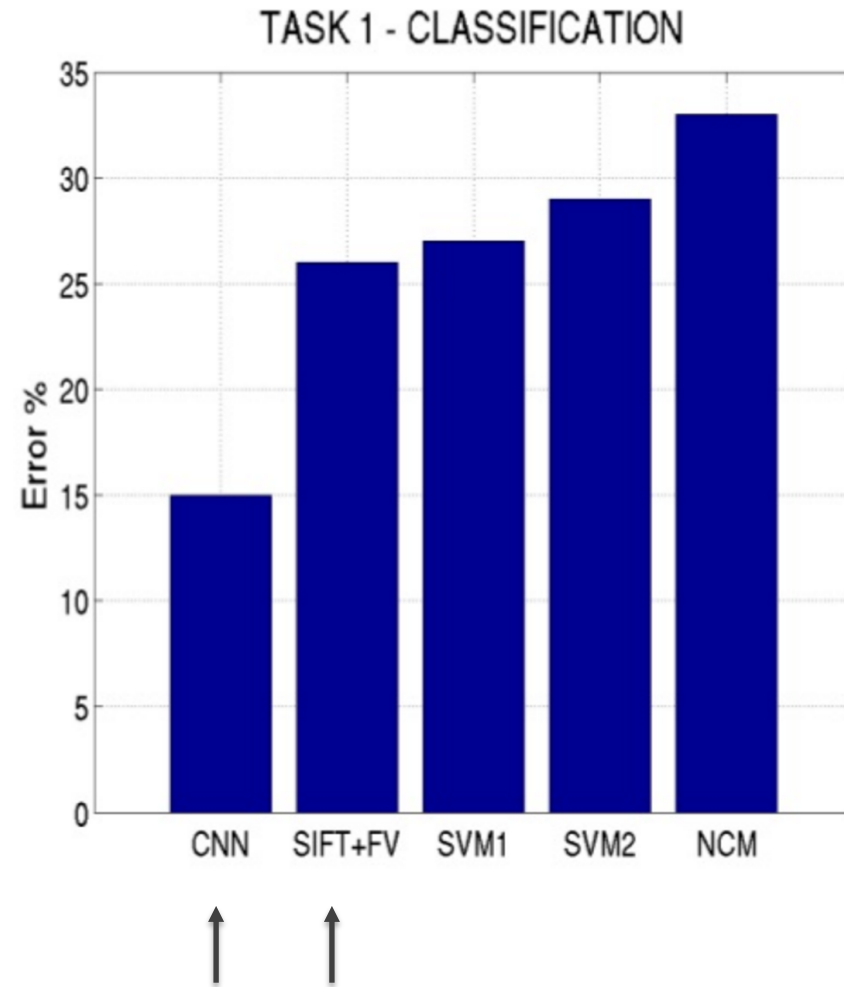
red fox (0.11)

Abyssinian cat (0.10)

