Deep Learning for Vision & Language

Computer Vision I: Convolutional Neural Networks

RICE UNIVERSITY

About the class

- COMP 646: Deep Learning for Vision and Language
- Instructor: **Vicente** Ordóñez (Vicente Ordóñez Román)
- Website: https://www.cs.rice.edu/~vo9/deep-vislang
- Location: Herzstein Hall 210
- Times: Tuesdays and Thursdays
  from 4pm to 5:15pm
- Office Hours: Tuesdays 10am to 11am (DH3098)
- Teaching Assistants: **Arnold, Jefferson, Sangwon, Gaotian**
- Discussion Forum: Piazza (Sign-up Link on Rice Canvas and Class Website)

Teaching Assistants (TAs)

Jefferson Hernandez

Mondays 2:30pm
DH 3036

Sangwon Seo

Wednesdays 10am
DH 3002

Gaotian Wang

Wednesdays 3pm
DH 3036

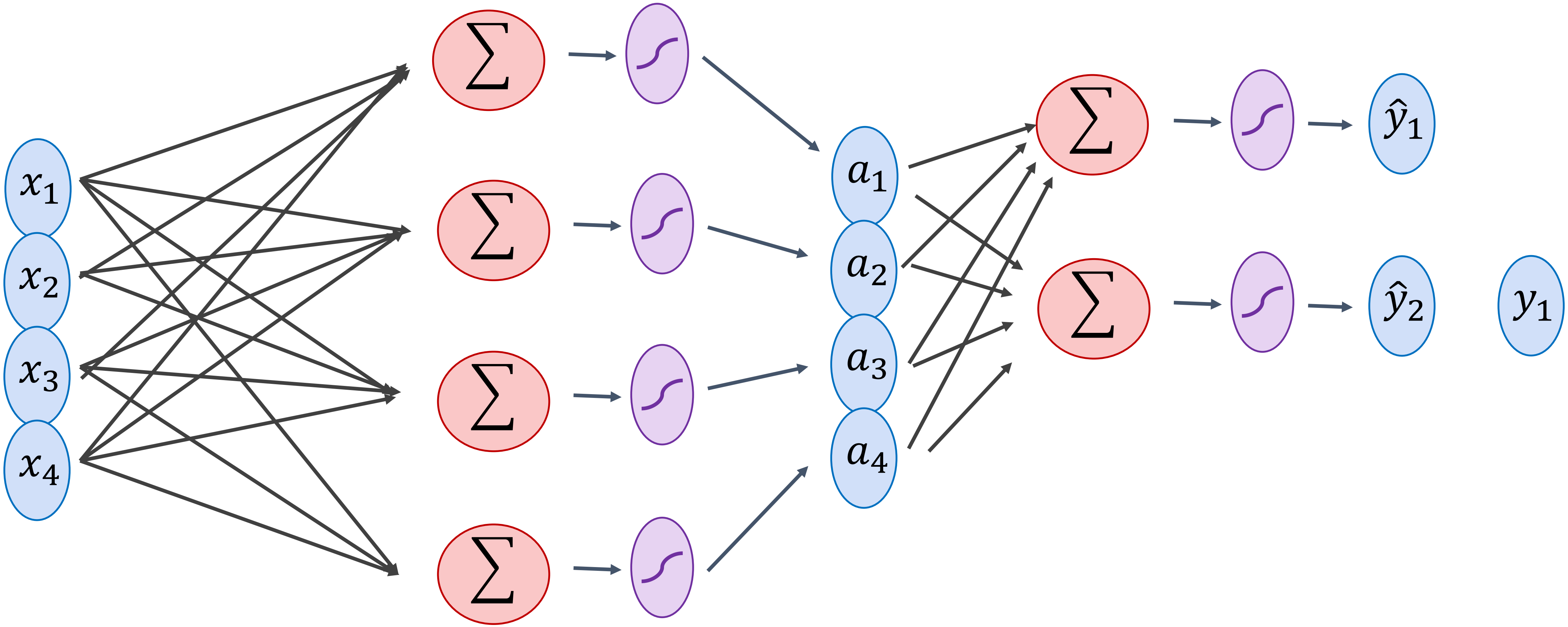
Arnold Kazadi

Thursdays 11am
DH 3036

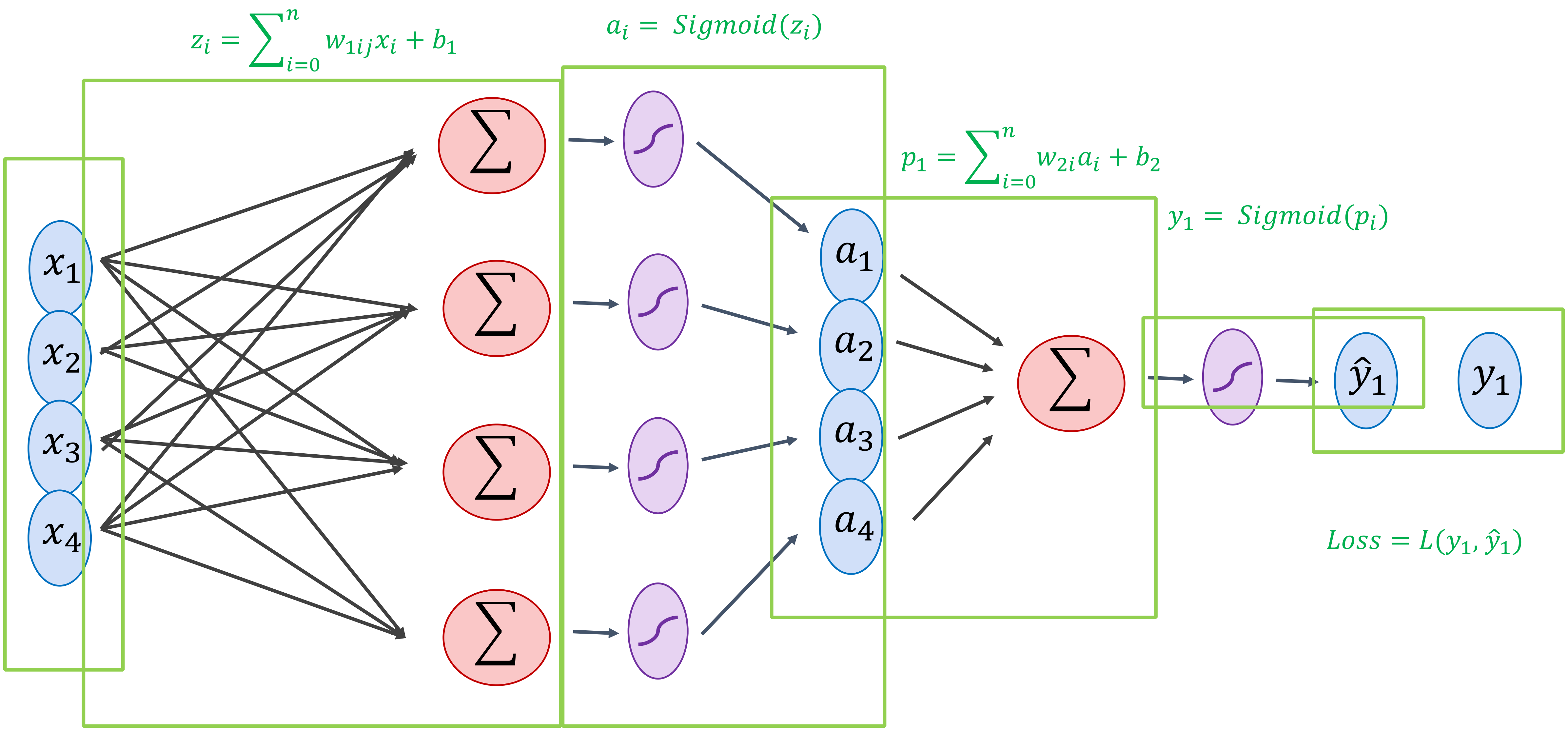
Assignment 1

- Due next Monday at midnight (No extension for any reason but especially not due to errors/outages in Google Colab)
  – Please submit early.

Forward pass (Forward-propagation)



Forward pass (Forward-propagation)



How to train the parameters?

$x_i = [x_{i1} \quad x_{i2} \quad x_{i3} \quad x_{i4}]$ $y_i = [1 \quad 0 \quad 0]$ $\hat{y}_i = [f_c \quad f_d \quad f_b]$

$$a_1 = sigmoid(w_{[1]}x^T + b_{[1]}^T)$$

$$a_2 = sigmoid(w_{[2]}a_1^T + b_{[2]}^T)$$

...

$$a_k = sigmoid(w_{[k]}a_{k-1}^T + b_{[i]}^T)$$

...

$$f = softmax(w_{[n]}a_{n-1}^T + b_{[n]}^T)$$

We can still use SGD

We need!

$$\frac{\partial l}{\partial w_{[k]ij}} \qquad \frac{\partial l}{\partial b_{[k]i}}$$

How to train the parameters?

$x_i = [x_{i1} \quad x_{i2} \quad x_{i3} \quad x_{i4}]$

$y_i = \quad [1 \quad 0 \quad 0]$

$\hat{y}_i = \quad [f_c \quad f_d \quad f_b]$

$$a_1 = sigmoid(w_{[1]}x^T + b_{[1]}^T)$$

$$a_2 = sigmoid(w_{[2]}a_1^T + b_{[2]}^T)$$

...

$$a_i = sigmoid(w_{[k]}a_{k-1}^T + b_{[k]}^T)$$

...

$$f = softmax(w_{[n]}a_{n-1}^T + b_{[n]}^T)$$

$$l = loss(f, y)$$

We can still use SGD

We need!

$$\frac{\partial l}{\partial w_{[k]ij}} \qquad \frac{\partial l}{\partial b_{[k]i}}$$

How to train the parameters?

$x_i = [x_{i1} \quad x_{i2} \quad x_{i3} \quad x_{i4}]$

$y_i = [1 \quad 0 \quad 0]$

$\hat{y}_i = [f_c \quad f_d \quad f_b]$

$$a_1 = sigmoid(w_{[1]}x^T + b_{[1]}^T)$$

$$a_2 = sigmoid(w_{[2]}a_1^T + b_{[2]}^T)$$

$$\dots$$

$$a_i = sigmoid(w_{[k]}a_{k-1}^T + b_{[k]}^T)$$

$$\dots$$

$$f = softmax(w_{[n]}a_{n-1}^T + b_{[n]}^T)$$

$$l = loss(f, y)$$

We can still use SGD

We need!

$$\boxed{\frac{\partial l}{\partial w_{[k]ij}}} \qquad \frac{\partial l}{\partial b_{[k]i}}$$

How to train the parameters?

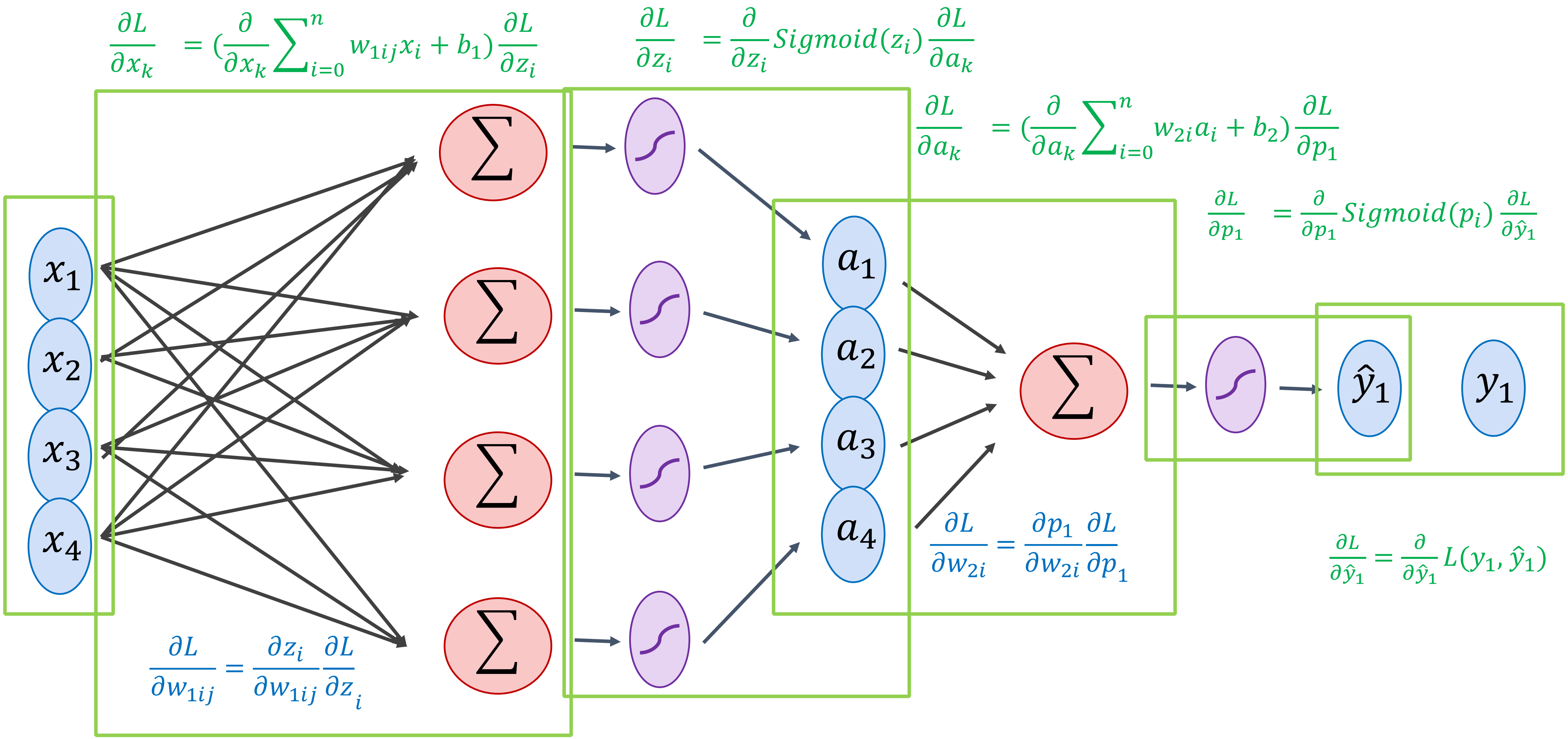
$x_i = [x_{i1} \quad x_{i2} \quad x_{i3} \quad x_{i4}]$ $y_i = [1 \quad 0 \quad 0]$ $\hat{y}_i = [f_c \quad f_d \quad f_b]$

$$a_1 = sigmoid(w_{[1]}x^T + b_{[1]}^T)$$
$$a_2 = sigmoid(w_{[2]}a_1^T + b_{[2]}^T)$$
$$\dots$$
$$a_i = sigmoid(w_{[k]}a_{k-1}^T + b_{[k]}^T)$$
$$\dots$$
$$f = softmax(w_{[n]}a_{n-1}^T + b_{[n]}^T)$$

$$l = loss(f, y)$$

$$\frac{\partial l}{\partial w_{[k]ij}} = \frac{\partial l}{\partial a_{n-1}} \frac{\partial a_{n-1}}{\partial a_{n-2}} \cdots \frac{\partial a_{k-2}}{\partial a_{k-1}} \frac{\partial a_{k-1}}{\partial w_{[k]ij}}$$

Backward pass (Back-propagation)



```python
# This class combines Softmax + Negative-log likelihood loss.
# Similar to the previous lab, but this implementation works for
# batches of inputs and not just individual input vectors.
# Here "inputs" is batchSize x sizePredictionScores, and
#      "labels" is a vector of size batchSize.
class toynn_CrossEntropyLoss(object):

    # Forward pass: -log softmax(input_{label})
    def forward(self, scores, labels):

        # 1. Computing the softmax: exp(x) / sum (exp(x))
        max_val = scores.max()  # This is to avoid variable overflows.
        exp_inputs = (scores - max_val).exp()
        # This is different than in the previous lab. Avoiding for loops here.
        denominators = exp_inputs.sum(1).repeat(scores.size(1), 1).t()
        self.predictions = torch.mul(exp_inputs, 1 / denominators)

        # 2. Computing the loss: -log(y_label).
        # Check what gather does. Just avoiding another for loop here.
        return -self.predictions.log().gather(1, labels.view(-1, 1)).mean()

    # Backward pass: y_hat - y
    def backward(self, scores, labels):

        # Here we avoid computing softmax again in backward pass.
        grad_inputs = self.predictions.clone()

        # Ok, Here we will use a for loop (but it is avoidable too).
        for i in range(0, scores.size(0)):
            grad_inputs[i][labels[i]] = grad_inputs[i][labels[i]] - 1

        return grad_inputs
```

Softmax + Negative Log Likelihood

$$\ell = -\log\left(\frac{\exp(a_{label})}{\sum_{k=1}^{10} \exp(a_k)}\right)$$

$$\frac{\partial \ell}{\partial a_i} = \hat{y}_i - y_i$$

```python
class toynn_Linear(object):
    def __init__(self, numInputs, numOutputs):
        # Allocate tensors for the weight and bias parameters.
        self.weight = torch.Tensor(numInputs, numOutputs).normal_(0, 0.01)
        self.weight_grads = torch.Tensor(numInputs, numOutputs)
        self.bias = torch.Tensor(numOutputs).zero_()
        self.bias_grads = torch.Tensor(numOutputs)

    # Forward pass, inputs is a matrix of size batchSize x numInputs.
    # Notice that compared to the previous assignment, each input vector
    # is a row in this matrix.
    def forward(self, inputs):
        # This one needs no change, it just becomes
        # a matrix x matrix multiplication
        # as opposed to just vector x matrix multiplication as we had before.
        return torch.matmul(inputs, self.weight) + self.bias

    # Backward pass, in addition to compute gradients for the weight and bias.
    # It has to compute gradients with respect to inputs.
    def backward(self, inputs, scores_grads):
        self.weight_grads = torch.matmul(inputs.t(), scores_grads)
        self.bias_grads = scores_grads.sum(0)
        return torch.matmul(scores_grads, self.weight.t())
```

Linear layer

```python
class toynn_ReLU(object):

    # Forward operation: f(x_i) = max(0, x_i)
    def forward(self, inputs):
        outputs = inputs.clone()
        outputs[outputs < 0] = 0
        return outputs

    # Make sure the backward pass is absolutely clear.
    def backward(self, inputs, outputs_grad):
        inputs_grad = outputs_grad.clone() # 1 * previous_grads
        inputs_grad[inputs < 0] = 0  # or zero.
        return inputs_grad
```

ReLU
layer

Two-layer Neural Network – Forward Pass

```
# Setup the input variable x.
img, label = trainset[0]
x = img.view(1, 1 * 28 * 28)

# Setup the number of inputs, hidden neurons, and outputs.
nInputs = 1 * 28 * 28
nHidden = 512
nOutputs = 10

# Create the model here.
linear_fn1 = toynn_Linear(nInputs, nHidden)
relu_fn = toynn_ReLU()
linear_fn2 = toynn_Linear(nHidden, nOutputs)

# Make predictions.
x = linear_fn1.forward(x)
x = relu_fn.forward(x)
x = linear_fn2.forward(x)

# Show the prediction scores for each class.
# Yes, pytorch tensors already come with a softmax function.
# We need it here because we hard-coded the softmax inside
# the loss function.
print(x.softmax(dim = 1))
```

Two-layer Neural Network – Backward Pass

```
# Create the model here.
linear_fn1 = toynn_Linear(nInputs, nHidden)
relu_fn = toynn_ReLU()
linear_fn2 = toynn_Linear(nHidden, nOutputs)
loss_fn = toynn_CrossEntropyLoss()

# Make predictions (forward pass).
a = linear_fn1.forward(x)
z = relu_fn.forward(a)
yhat = linear_fn2.forward(z)

# Compute loss.
loss = loss_fn.forward(yhat, label)
yhat_grads = loss_fn.backward(yhat, label)

# Compute gradients (backward pass).
z_grads = linear_fn2.backward(z, yhat_grads)
a_grads = relu_fn.backward(a, z_grads)
x_grads = linear_fn1.backward(x, a_grads)

# Update parameters:
learningRate = 0.2
linear_fn1.weight.add_(-learningRate, linear_fn1.weight_grads)
linear_fn1.bias.add_(-learningRate, linear_fn1.bias_grads)
linear_fn2.weight.add_(-learningRate, linear_fn2.weight_grads)
linear_fn2.bias.add_(-learningRate, linear_fn2.bias_grads)
```

Automatic Differentiation

You only need to write code for the forward pass,
backward pass is computed automatically.

