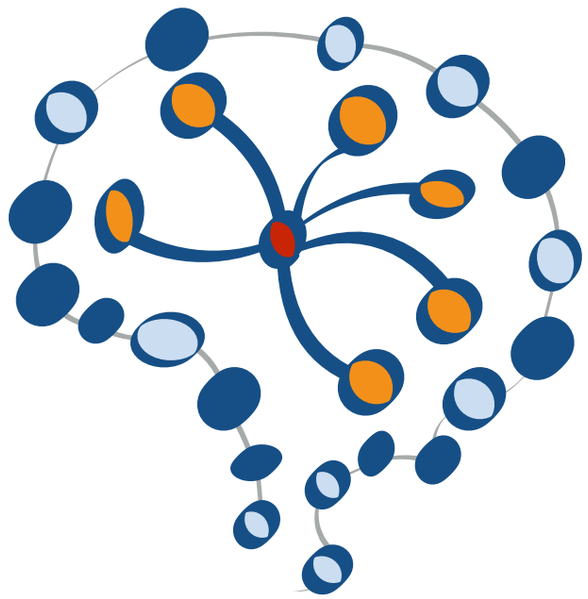
STAT 453: Introduction to Deep Learning and Generative Models

Sebastian Raschka

http://stat.wisc.edu/~sraschka/teaching



Lecture 13

Introduction to Convolutional Neural Networks

Lecture Overview

1. What CNNs Can Do
2. Image Classification
3. Convolutional Neural Network Basics
4. Convolutional Filters and Weight-Sharing
5. Cross-Correlation vs Convolution
6. CNNs & Backpropagation
7. CNN Architectures
8. What a CNN Can See
9. CNNs in PyTorch

Next Lecture

1. Padding (control output size in addition to stride)
2. Spatial Dropout and BatchNorm
3. Considerations for CNNs on GPUs
4. Common Architectures
    A. VGG16 (simple, deep CNN)
    B. ResNet and skip connections
    C. Fully convolutional networks (no fully connected layers)
    D. Inception (parallel convolutions and auxiliary losses)
5. Transfer learning

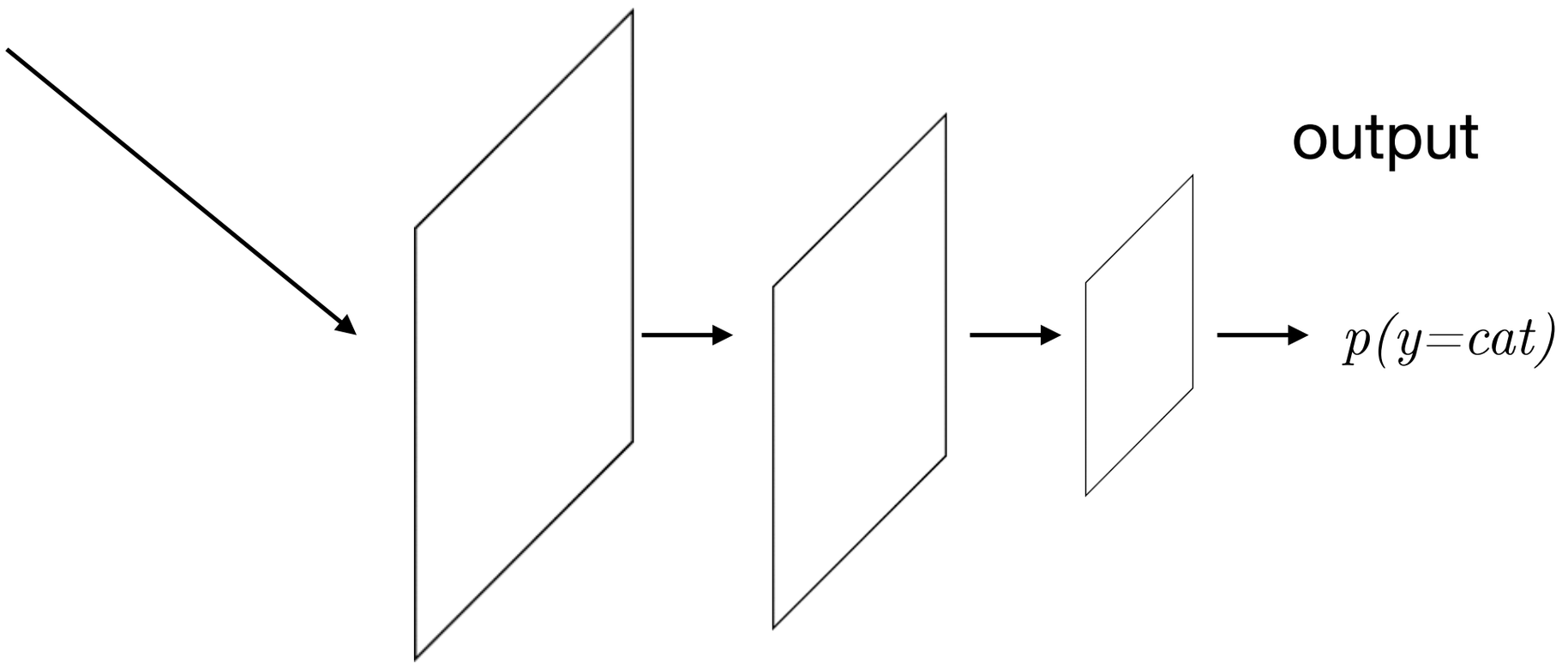
Common Applications of CNNs

1. **What CNNs Can Do**
2. Image Classification
3. Convolutional Neural Network Basics
4. Convolutional Filters and Weight-Sharing
5. Cross-Correlation vs Convolution
6. CNNs & Backpropagation
7. CNN Architectures
8. What a CNN Can See
9. CNNs in PyTorch

CNNs for Image Classification

Image Source: twitter.com%2Fcats&psig=AOvVaw30_o-PCM-K21DiMAJQimQ4&ust=1553887775741551

Image Source: https://www.pinterest.com/pin/244742560974520446



Object Detection

Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You only look once: Unified, real-time object detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (pp. 779-788).

Object Segmentation

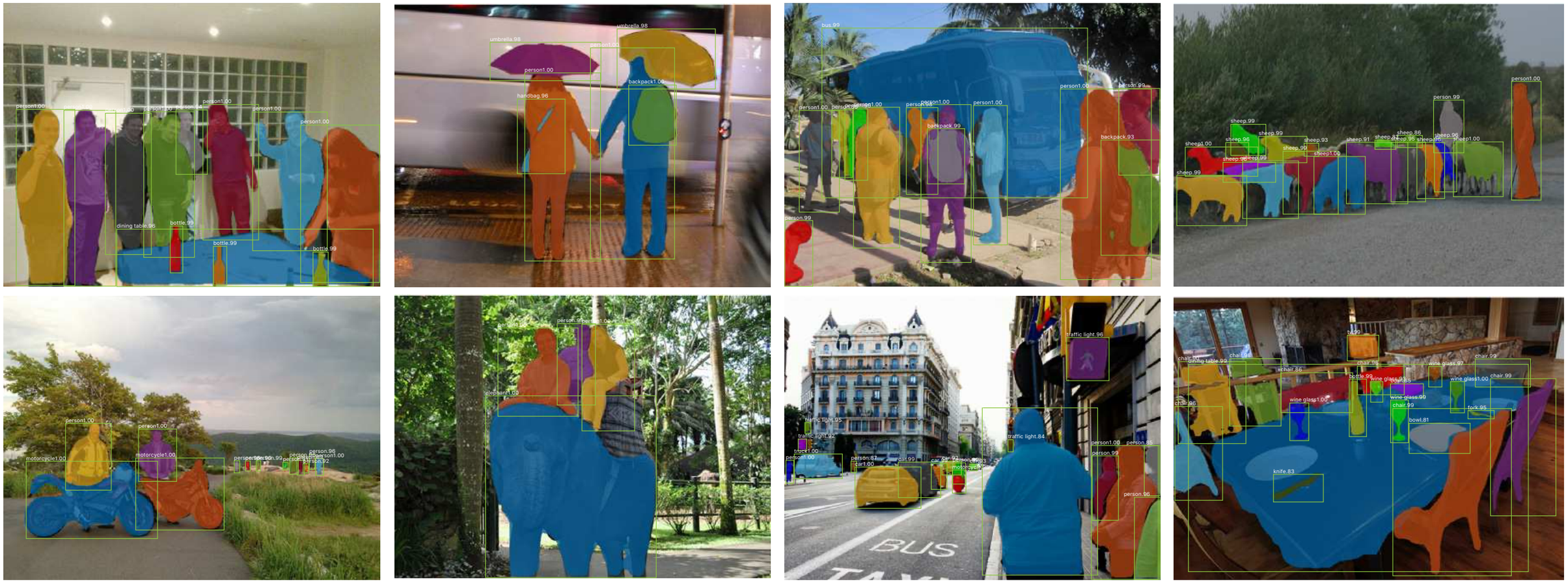


Figure 2. **Mask R-CNN** results on the COCO test set. These results are based on ResNet-101 [15], achieving a *mask* AP of 35.7 and running at 5 fps. Masks are shown in color, and bounding box, category, and confidences are also shown.

He, Kaiming, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. "Mask R-CNN." In *Proceedings of the IEEE International Conference on Computer Vision*, pp. 2961-2969. 2017.

Face Recognition

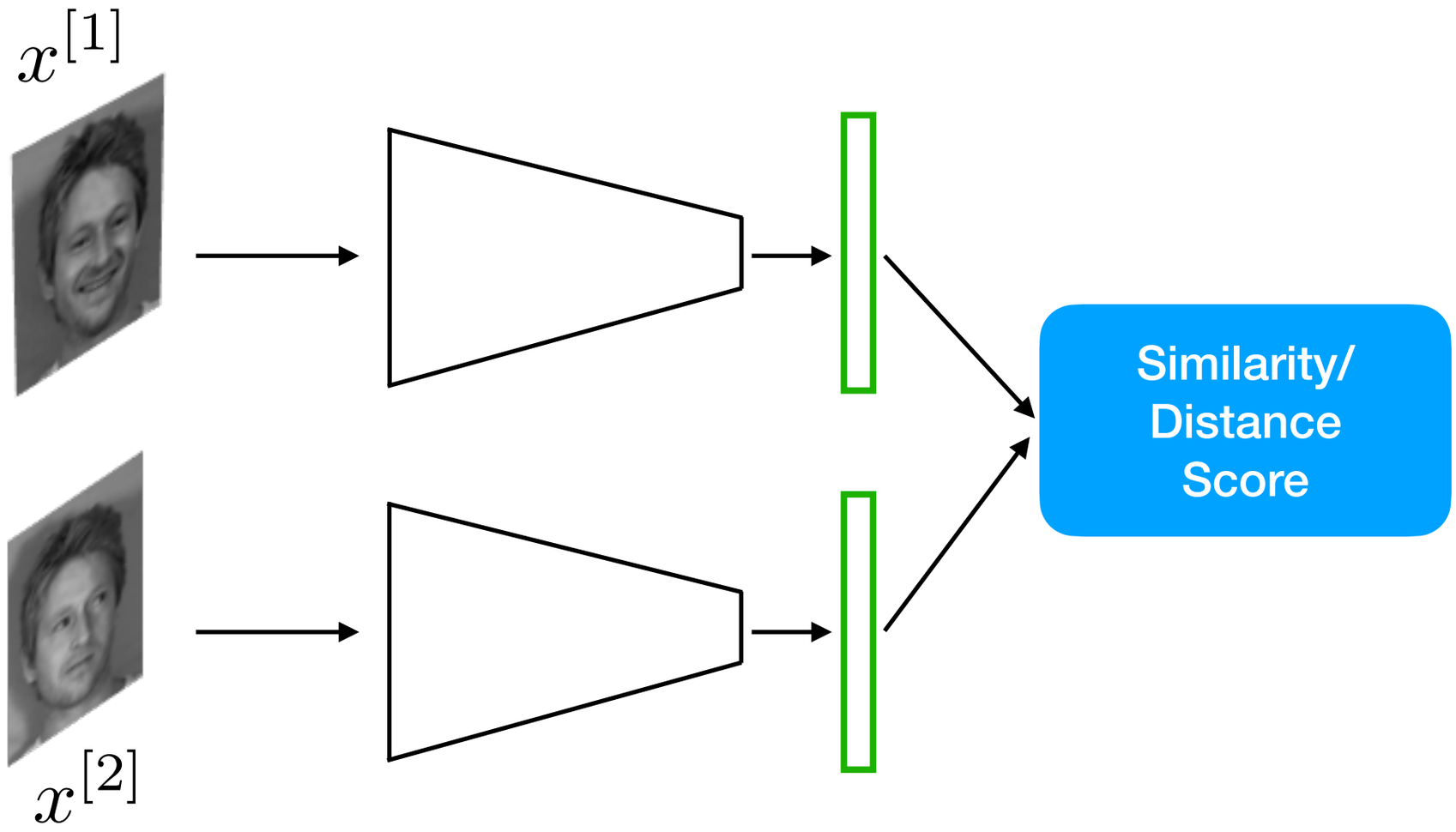
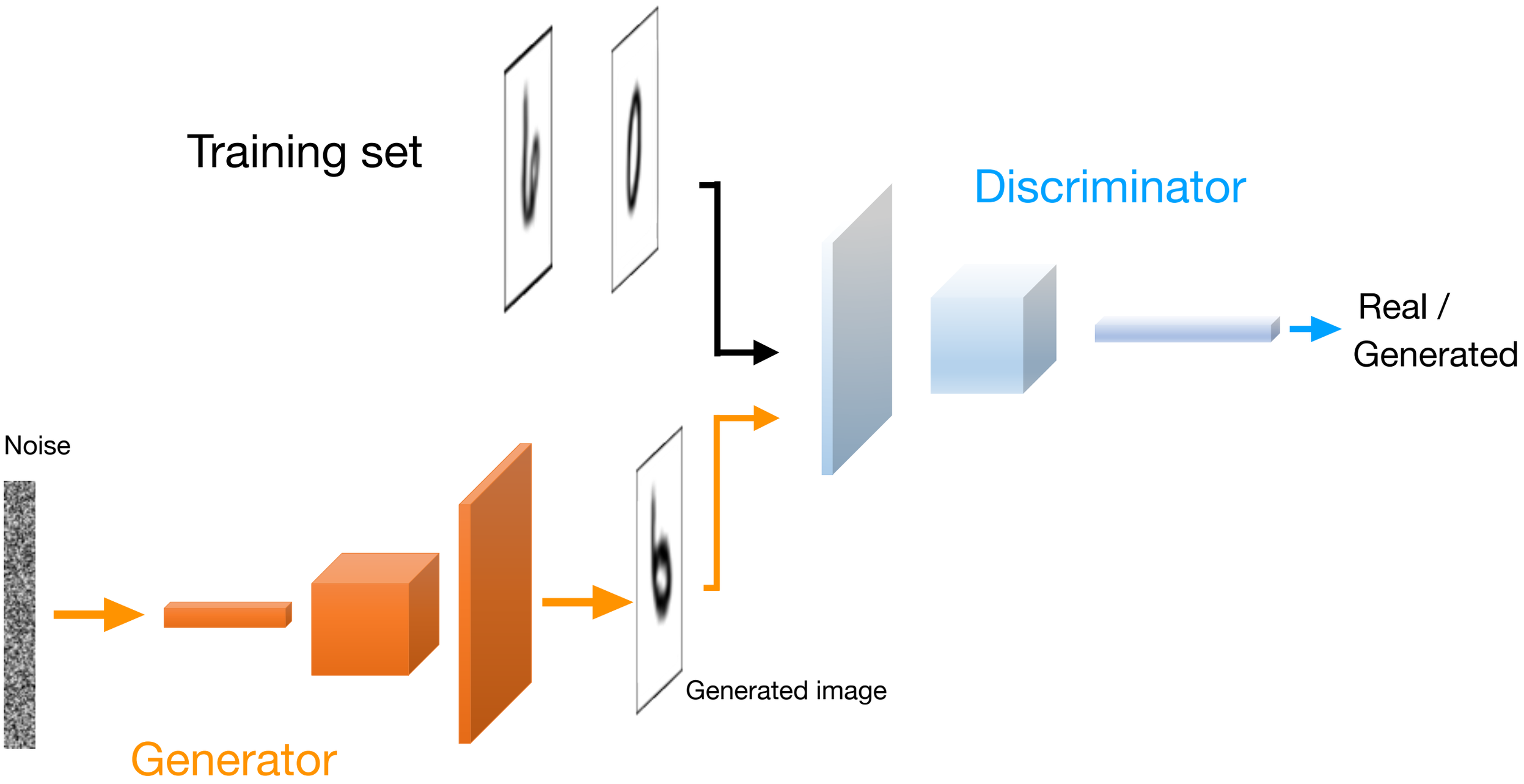


Image Synthesis



Why Computer Vision is (was) Hard: Challenges with Image Classification

1. What CNNs Can Do
2. **Image Classification**
3. Convolutional Neural Network Basics
4. Convolutional Filters and Weight-Sharing
5. Cross-Correlation vs Convolution
6. CNNs & Backpropagation
7. CNN Architectures
8. What a CNN Can See
9. CNNs in PyTorch

Why Image Classification is Hard

Different lighting, contrast, viewpoints, etc.