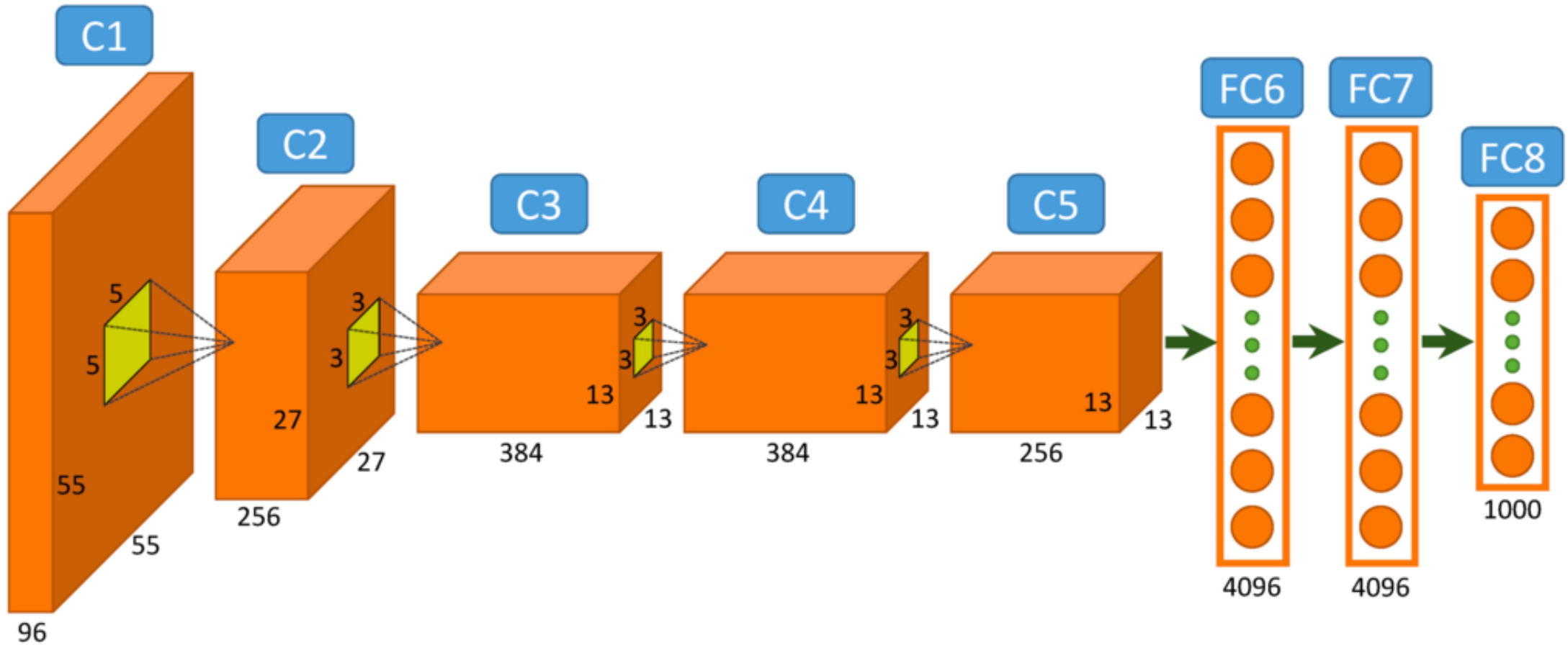
French terrier (0.03)

.....

Top-5 error on this competition (2012)