Pytorch (Facebook -- mostly):	https://pytorch.org/
Tensorflow (Google -- mostly):	https://www.tensorflow.org/
MXNet (Amazon -- mostly):	https://mxnet.apache.org/versions/1.9.0/

Defining a Model in Pytorch (Two Layer NN)

```
import torch.nn as nn
import torch.nn.functional as F

class TwoLayerNN(nn.Module):
  def __init__(self):
    super(TwoLayerNN, self).__init__()

    self.linear1 = nn.Linear(1 * 28 * 28, 512)
    self.linear2 = nn.Linear(512, 10)

  def forward(self, x):
    x = x.view(batchSize, 1 * 28 * 28)
    z = F.relu(self.linear1(x))
    return self.linear2(z)
```

1. Creating Model, Loss, Optimizer

```
# Create the model.
model = TwoLayerNN()
loss_fn = nn.CrossEntropyLoss()
```

```
# Define a learning rate.
learningRate = 5e-2

# Optimizer.
optimizer = optim.SGD(model.parameters(), lr = learningRate,
                      momentum = 0.9, weight_decay = 1e-4)
```

2. Running forward and backward on a batch

```
# Forward pass. (Prediction stage)
scores = model(inputs)
loss = loss_fn(scores, labels)
```

```
# Zero the gradients in the network.
optimizer.zero_grad()

#Backward pass. (Gradient computation stage)
loss.backward()

# Parameter updates (SGD step) -- if done with torch.optim!
optimizer.step()
```

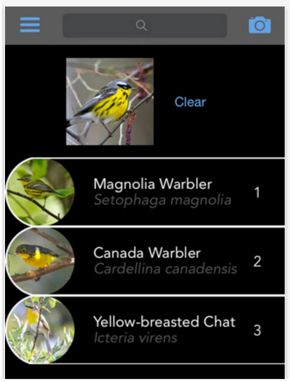
Today: Computer Vision

- Why is it hard?
- Image Processing
- The Convolutional Operator: Filtering
- Convolutional Neural Networks

kaggle

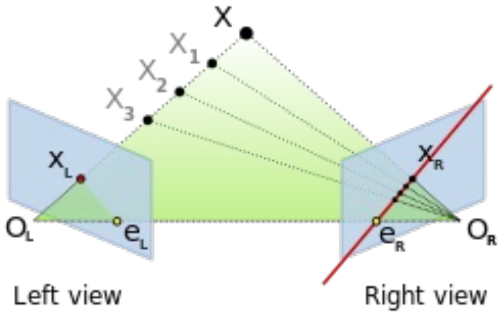
Create an algorithm to distinguish dogs from cats

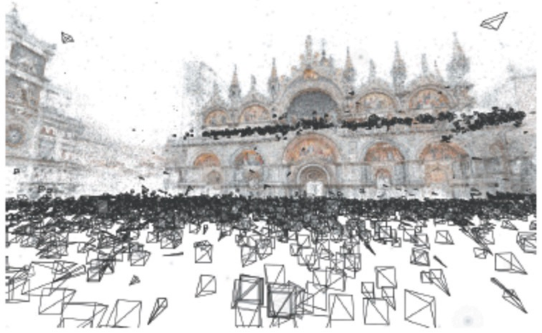
Birdsnap

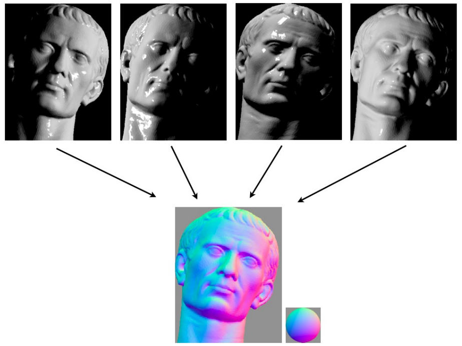


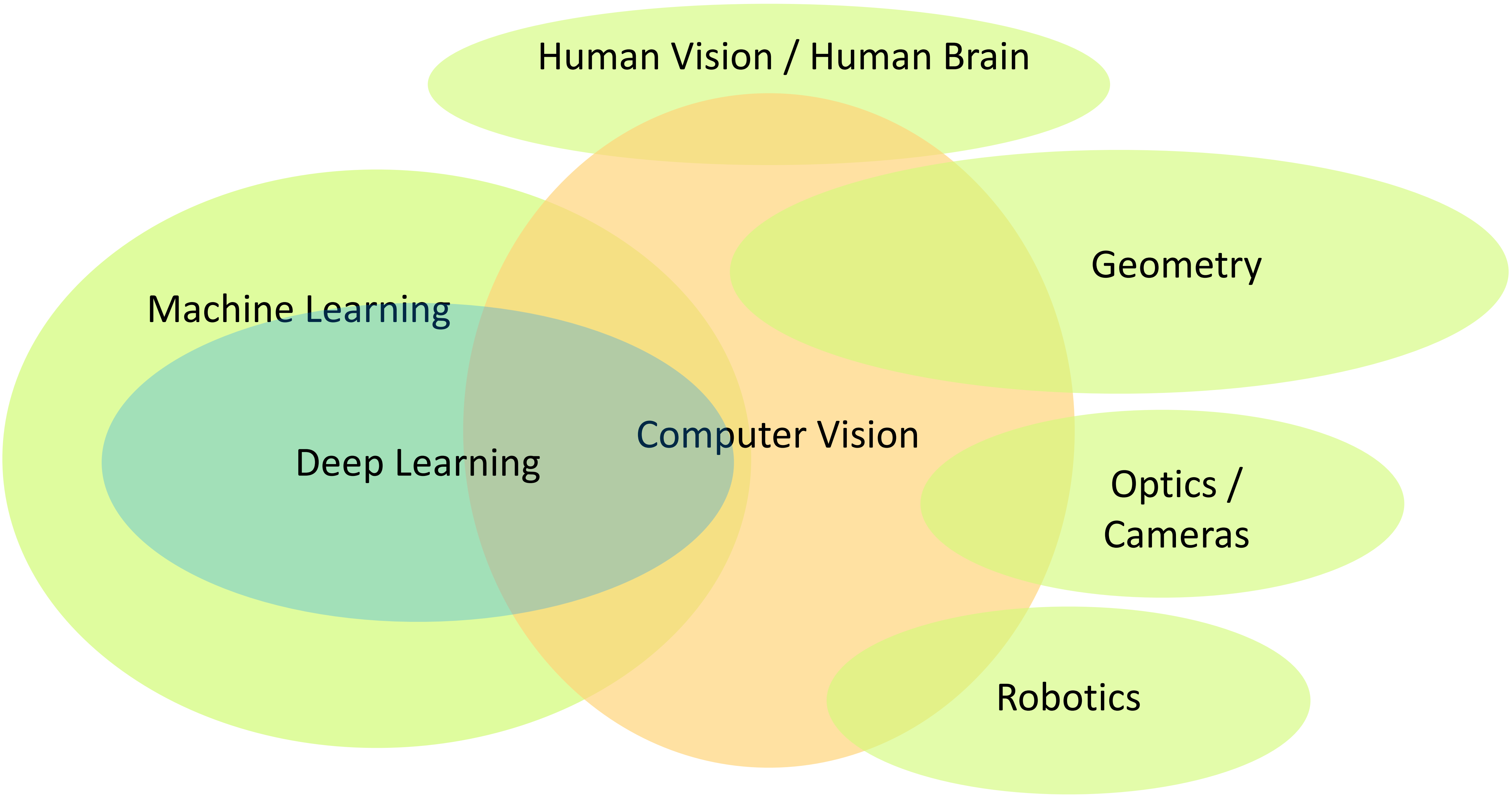
Face Detection in Cameras

Computer Vision



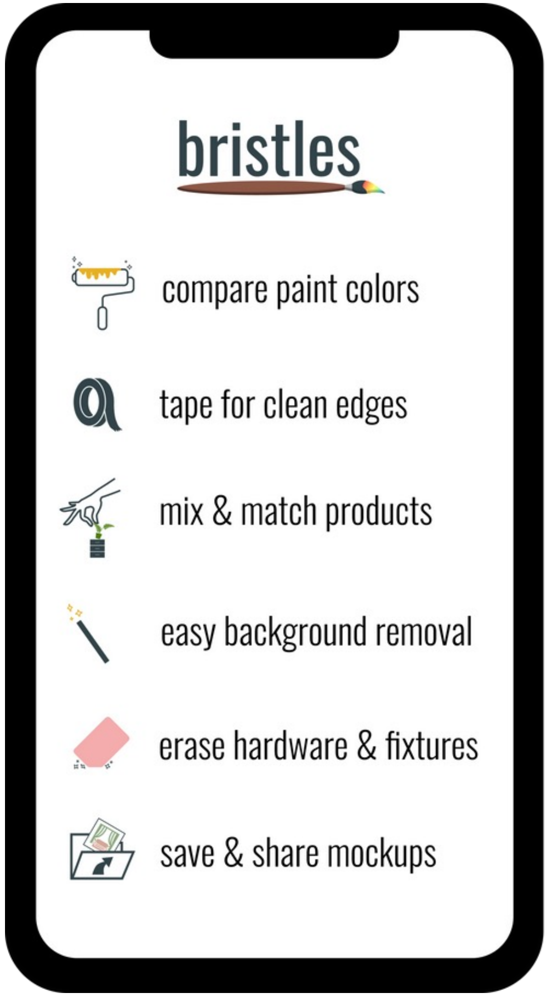


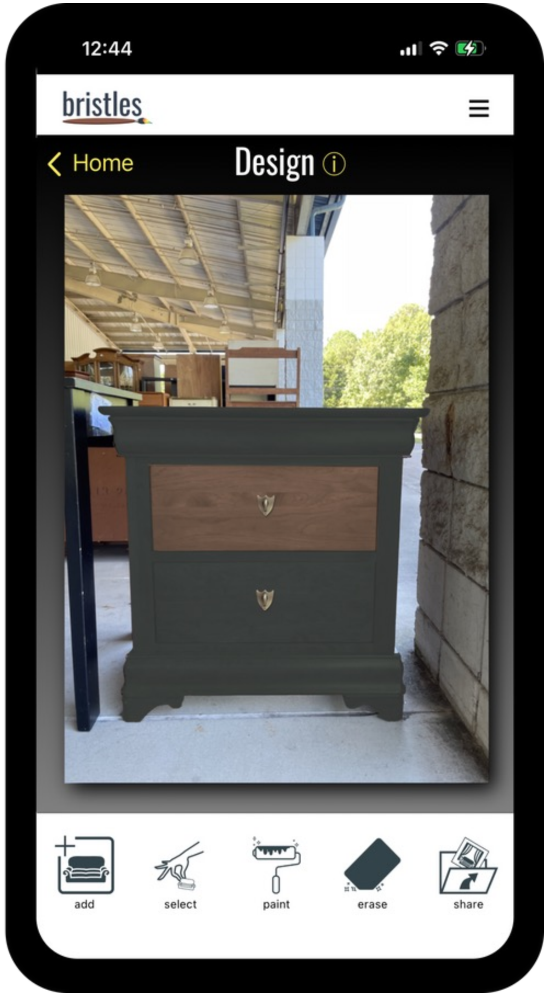



Human Vision / Human Brain
Geometry
Machine Learning
Computer Vision
Deep Learning
Optics /
Cameras
Robotics

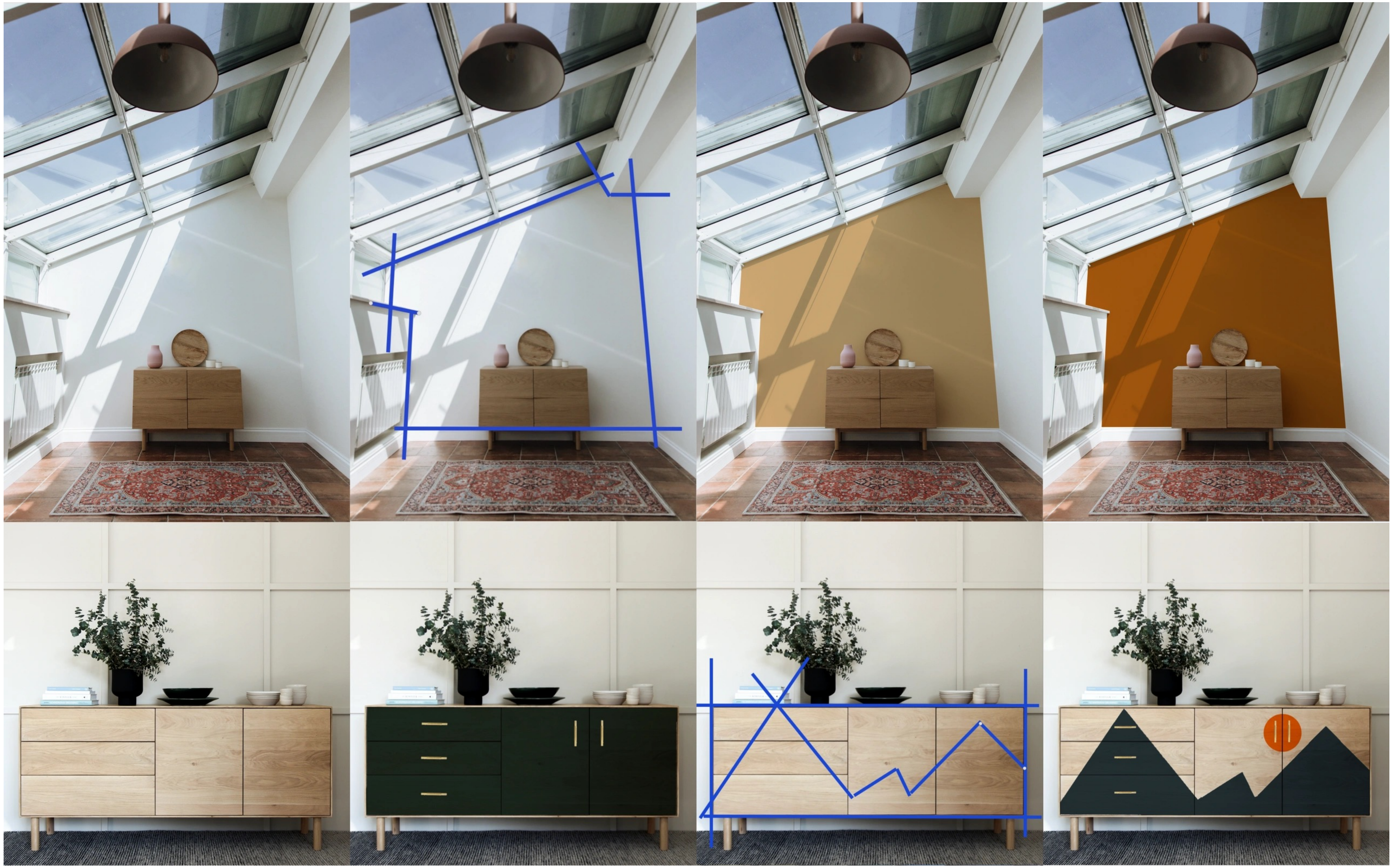
Who is using Computer Vision?

- Facebook – Oculus VR, Image Search, Image tagging, Content filtering, Instagram, etc.
- Google/Alphabet – Waymo, DeepMind, Image Search, Google Earth/Maps, Street View, Google Photos, etc.
- Adobe – Photoshop, Premiere, Lightroom, etc.
- Snap Inc – Snapchat, Smart Goggles, Filters, Face Detection, Style Transfer, etc.
- eBay Inc – Product Search, Product Matching, Content Filtering, Duplicate Removal, etc.
- Amazon – Warehouse robotics, Smart Stores, Product Search.
- IBM – Image Retrieval, Medical Applications, Product Quality.
- Microsoft – Hololens, Optical Character Recognition (OCR), Face Detection, Cloud Services.
- Apple – Face Verification, Enhanced cameras and chips for image processing.



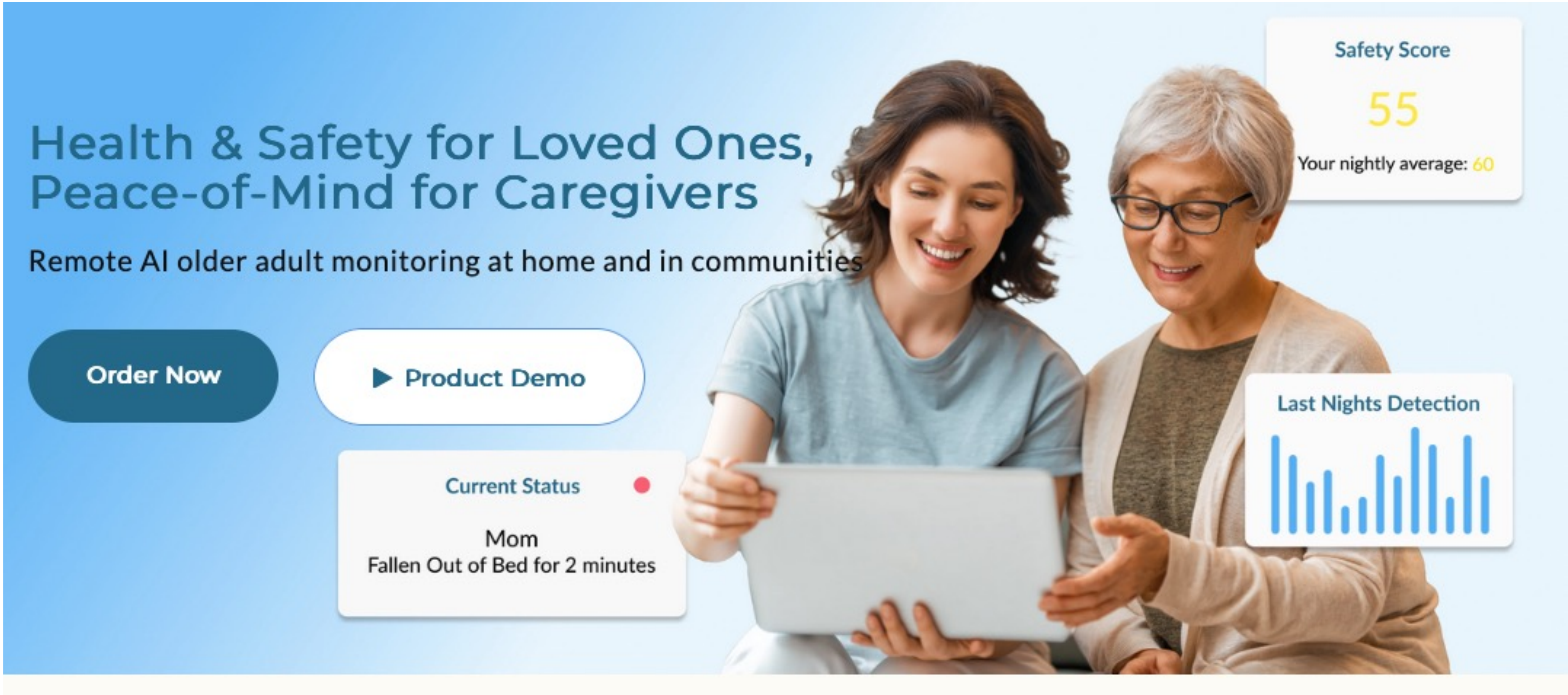


https://bristles.ai/



https://bristles.ai/

MercuryAlert



https://www.mercuryalert.ai/

PHIAR

Phiar.ai (now part of Google)

Images

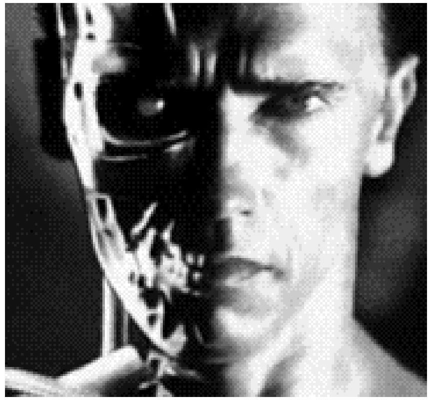
- Can be viewed as a matrix with pixel values

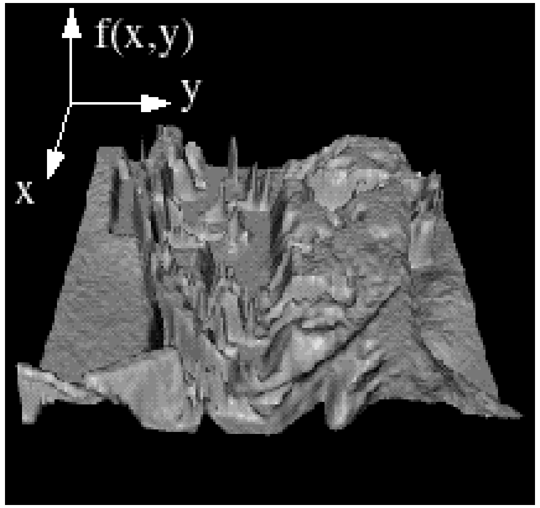
0	3	2	5	4	7	6	9	8
3	0	1	2	3	4	5	6	7
2	1	0	3	2	5	4	7	6
5	2	3	0	1	2	3	4	5
4	3	2	1	0	3	2	5	4
7	4	5	2	3	0	1	2	3
6	5	4	3	2	1	0	3	2
9	6	7	4	5	2	3	0	1
8	7	6	5	4	3	2	1	0

Images

- Or as a function in a 2D domain

$$z = f(x, y)$$

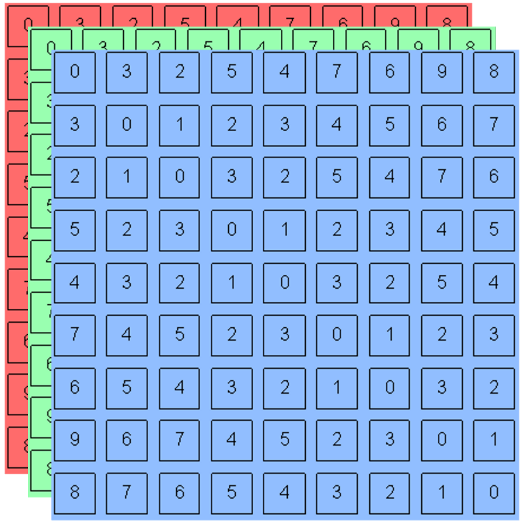




Color Images

- Can be viewed as tensors (3-dimensional arrays)