Image Source: twitter.com%2Fcats&psig=AOvVaw30_o-PCM-K21DiMAJQimQ4&ust=1553887775741551

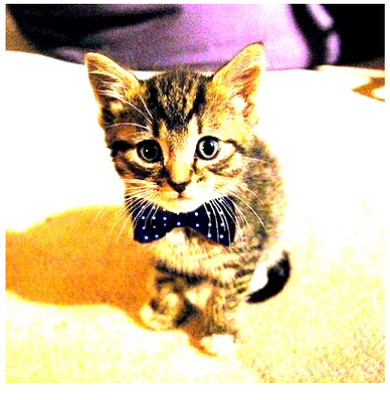
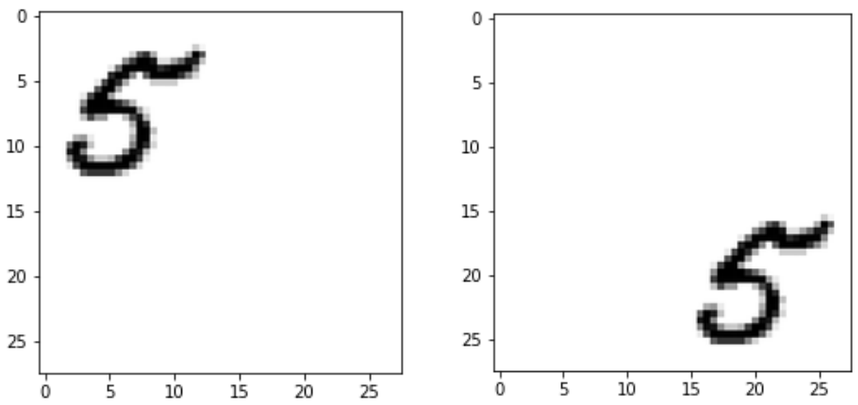


Image Source: https://www.123rf.com/photo_76714328_side-view-of-tabby-cat-face-over-white.html

Or even simple translation



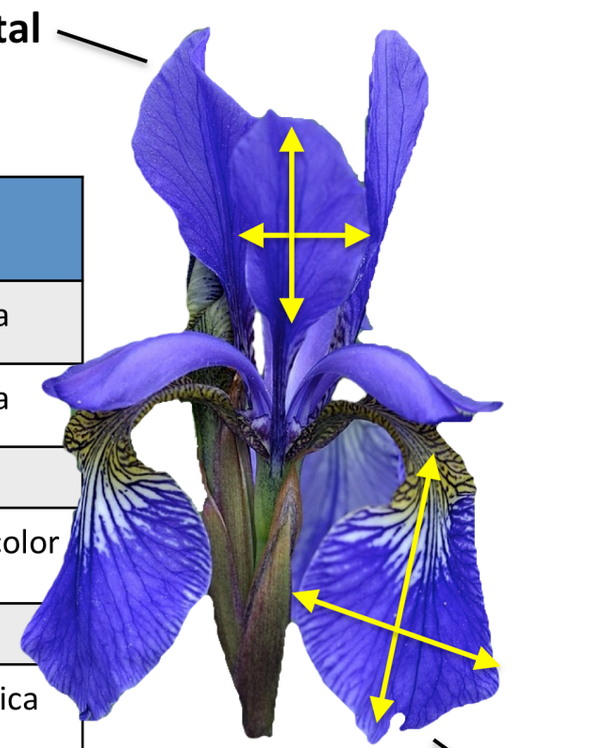
This is hard for traditional methods like multi-layer perceptrons, because the prediction is basically based on a sum of pixel intensities

Traditional Approaches

a) Use hand-engineered features

Samples
(instances, observations)

	Sepal length	Sepal width	Petal length	Petal width	Class label
1	5.1	3.5	1.4	0.2	Setosa
2	4.9	3.0	1.4	0.2	Setosa
...					
50	6.4	3.5	4.5	1.2	Versicolor
...					
150	5.9	3.0	5.0	1.8	Virginica

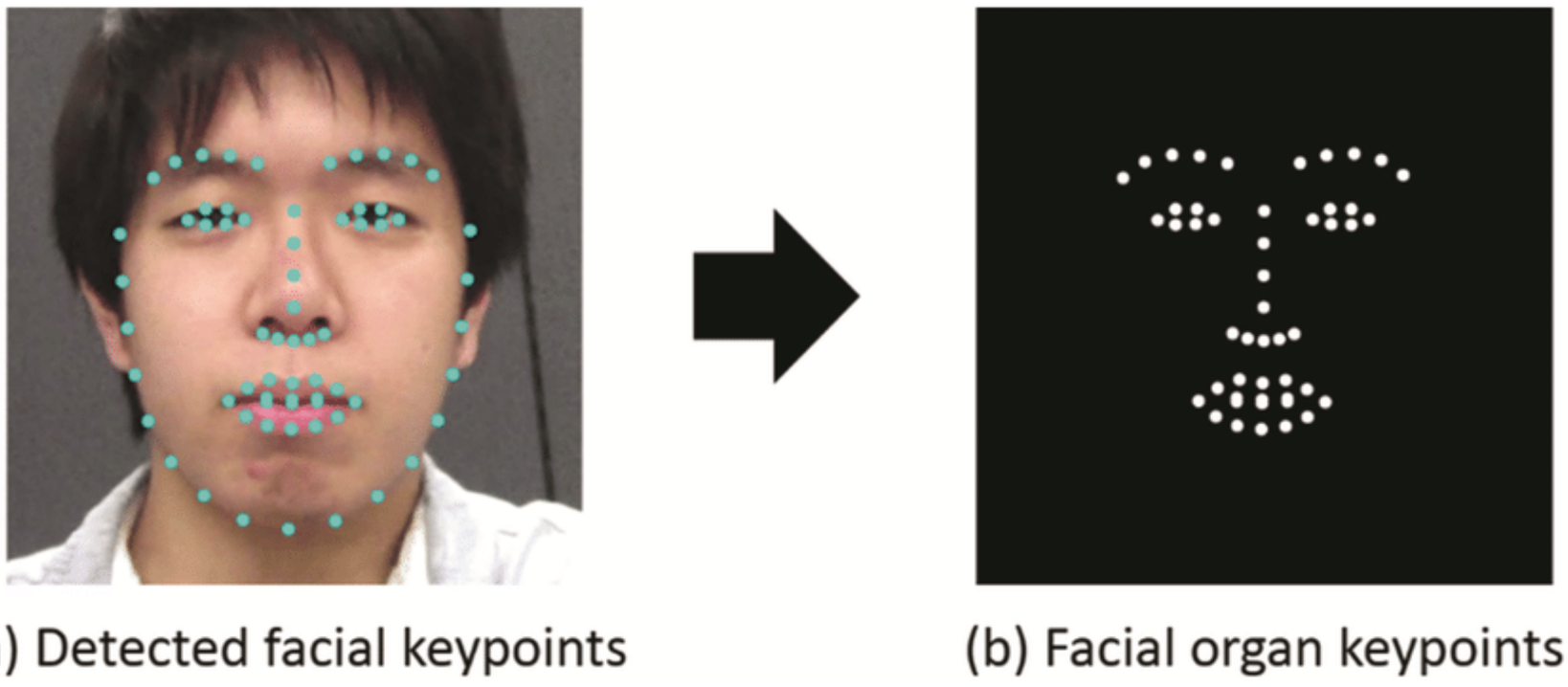


Features
(attributes, measurements, dimensions)

Class labels
(targets)

Traditional Approaches

a) Use hand-engineered features



(a) Detected facial keypoints (b) Facial organ keypoints

Sasaki, K., Hashimoto, M., & Nagata, N. (2016). Person Invariant Classification of Subtle Facial Expressions Using Coded Movement Direction of Keypoints. In *Video Analytics. Face and Facial Expression Recognition and Audience Measurement* (pp. 61-72). Springer, Cham.

Traditional Approaches

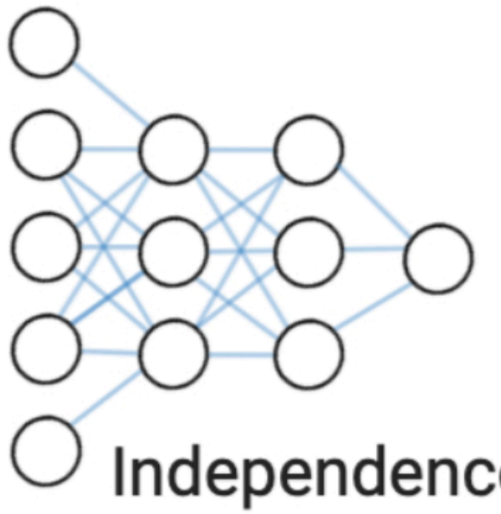
b) Preprocess images (centering, cropping, etc.)

Image Source: https://www.tokkoro.com/2827328-cat-animals-nature-feline-park-green-trees-grass.html

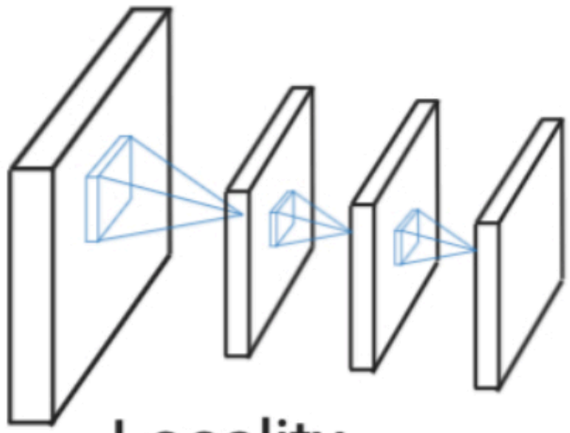
The Main Concepts Behind Convolutional Neural Networks

1. What CNNs Can Do
2. Image Classification
3. **Convolutional Neural Network Basics**
4. Convolutional Filters and Weight-Sharing
5. Cross-Correlation vs Convolution
6. CNNs & Backpropagation
7. CNN Architectures
8. What a CNN Can See
9. CNNs in PyTorch

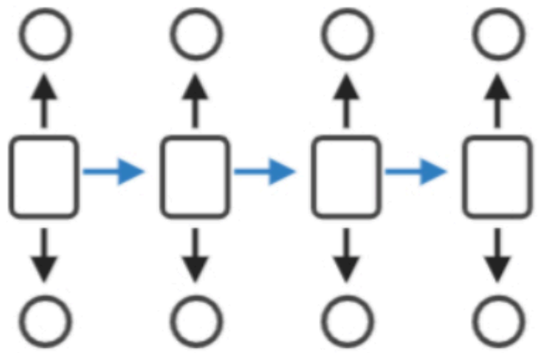
Relational Inductive Biases



Independence



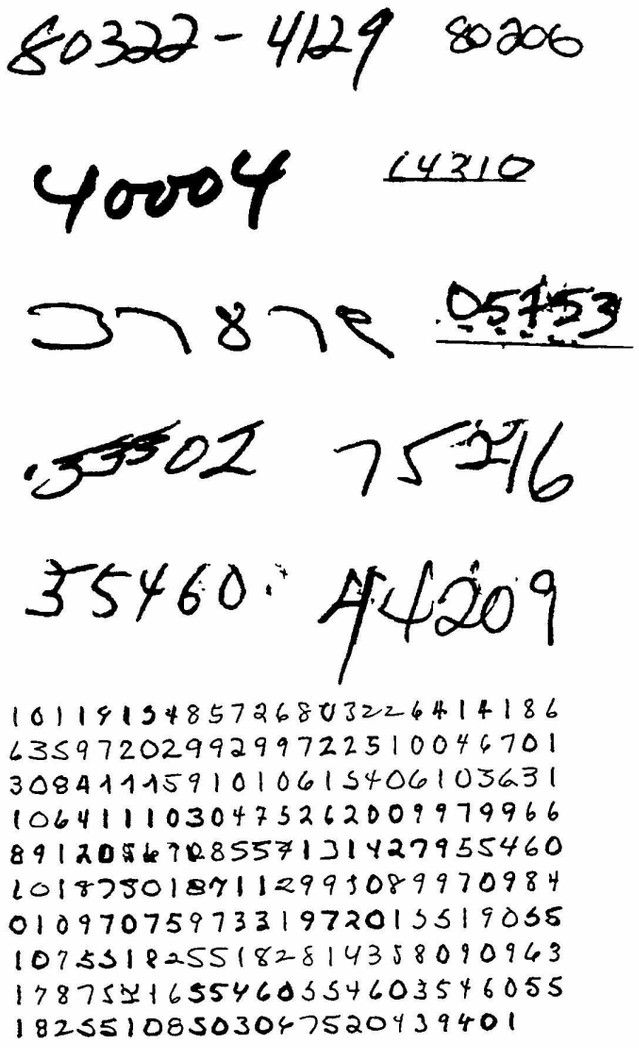
Locality

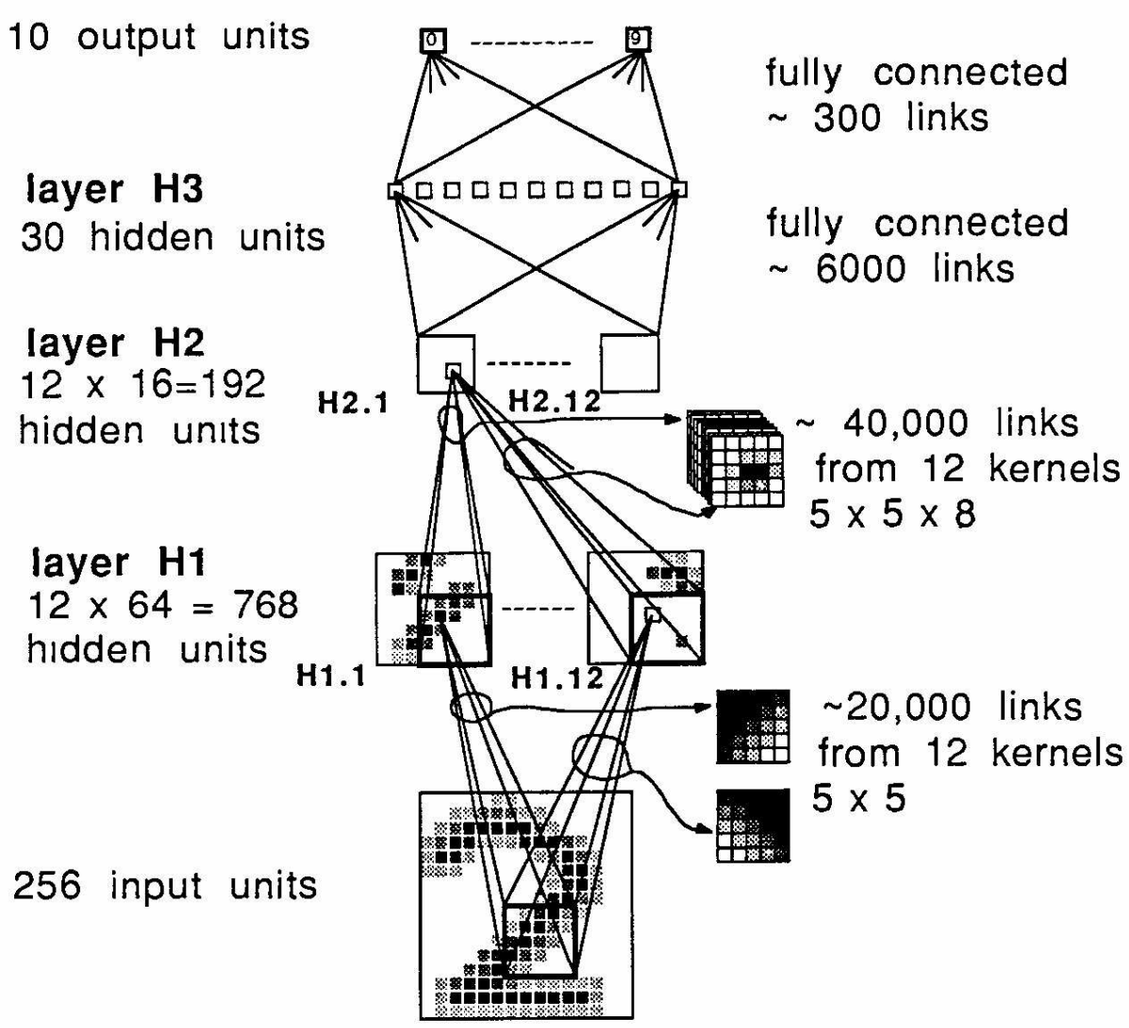


Sequentiality

https://sgfin.github.io/2020/06/22/Induction-Intro/

Convolutional Neural Networks





Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard and L. D. Jackel: Backpropagation Applied to Handwritten Zip Code Recognition, Neural Computation, 1(4):541-551, Winter 1989.

Convolutional Neural Networks

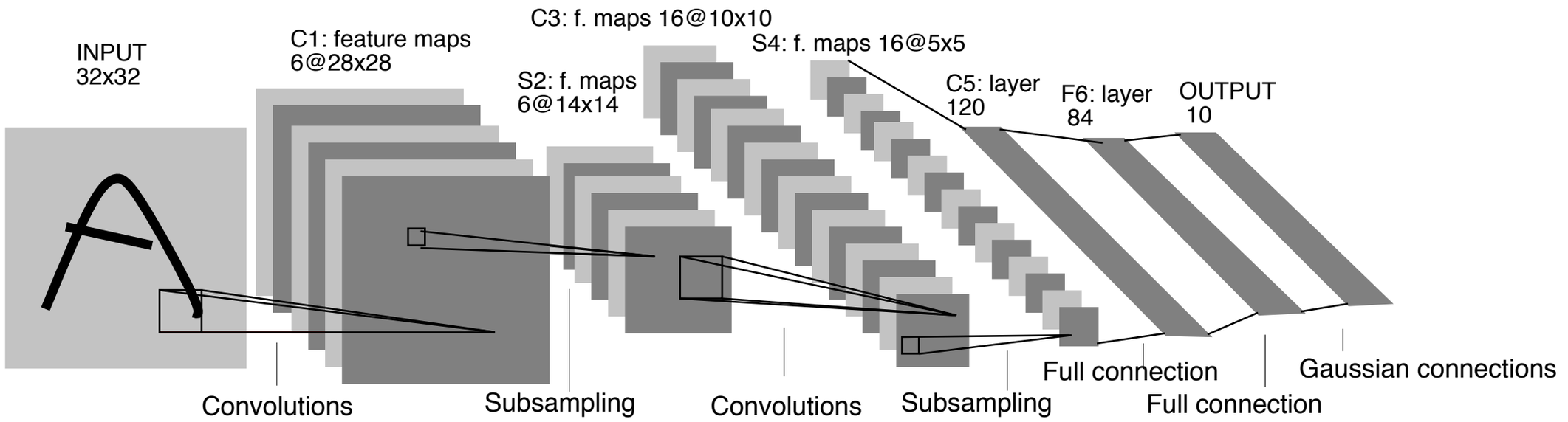
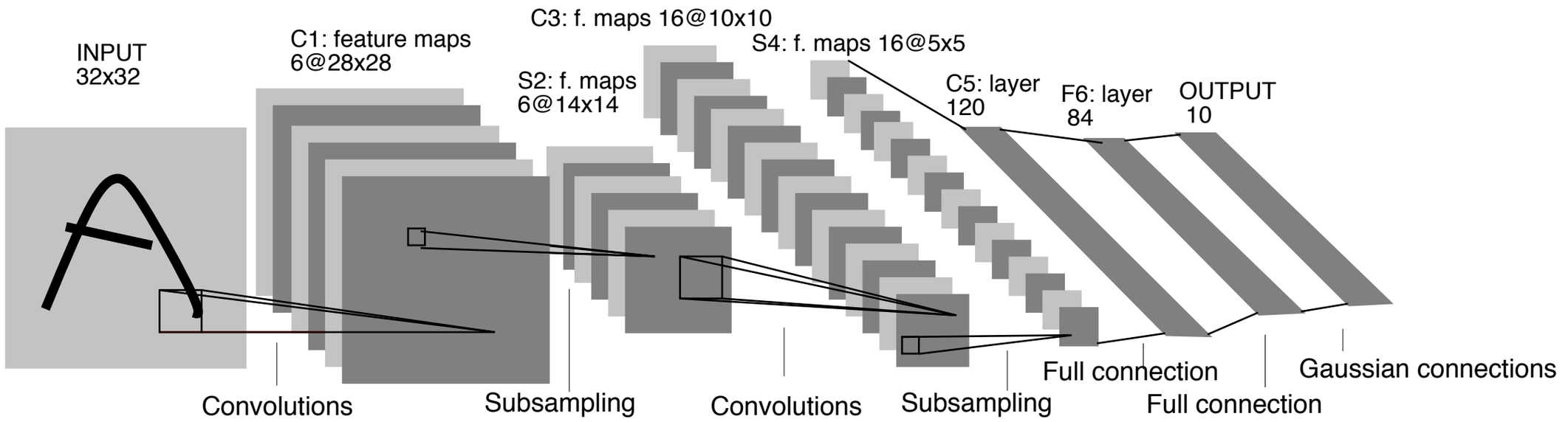
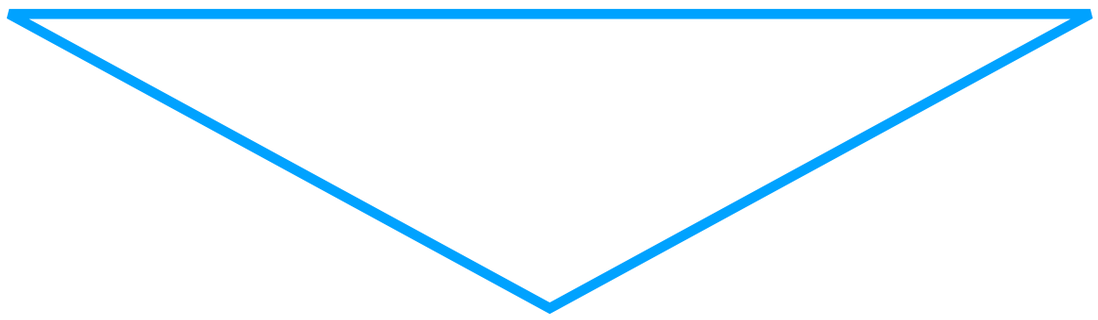


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

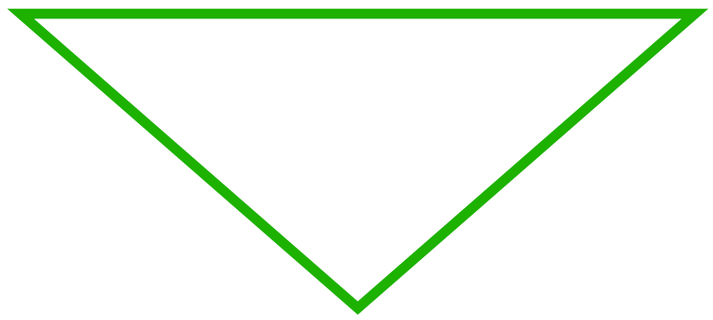
Yann LeCun, Léon Bottou, Yoshua Bengio and Patrick Haffner: Gradient Based Learning Applied to Document Recognition, Proceedings of IEEE, 86(11):2278–2324, 1998.