Alexnet

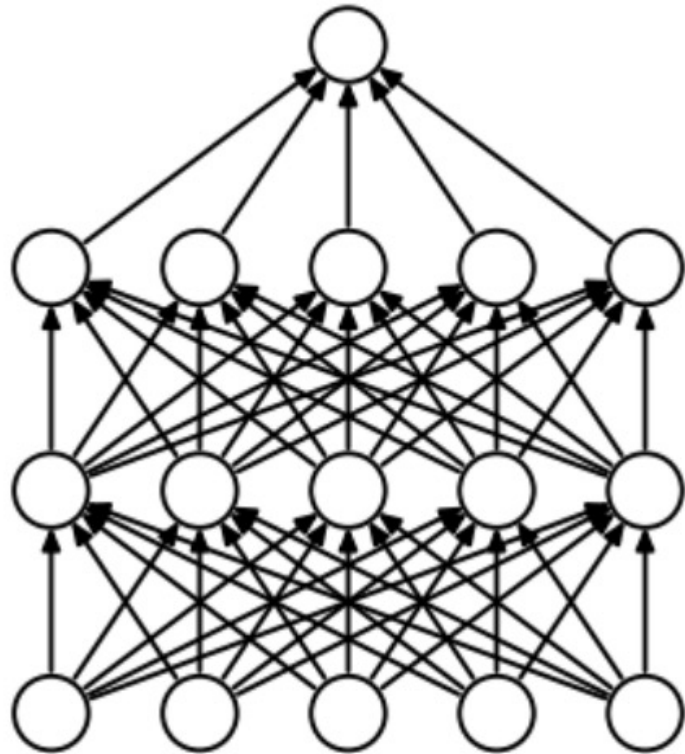


Pytorch Code for Alexnet

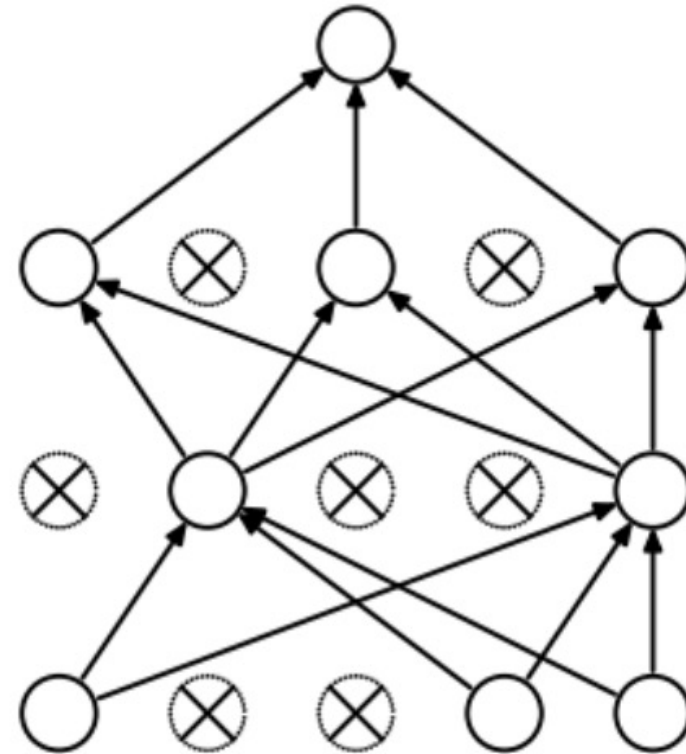
- In-class analysis

<https://github.com/pytorch/vision/blob/master/torchvision/models/alexnet.py>

Dropout Layer

