



# Introduction to Deep learning: a 2-weeks lecture Part 2

Presented by: Dra. Jeaneth Machicao  
machicao@usp.br

October/2020



# Course overview: STAT 453: Deep Learning, Spring 2020 by Prof. Sebastian Raschka

## Part1: Introduction

- Introduction to deep learning
- The brief history of deep learning
- Single-layer neural networks: The perceptron
- Motivation: cases of use
- Hands-on

## Part2: Mathematical and computational foundations

- Linear algebra and calculus for deep learning
- Parameter optimization with gradient descent
- Automatic differentiation & PyTorch

## Part3: Introduction to neural networks

- Multinomial logistic regression
- Multilayer perceptrons
- Regularization
- Input normalization and weight initialization
- Learning rate and advanced optimization algorithms

## Part4: DL for computer vision and language modeling

- Introduction to convolutional neural networks 1-2
  - CNNs Architectures Illustrated
- Introduction to recurrent neural networks 1-2

## Part5: Deep generative models

- Autoencoders,
- Autoregressive models
- Variational autoencoders
- Normalizing Flow models
- Generative adversarial networks
- Evaluating generative models

<http://stat.wisc.edu/~sraschka/teaching/stat453-ss2020/>  
<https://github.com/rasbt/stat453-deep-learning-ss20>

• [Course Playlist on youtube:](#)

[Prof. Dalcimar Casanova](#)

[https://www.youtube.com/watch?v=0VD\\_2t6EdS4&list=PL9At2PVRU0ZqVArhU9QMyl3jSe113\\_m2-](https://www.youtube.com/watch?v=0VD_2t6EdS4&list=PL9At2PVRU0ZqVArhU9QMyl3jSe113_m2-)

[Prof. Sebastian Raschka](#)

[https://www.youtube.com/watch?v=e\\_l0q3mmfw4&list=PLTKMiZHVd\\_2JkR6QtQEnml7swCnFBtq4P](https://www.youtube.com/watch?v=e_l0q3mmfw4&list=PLTKMiZHVd_2JkR6QtQEnml7swCnFBtq4P)



# Overview of our 2-weeks lecture!

## 1st week

### 1: Introduction

- Introduction to deep learning
- The brief history of deep learning
- Single-layer neural networks: The perceptron
- Motivation: cases of use
- Hands-on (report)

### 2: Mathematical and computational foundations

- Linear algebra and calculus for deep learning
- Parameter optimization with gradient descent
- Automatic differentiation & PyTorch

### 3: Introduction to neural networks

- Multinomial logistic regression
- Multilayer perceptrons
- Regularization
- Input normalization and weight initialization
- Learning rate and advanced optimization algorithms

## 2nd week

### 4: DL for computer vision and language modeling

- Introduction to convolutional neural networks 1-2
  - CNNs Architectures Illustrated
- Introduction to recurrent neural networks 1-2

## 3rd week

- Deliver report of the hands-on

<http://stat.wisc.edu/~sraschka/teaching/stat453-ss2020/>  
<https://github.com/rasbt/stat453-deep-learning-ss20>

- [Course Playlist on youtube:](#)  
[Prof. Dalcimar Casanova](#)  
[Prof. Sebastian Raschka](#)