T =

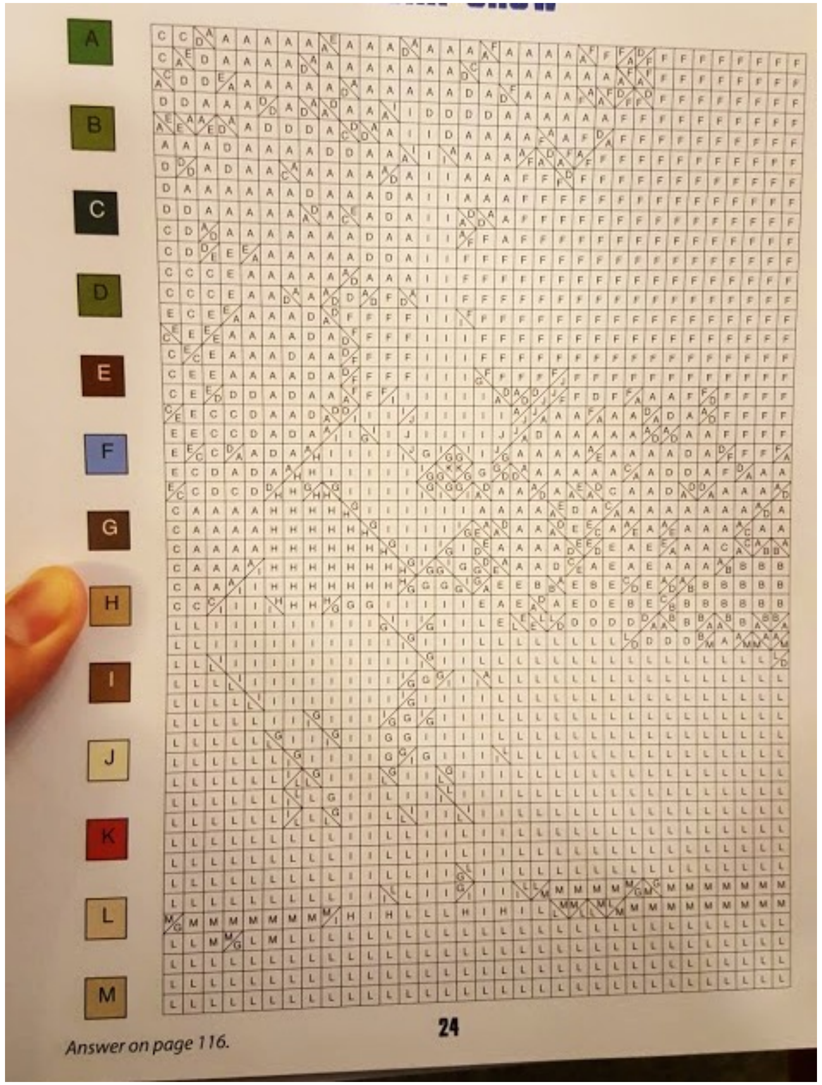


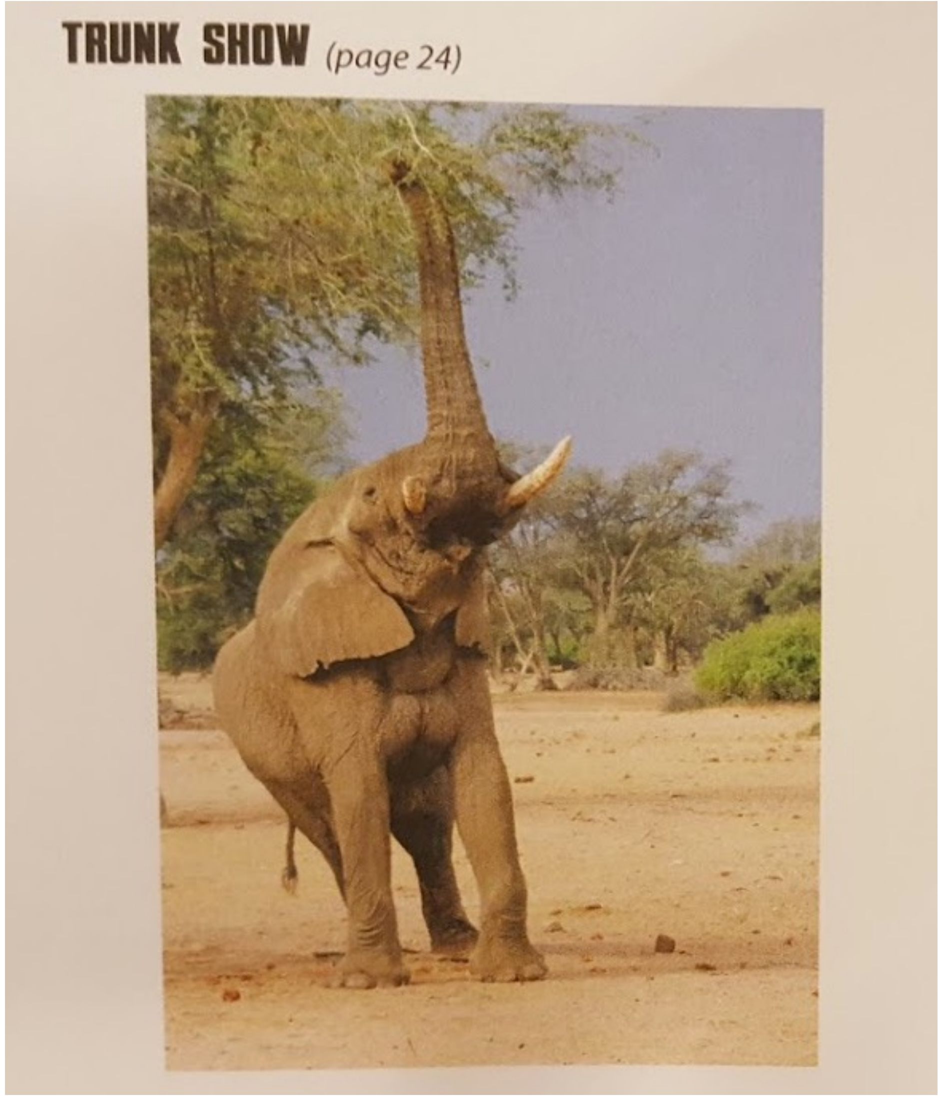
sizeof(T) = 3 x height x width

Channels are usually RGB: Red, Green, and Blue

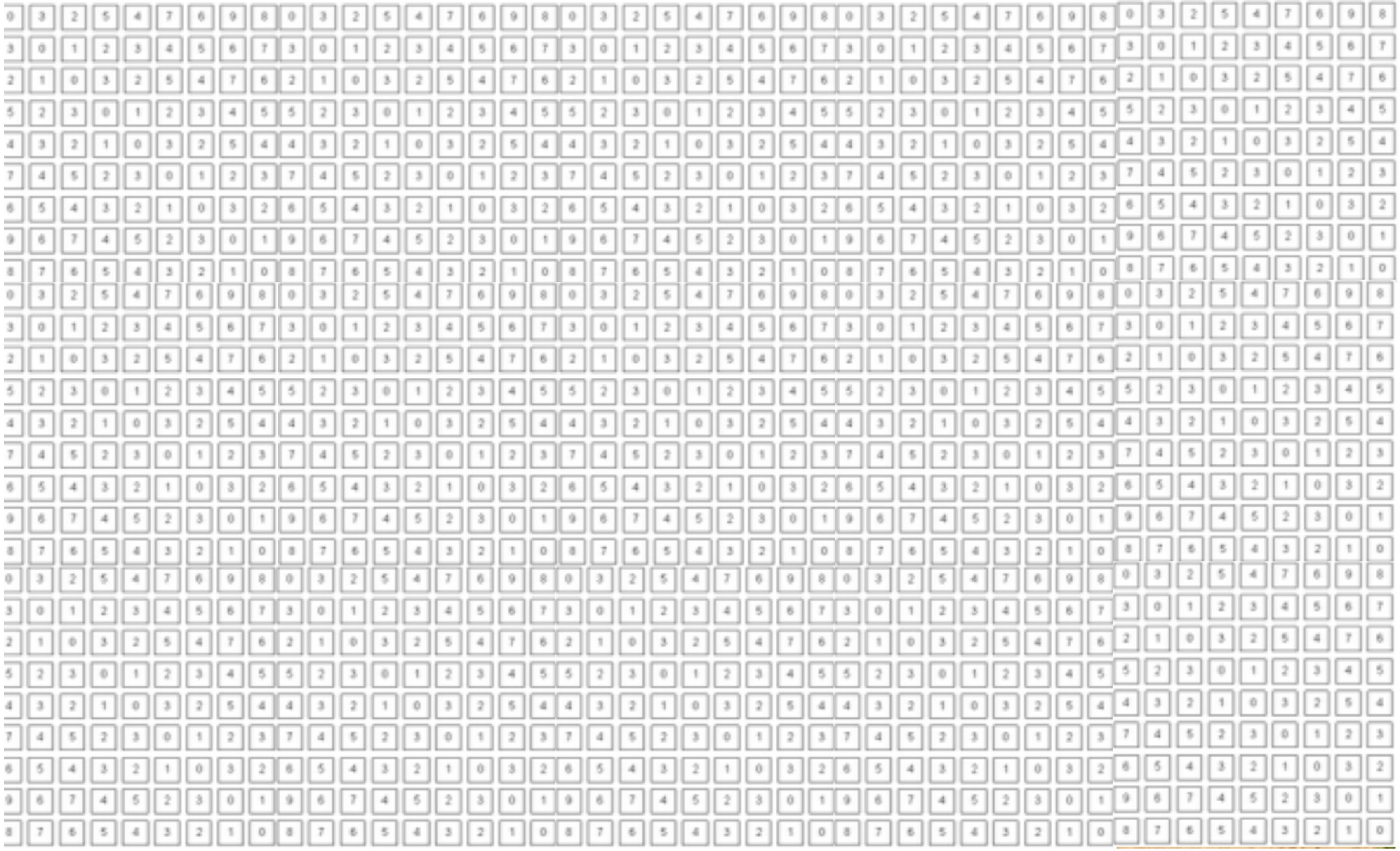
Other color spaces: HSV, HSL, LUV, XYZ, Lab, CMYK, etc

Why is it hard?



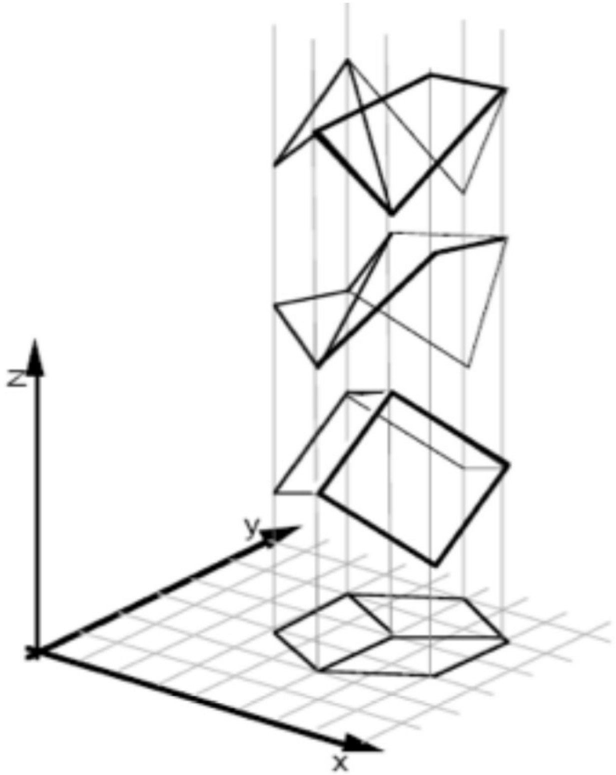


This is just as hard for computers



Why is Computer Vision hard?

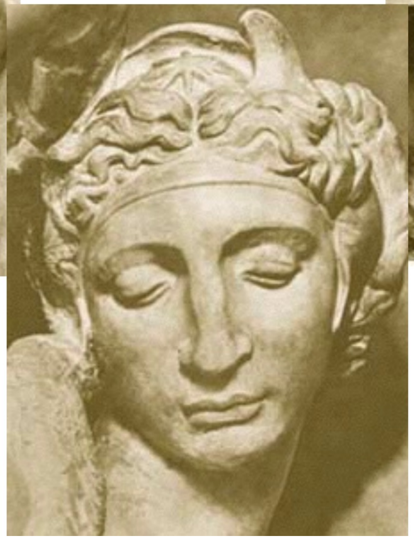
Ambiguities due to viewpoints



[Sinha and Adelson 1993]

Why is Computer Vision hard?

Ambiguities due to viewpoints

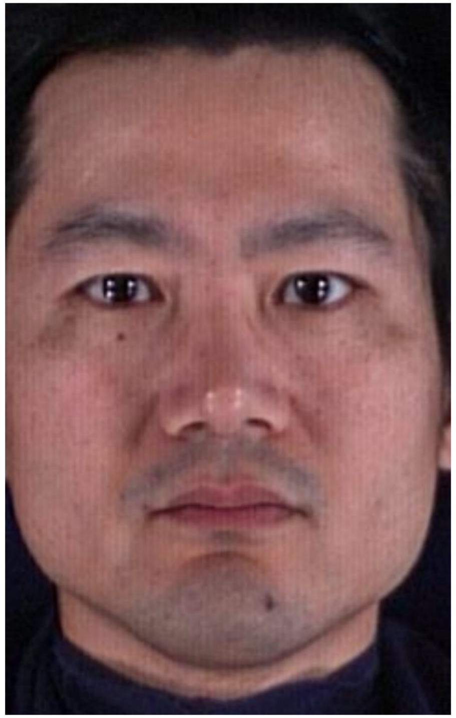




Michelangelo 1475-1564

slide by Fei Fei, Fergus & Torralba

Why is Computer Vision hard?

Issues with Illumination



slide credit: S. Ullman

Why is Computer Vision hard?

Background clutter

Why is Computer Vision hard?

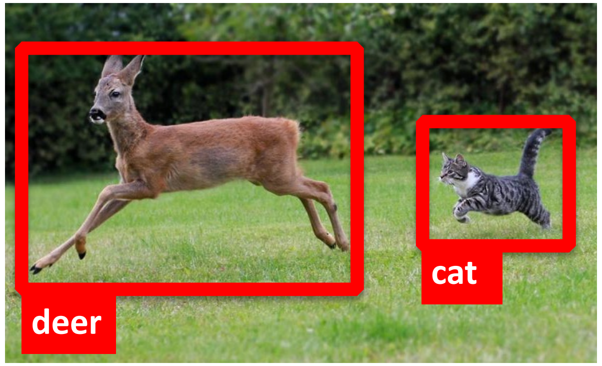
Intra-class variation

slide by Fei-Fei, Fergus & Torralba

Computer Vision vs Image Processing

- Computer Vision: Image → Knowledge

→



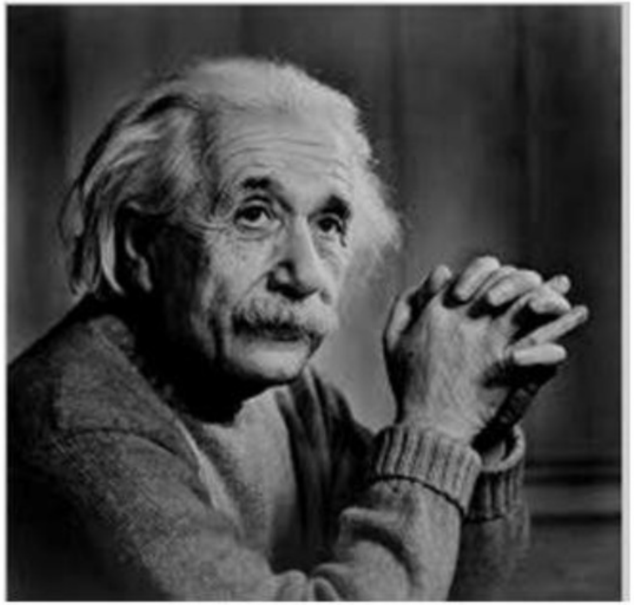
Computer Vision vs Image Processing

- Image Processing: Image ⟶ Image

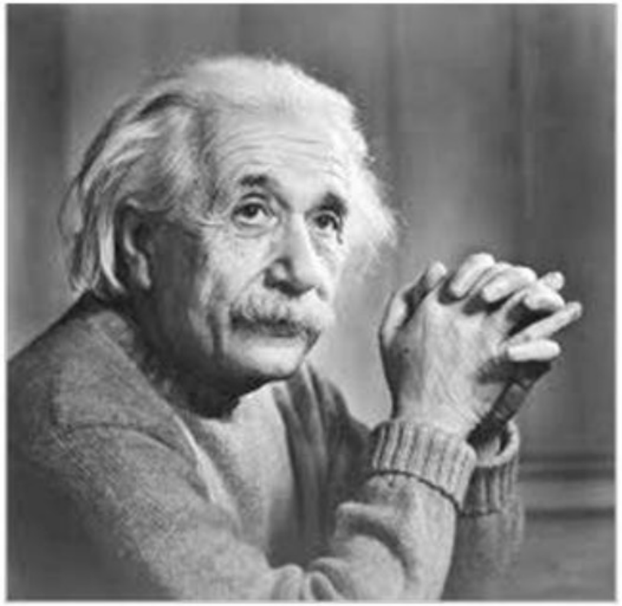
⟶

Basic Image Processing

I



αI



$\alpha > 1$

Primer on Image Processing: https://bit.ly/3lGEdwv

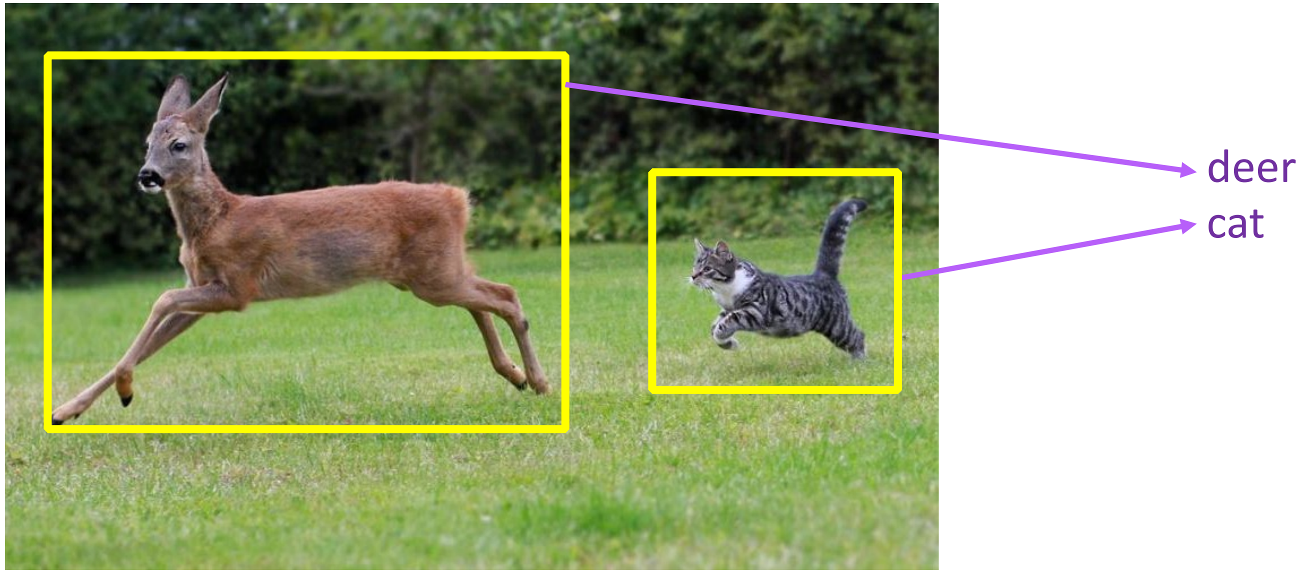
Common tasks in Computer Vision

Image tagging

deer
cat
trees
grass

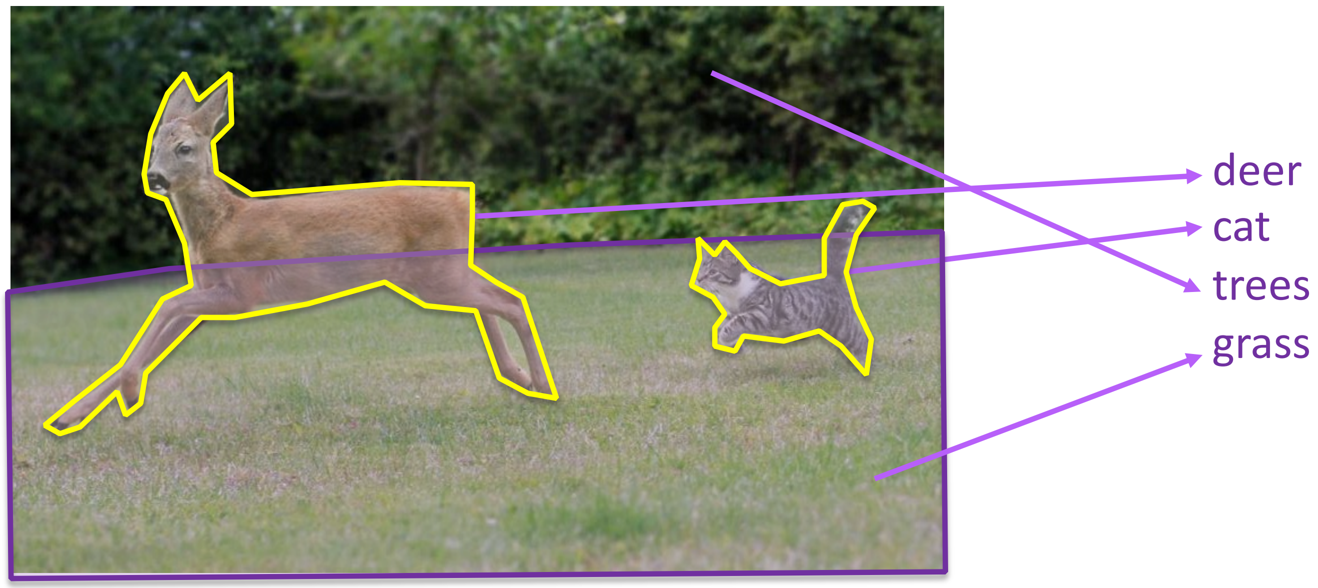
Common tasks in Computer Vision

Object detection



Common tasks in Computer Vision

Semantic segmentation



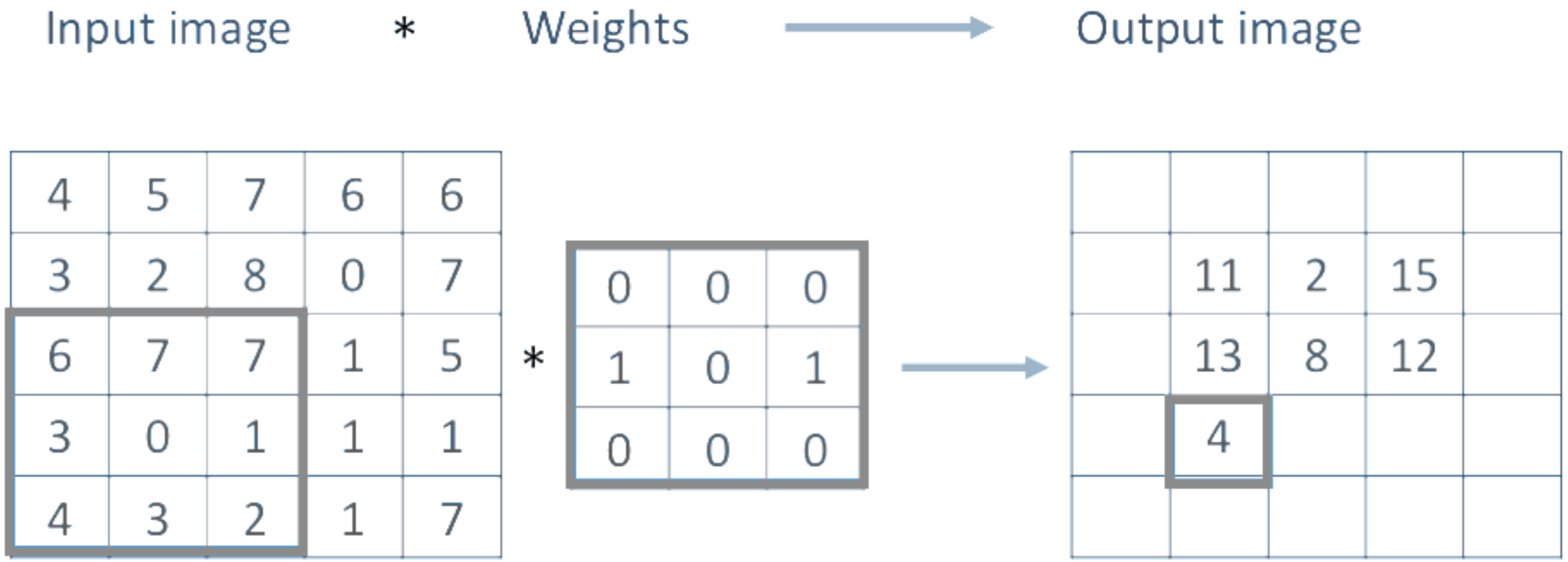
This class -> Vision and Language Tasks!