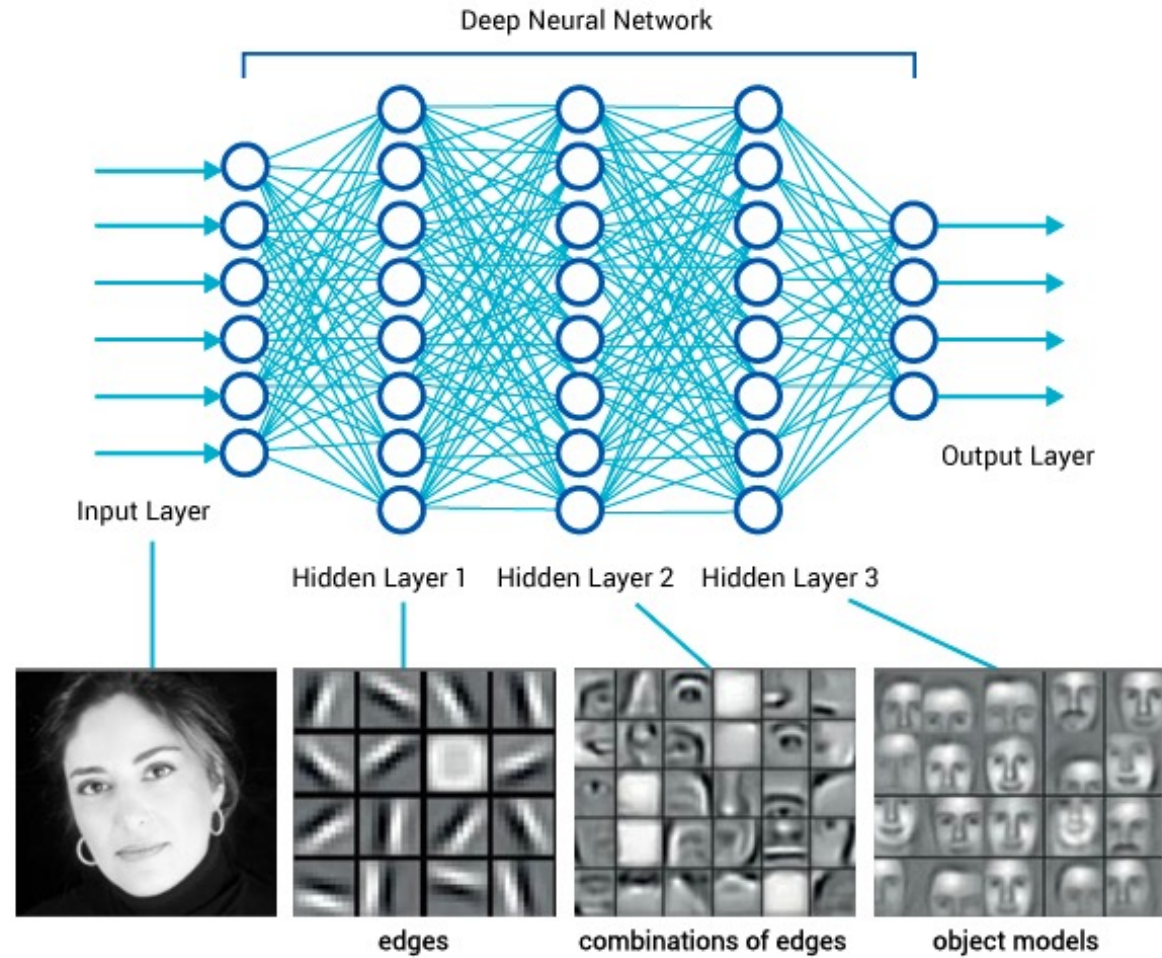


(a) Standard Neural Net

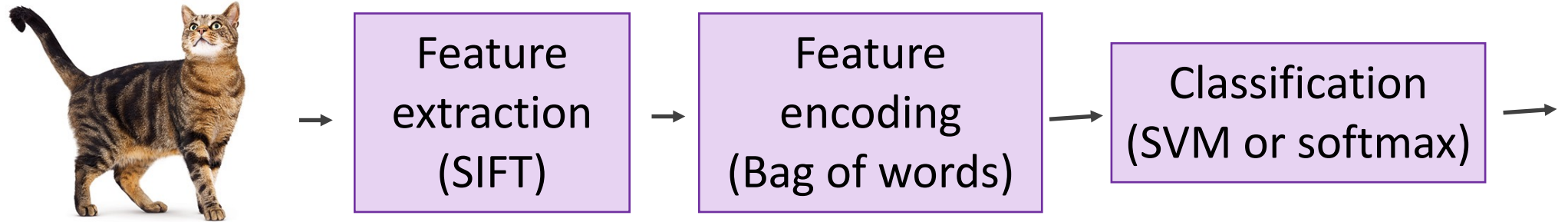


(b) After applying dropout.

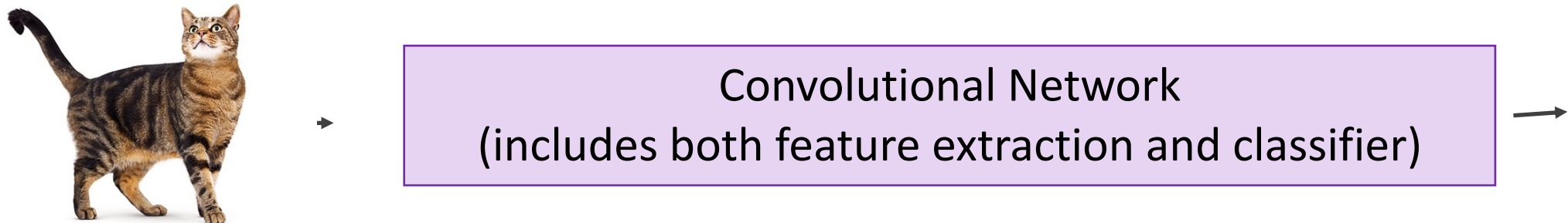
What is happening?



SIFT + FV + SVM (or softmax)