Hidden Layers





"Automatic feature extractor"



"Regular classifier"

Hidden Layers

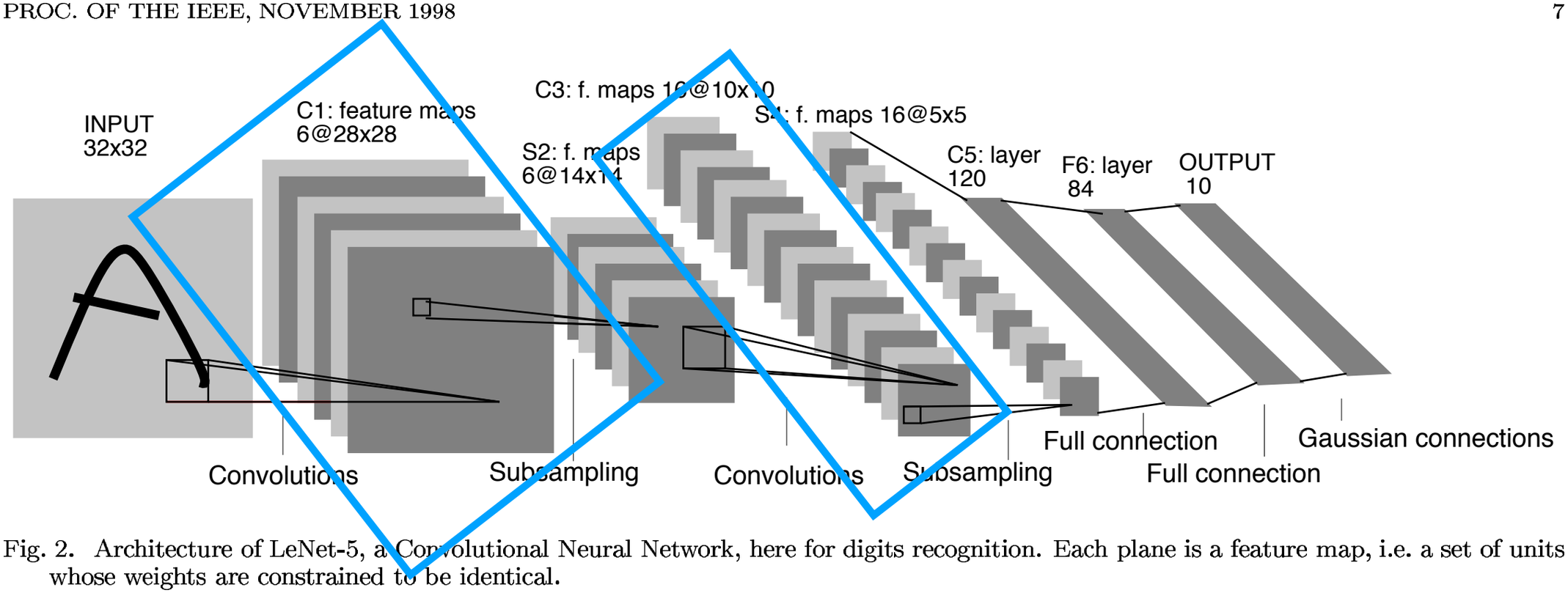


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

Each "bunch" of feature maps represents one hidden layer in the neural network.

Counting the FC layers, this network has 5 layers

Convolutional Neural Networks

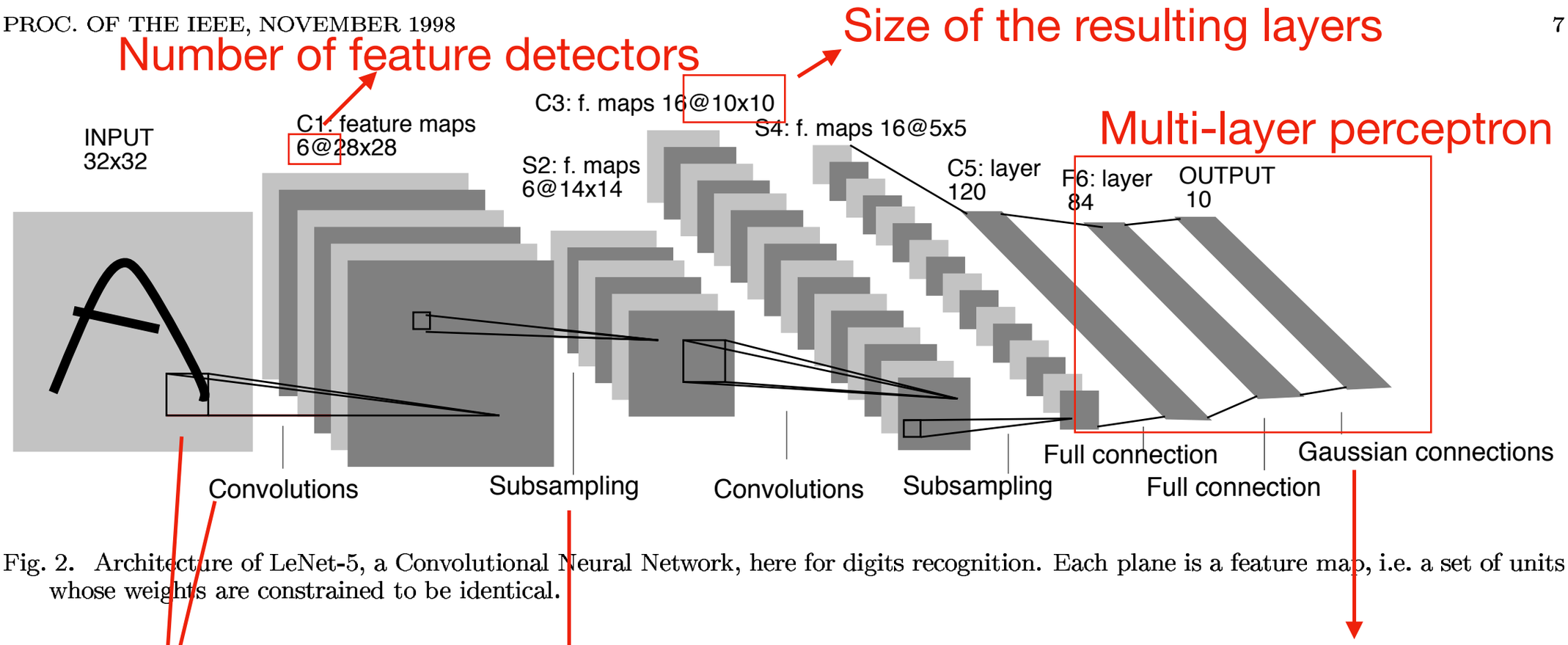


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

nowadays called "pooling"

basically a fully-connected layer + MSE loss
(nowadays common to use fc-layer + softmax + cross entropy)

"Feature detectors" (weight matrices) that are being reused ("weight sharing") => also called "kernel" or "filter"

Yann LeCun, Léon Bottou, Yoshua Bengio and Patrick Haffner: Gradient Based Learning Applied to Document Recognition, Proceedings of IEEE, 86(11):2278–2324, 1998.

Main Concepts Behind Convolutional Neural Networks

- **Sparse-connectivity:** A single element in the feature map is connected to only a small patch of pixels. (This is very different from connecting to the whole input image, in the case of multi-layer perceptrons.)

- **Parameter-sharing:** The same weights are used for different patches of the input image.

- **Many layers:** Combining extracted local patterns to global patterns

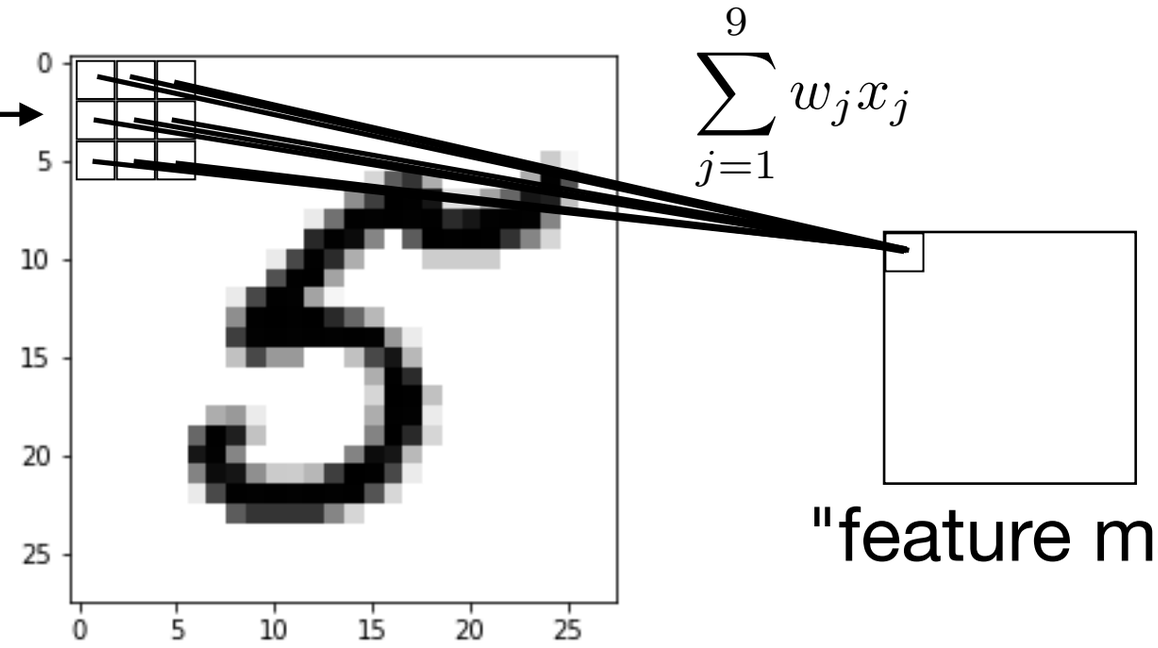
A Closer Look at the Convolutional Layer

1. What CNNs Can Do
2. Image Classification
3. Convolutional Neural Network Basics
4. **Convolutional Filters and Weight-Sharing**
5. Cross-Correlation vs Convolution
6. CNNs & Backpropagation
7. CNN Architectures
8. What a CNN Can See
9. CNNs in PyTorch

Weight Sharing

A "feature detector" (filter, kernel) slides over the inputs to generate a feature map

The pixels are referred to as "receptive field"

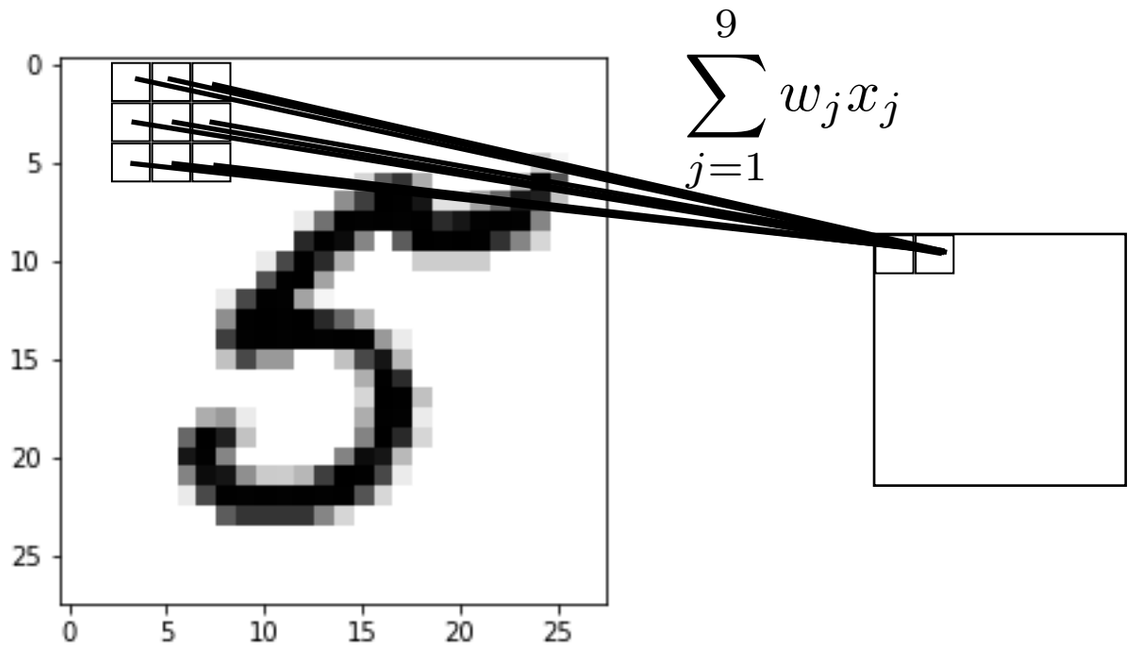


Rationale: A feature detector that works well in one region may also work well in another region