Reasoning about Language!

a cat is chasing a young deer

Most important operation for Computer Vision

- The Convolution Operation



http://www.cs.virginia.edu/~vicente/recognition/animation.gif

Most important operation for Computer Vision

- The Convolution Operation

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	
	4			

Convolutional filter
Convolutional kernel
Filter
Kernel

Most important operation for Computer Vision

- The Convolution Operation

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

$f(x, y)$

*

0	0	0
1	0	1
0	0	0

$k(x, y)$

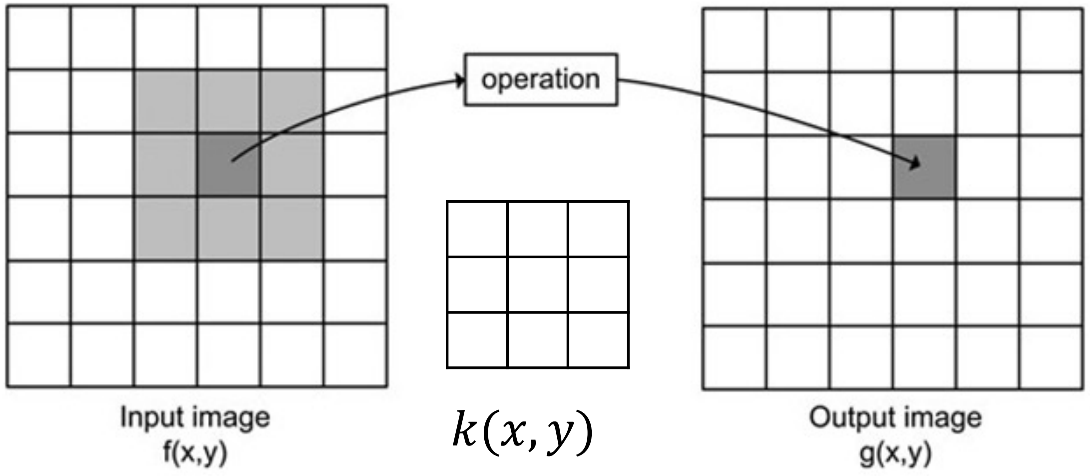
→

	11	2	15	
	13	8	12	
	4			

$g(x, y)$

$$g(x, y) = \sum_{v} \sum_{u} k(u, v) f(x - u, y - v)$$

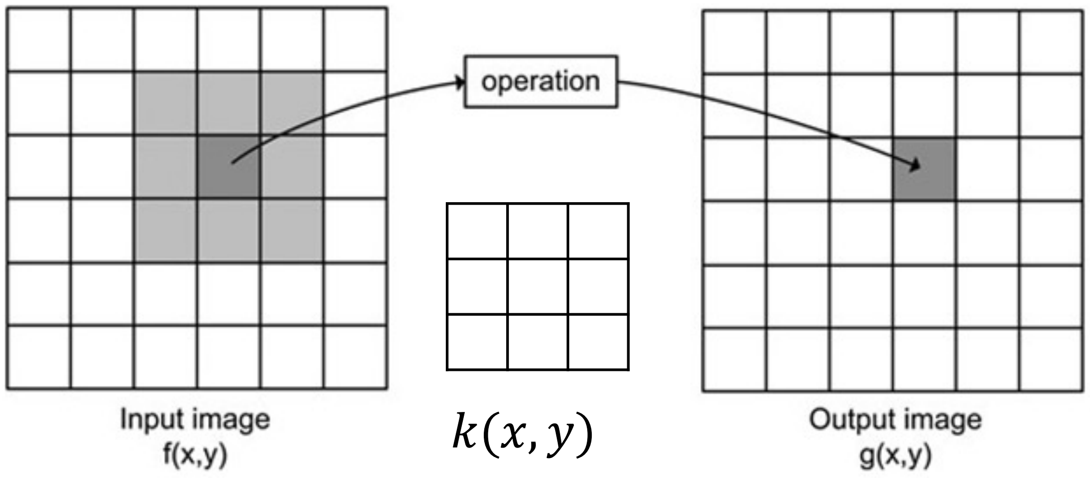
Image filtering: Convolution operator
e.g. mean filter



$$k(x,y) = \begin{array}{|c|c|c|}\hline 1/9 & 1/9 & 1/9 \\ \hline 1/9 & 1/9 & 1/9 \\ \hline 1/9 & 1/9 & 1/9 \\ \hline \end{array}$$

Image Credit: http://what-when-how.com/introduction-to-video-and-image-processing/neighborhood-processing-introduction-to-video-and-image-processing-part-1/

Image filtering: Convolution operator
e.g. mean filter



$$k(x, y) = \begin{bmatrix} 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \\ 1/9 & 1/9 & 1/9 \end{bmatrix}$$

Image Credit: http://what-when-how.com/introduction-to-video-and-image-processing/neighborhood-processing-introduction-to-video-and-image-processing-part-1/

Example: box filter

$$\mathrm{g}[\cdot\,,\cdot\,]$$

$$\frac{1}{9}\begin{array}{|c|c|c|}\hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline 1 & 1 & 1 \\ \hline\end{array}$$

Slide credit: David Lowe (UBC)

Image filtering

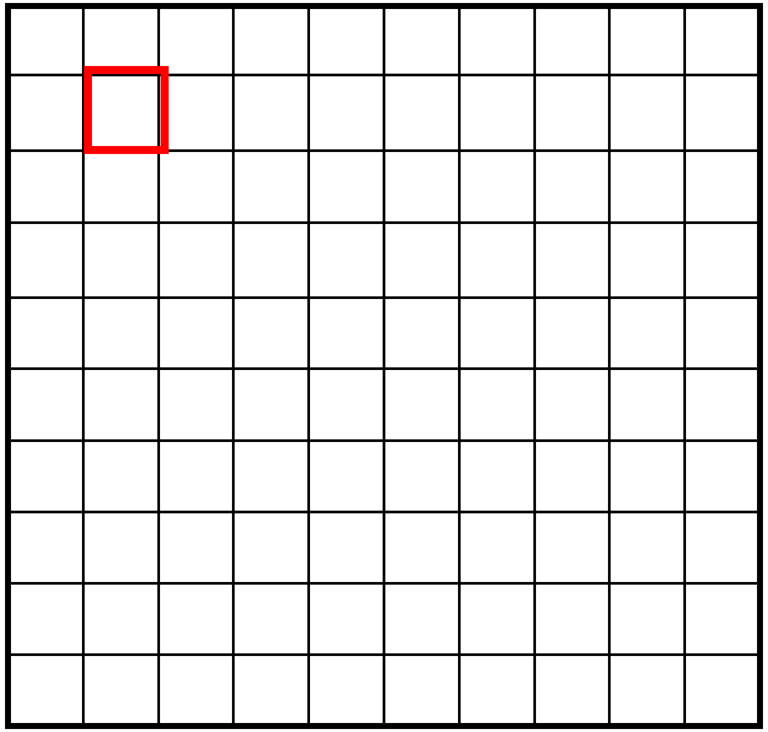
$g[\cdot,\cdot]\ \frac{1}{9}$

1	1	1
1	1	1
1	1	1

$f[.,.]$

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

$h[.,.]$



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

Credit: S. Seitz

Image filtering

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

$f[.,.]$

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

$h[.,.]$

	0	10							

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

Credit: S. Seitz

Image filtering

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

$f[.,.]$

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

$h[.,.]$

	0	10	20						

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

Credit: S. Seitz

Image filtering

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

$f[.,.]$

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

$h[.,.]$

	0	10	20	30					

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

Credit: S. Seitz

Image filtering

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

$f[.,.]$

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

$h[.,.]$

	0	10	20	30	30				

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

Credit: S. Seitz

Image filtering

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

$f[.,.]$

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

$h[.,.]$

	0	10	20	30	30				
				?					

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

Credit: S. Seitz

Image filtering

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

1	1	1
1	1	1
1	1	1

$f[.,.]$

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

$h[.,.]$

	0	10	20	30	30				
						?			
				50					

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

Credit: S. Seitz

Image filtering

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

$f[.,.]$

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

$h[.,.]$

	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	

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

Credit: S. Seitz

Box Filter

What does it do?

- Replaces each pixel with an average of its neighborhood

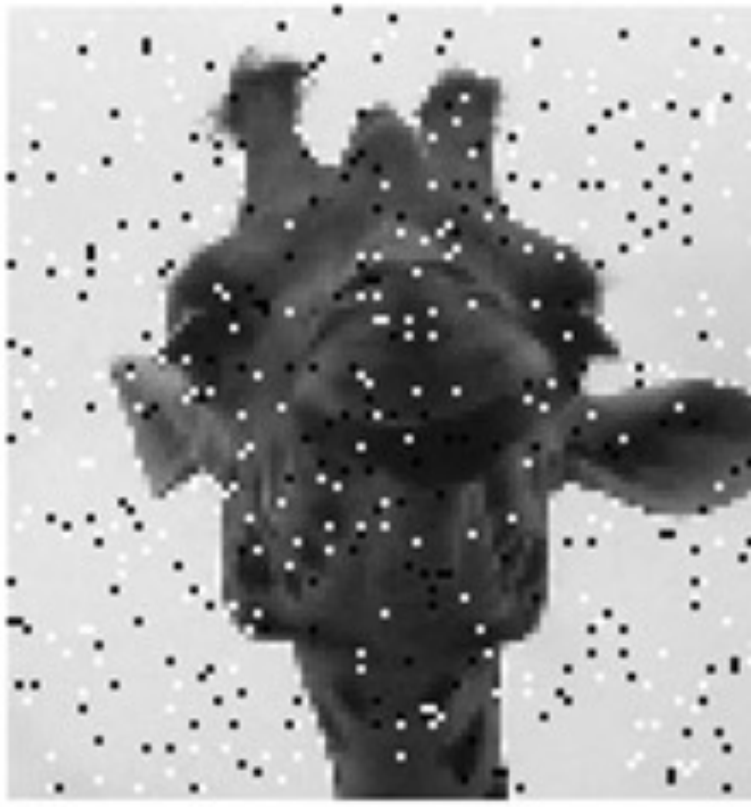
- Achieve smoothing effect (remove sharp features)

$$g[\cdot\,,\cdot\,]$$

$$\frac{1}{9}\begin{bmatrix} 1 & 1 & 1 \\ 1 & 1 & 1 \\ 1 & 1 & 1 \end{bmatrix}$$

Slide credit: David Lowe (UBC)

Image filtering: e.g. Mean Filter



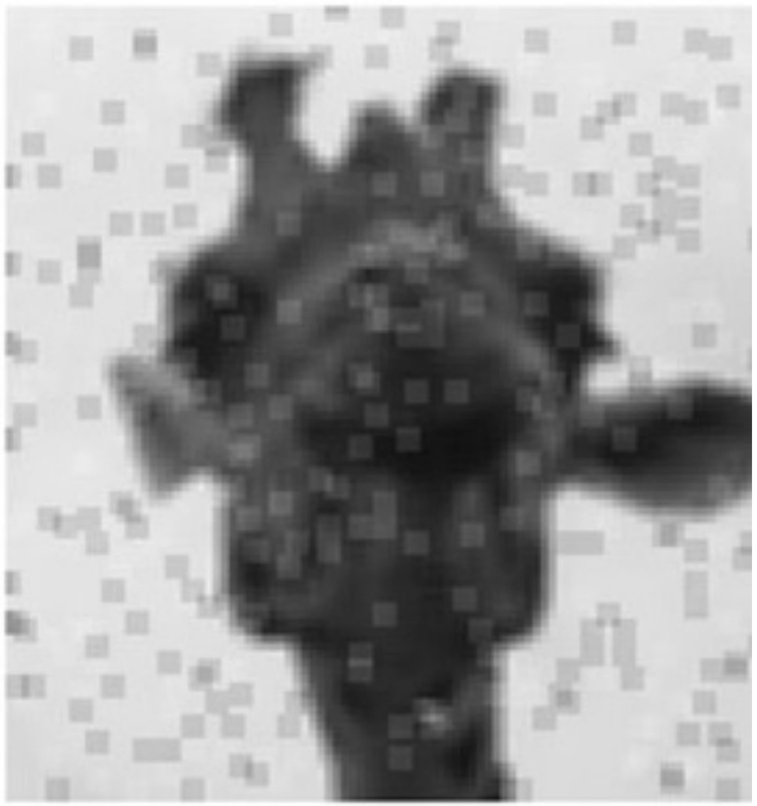
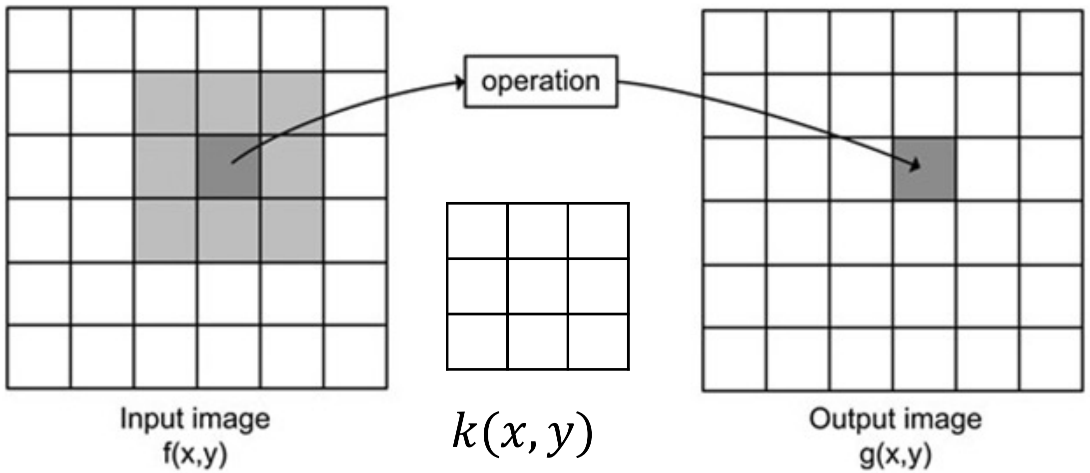


Image filtering: Convolution operator
Important filter: gaussian filter (gaussian blur)



$k(x, y) =$

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

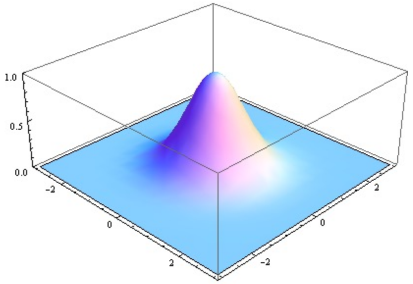
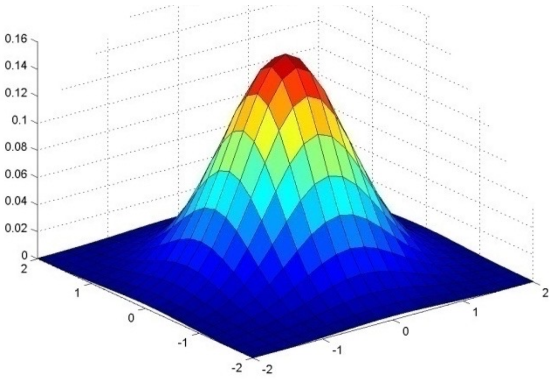
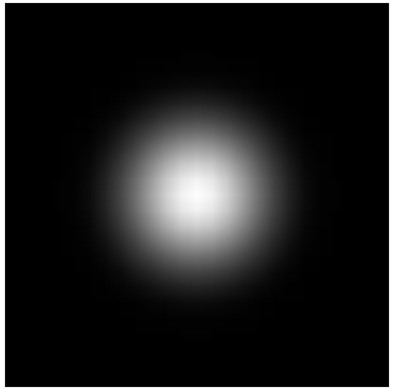


Image Credit: http://what-when-how.com/introduction-to-video-and-image-processing/neighborhood-processing-introduction-to-video-and-image-processing-part-1/

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, σ = 1

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

Slide credit: Christopher Rasmussen

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

Image Credit: http://what-when-how.com/introduction-to-video-and-image-processing/neighborhood-processing-introduction-to-video-and-image-processing-part-1/

Practical matters

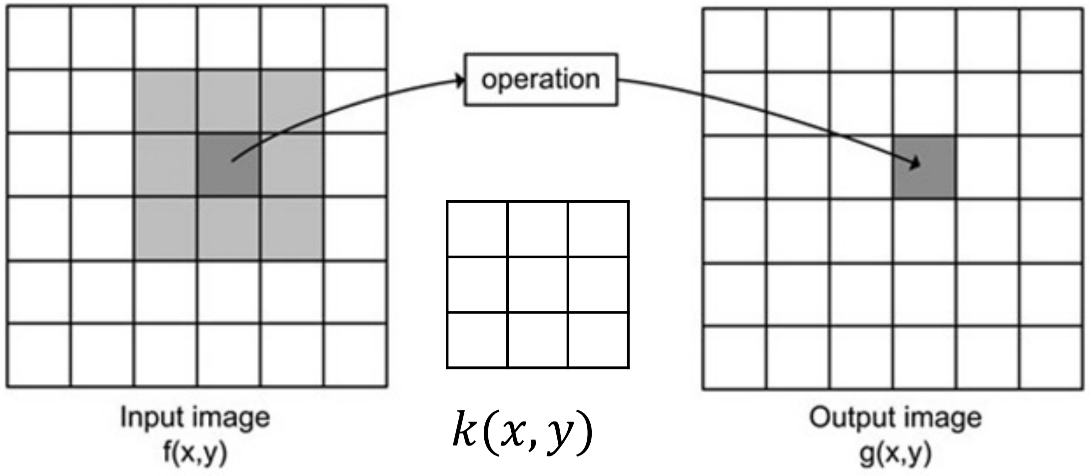
- What about near the edge?
  - the filter window falls off the edge of the image
  - need to extrapolate
  - methods:
    - clip filter (black)
    - wrap around
    - copy edge
    - reflect across edge