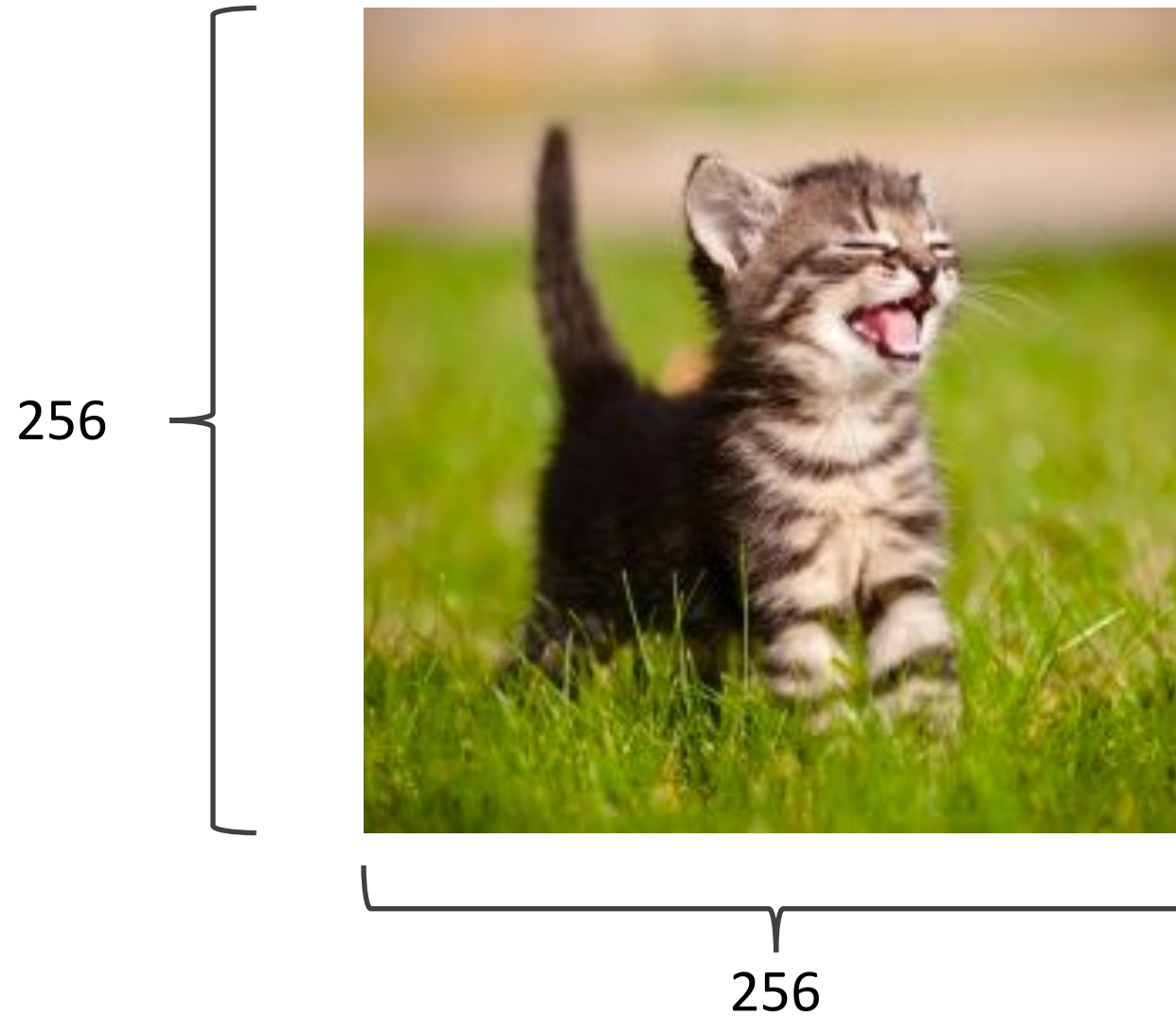
Deep Learning



Preprocessing and Data Augmentation



Preprocessing and Data Augmentation



Preprocessing and Data Augmentation

224x224



Preprocessing and Data Augmentation

224x224





True label: Abyssinian cat

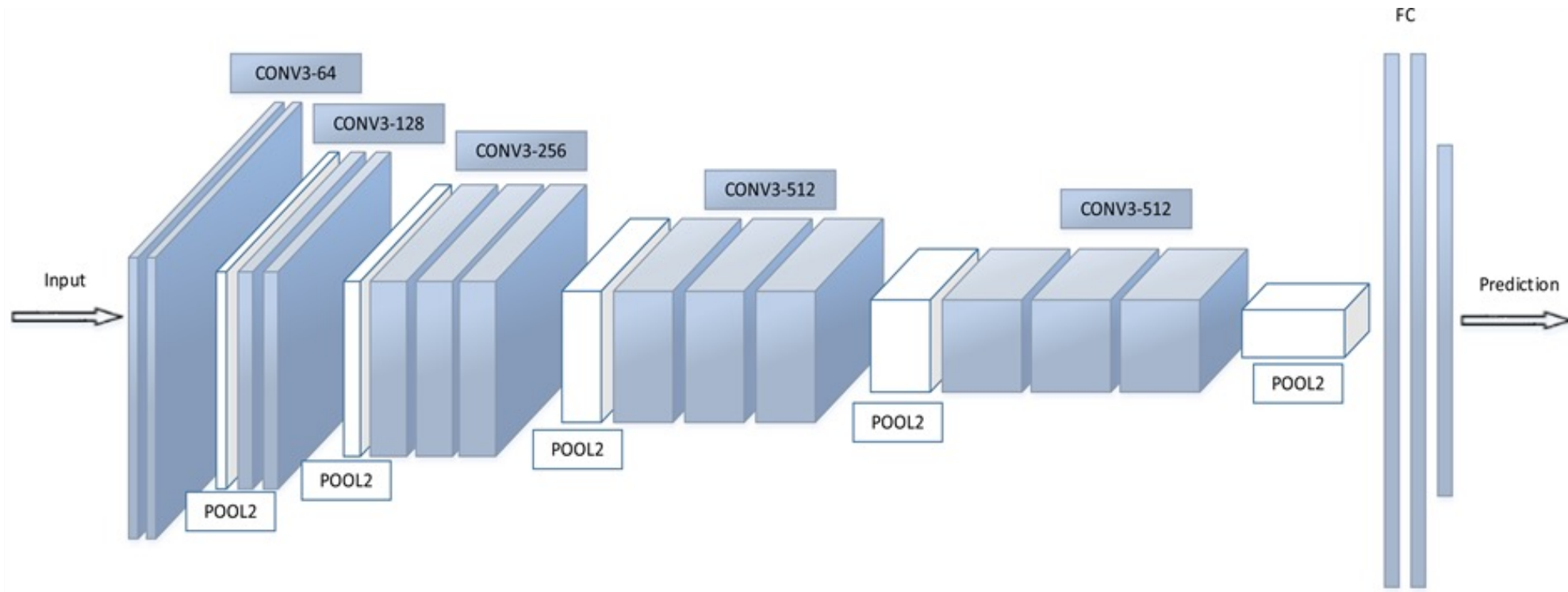
Other Important Aspects

- Using ReLUs instead of Sigmoid or Tanh