Plus, it is a nice reduction in parameters to fit

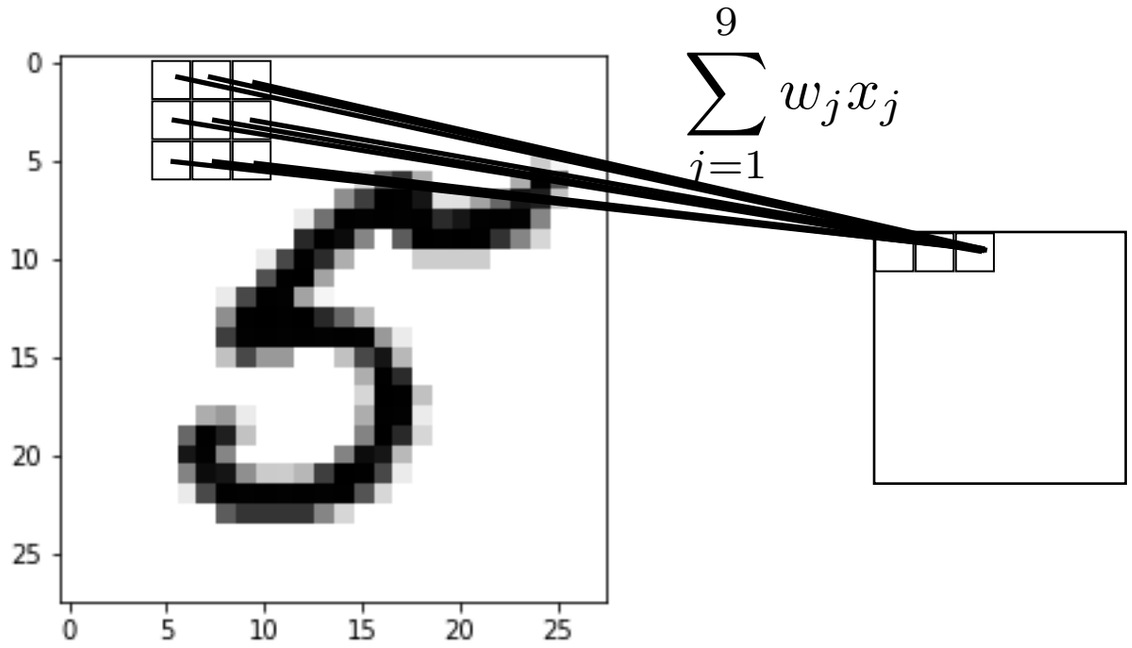
Weight Sharing

A "feature detector" (kernel) slides over the inputs to generate a feature map



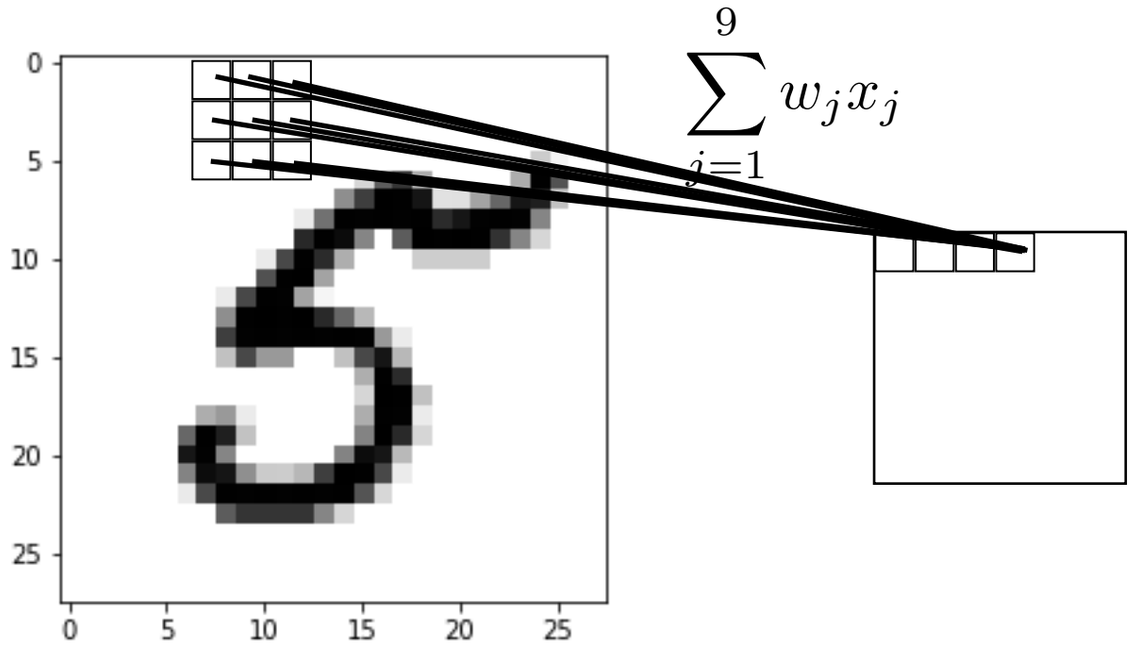
Weight Sharing

A "feature detector" (kernel) slides over the inputs to generate a feature map



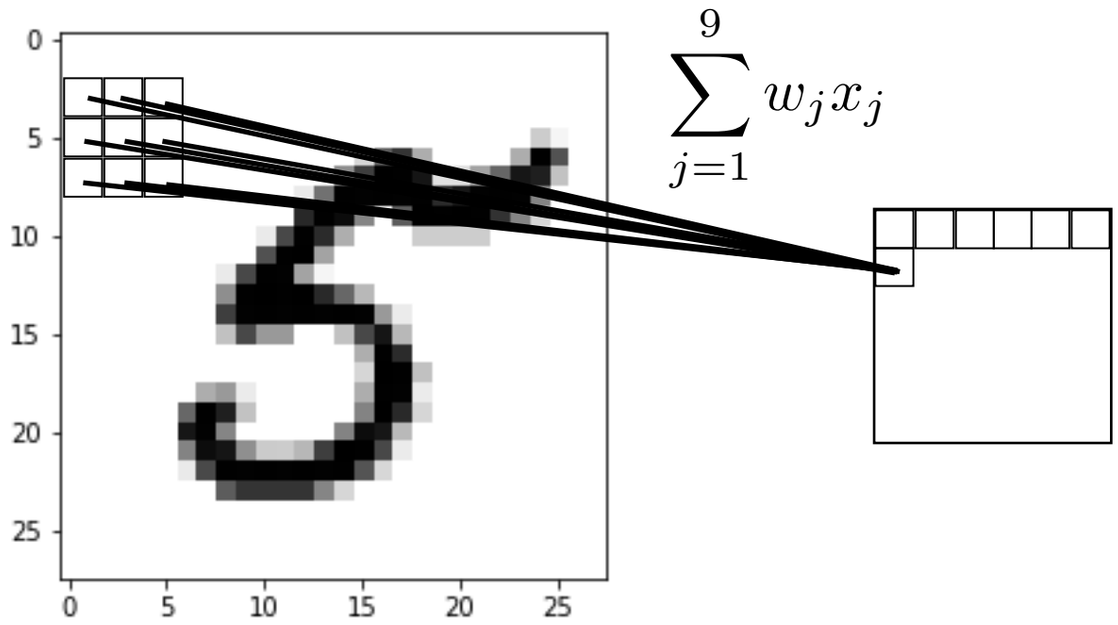
Weight Sharing

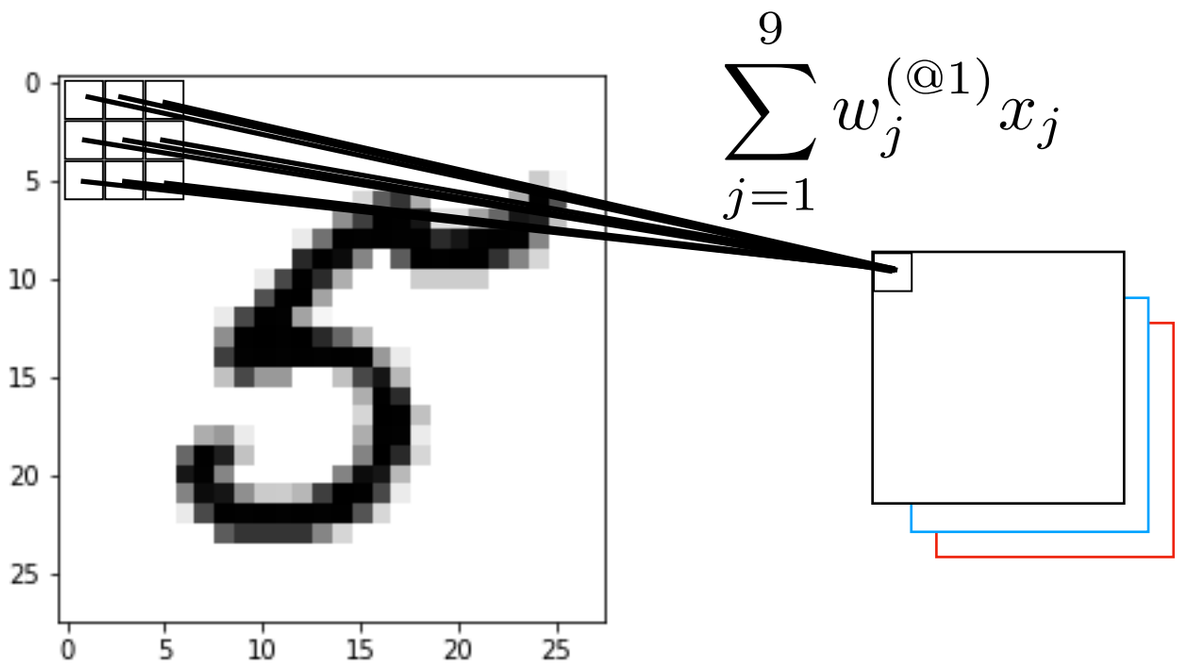
A "feature detector" (kernel) slides over the inputs to generate a feature map

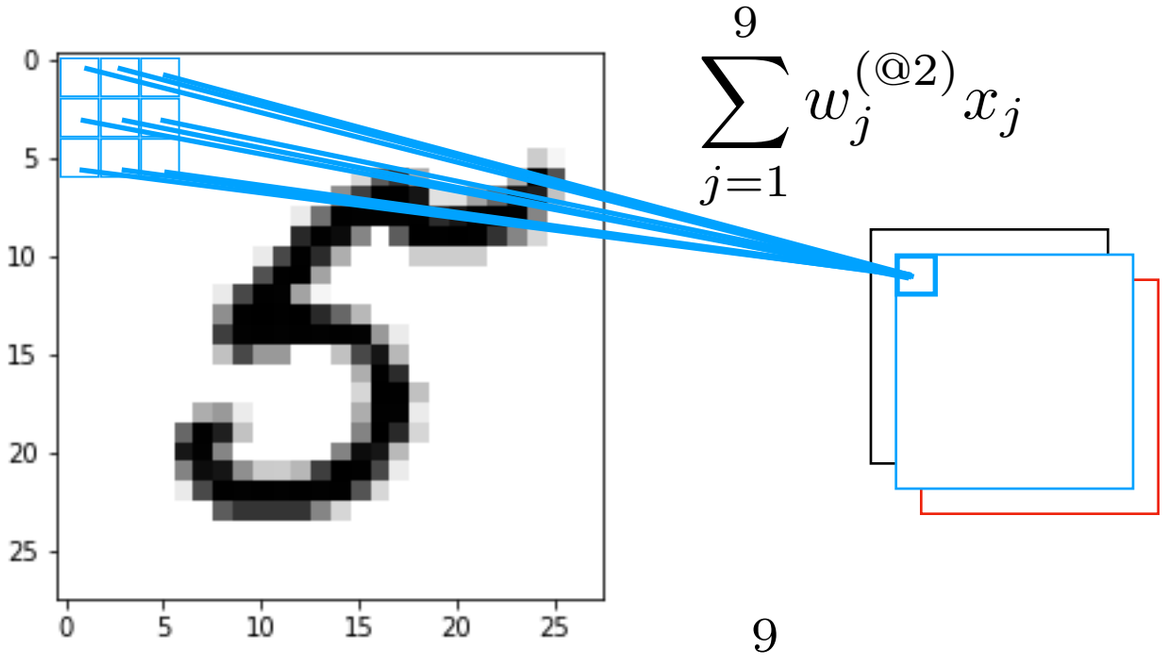


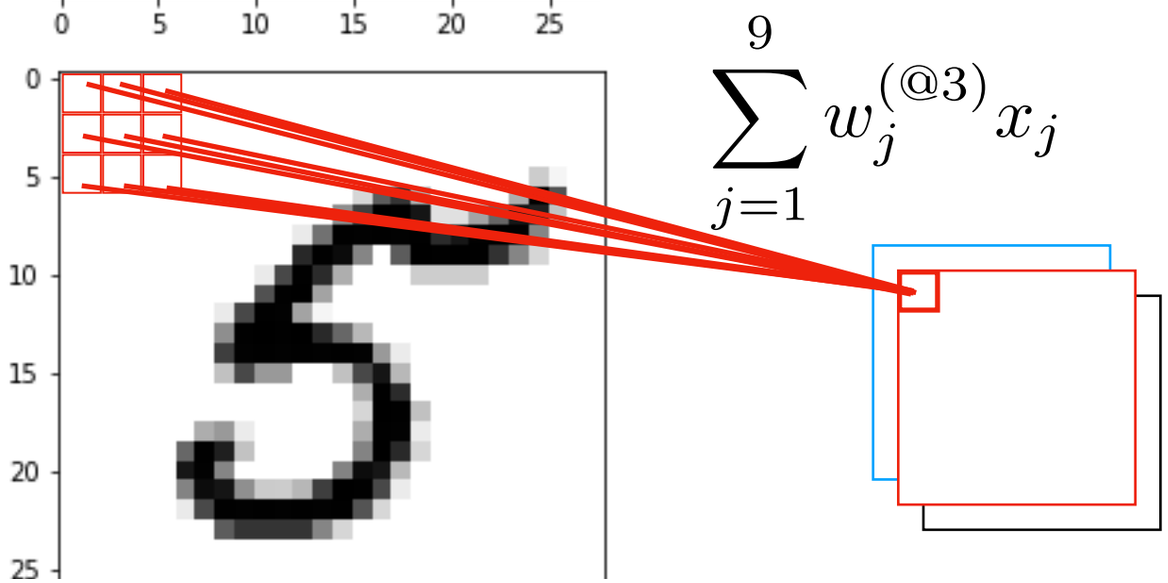
Weight Sharing

A "feature detector" (kernel) slides over the inputs to generate a feature map





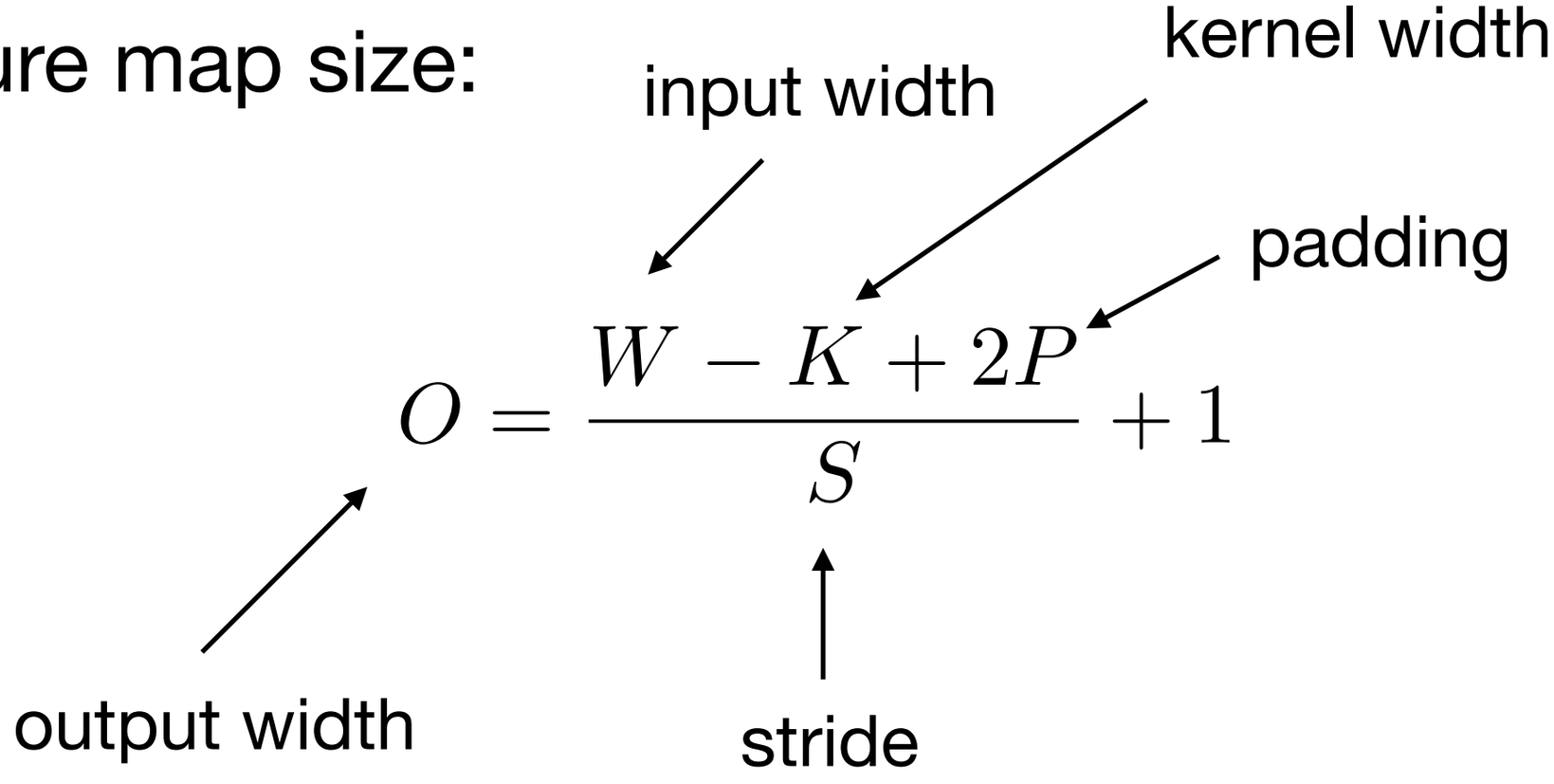




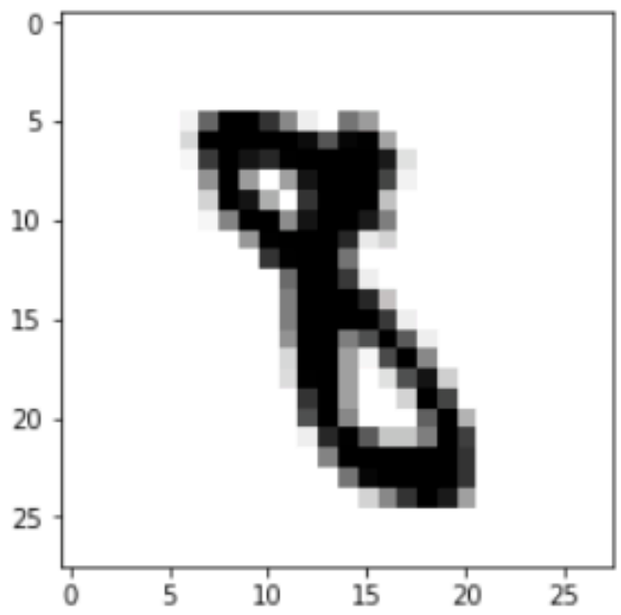
Multiple "feature detectors" (kernels) are used to create multiple feature maps

Size Before and After Convolutions

Feature map size:



Kernel Dimensions and Trainable Parameters




```
a.shape
```

```
(1, 28, 28)
```

```
import torch
```

```
conv = torch.nn.Conv2d(in_channels=1,
                       out_channels=8,
                       kernel_size=(5, 5),
                       stride=(1, 1))
```

```
conv.weight.size()
```

```
torch.Size([8, 1, 5, 5])
```

```
conv.bias.size()
```

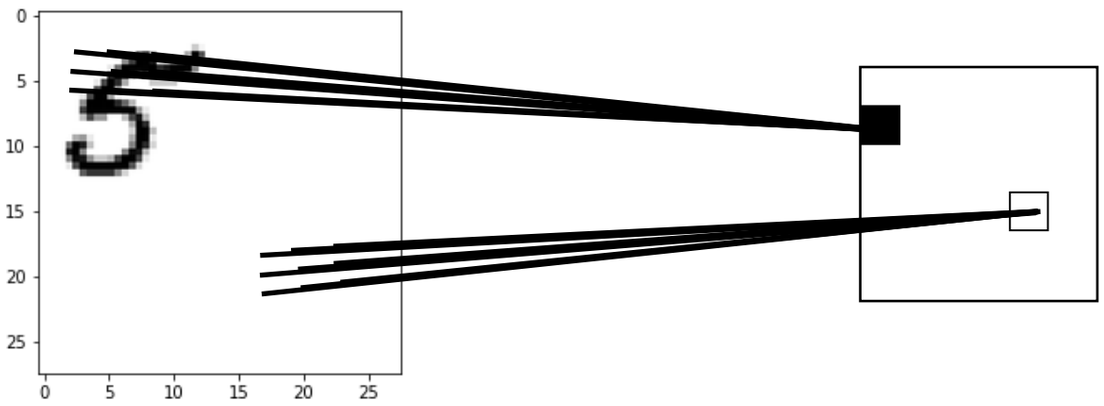
```
torch.Size([8])
```

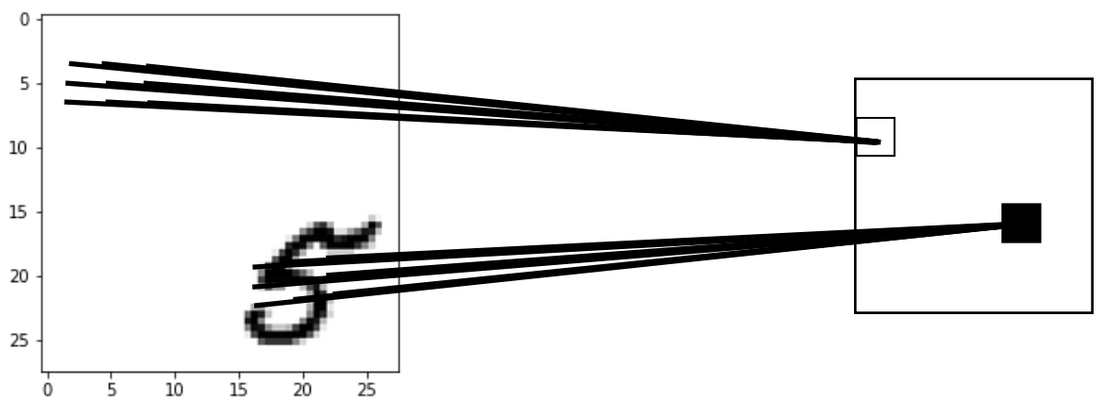
For a grayscale image with a 5x5 feature detector (kernel), we have the following dimensions (number of parameters to learn)

What do you think is the output size for this 28x28 image?

CNNs and Translation/Rotation/Scale Invariance

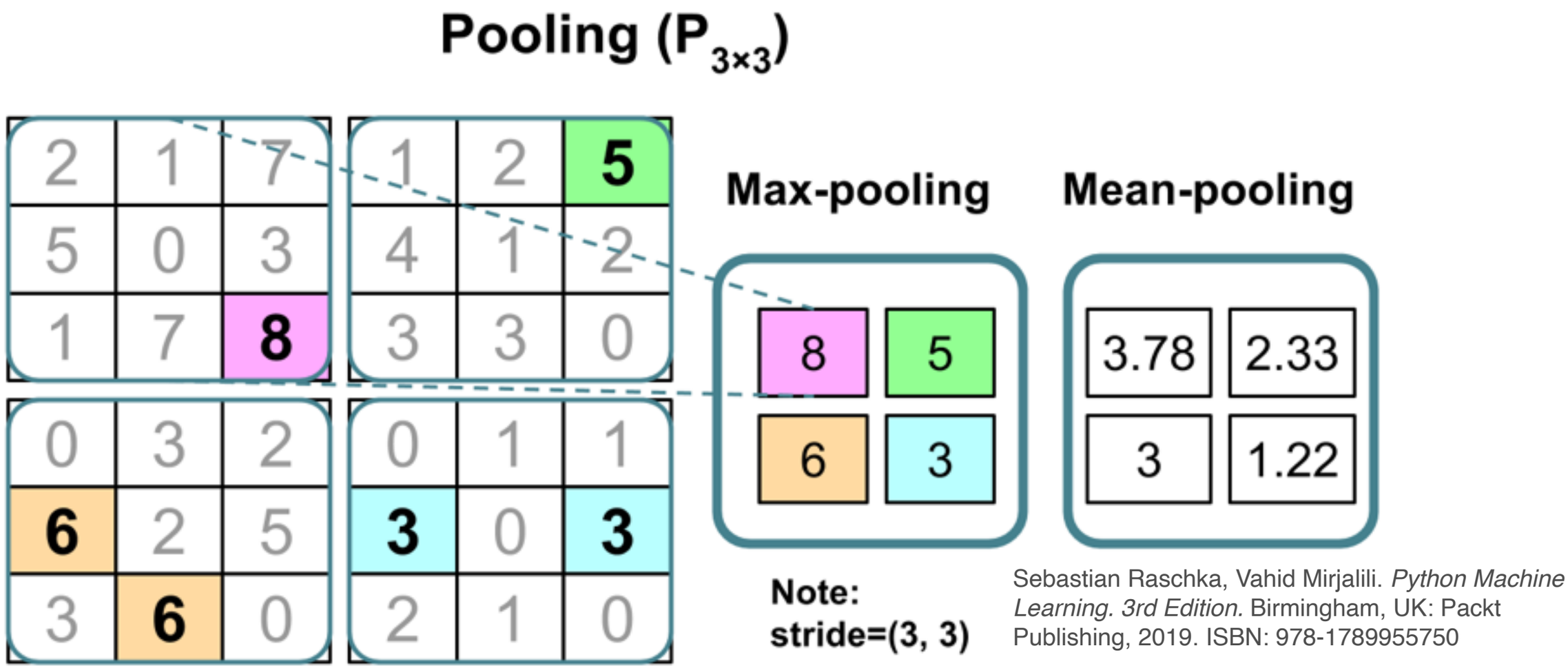
Note that CNNs are not really invariant to scale, rotation, translation, etc.