Source: S. Marschner

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



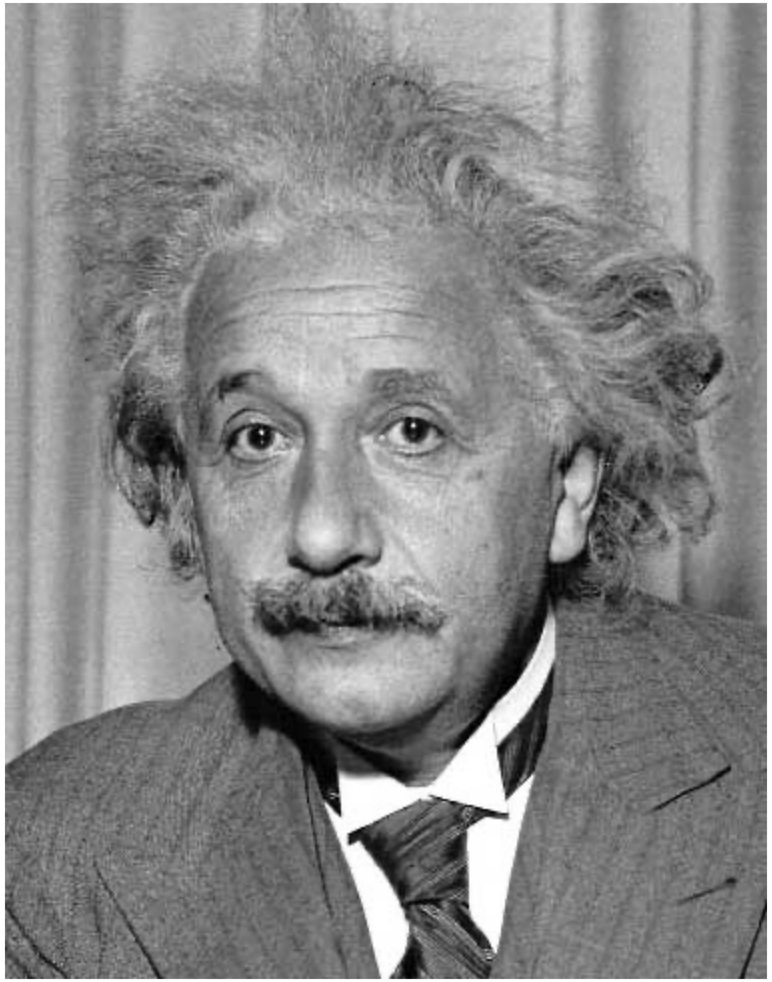
$k(x, y)$

$k(x, y) =$

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

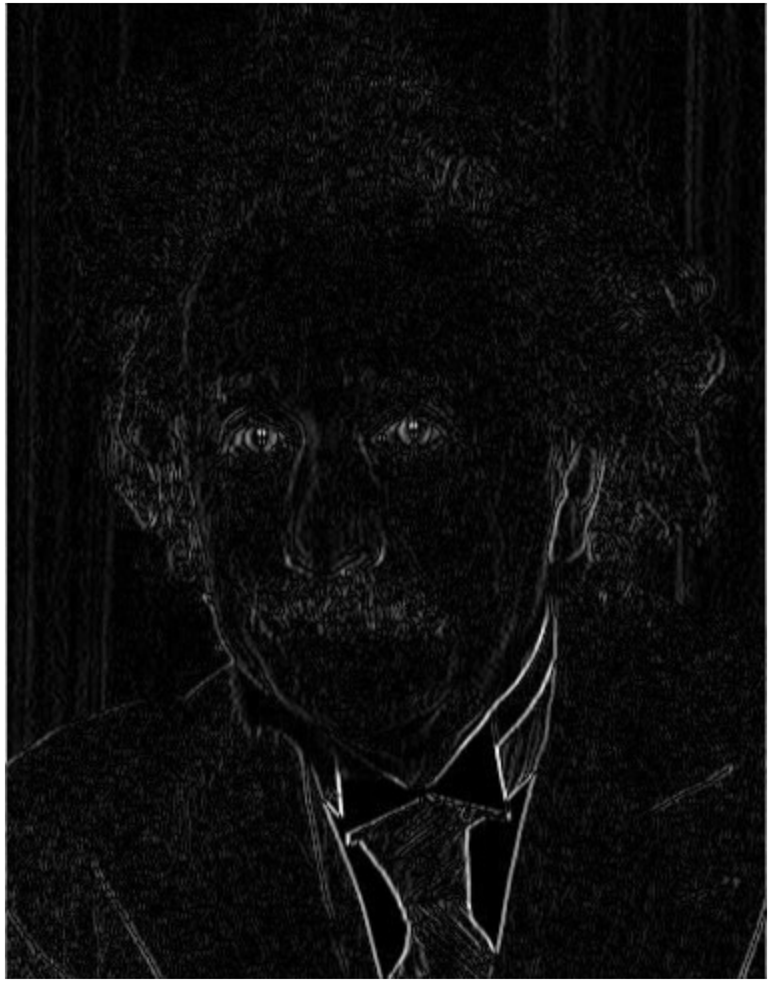
Image Credit: http://what-when-how.com/introduction-to-video-and-image-processing/neighborhood-processing-introduction-to-video-and-image-processing-part-1/

Other filters



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

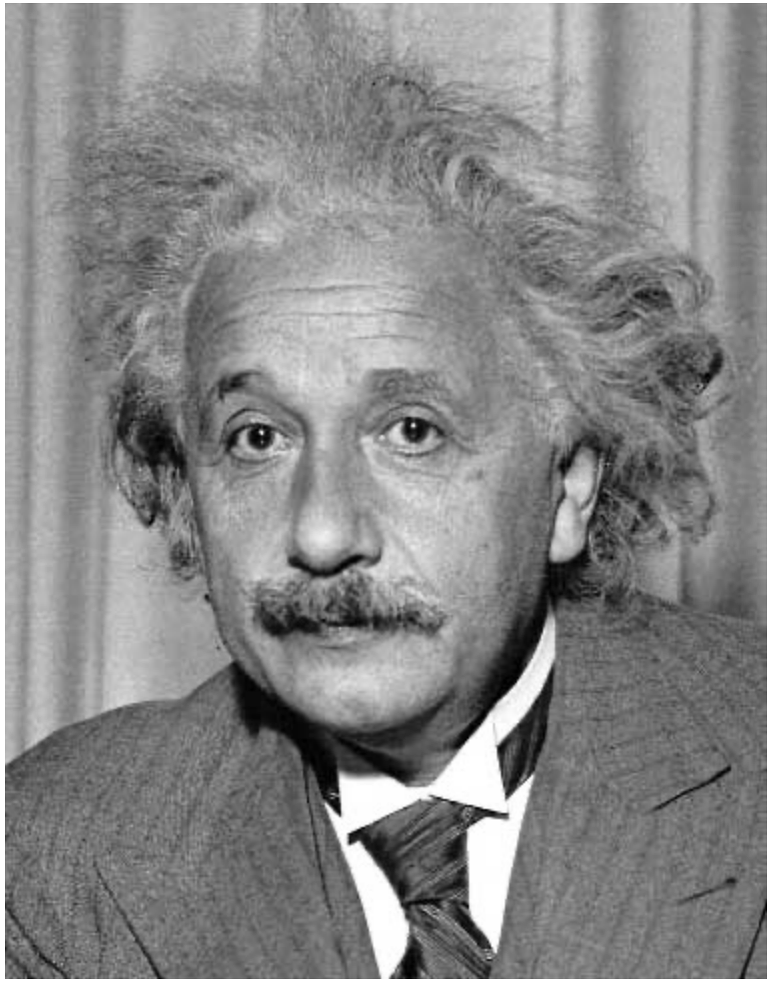
Sobel



Vertical Edge
(absolute value)

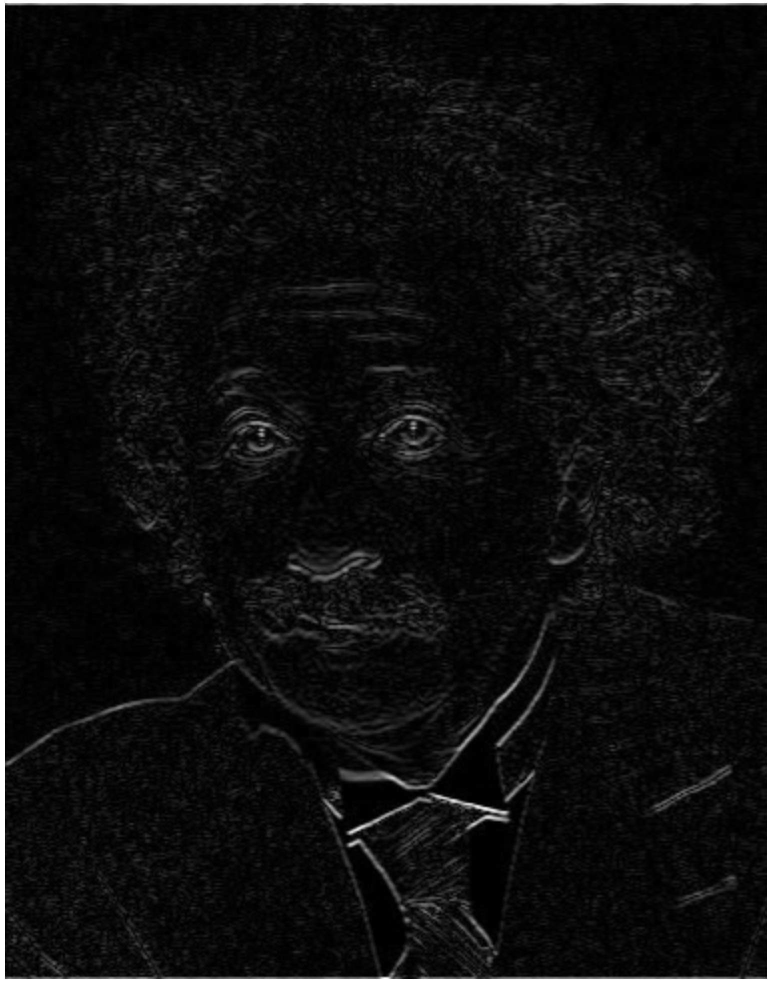
Slide by James Hays

Other filters



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

Sobel



Horizontal Edge
(absolute value)

Slide by James Hays

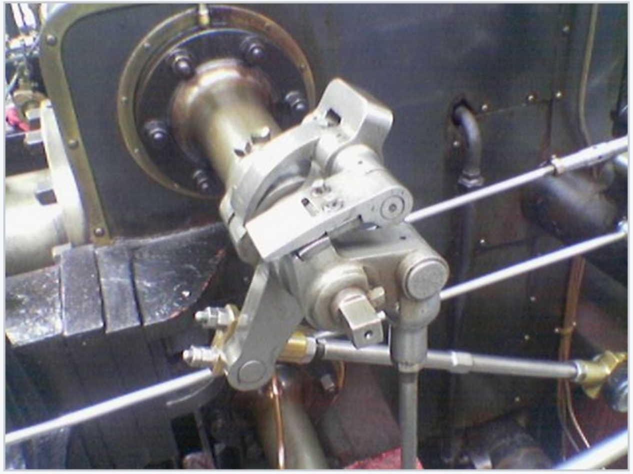
Sobel operators are equivalent to 2D partial derivatives of the image

- Vertical sobel operator – Partial derivative in X (width)
- Horizontal sobel operator – Partial derivative in Y (height)

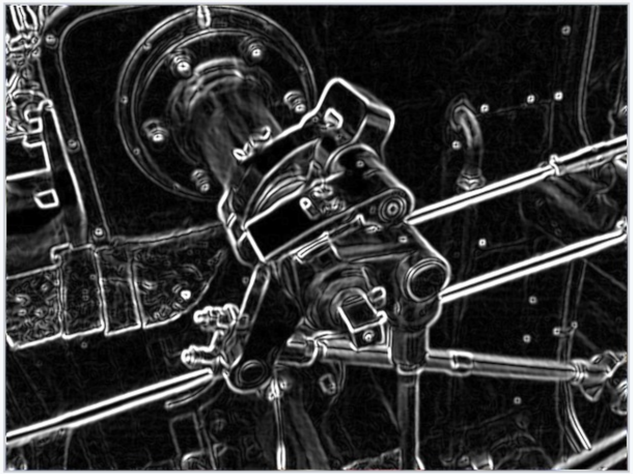
- Can compute magnitude and phase at each location.

- 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:

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

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

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

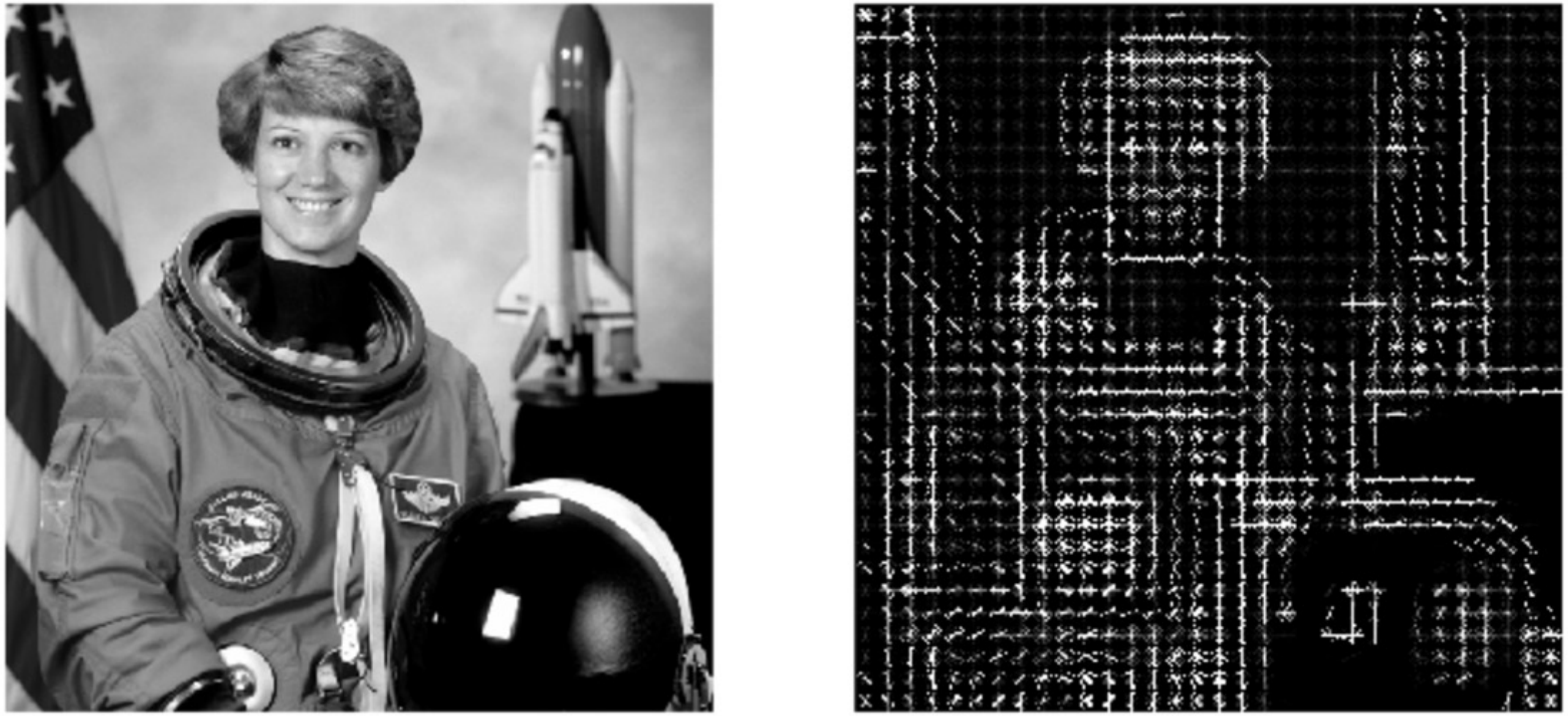
$$k(x, y) = \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} * \begin{bmatrix} 1 & 0 & -1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix}$$

Similarly to differentiate in Y

Image Features: HoG

Input image

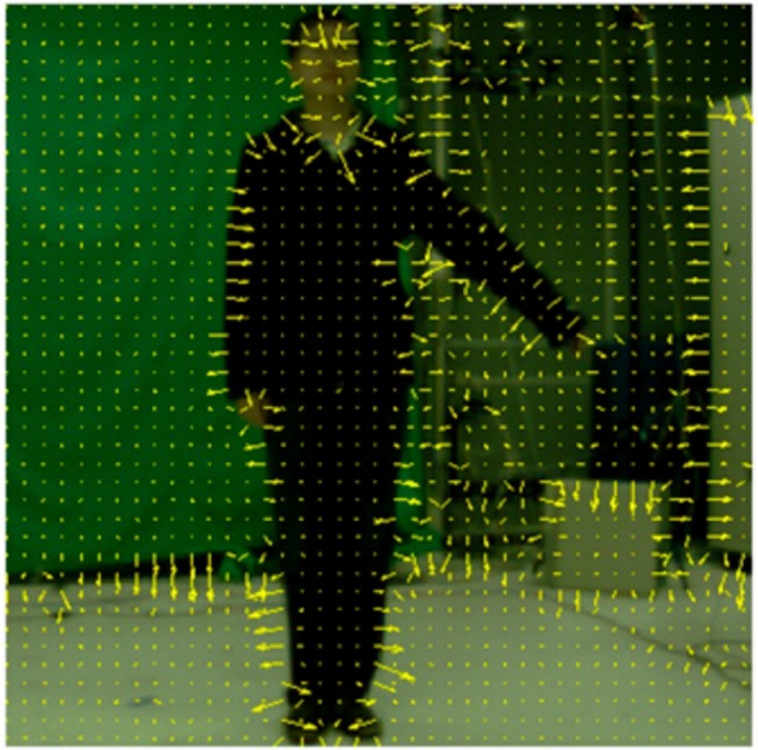
Histogram of Oriented Gradients

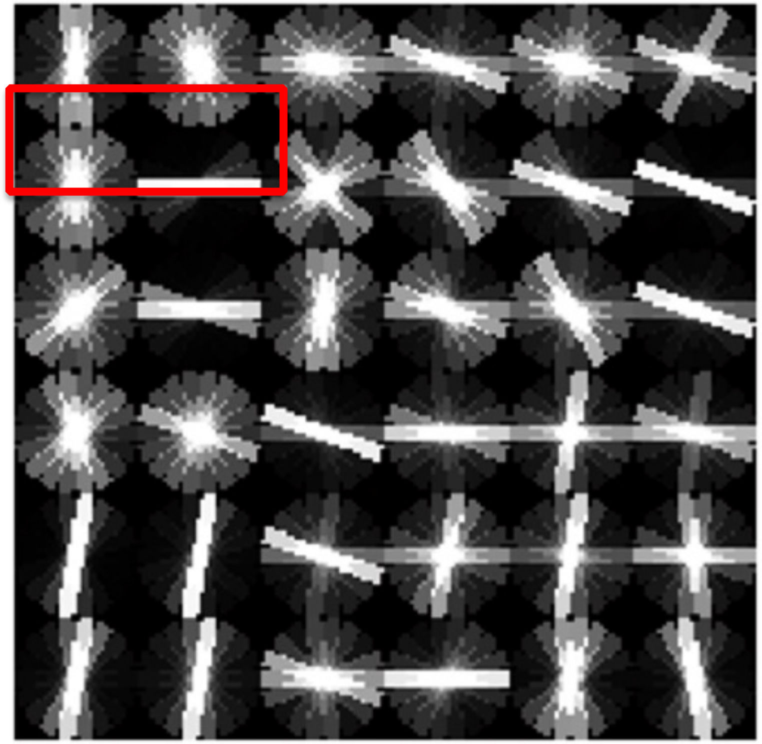


Paper by Navneet Dalal & Bill Triggs presented at CVPR 2005 for detecting people.

Scikit-image implementation

Image Features: HoG

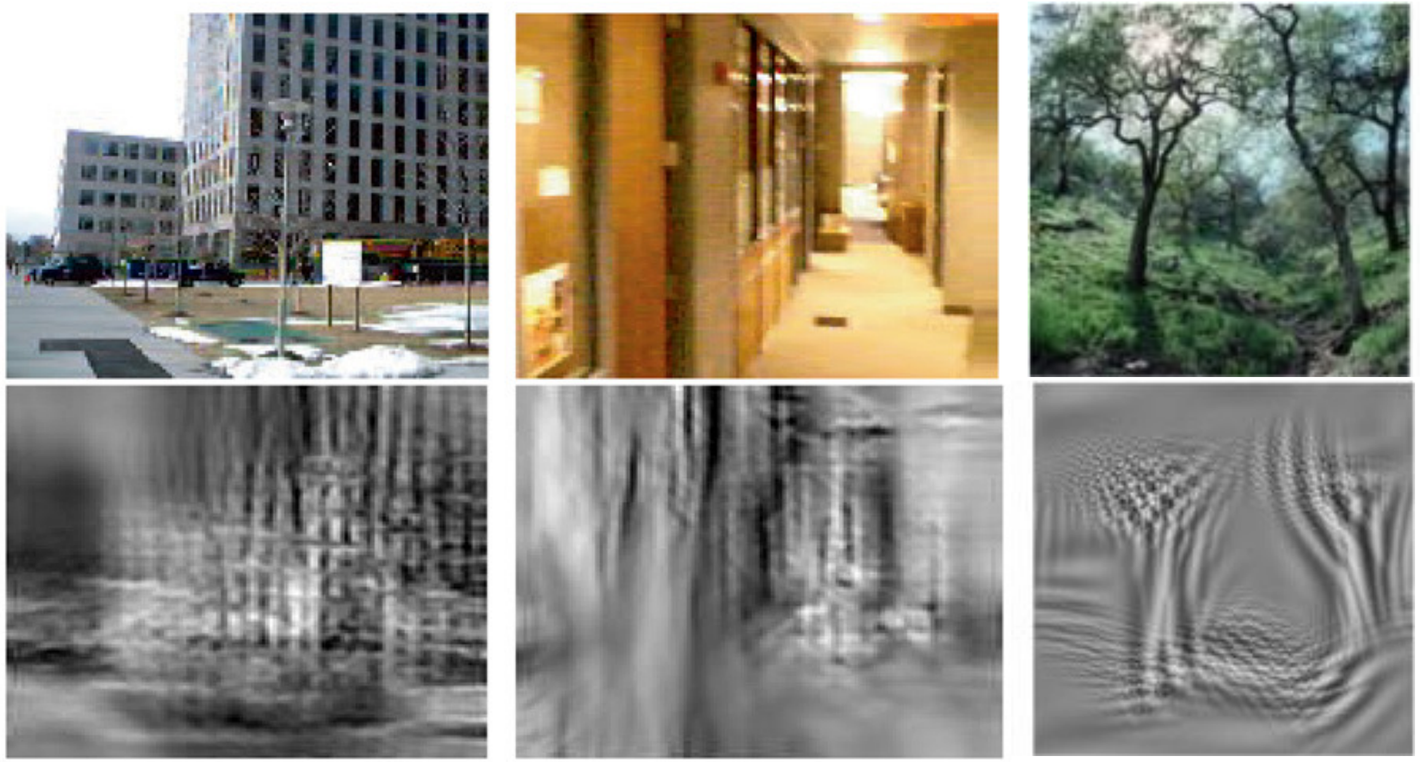




+ Block Normalization

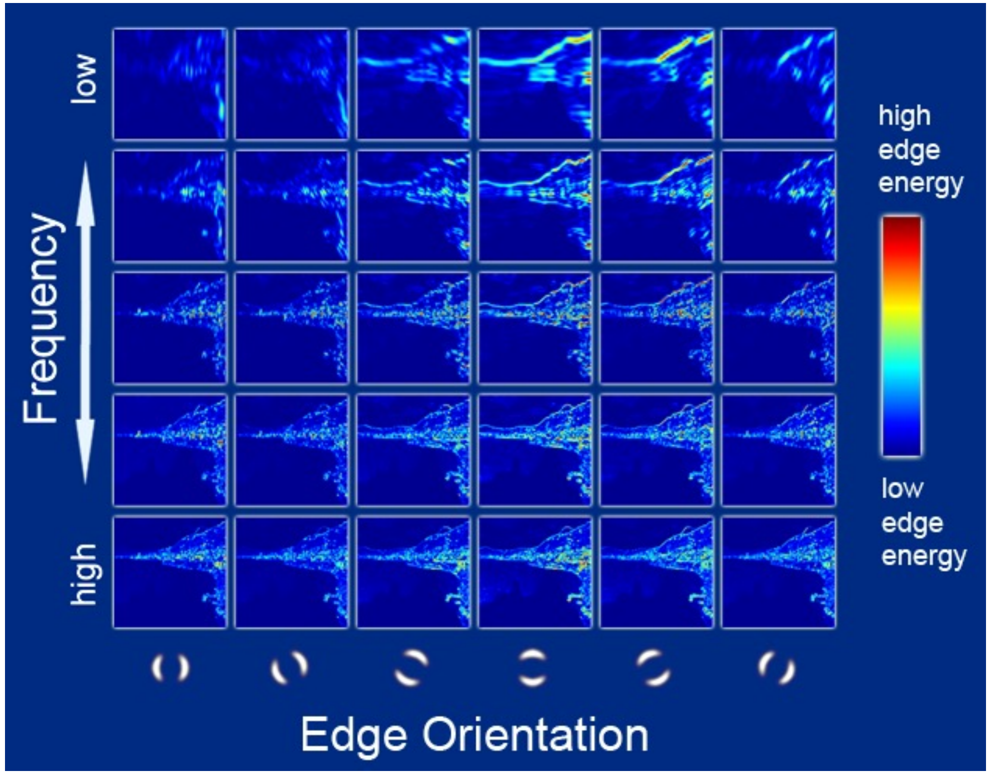
Paper by Navneet Dalal & Bill Triggs presented at CVPR 2005 for detecting people.
Figure from Zhuolin Jiang, Zhe Lin, Larry S. Davis, ICCV 2009 for human action recognition.