- Momentum + Weight Decay
- Dropout (Randomly sets Unit outputs to zero during training)
- GPU Computation!

Model	Top-1	Top-5
<i>Sparse coding [2]</i>	47.1%	28.2%
<i>SIFT + FVs [24]</i>	45.7%	25.7%
CNN	37.5%	17.0%

VGG Network

Top-5:

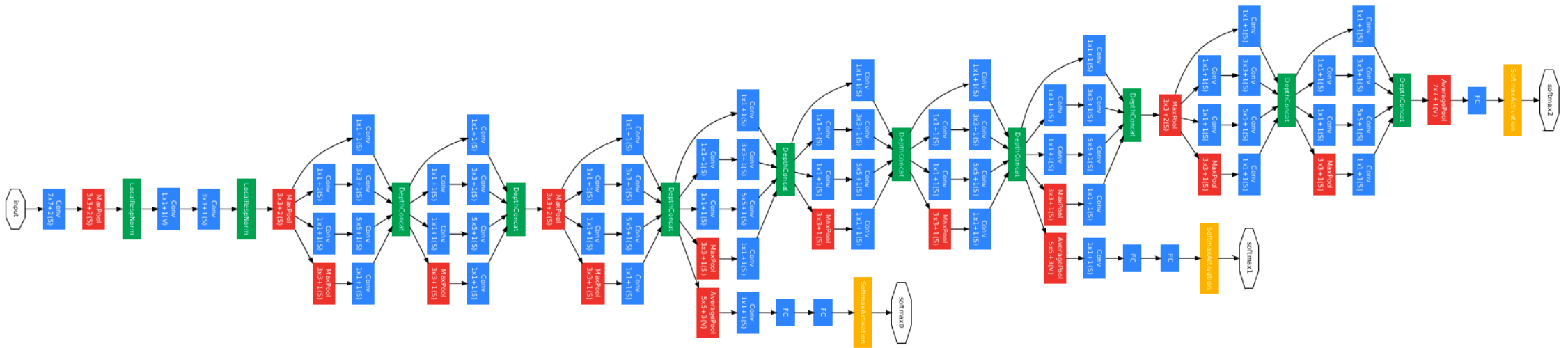


<https://github.com/pytorch/vision/blob/master/torchvision/models/vgg.py>

Simonyan and Zisserman, 2014.