The activations are still dependent on the location, etc.

Pooling Layers Can Help With Local Invariance

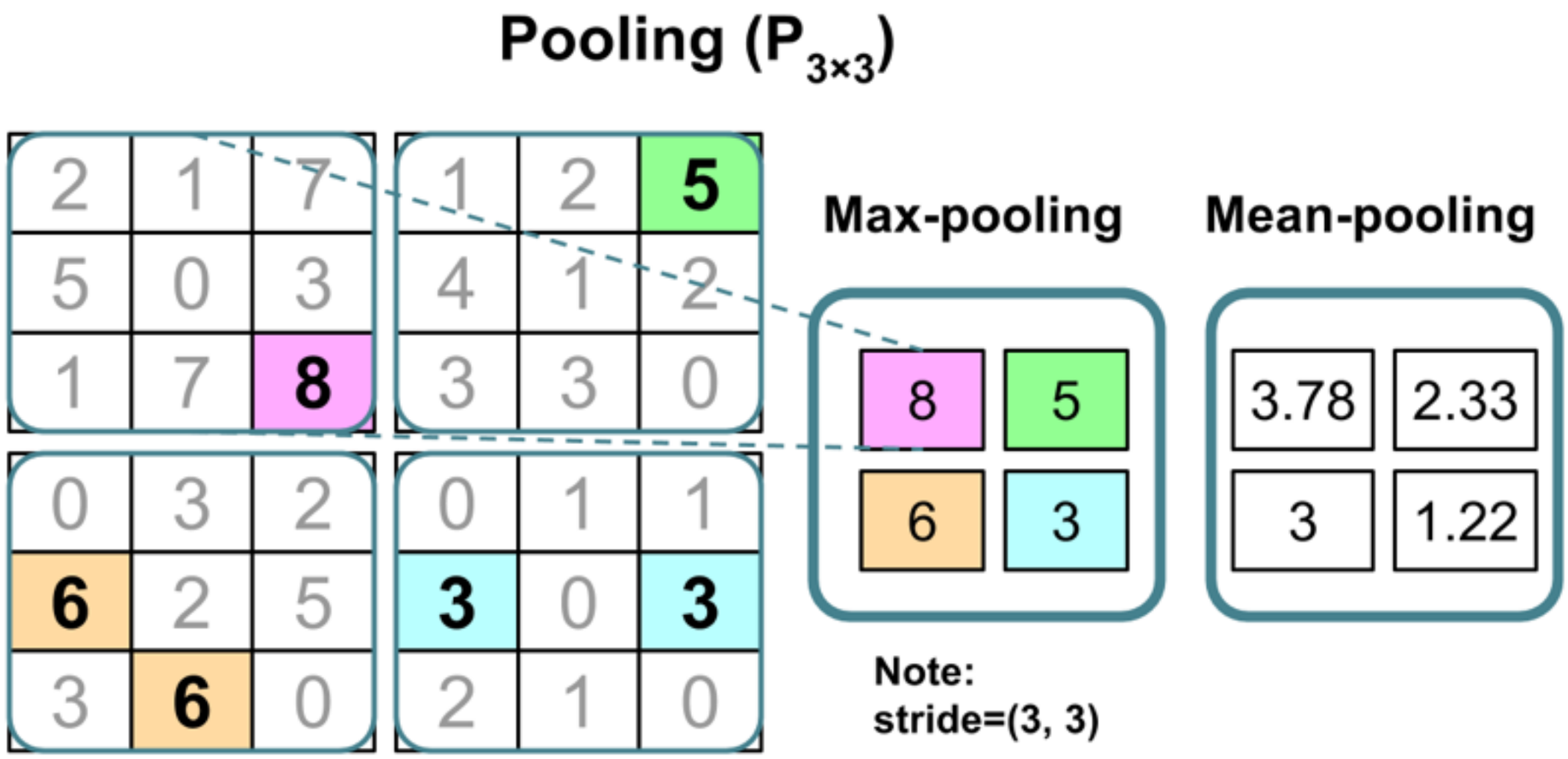


Sebastian Raschka, Vahid Mirjalili. *Python Machine Learning. 3rd Edition.* Birmingham, UK: Packt Publishing, 2019. ISBN: 978-1789955750

Downside: Information is lost.
May not matter for classification, but applications where relative position is important (like face recognition)

In practice for CNNs: some image preprocessing still recommended

Pooling Layers Can Help With Local Invariance



Note that typical pooling layers do not have any learnable parameters

Downside: Information is lost.
May not matter for classification, but applications where relative position is important (like face recognition)

In practice for CNNs: some image preprocessing still recommended

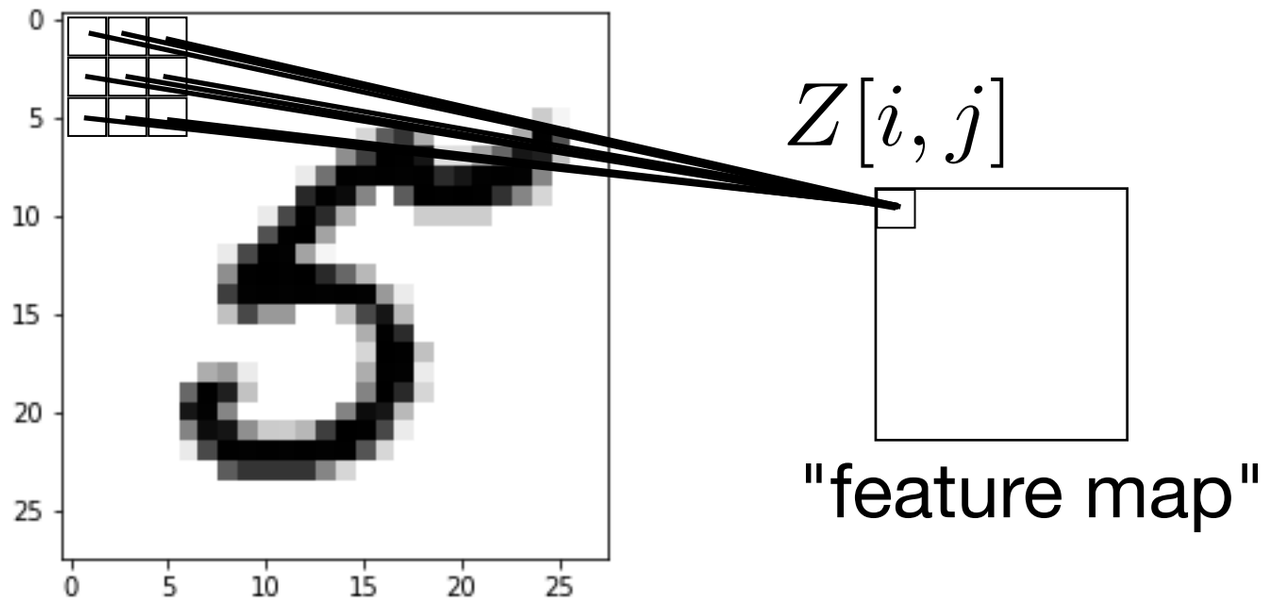
Why are Convolutional Nets Using Cross-Correlation?

1. What CNNs Can Do
2. Image Classification
3. Convolutional Neural Network Basics
4. Convolutional Filters and Weight-Sharing
5. **Cross-Correlation vs Convolution**
6. CNNs & Backpropagation
7. CNN Architectures
8. What a CNN Can See
9. CNNs in PyTorch

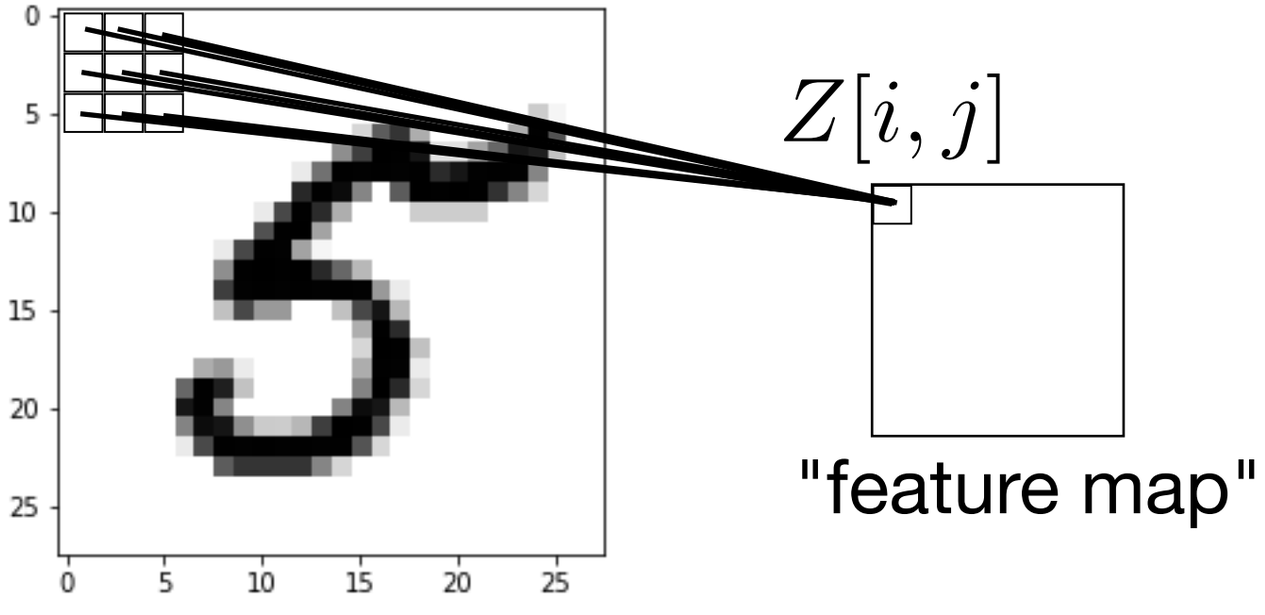
Cross-Correlation vs Convolution

Deep Learning Jargon: convolution in DL is actually cross-correlation

Cross-correlation is our sliding dot product over the image



Cross-Correlation vs Convolution



Cross-Correlation:

$$Z[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} K[u,v]A[i+u, j+v]$$

$$Z[i,j] = K \otimes A$$

Cross-Correlation vs Convolution

Cross-Correlation:

$$Z[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} K[u,v] A[i+u, j+v]$$

$$Z[i,j] = K \otimes A$$

Looping direction indicated via the red numbers

1) -1,-1	2) -1,0	3) -1,1
4) 0,-1	5) 0,0	6) 0,1
7) 1,-1	8) 1,0	9) 1,1

Cross-Correlation vs Convolution

Cross-Correlation: $Z[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} K[u,v]A[i+u,j+v]$ $\qquad Z[i,j] = K \otimes A$

Convolution: $$Z[i,j] = \sum_{u=-k}^{k} \sum_{v=-k}^{k} K[u,v]A[i-u,j-v]$$

$$Z[i,j] = K * A$$

Basically, we are flipping the kernel (or the receptive field) horizontally and vertically

9) -1,-1	8) -1,0	7) -1,1
6) 0,-1	5) 0,0	4) 0,1
3) 1,-1	2) 1,0	1) 1,1

[1]:
```
from scipy.signal import correlate2d, convolve2d
import torch
```

[2]:
```
a = torch.tensor([[1.1, 1.2, 1.3],
                  [2.1, 2.2, 2.3],
                  [3.1, 3.2, 3.3]])
```

[3]:
```
conv_pytorch = torch.nn.Conv2d(in_channels=1, out_channels=1, kernel_size=(3, 3))
with torch.no_grad():
    conv_pytorch.bias.zero_()

conv_pytorch.weight
```

[3]:
```
Parameter containing:
tensor([[[[ 0.1189, -0.1202,  0.2584],
          [-0.2276, -0.1327, -0.0296],
          [ 0.1741, -0.3225, -0.0288]]]], requires_grad=True)
```

[4]:
```
conv_weight_numpy = conv_pytorch.weight.detach().numpy().reshape(3, 3)
conv_weight_numpy
```

[4]:
```
array([[ 0.11893535, -0.12021737,  0.2583796 ],
       [-0.22761738, -0.13269596, -0.02961969],
       [ 0.17413345, -0.3224947 , -0.02875605]], dtype=float32)
```

Cross-correlation

[5]:
```
conv_pytorch(a.view(1, 1, 3, 3))
```

[5]:
```
tensor([[[[-1.1027]]]])
```

[6]:
```
correlate2d(a.numpy(), conv_weight_numpy, mode='valid')
```

[6]:
```
array([[-1.1026558]], dtype=float32)
```

[1]:
```
from scipy.signal import correlate2d, convolve2d
import torch
```

[2]:
```
a = torch.tensor([[1.1, 1.2, 1.3],
                  [2.1, 2.2, 2.3],
                  [3.1, 3.2, 3.3]])
```

[3]:
```
conv_pytorch = torch.nn.Conv2d(in_channels=1, out_channels=1, kernel_size=(3, 3))
with torch.no_grad():
    conv_pytorch.bias.zero_()

conv_pytorch.weight
```

[3]:
```
Parameter containing:
tensor([[[[ 0.1189, -0.1202,  0.2584],
          [-0.2276, -0.1327, -0.0296],
          [ 0.1741, -0.3225, -0.0288]]]], requires_grad=True)
```

[4]:
```
conv_weight_numpy = conv_pytorch.weight.detach().numpy().reshape(3, 3)
conv_weight_numpy
```

[4]:
```
array([[ 0.11893535, -0.12021737,  0.2583796 ],
       [-0.22761738, -0.13269596, -0.02961969],
       [ 0.17413345, -0.3224947 , -0.02875605]], dtype=float32)
```

Real convolution

[7]:
```
convolve2d(a.numpy(), conv_weight_numpy, mode='valid')
```

[7]:
```
array([[-0.2611365]], dtype=float32)
```

[8]:
```
a_mod = torch.tensor([[3.3, 3.2, 3.1],
                      [2.3, 2.2, 2.1],
                      [1.3, 1.2, 1.1]])


conv_pytorch(a_mod.view(1, 1, 3, 3))
```

[8]:
```
tensor([[[[-0.2611]]]])
```

Cross-Correlation vs Convolution

Deep Learning Jargon: convolution in DL is actually cross-correlation

"Real" convolution has the nice associative property:

$$(A * B) * C = A * (B * C)$$

In DL, we usually don't care about that (as opposed to many traditional computer vision and signal processing applications).

Also, cross-correlation is easier to implement.

Maybe the term "convolution" for cross-correlation became popular, because "Cross-Correlational Neural Network" sounds weird ;)

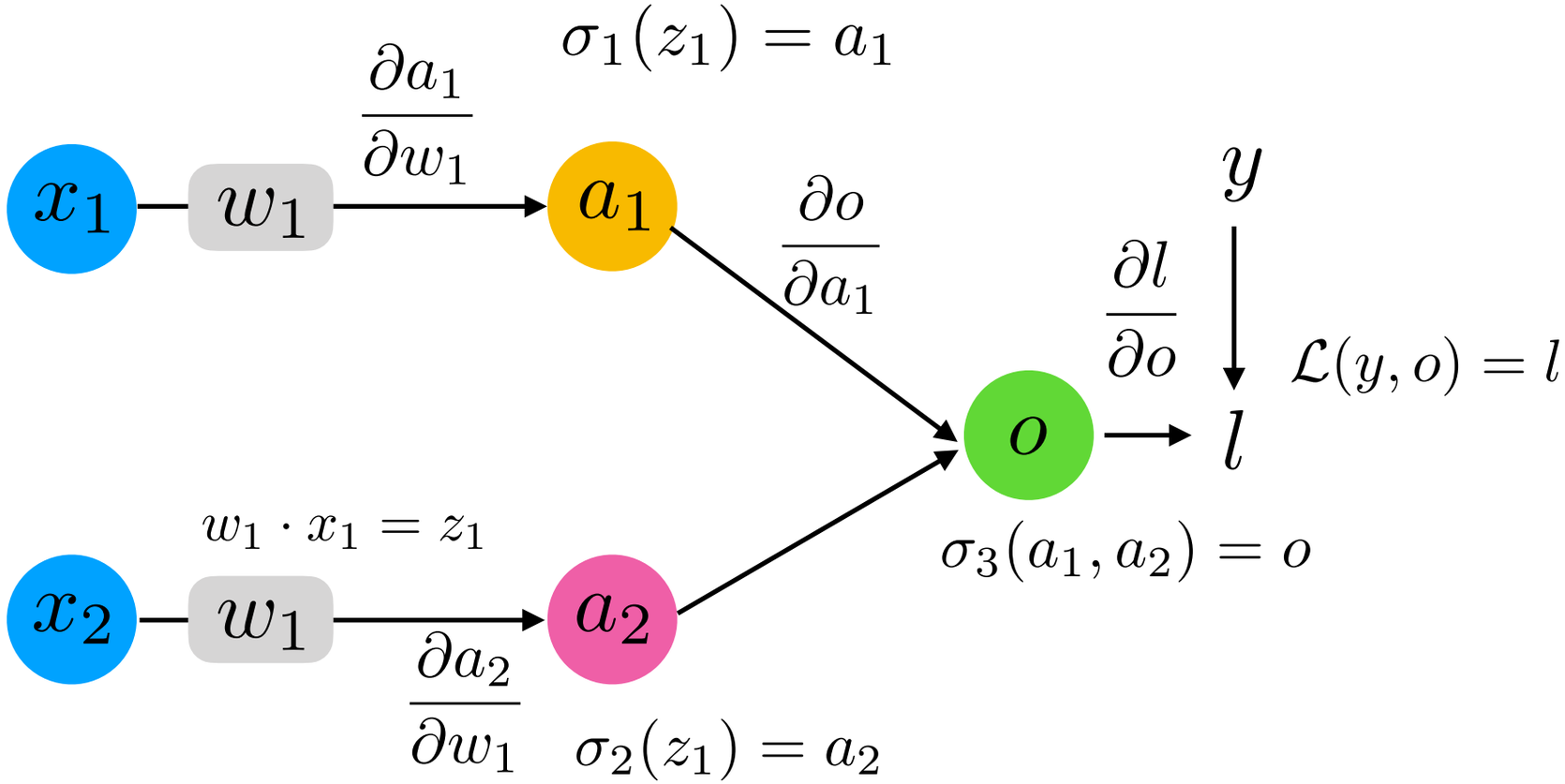
How does Backpropagation Work in CNNs?