Image Features: GIST



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

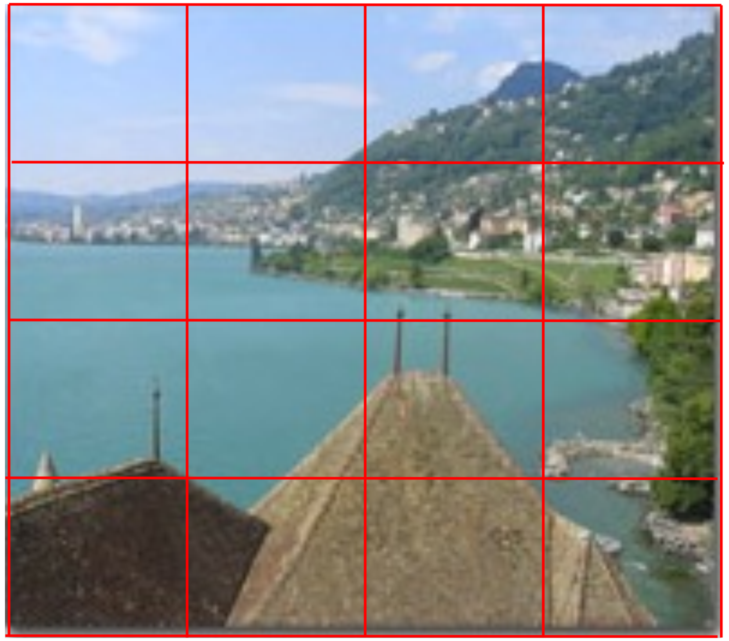
Image Features: GIST

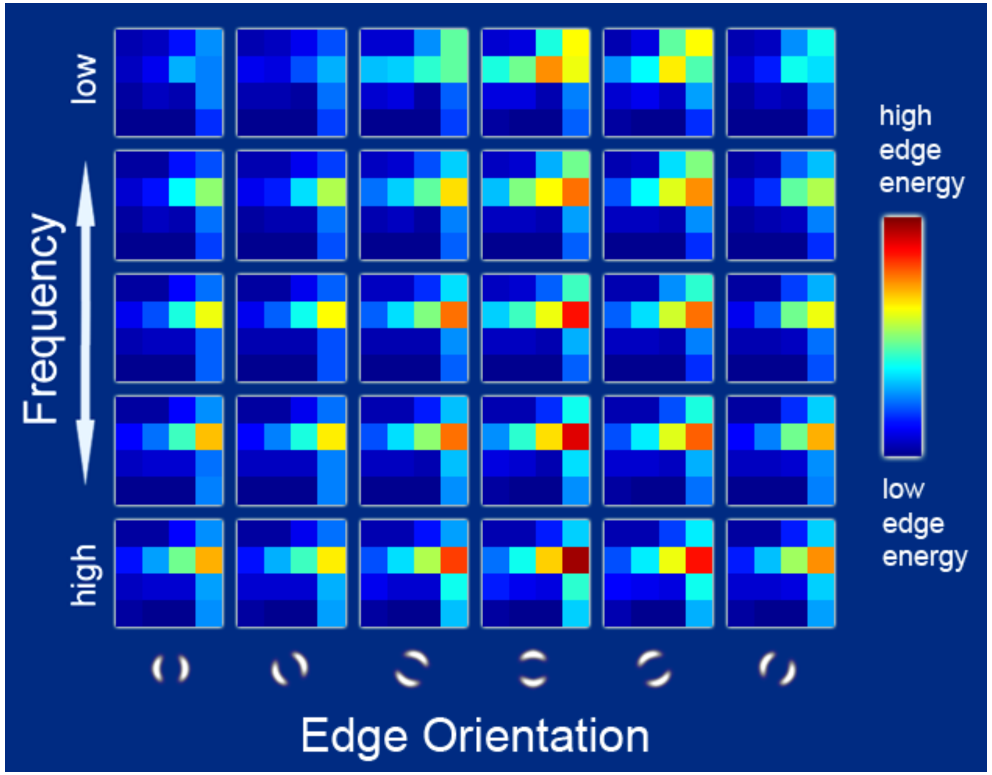


Oriented edge response at multiple scales (5 spatial scales, 6 edge orientations)

Hays and Efros, SIG
2007

Image Features: GIST





Aggregated edge responses over 4x4 windows

Hays and Efros, SIG
2007

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

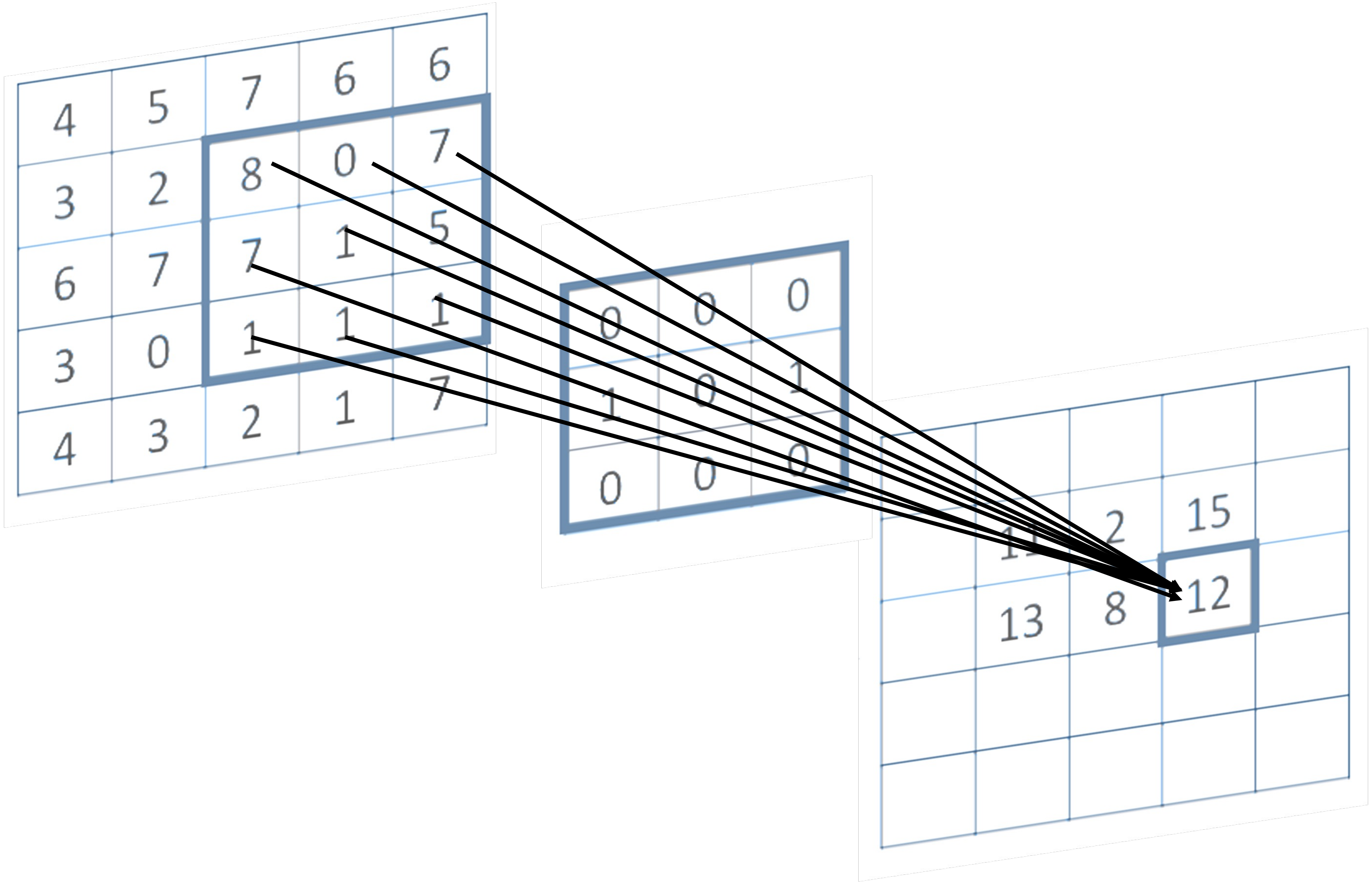
*

0	0	0
1	0	1
0	0	0

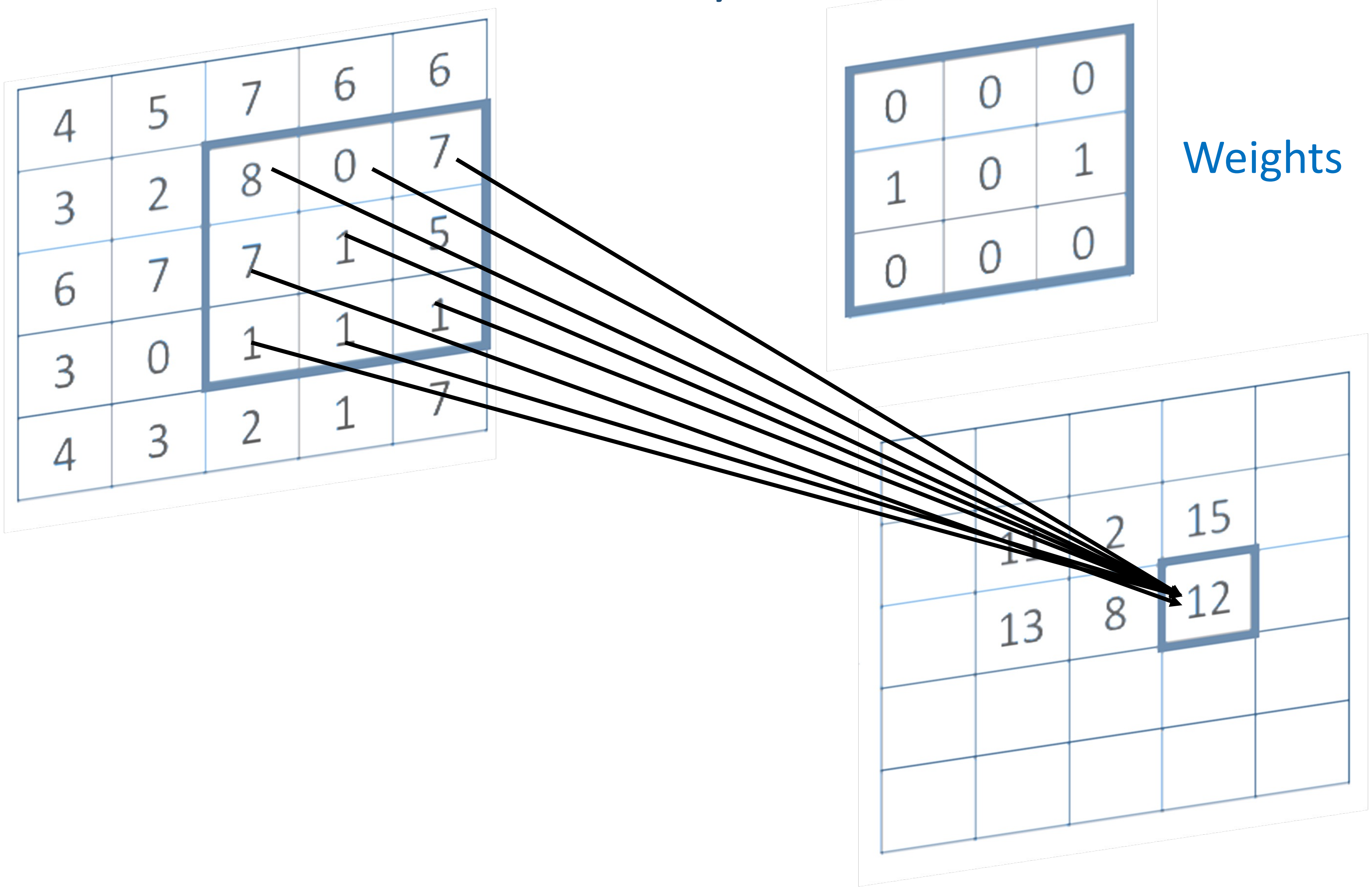
→

	11	2	15	
	13	8	12	

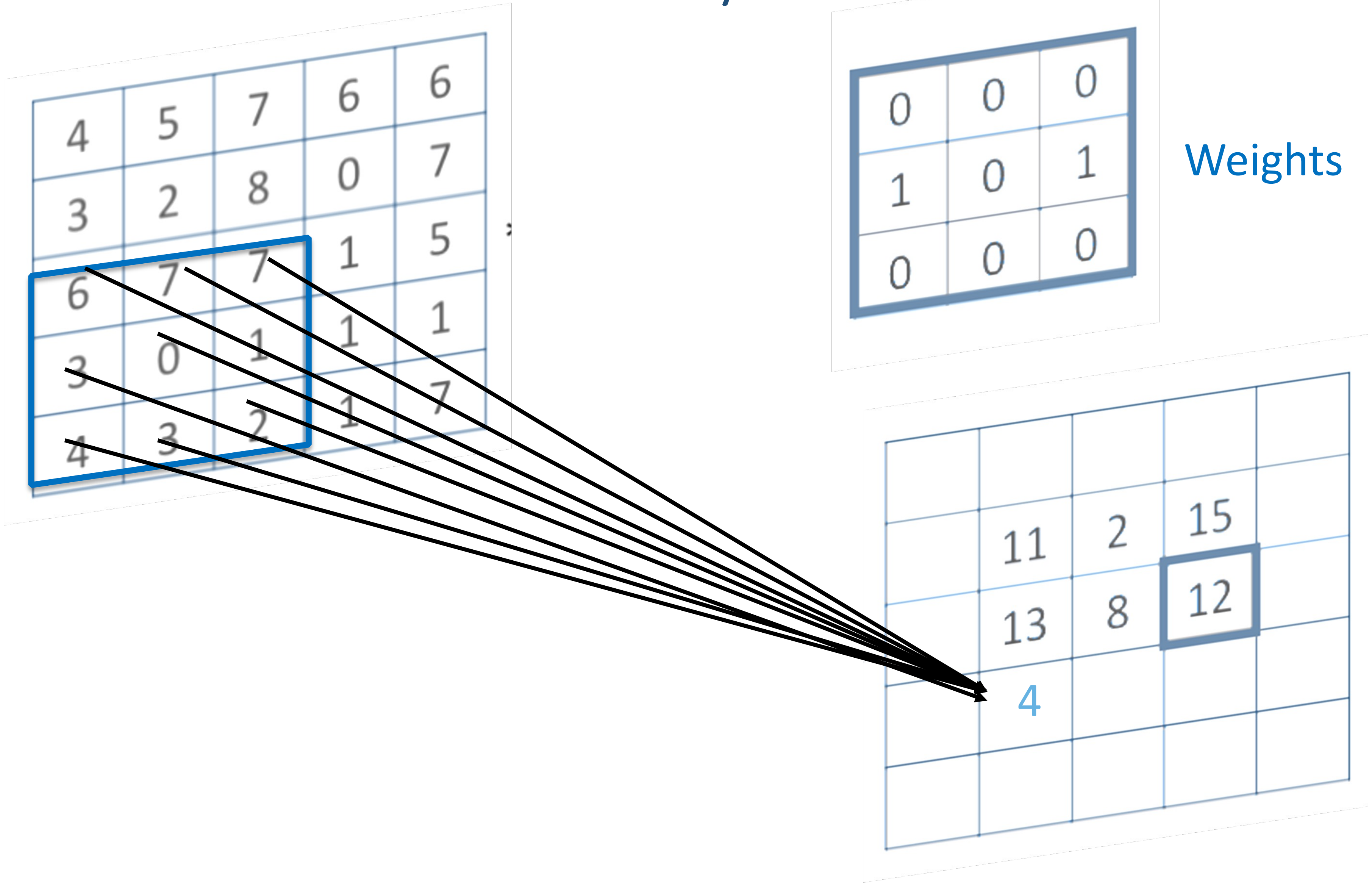
The 2D Convolutional Layer in a Neural Network



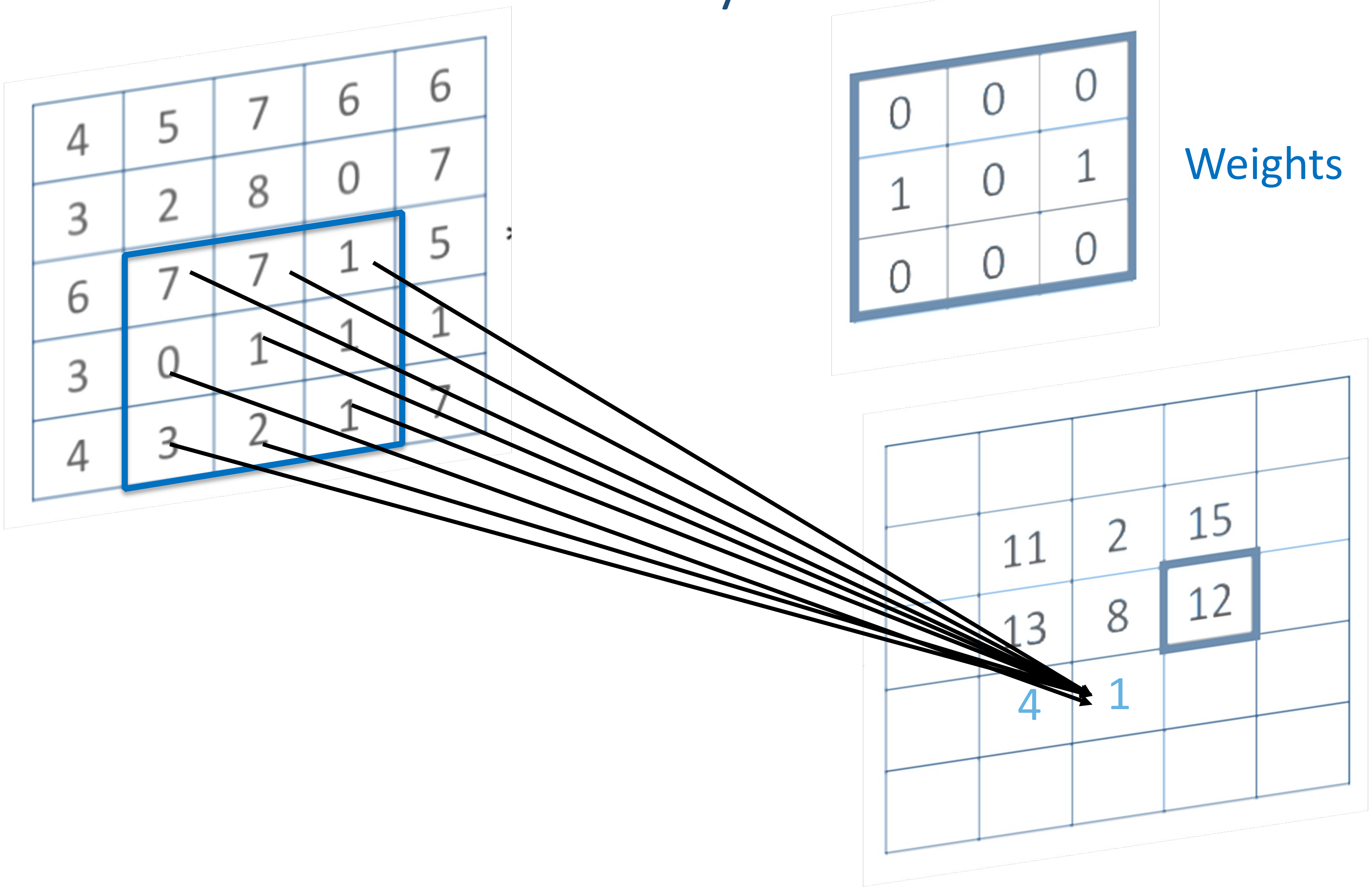
The 2D Convolutional Layer in a Neural Network