<https://arxiv.org/pdf/1409.1556.pdf>

GoogLeNet

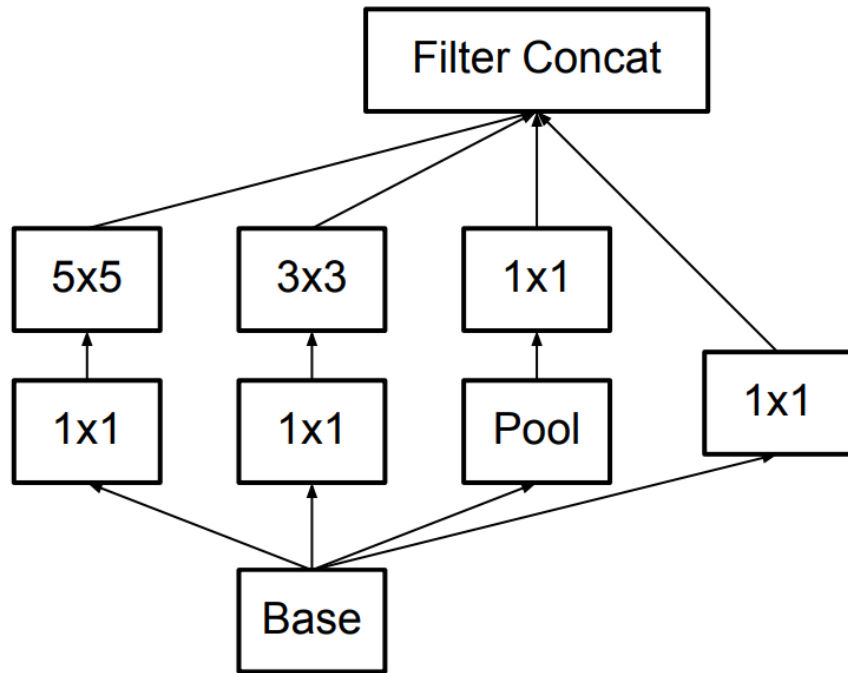


<https://github.com/kuangliu/pytorch-cifar/blob/master/models/googlenet.py>

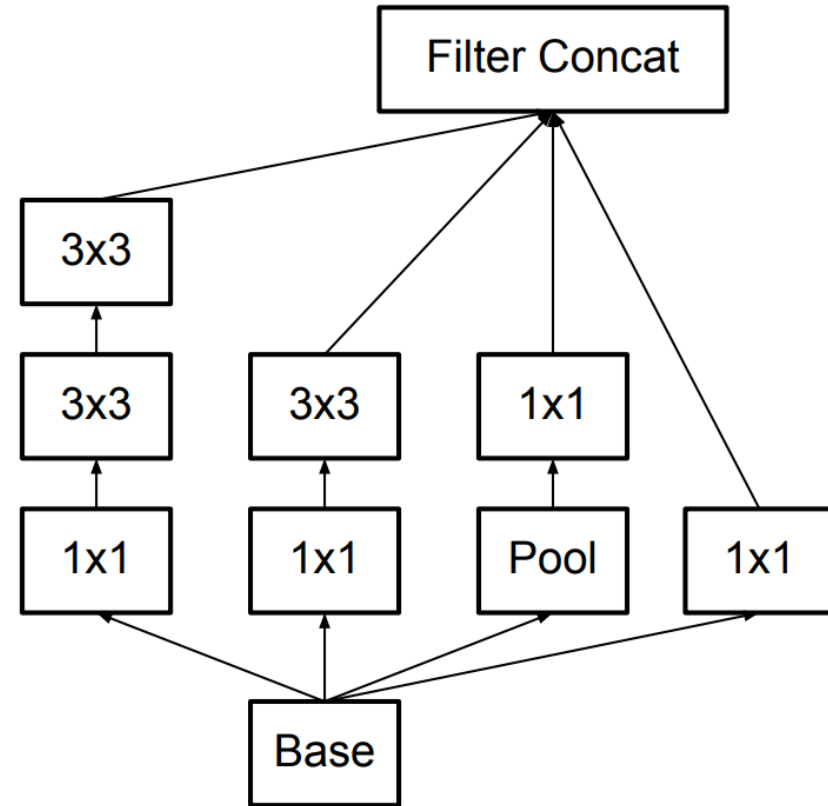
Szegedy et al. 2014

<https://www.cs.unc.edu/~wliu/papers/GoogLeNet.pdf>

Further Refinements – Inception v3, e.g.