1. What CNNs Can Do
2. Image Classification
3. Convolutional Neural Network Basics
4. Convolutional Filters and Weight-Sharing
5. Cross-Correlation vs Convolution
6. **CNNs & Backpropagation**
7. CNN Architectures
8. What a CNN Can See
9. CNNs in PyTorch

Backpropagation in CNNs

Same overall concept as before: Multivariable chain rule, but now with an additional weight sharing constraint

Remember Lecture 6? Graph with Weight Sharing



Upper path

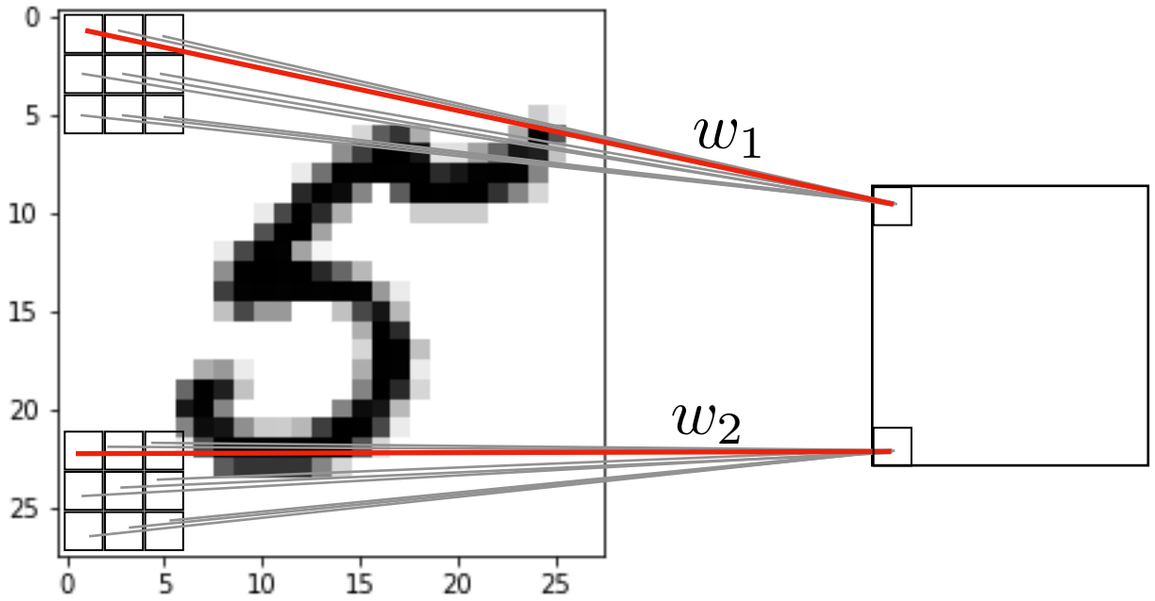
$$\frac{\partial l}{\partial w_1} = \frac{\partial l}{\partial o} \cdot \frac{\partial o}{\partial a_1} \cdot \frac{\partial a_1}{\partial w_1} + \frac{\partial l}{\partial o} \cdot \frac{\partial o}{\partial a_2} \cdot \frac{\partial a_2}{\partial w_1}$$ (multivariable chain rule)

Lower path

Backpropagation in CNNs

Same overall concept as before: Multivariable chain rule, but now with an additional weight sharing constraint

Due to weight sharing: $w_1 = w_2$



Optional averaging

weight update:

$$w_1 := w_2 := w_1 - \eta \cdot \frac{1}{2}\left(\frac{\partial \mathcal{L}}{\partial w_1} + \frac{\partial \mathcal{L}}{\partial w_2}\right)$$

What Are Some of the Common Cnn Architectures?

1. What CNNs Can Do
2. Image Classification
3. Convolutional Neural Network Basics
4. Convolutional Filters and Weight-Sharing
5. Cross-Correlation vs Convolution
6. CNNs & Backpropagation
7. **CNN Architectures**
8. What a CNN Can See
9. CNNs in PyTorch

Main Breakthrough for CNNs: AlexNet & ImageNet

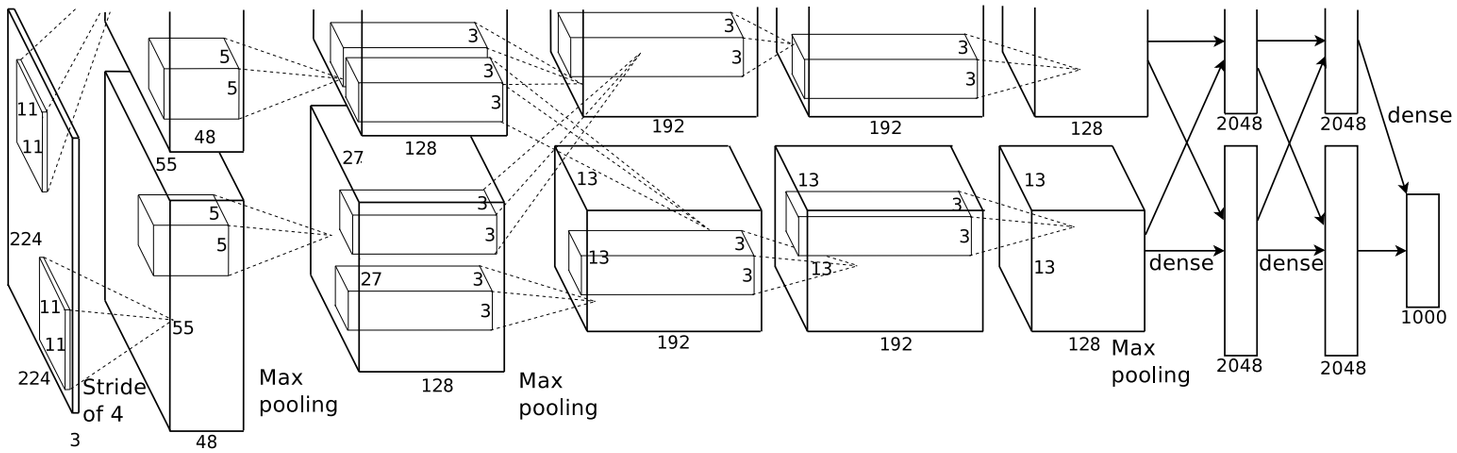


Figure 2: An illustration of the architecture of our CNN, explicitly showing the delineation of responsibilities between the two GPUs. One GPU runs the layer-parts at the top of the figure while the other runs the layer-parts at the bottom. The GPUs communicate only at certain layers. The network's input is 150,528-dimensional, and the number of neurons in the network's remaining layers is given by 253,440–186,624–64,896–64,896–43,264–4096–4096–1000.

AlexNet achieved 15.4% error on top-5 in 2012
2nd best was not even close: 26.2%
(nowadays ~3% error on ImageNet)

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (pp. 1097-1105).

Main Breakthrough for CNNs: AlexNet & ImageNet

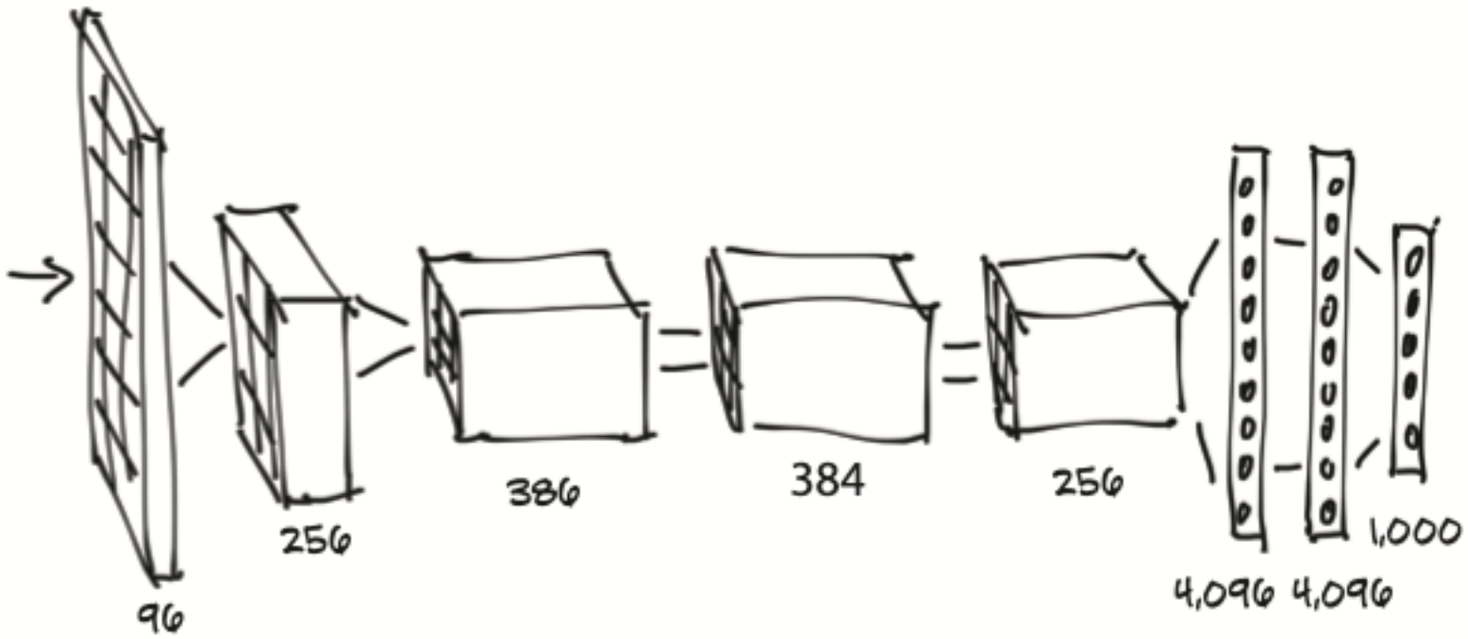


Figure 2.3 The AlexNet architecture

Stevens, Eli, Luca Antiga, and Thomas Viehmann. *Deep learning with PyTorch*. Manning Publications, 2020

Main Breakthrough for CNNs: AlexNet & ImageNet

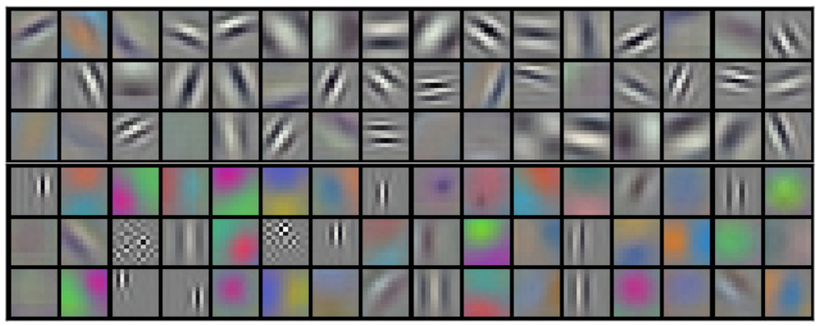


Figure 3: 96 convolutional kernels of size $11 \times 11 \times 3$ learned by the first convolutional layer on the $224 \times 224 \times 3$ input images. The top 48 kernels were learned on GPU 1 while the bottom 48 kernels were learned on GPU 2. See Section 6.1 for details.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition* (pp. 248–255).

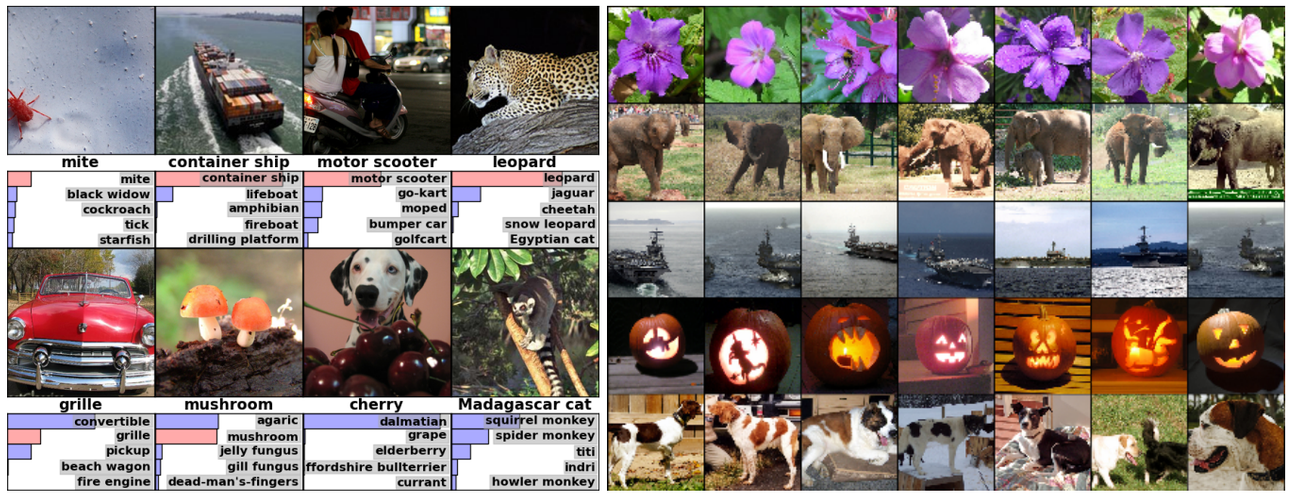
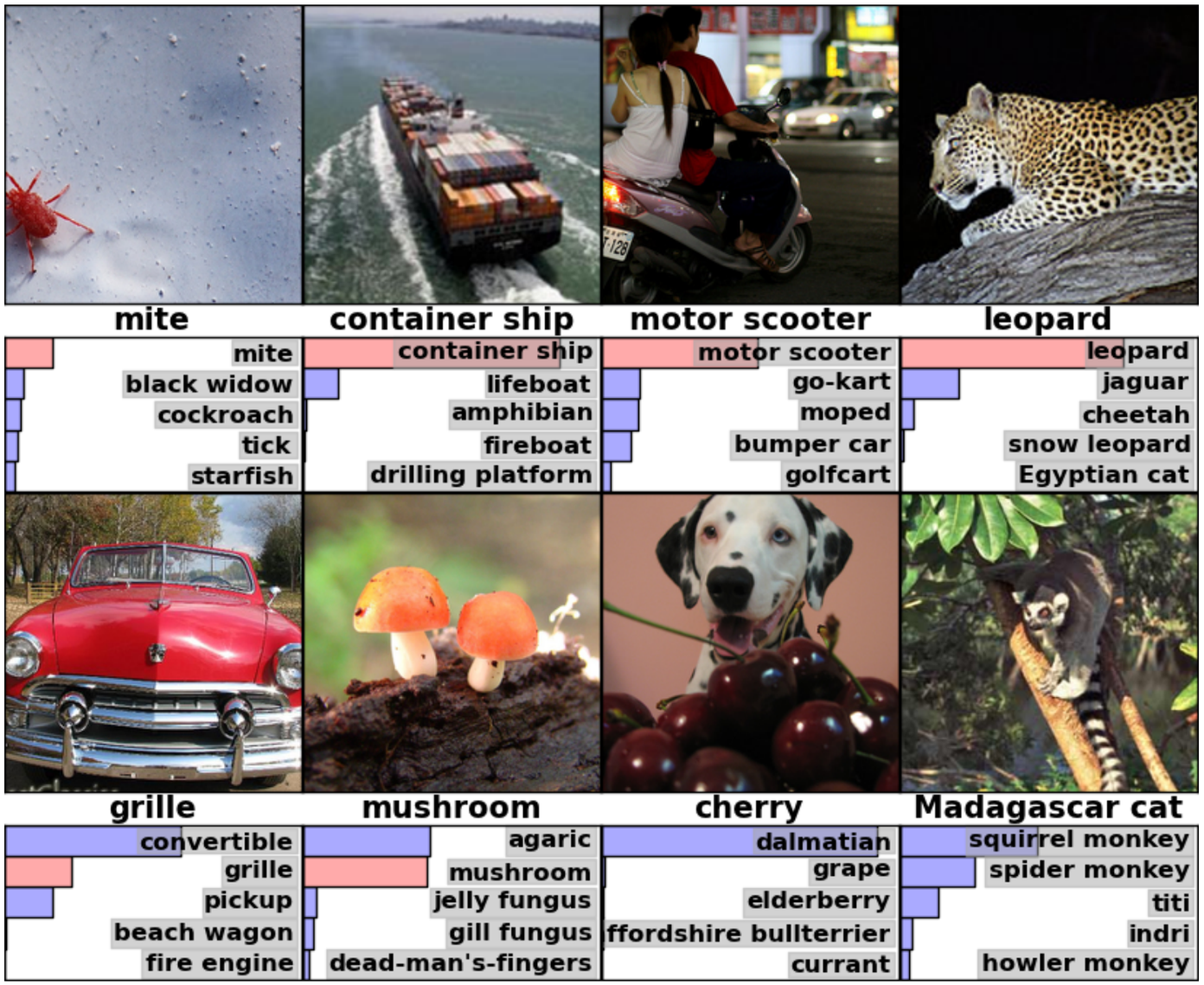


Figure 4: **(Left)** Eight ILSVRC-2010 test images and the five labels considered most probable by our model. The correct label is written under each image, and the probability assigned to the correct label is also shown with a red bar (if it happens to be in the top 5). **(Right)** Five ILSVRC-2010 test images in the first column. The remaining columns show the six training images that produce feature vectors in the last hidden layer with the smallest Euclidean distance from the feature vector for the test image.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (pp. 1097-1105).

Main Breakthrough for CNNs: AlexNet & ImageNet

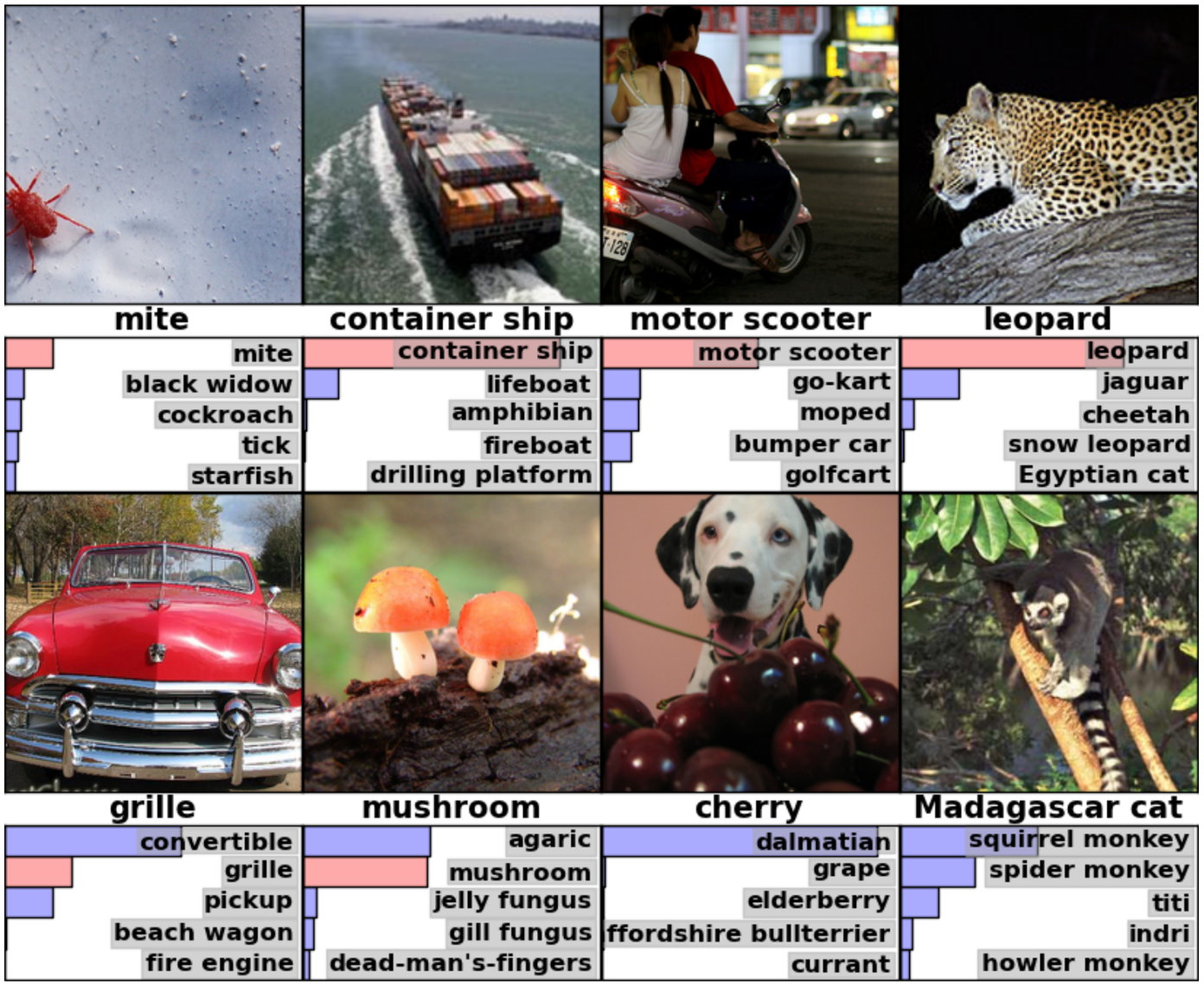


The ImageNet set that was used has ~1.2 million images and 1000 classes

Accuracy is measured as top-5 performance:
Correct prediction if the true label matches one of the top 5 predictions of the model

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (pp. 1097-1105).

Main Breakthrough for CNNs: AlexNet & ImageNet



Note that the actual network inputs were still 224x224 images
(random crops from downsampled 256x256 images)

224x224 is still a good/ reasonable size today
(224*224*3 = 150,528 features)

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (pp. 1097-1105).