The 2D Convolutional Layer in a Neural Network



The 2D Convolutional Layer in a Neural Network



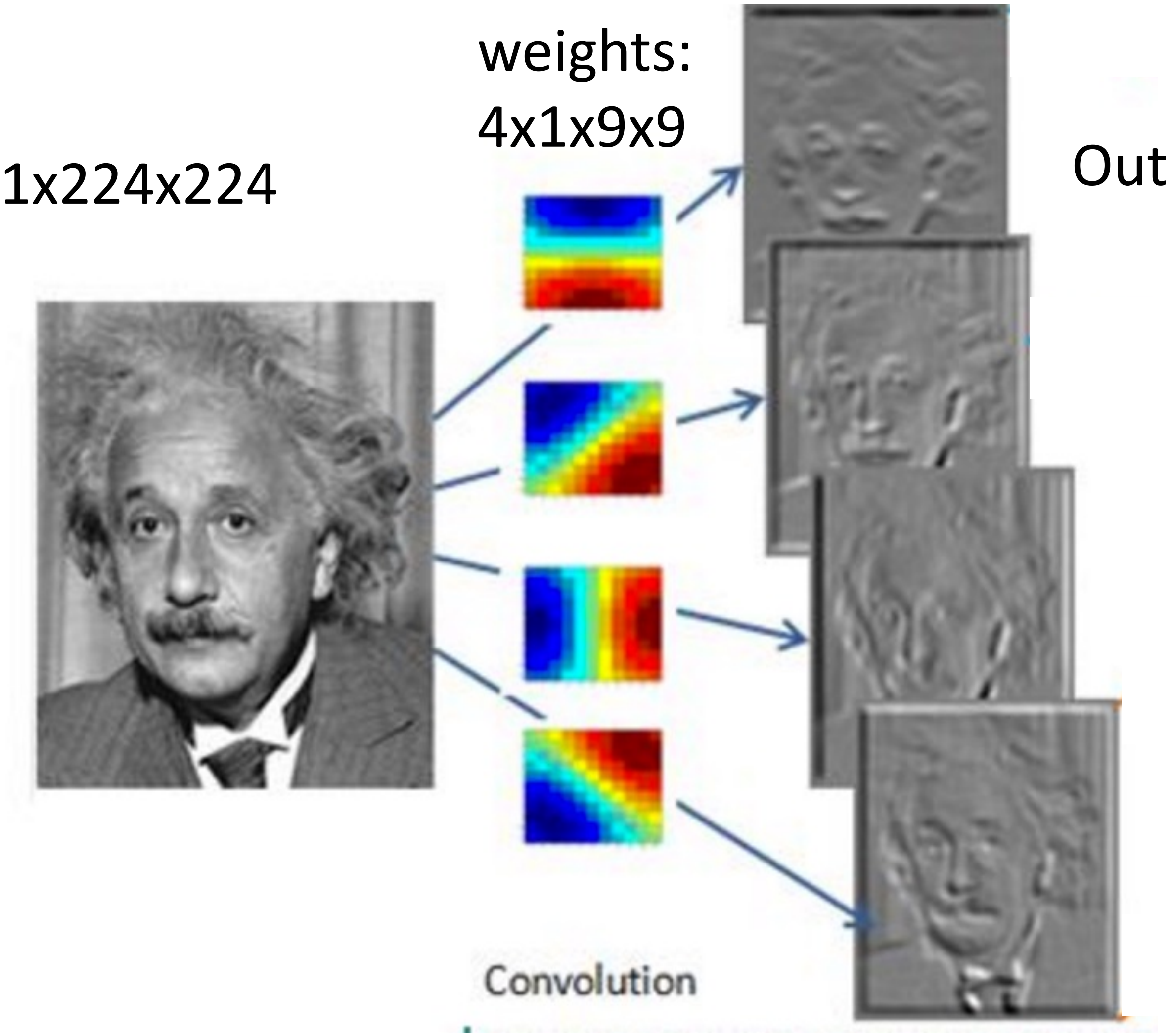
Convolutional Layer (with 4 filters)

weights:
4x1x9x9

Input: 1x224x224

Output: 4x224x224

if zero padding,
and stride = 1

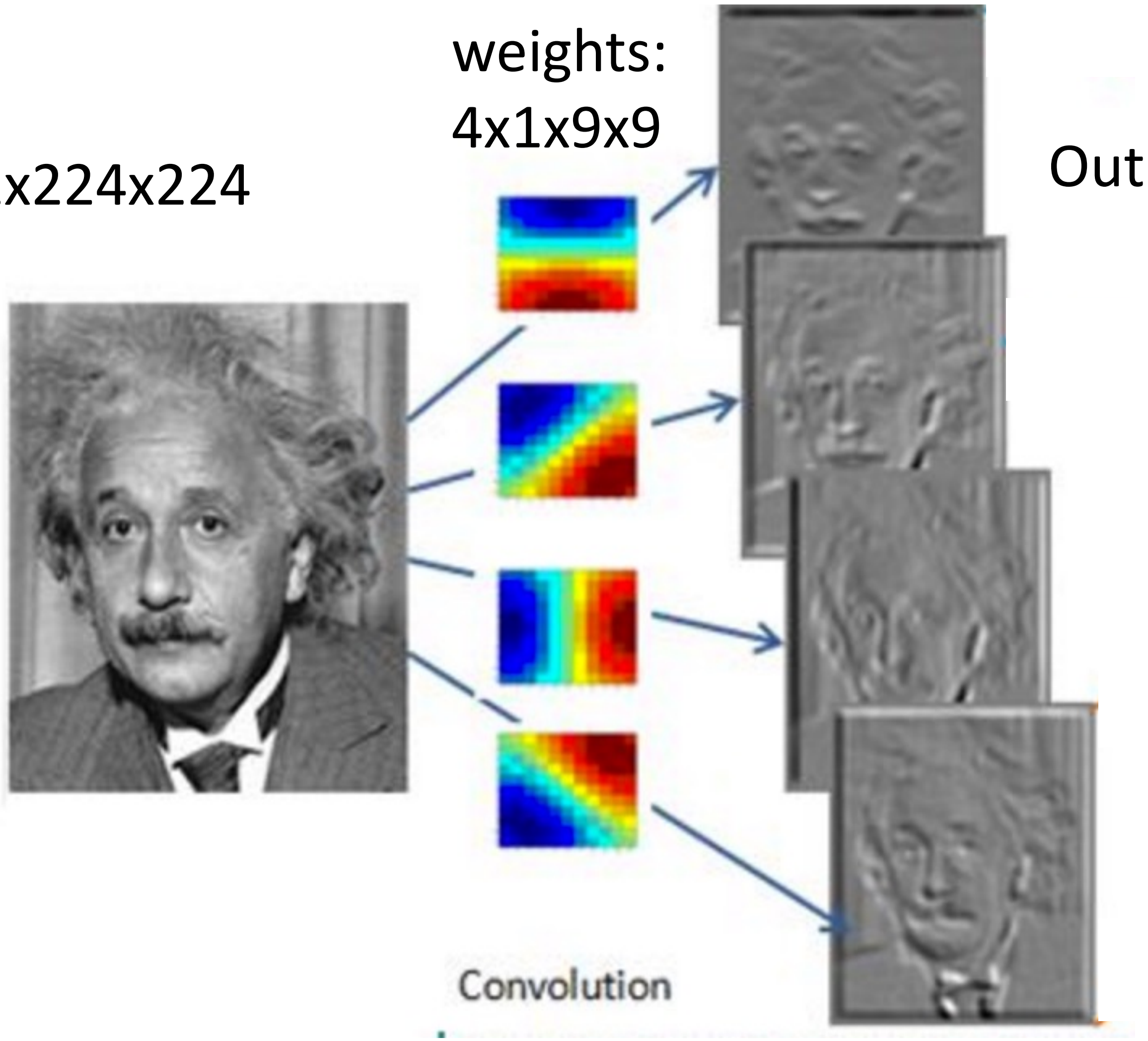


Convolutional Layer (with 4 filters)

weights:
4x1x9x9

Input: 1x224x224

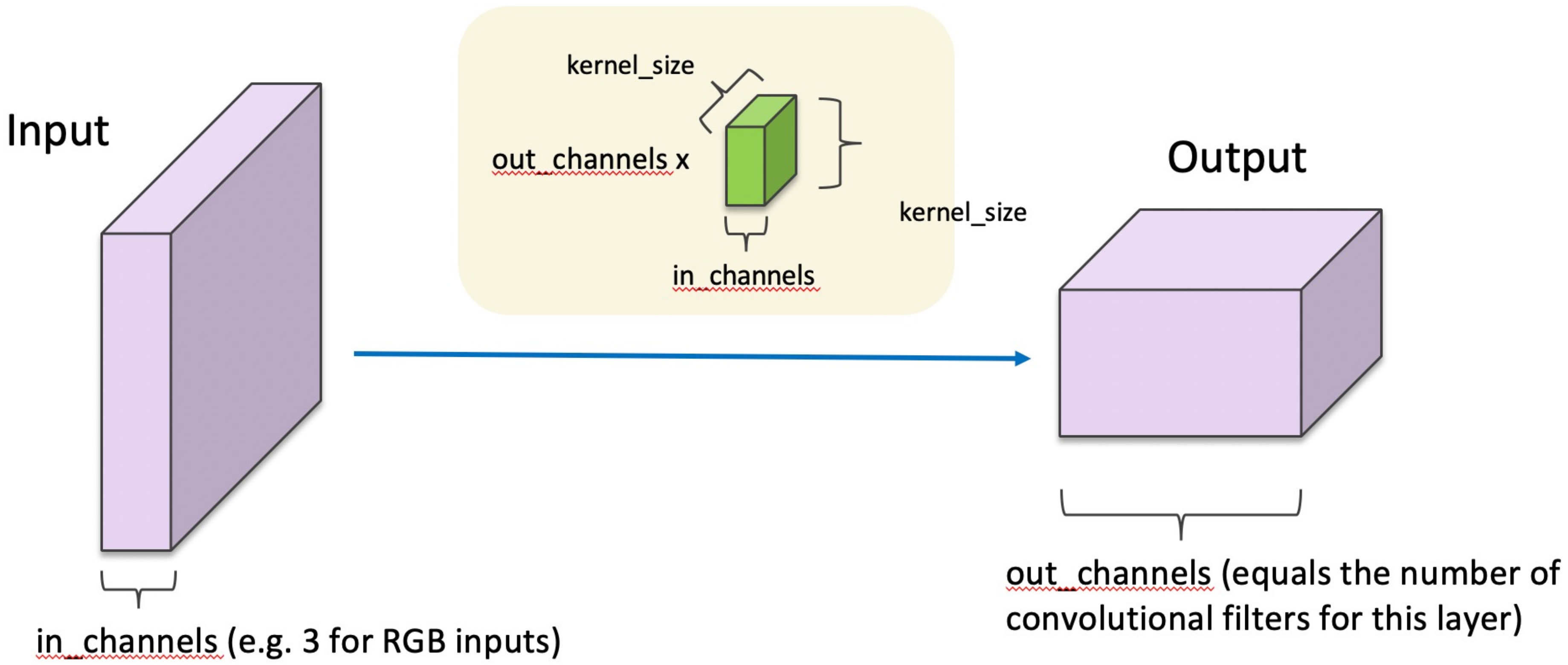
Output: 4x112x112



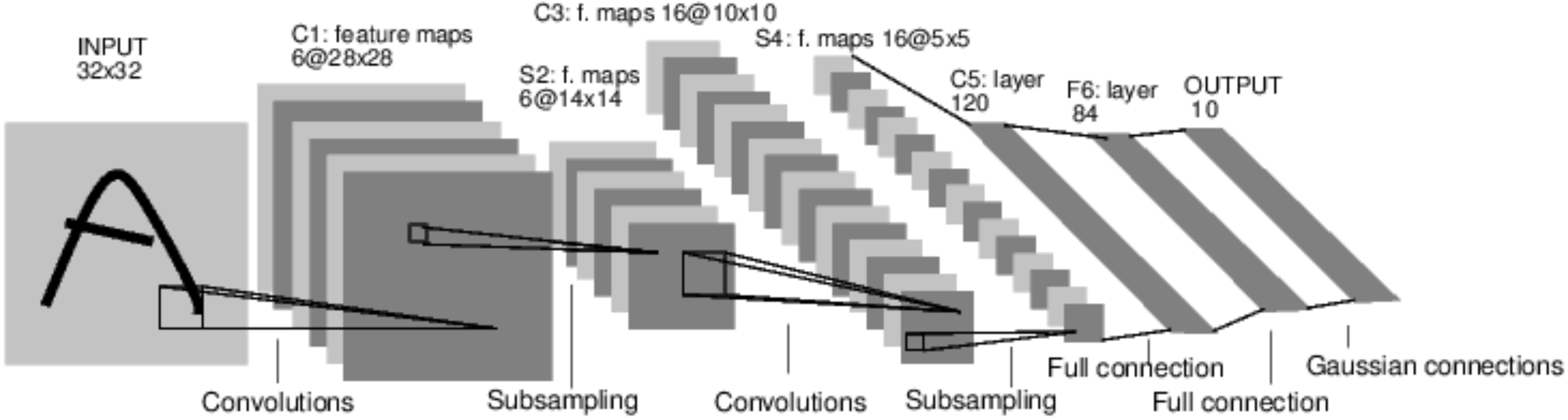
if zero padding,
but stride = 2

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 Y LeCun, L Bottou, Y Bengio, P Haffner Proceedings of the IEEE 86 (11), 2278-2324	11736	1998