GoogLeNet (Inceptionv1)



Inception v3

Batch Normalization Layer

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\dots m}\}$;

Parameters to be learned: γ, β

Output: $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{ mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{ mini-batch variance}$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{ normalize}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) \quad // \text{ scale and shift}$$

ResNet (He et al CVPR 2016)

Sorry, does not fit in slide.

<http://felixlaumon.github.io/assets/kaggle-right-whale/resnet.png>

<https://github.com/pytorch/vision/blob/master/torchvision/models/resnet.py>

Revolution of Depth

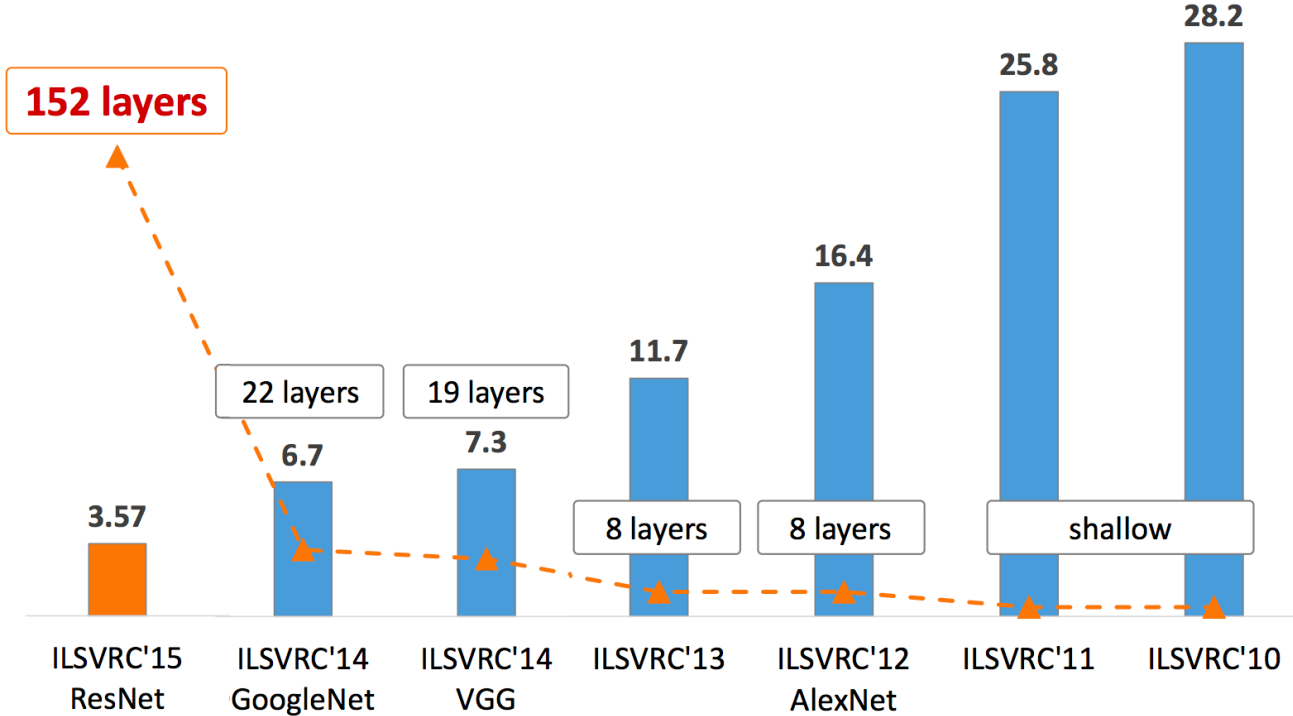
AlexNet, 8 layers
(ILSVRC 2012)



VGG, 19 layers
(ILSVRC 2014)



ResNet, 152 layers
(ILSVRC 2015)



Questions