Common CNN Architectures

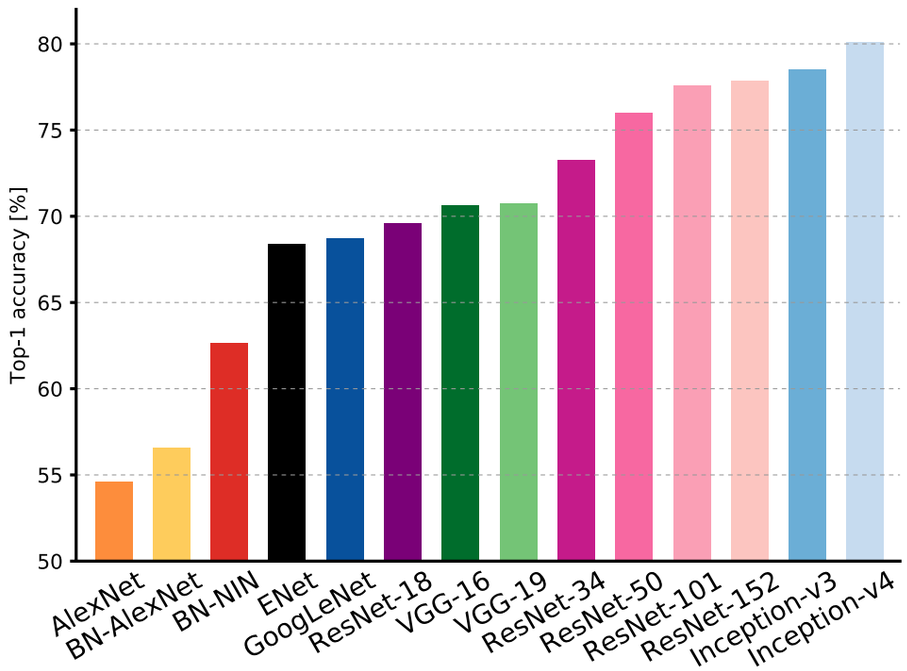


Figure 1: **Top1 *vs.* network.** Single-crop top-1 validation accuracies for top scoring single-model architectures. We introduce with this chart our choice of colour scheme, which will be used throughout this publication to distinguish effectively different architectures and their correspondent authors. Notice that networks of the same group share the same hue, for example ResNet are all variations of pink.

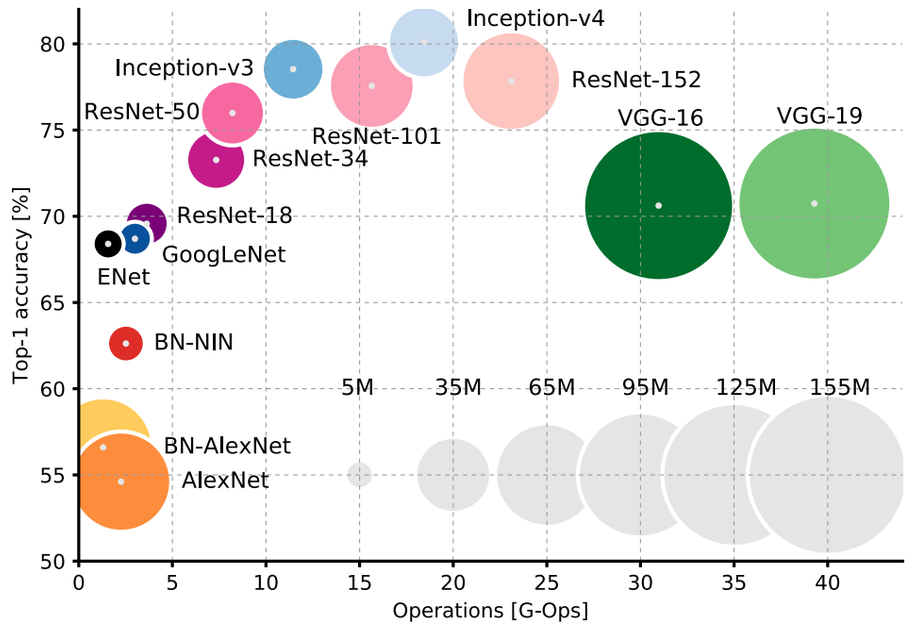
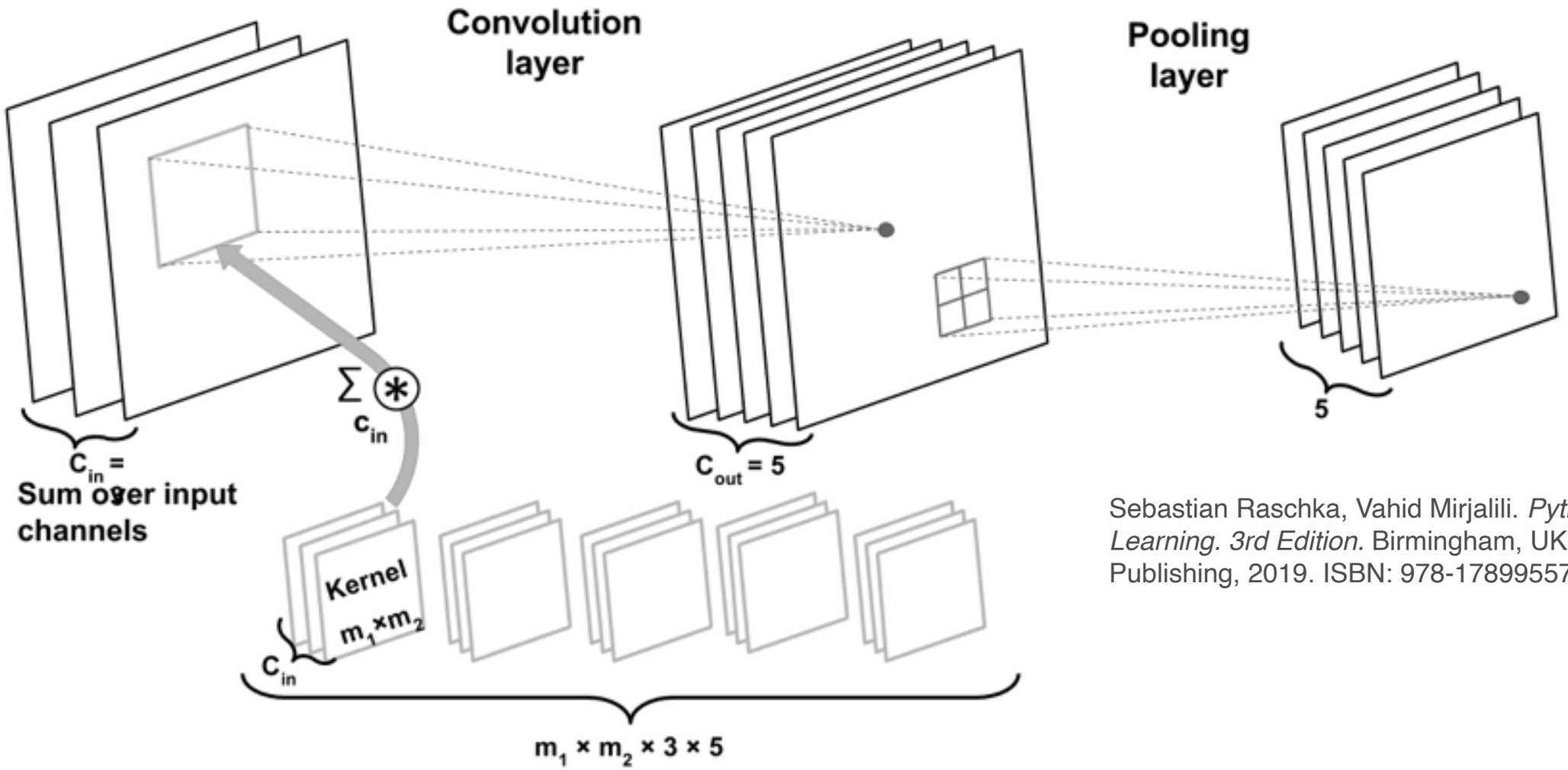


Figure 2: **Top1 *vs.* operations, size $\propto$ parameters.** Top-1 one-crop accuracy versus amount of operations required for a single forward pass. The size of the blobs is proportional to the number of network parameters; a legend is reported in the bottom right corner, spanning from 5×10^6 to 155×10^6 params. Both these figures share the same y-axis, and the grey dots highlight the centre of the blobs.

Canziani, A., Paszke, A., & Culurciello, E. (2016). An analysis of deep neural network models for practical applications. *arXiv preprint arXiv:1605.07678*.

Convolutions with Color Channels



Sebastian Raschka, Vahid Mirjalili. *Python Machine Learning. 3rd Edition.* Birmingham, UK: Packt Publishing, 2019. ISBN: 978-1789955750

Image dimension: $\mathbf{X} \in \mathbb{R}^{n_1 \times n_2 \times c_{in}}$ in NWHC format,
CUDA & PyTorch use NCWH

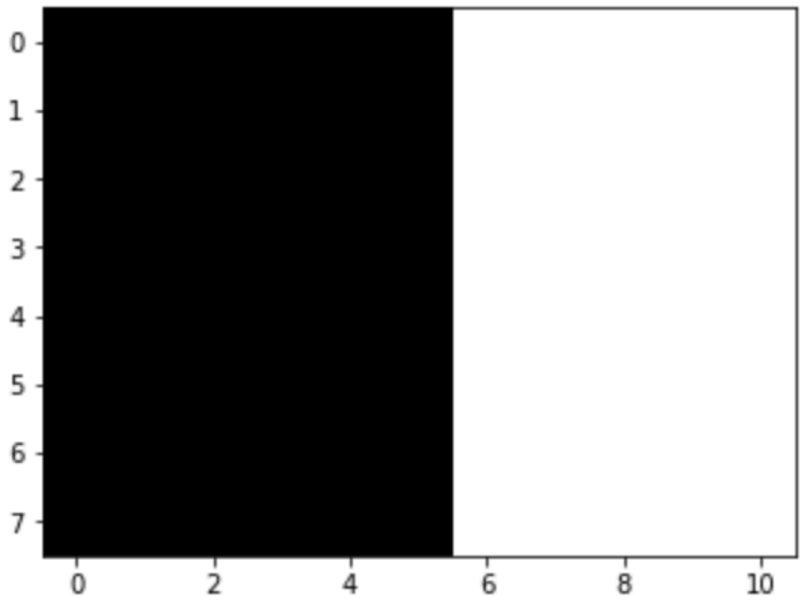
Interpreting (the Hidden) ConvNet Layers

1. What CNNs Can Do
2. Image Classification
3. Convolutional Neural Network Basics
4. Convolutional Filters and Weight-Sharing
5. Cross-Correlation vs Convolution
6. CNNs & Backpropagation
7. CNN Architectures
8. **What a CNN Can See**
9. CNNs in PyTorch

What a CNN Can See

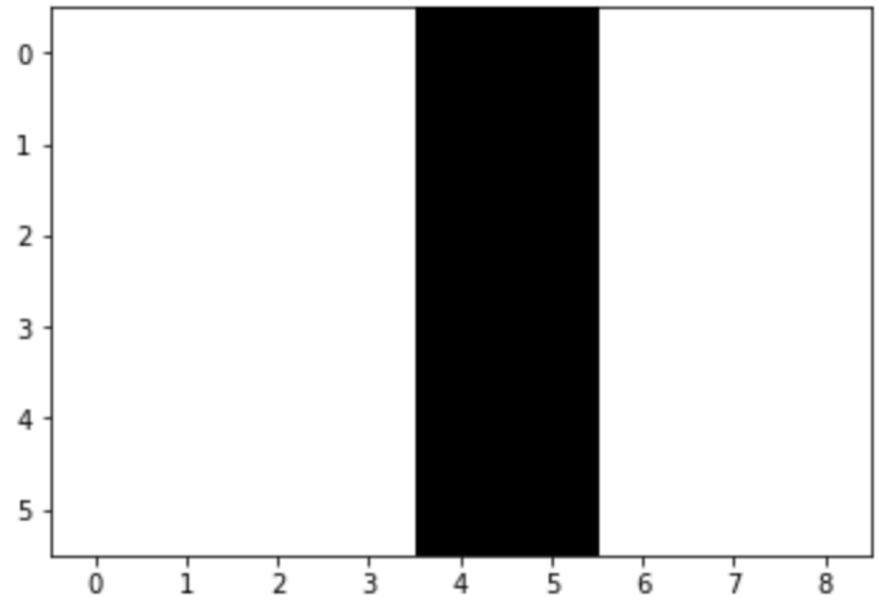
Simple example: vertical edge detector

```
conv.weight[0, 0, :, :] = torch.tensor([[1, 0, -1],
                                        [1, 0, -1],
                                        [1, 0, -1]]).float()

t = torch.tensor([
[0., 0., 0., 0., 0., 0., 1., 1., 1., 1., 1.],
[0., 0., 0., 0., 0., 0., 1., 1., 1., 1., 1.],
[0., 0., 0., 0., 0., 0., 1., 1., 1., 1., 1.],
[0., 0., 0., 0., 0., 0., 1., 1., 1., 1., 1.],
[0., 0., 0., 0., 0., 0., 1., 1., 1., 1., 1.],
[0., 0., 0., 0., 0., 0., 1., 1., 1., 1., 1.],
[0., 0., 0., 0., 0., 0., 1., 1., 1., 1., 1.],
[0., 0., 0., 0., 0., 0., 1., 1., 1., 1., 1.],
])

plt.imshow(t, cmap='gray');
```



(From classical computer vision research)

```
tt = torch.zeros([1, 1] + list(t.size()))
tt[0, 0, :, :] = t
after = conv(tt)
plt.imshow(after[0, 0, :, :].detach().numpy(), cmap='gray');
```



What a CNN Can See

Simple example: vertical edge detector

```
conv = torch.nn.Conv2d(in_channels=1,
                       out_channels=1,
                       kernel_size=(3, 3))
```

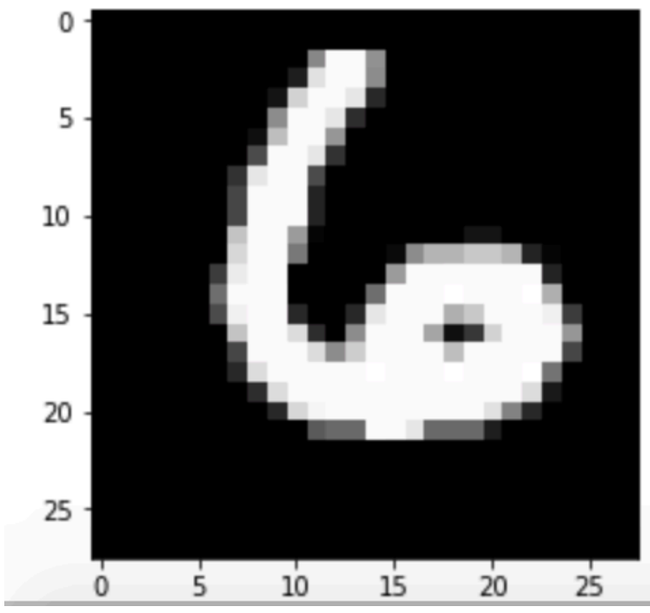
```
conv.weight.size()
```

```
torch.Size([1, 1, 3, 3])
```

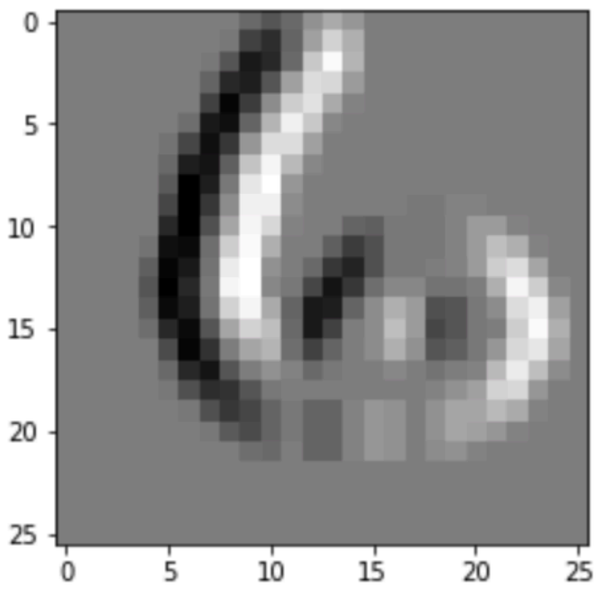
```
conv.weight[0, 0, :, :] = torch.tensor([[1, 0, -1],
                                        [1, 0, -1],
                                        [1, 0, -1]]).float()
conv.bias[0] = torch.tensor([0.]).float()
```

```
images_after = conv(images)
```

```
plt.imshow(images[5, 0], cmap='gray');
```




```
plt.imshow(images_after[5, 0].detach().numpy() , cmap='gray');
```



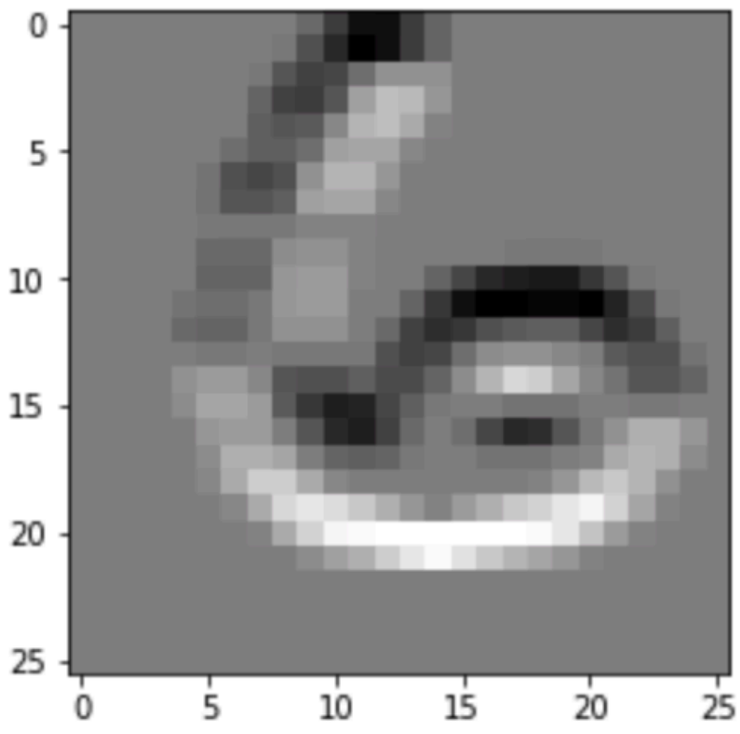
What a CNN Can See

Simple example: horizontal edge detector

```
conv.weight[0, 0, :, :] = torch.tensor([[1, 1, 1],
                                        [0, 0, 0],
                                        [-1, -1, -1]]).float()
conv.bias[0] = torch.tensor([0.]).float()
```

```
images_after2 = conv(images)
```

```
plt.imshow(images_after2[5, 0].detach().numpy() , cmap='gray');
```



A CNN can learn whatever it finds best based on optimizing the objective (e.g., minimizing a particular loss to achieve good classification accuracy)

Main Breakthrough for CNNs: AlexNet & ImageNet

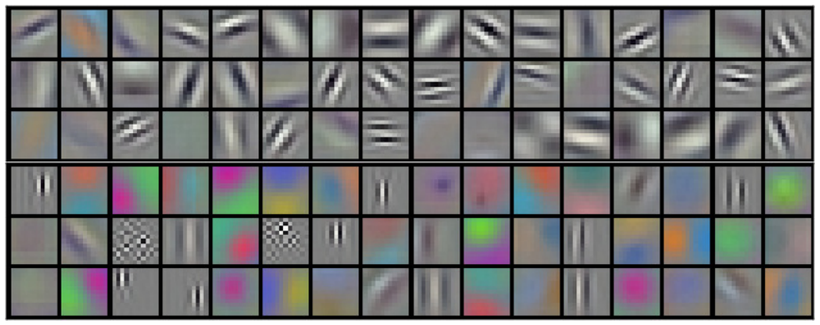


Figure 3: 96 convolutional kernels of size $11 \times 11 \times 3$ learned by the first convolutional layer on the $224 \times 224 \times 3$ input images. The top 48 kernels were learned on GPU 1 while the bottom 48 kernels were learned on GPU 2. See Section 6.1 for details.