LeNet in Pytorch

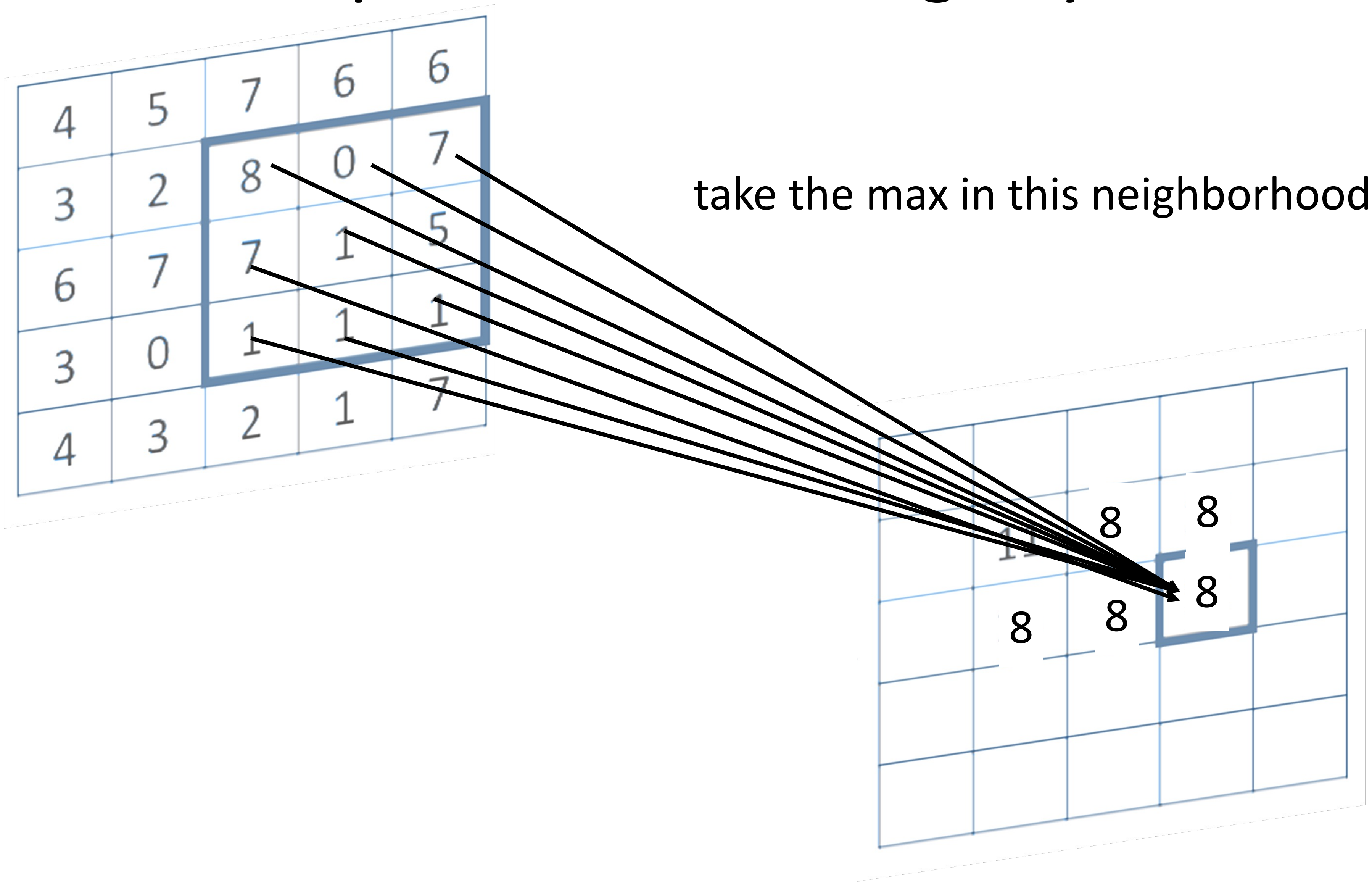
```python
# LeNet is French for The Network, and is taken from Yann Lecun's 98 paper
# on digit classification http://yann.lecun.com/exdb/lenet/
# This was also a network with just two convolutional layers.
class LeNet(nn.Module):
    def __init__(self):
        super(LeNet, self).__init__()
        # 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



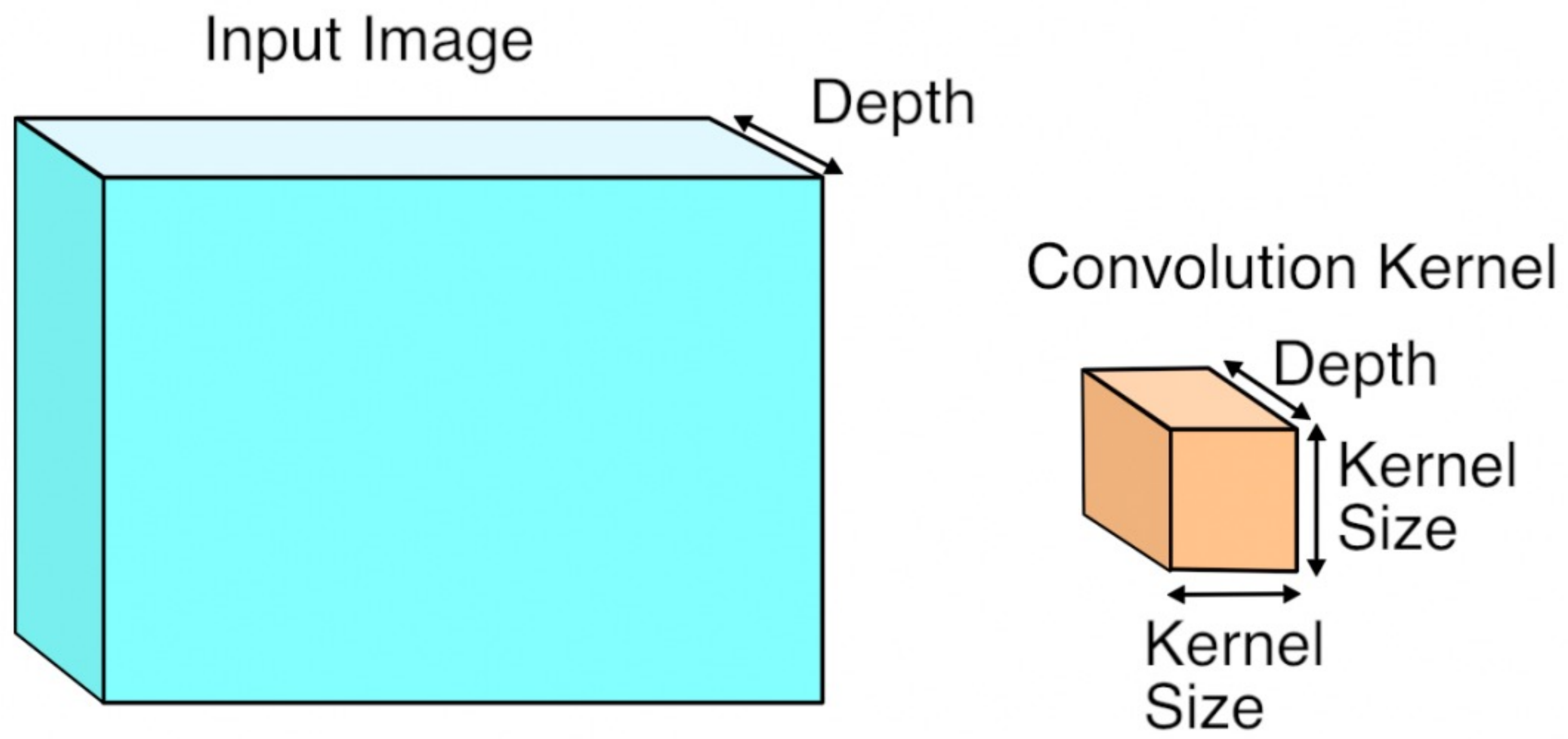
LeNet Summary

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

New Architectures Proposed

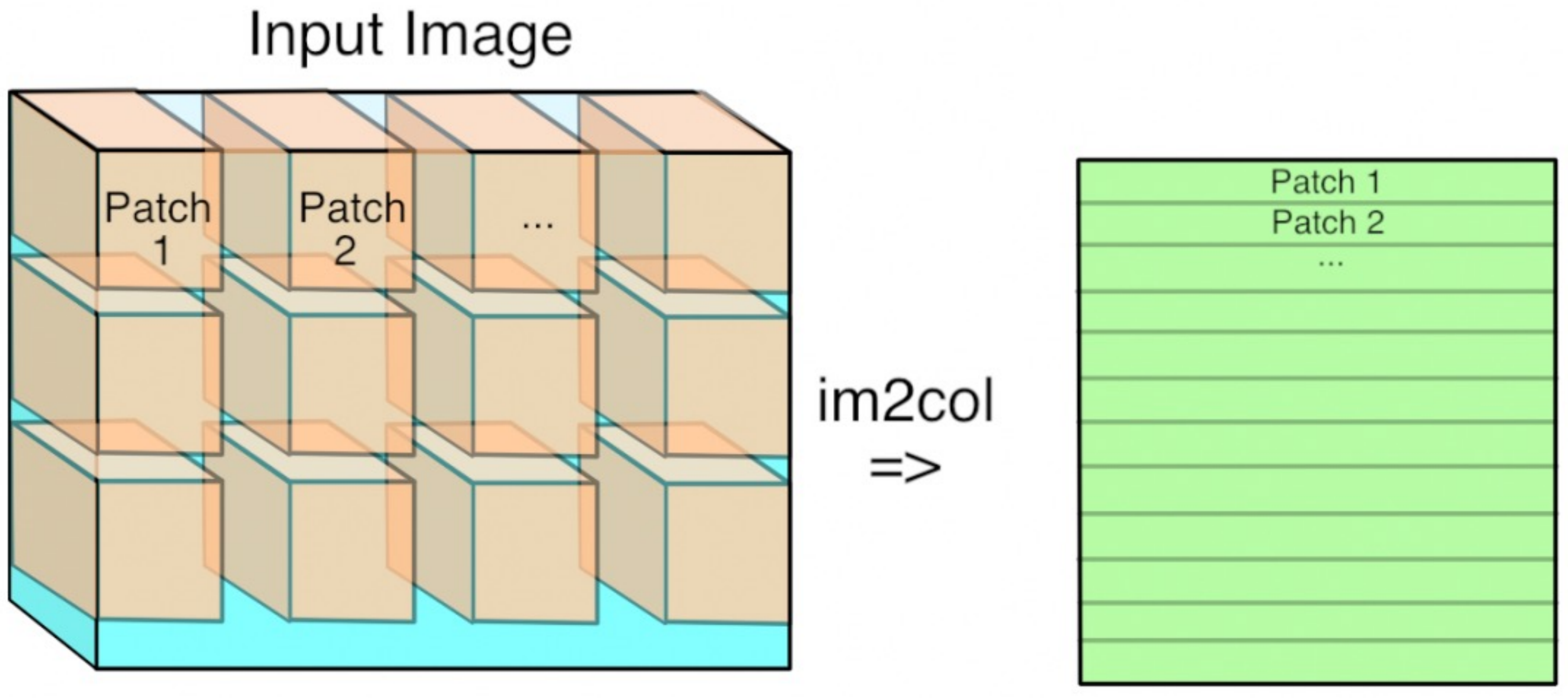
- Alexnet (Kriszhevsky et al NIPS 2012) **[Required Reading]**
- VGG (Simonyan and Zisserman 2014)
- GoogLeNet (Szegedy et al CVPR 2015)
- ResNet (He et al CVPR 2016)
- DenseNet (Huang et al CVPR 2017)

Convolutional Layers as Matrix Multiplication



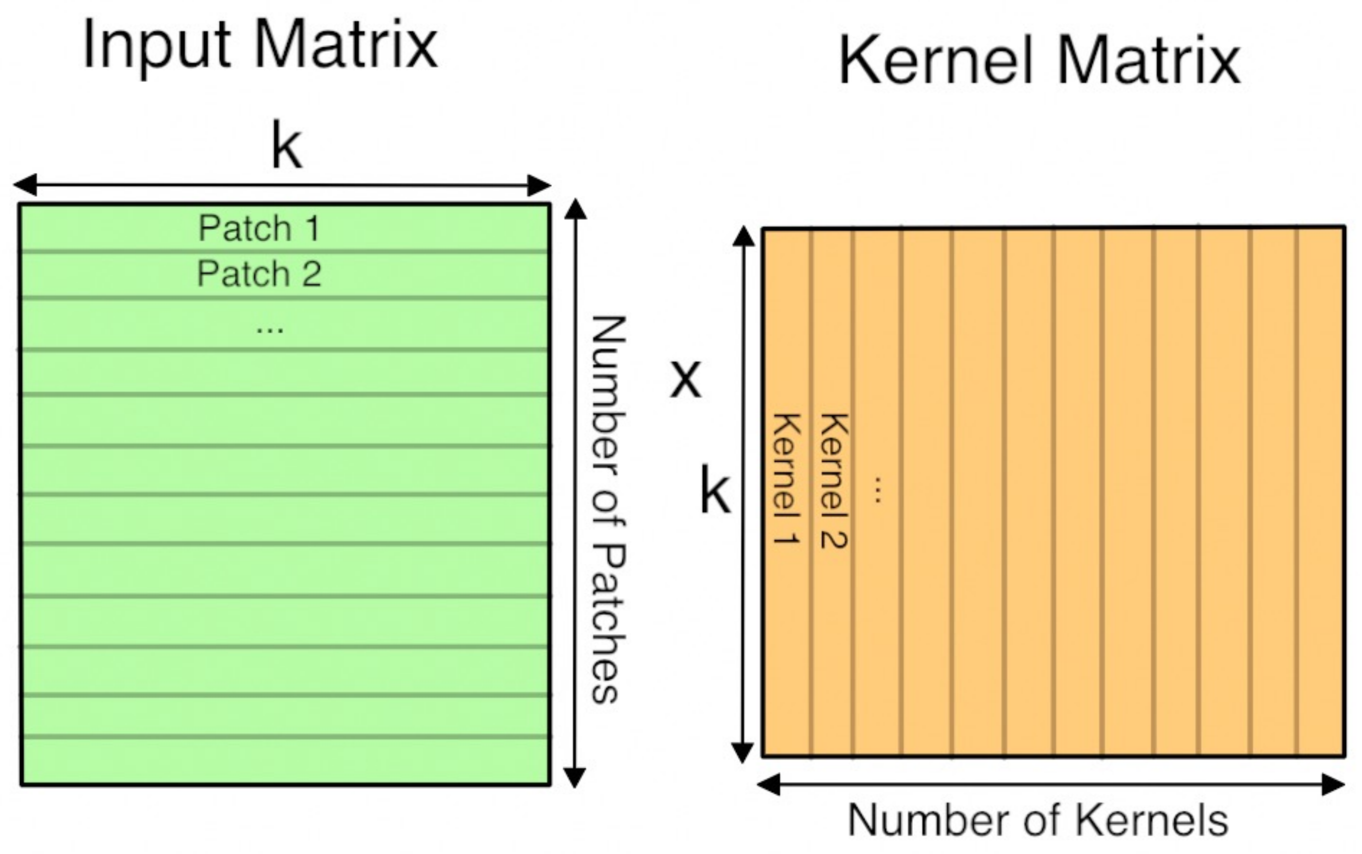
https://petewarden.com/2015/04/20/why-gemm-is-at-the-heart-of-deep-learning/

Convolutional Layers as Matrix Multiplication



https://petewarden.com/2015/04/20/why-gemm-is-at-the-heart-of-deep-learning/

Convolutional Layers as Matrix Multiplication



Pros?
Cons?

https://petewarden.com/2015/04/20/why-gemm-is-at-the-heart-of-deep-learning/

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)

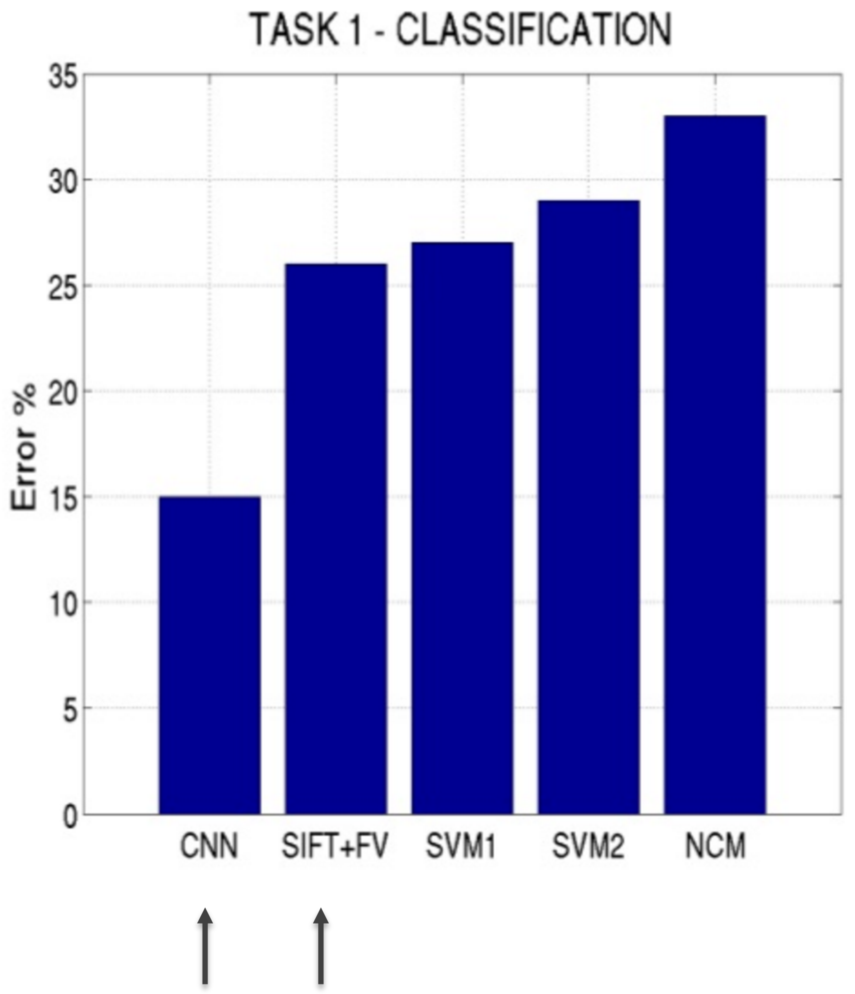
The Evaluation Metric: Top K-error

True label: Abyssinian cat

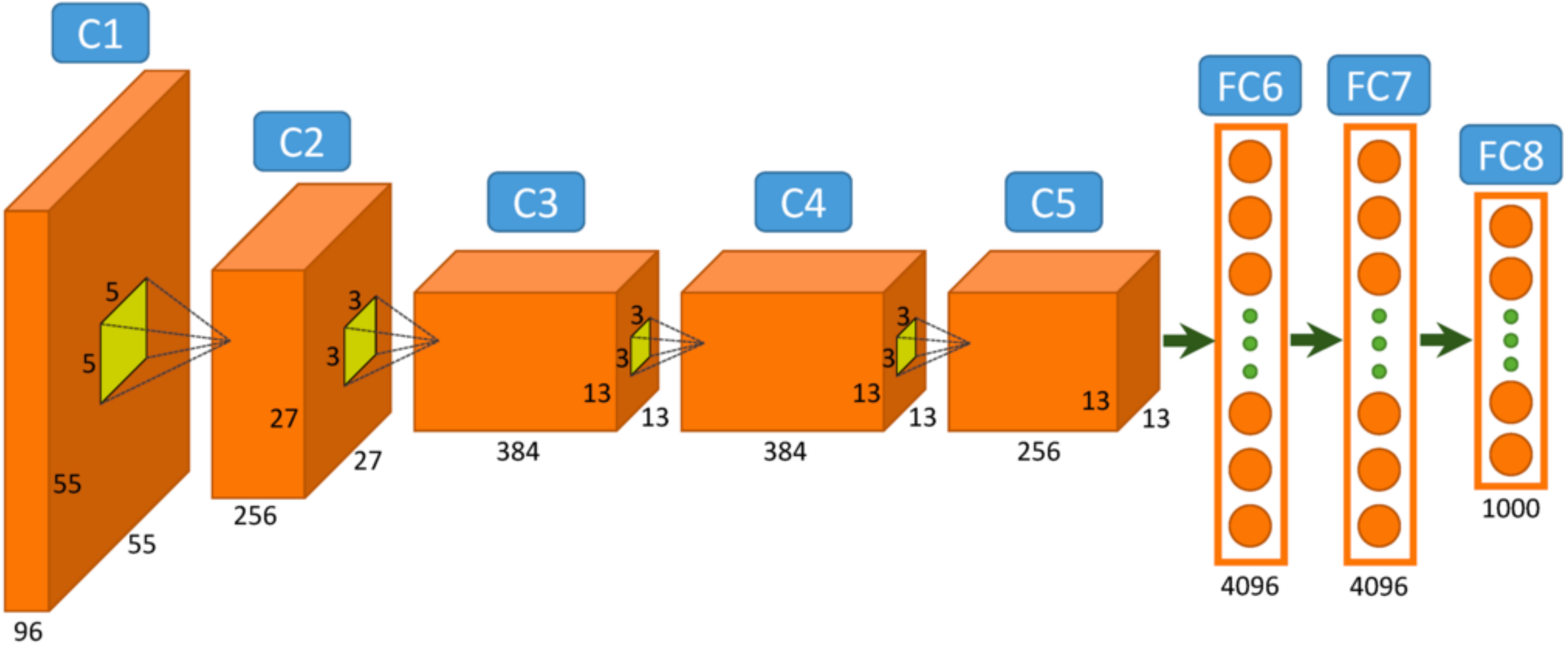
Top-1 error: 1.0	Top-1 accuracy: 0.0
Top-2 error: 1.0	Top-2 accuracy: 0.0
Top-3 error: 1.0	Top-3 accuracy: 0.0
Top-4 error: 0.0	Top-4 accuracy: 1.0
Top-5 error: 0.0	Top-5 accuracy: 1.0

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



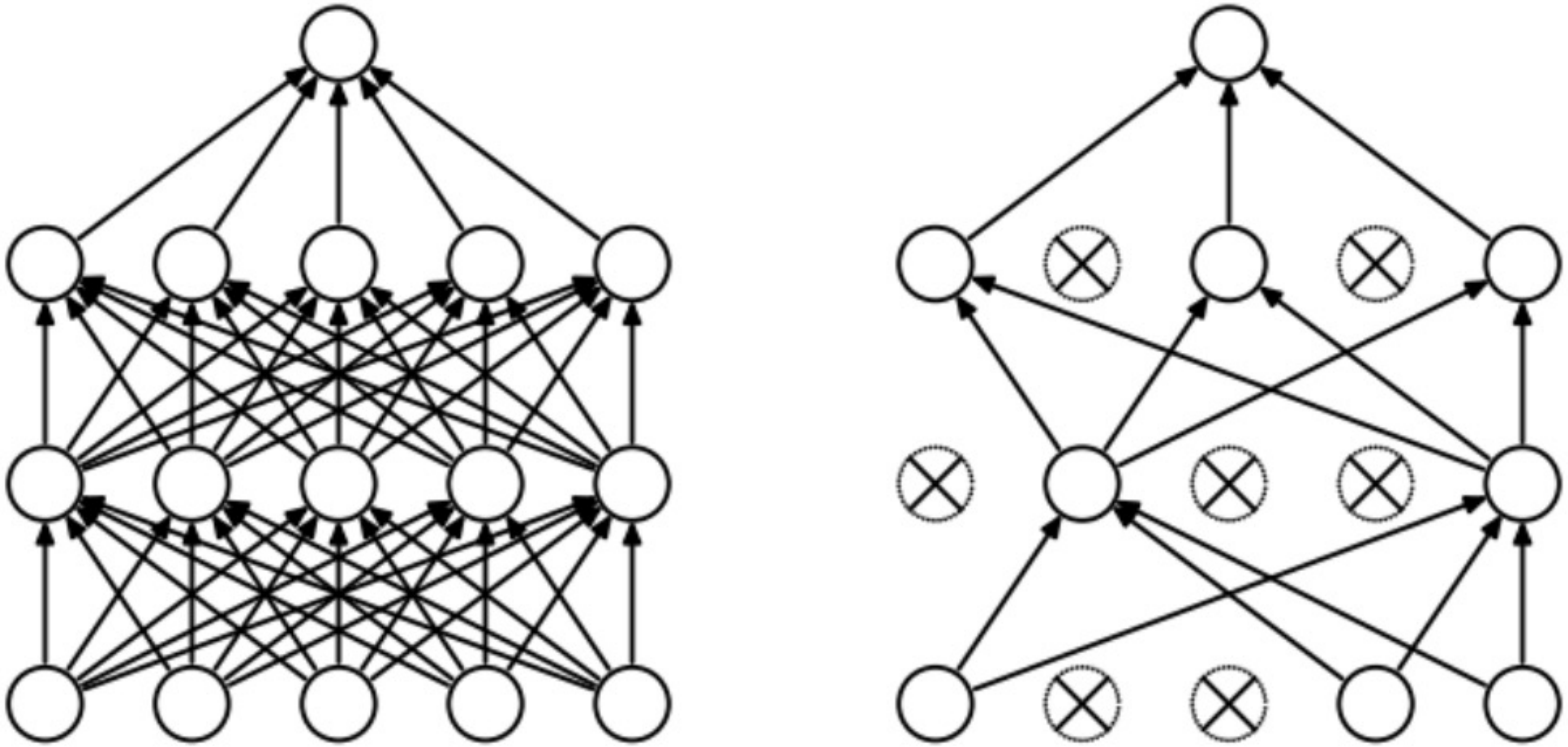
https://www.saagie.com/fr/blog/object-detection-part1

Pytorch Code for Alexnet

- In-class analysis

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

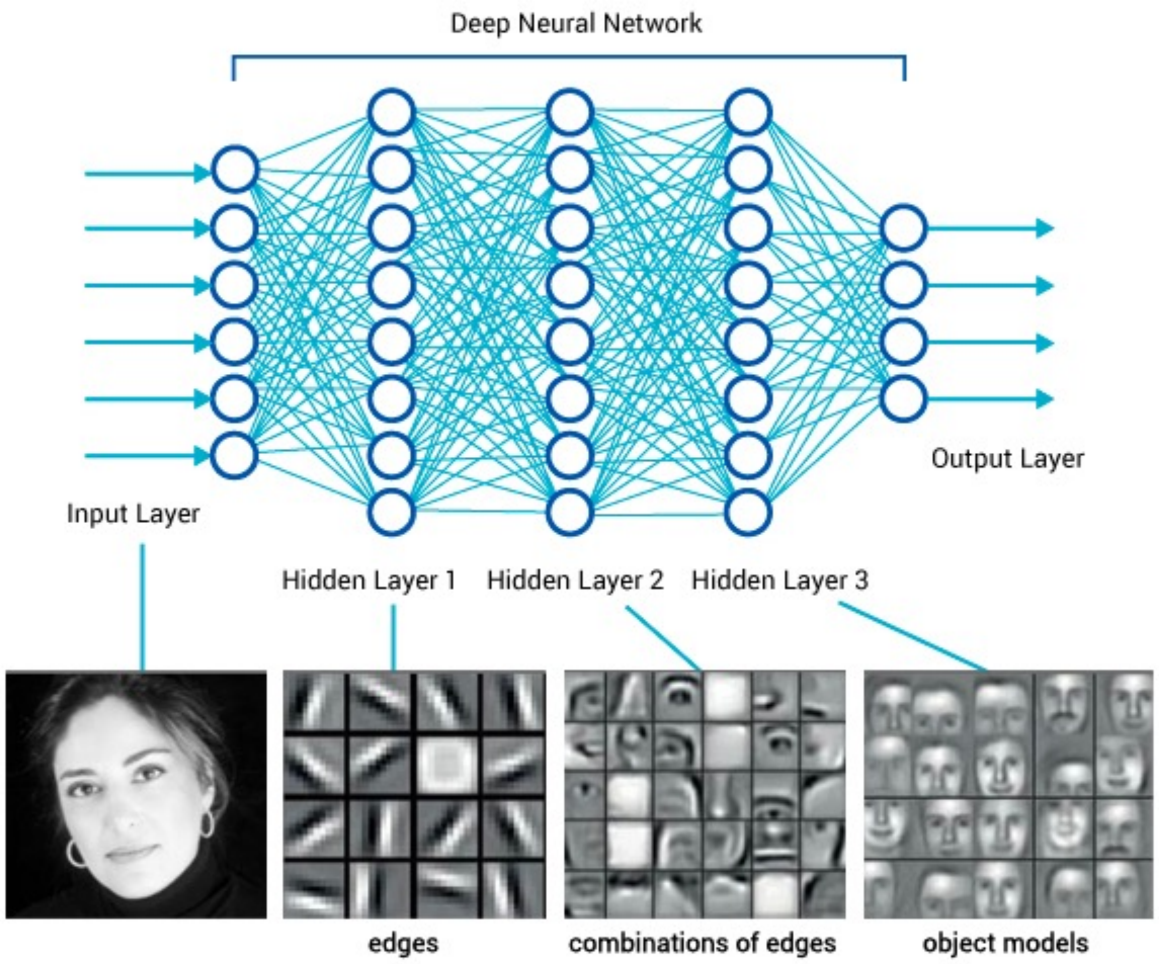
Dropout Layer



(a) Standard Neural Net (b) After applying dropout.

Srivastava et al 2014

What is happening?



https://www.saagie.com/fr/blog/object-detection-part1

SIFT + FV + SVM (or softmax)

Feature extraction (SIFT) → Feature encoding (Bag of words) → Classification (SVM or softmax) →

Deep Learning

Convolutional Network
(includes both feature extraction and classifier) →

Preprocessing and Data Augmentation

Preprocessing and Data Augmentation

256

256

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**
Sparse coding [2]	*47.1%*	*28.2%*
SIFT + FVs [24]	*45.7%*	*25.7%*
CNN	**37.5%**	**17.0%**