Deng, J., Dong, W., Socher, R., Li, L.-J., Li, K., & Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition* (pp. 248–255).

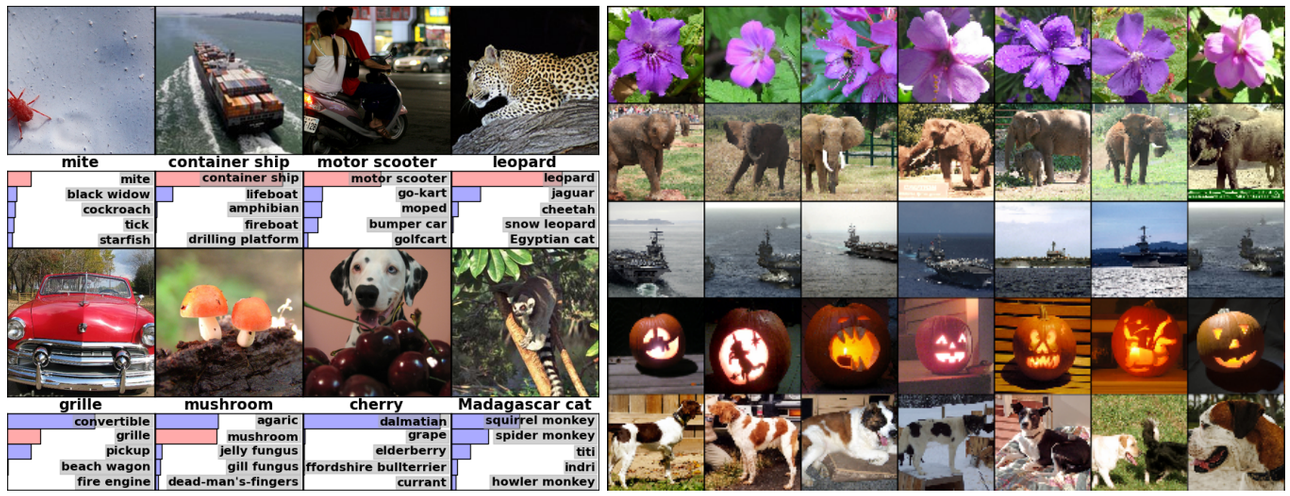


Figure 4: **(Left)** Eight ILSVRC-2010 test images and the five labels considered most probable by our model. The correct label is written under each image, and the probability assigned to the correct label is also shown with a red bar (if it happens to be in the top 5). **(Right)** Five ILSVRC-2010 test images in the first column. The remaining columns show the six training images that produce feature vectors in the last hidden layer with the smallest Euclidean distance from the feature vector for the test image.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems* (pp. 1097-1105).

What a CNN Can See

Which patterns from the training set activate the feature map?

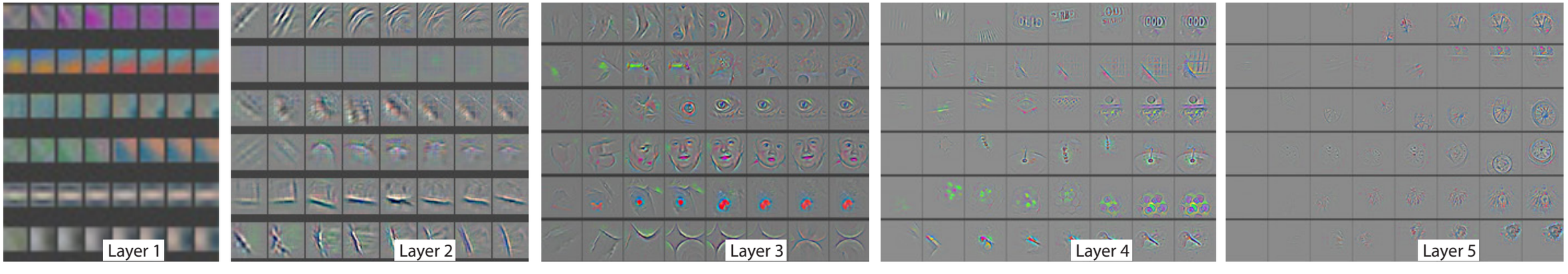


Fig. 4. Evolution of a randomly chosen subset of model features through training. Each layer's features are displayed in a different block. Within each block, we show a randomly chosen subset of features at epochs [1,2,5,10,20,30,40,64]. The visualization shows the strongest activation (across all training examples) for a given feature map, projected down to pixel space using our deconvnet approach. Color contrast is artificially enhanced and the figure is best viewed in electronic form.

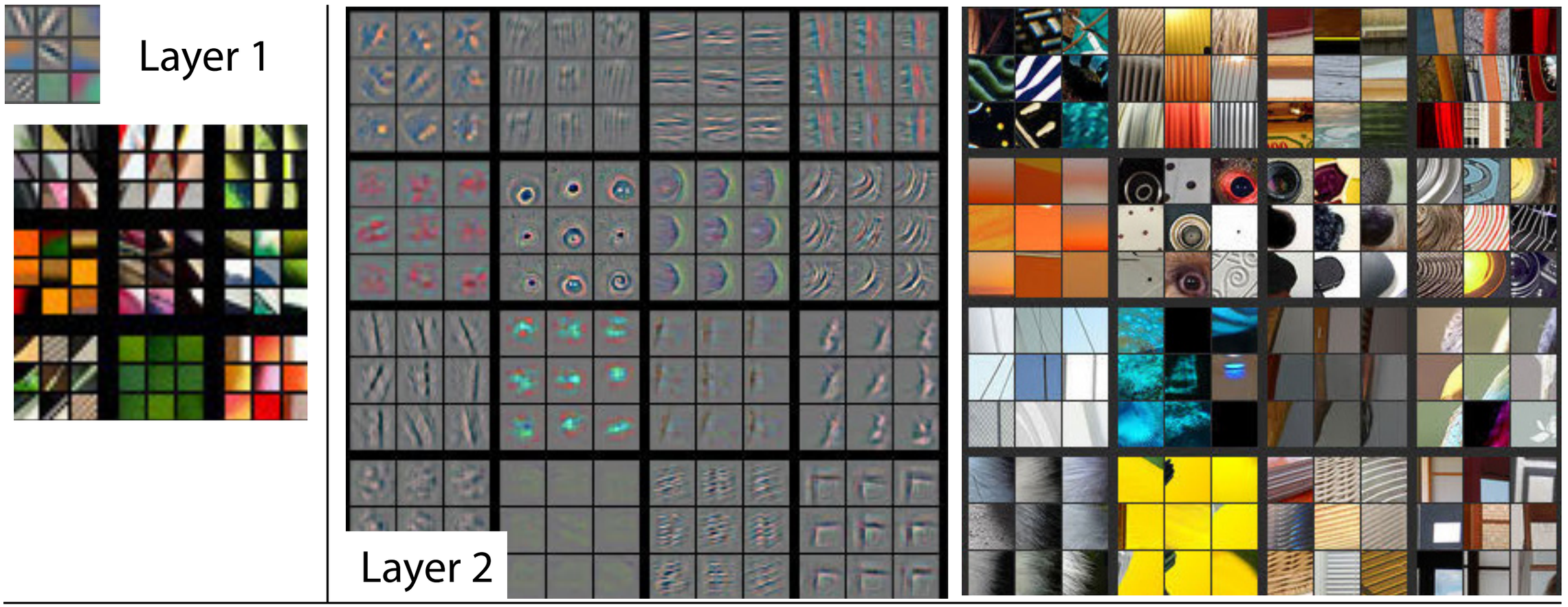
Zeiler, M. D., & Fergus, R. (2014, September). Visualizing and understanding convolutional networks. In *European conference on computer vision* (pp. 818-833). Springer, Cham.

Method: backpropagate strong activation signals in hidden layers to the input images, then apply "unpooling" to map the values to the original pixel space for visualization

What a CNN Can See

Which patterns from the training set activate the feature map?

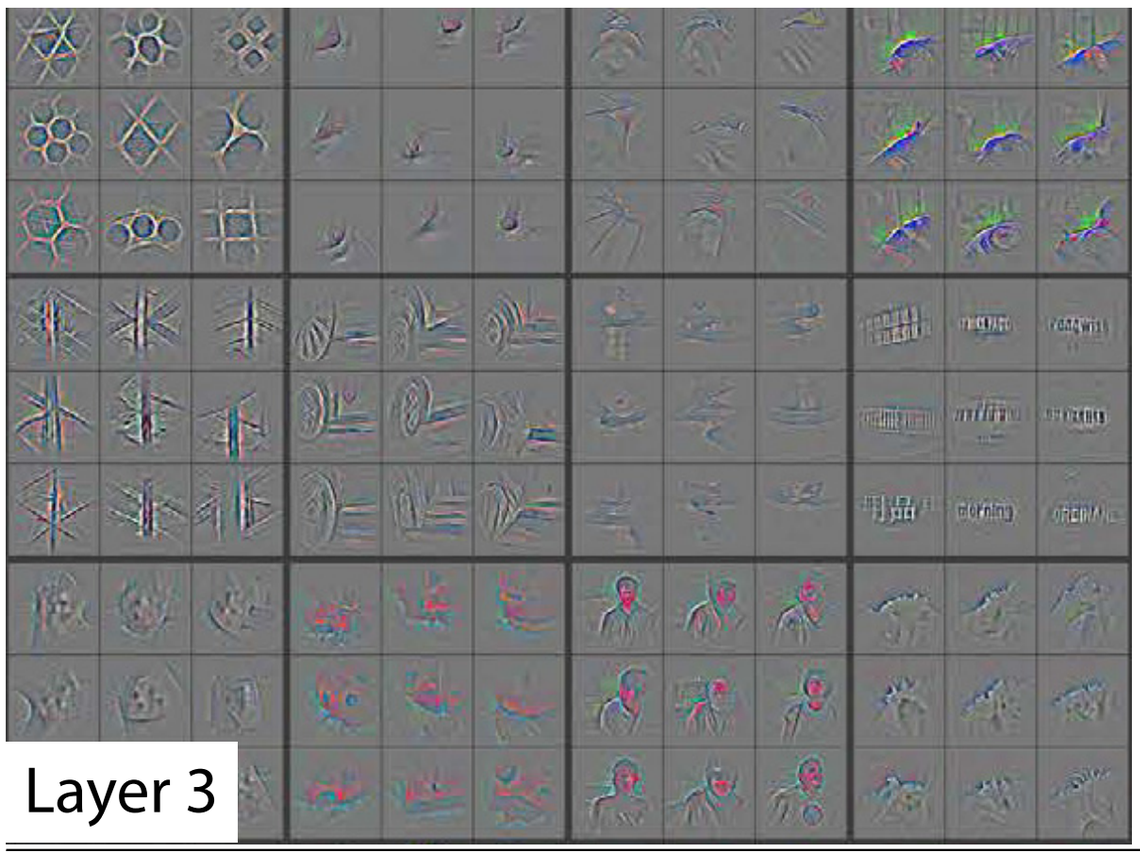
Zeiler, M. D., & Fergus, R. (2014, September). Visualizing and understanding convolutional networks. In *European conference on computer vision* (pp. 818-833). Springer, Cham.



What a CNN Can See

Which patterns from the training set activate the feature map?

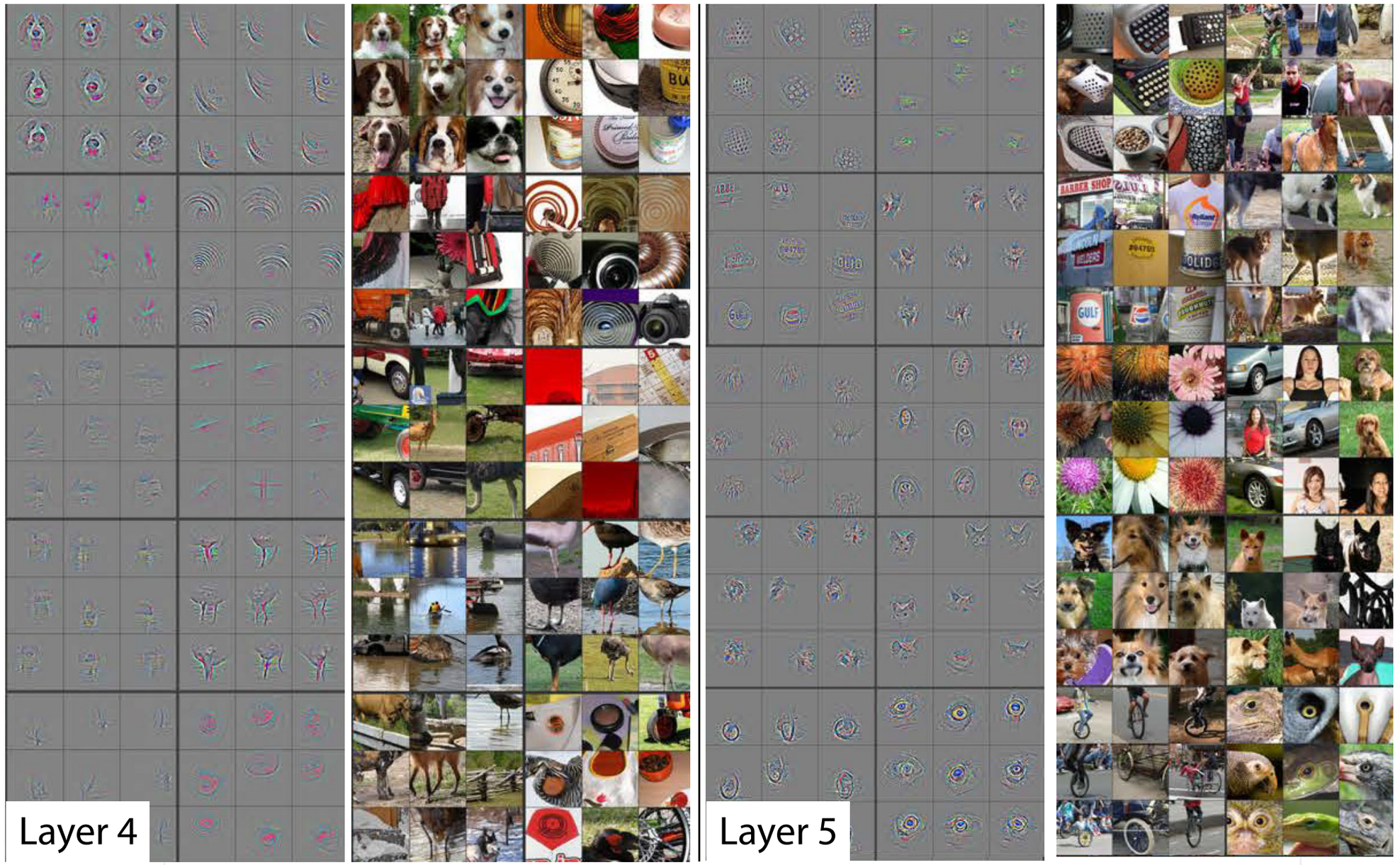
Zeiler, M. D., & Fergus, R. (2014, September). Visualizing and understanding convolutional networks. In *European conference on computer vision* (pp. 818-833). Springer, Cham.

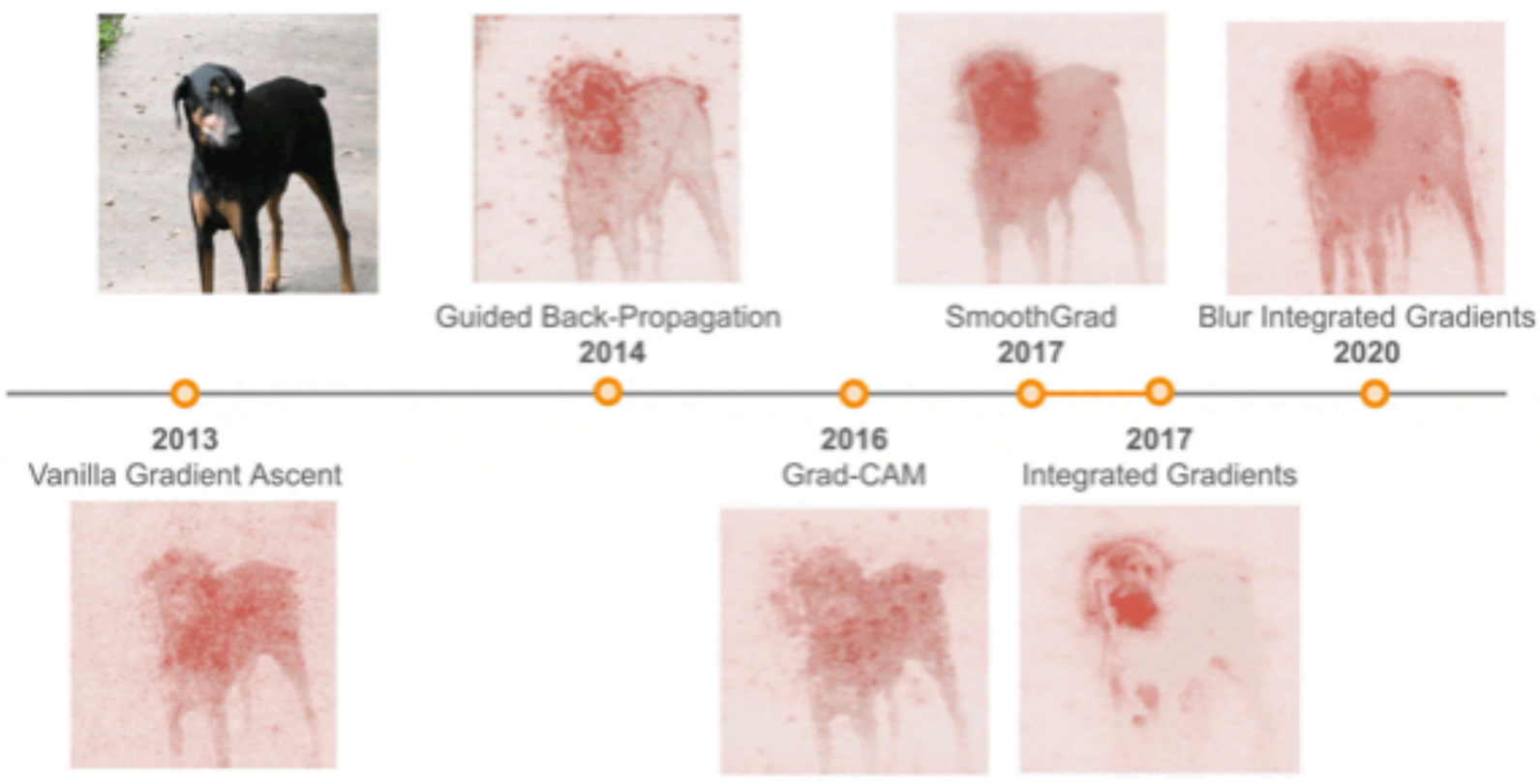


What a CNN Can See

Which patterns from the training set activate the feature map?

Zeiler, M. D., & Fergus, R. (2014, September). Visualizing and understanding convolutional networks. In *European conference on computer vision* (pp. 818-833). Springer, Cham.





https://thegradient.pub/a-visual-history-of-interpretation-for-image-recognition/

Let's Implement CNNs in PyTorch

1. What CNNs Can Do
2. Image Classification
3. Convolutional Neural Network Basics
4. Convolutional Filters and Weight-Sharing
5. Cross-Correlation vs Convolution
6. CNNs & Backpropagation
7. CNN Architectures
8. What a CNN Can See
9. **CNNs in PyTorch**