Lecture 12

# Introduction to Convolutional Neural Networks Part 1

STAT 453: Deep Learning, Spring 2020

Sebastian Raschka

<http://stat.wisc.edu/~sraschka/teaching/stat453-ss2020/>

<https://github.com/rasbt/stat453-deep-learning-ss20/tree/master/L12-cnns>

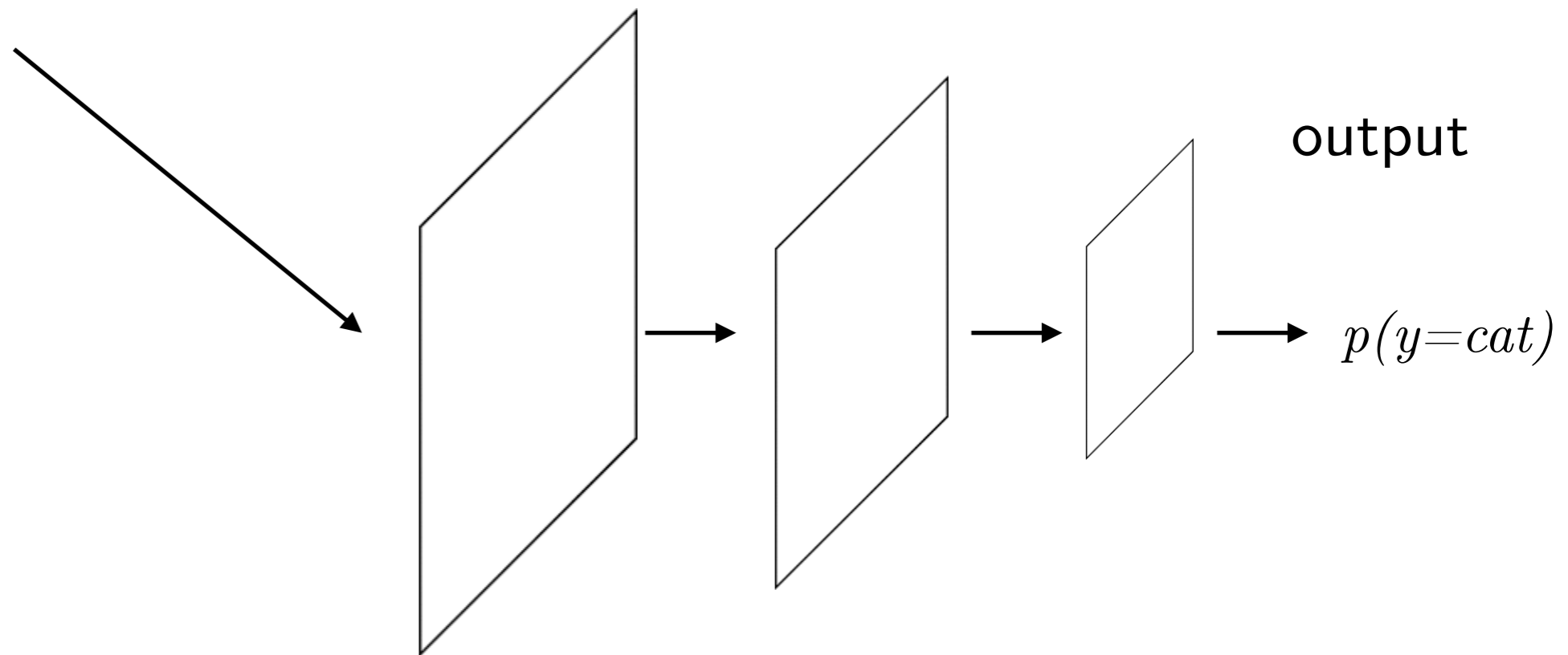
# CNNs for Image Classification



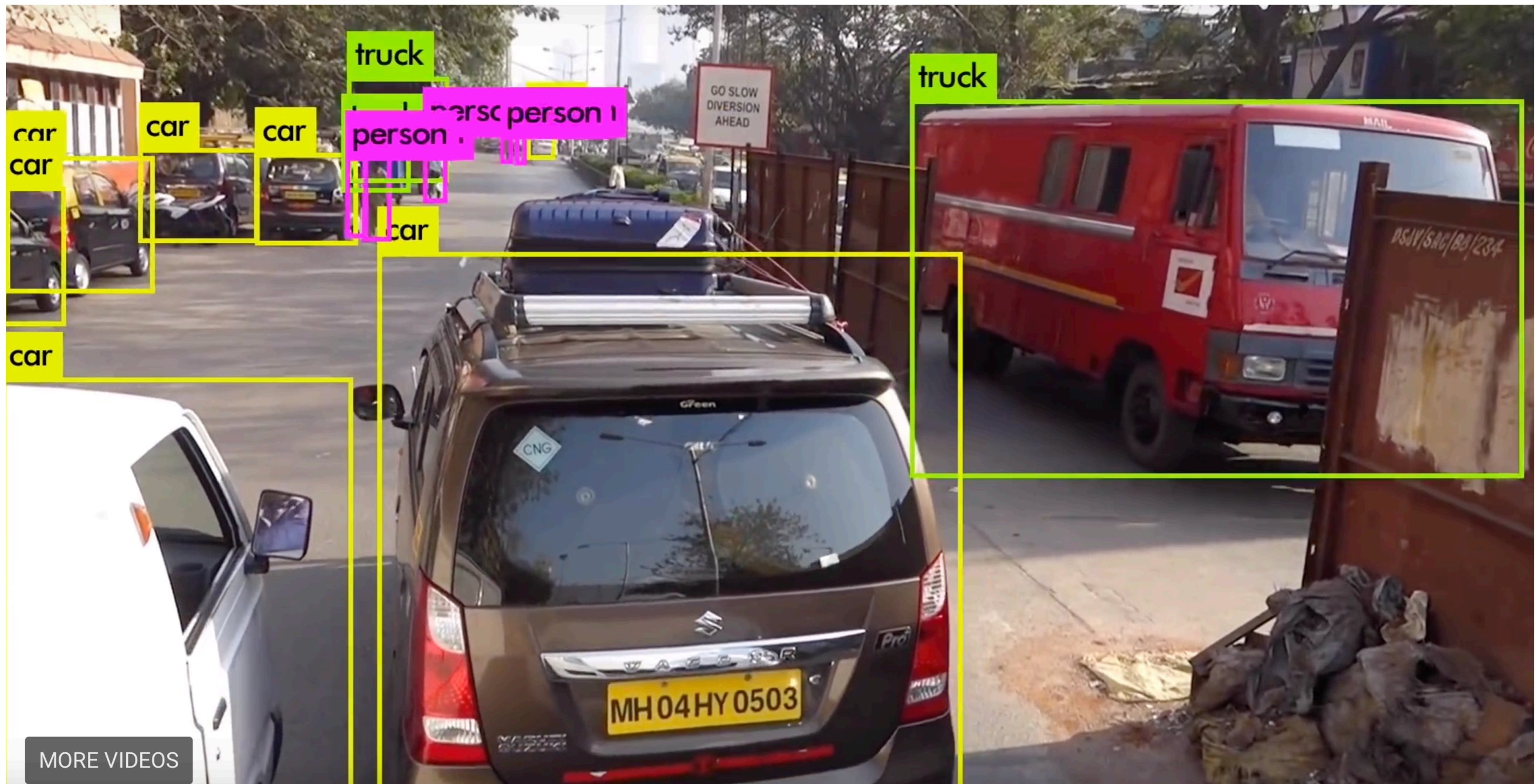
Image Source:  
[twitter.com/%2Fcats&psig=AOvVaw30\\_o-PCM-K21DiMAJQimQ4&ust=1553887775741551](https://twitter.com/%2Fcats&psig=AOvVaw30_o-PCM-K21DiMAJQimQ4&ust=1553887775741551)



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

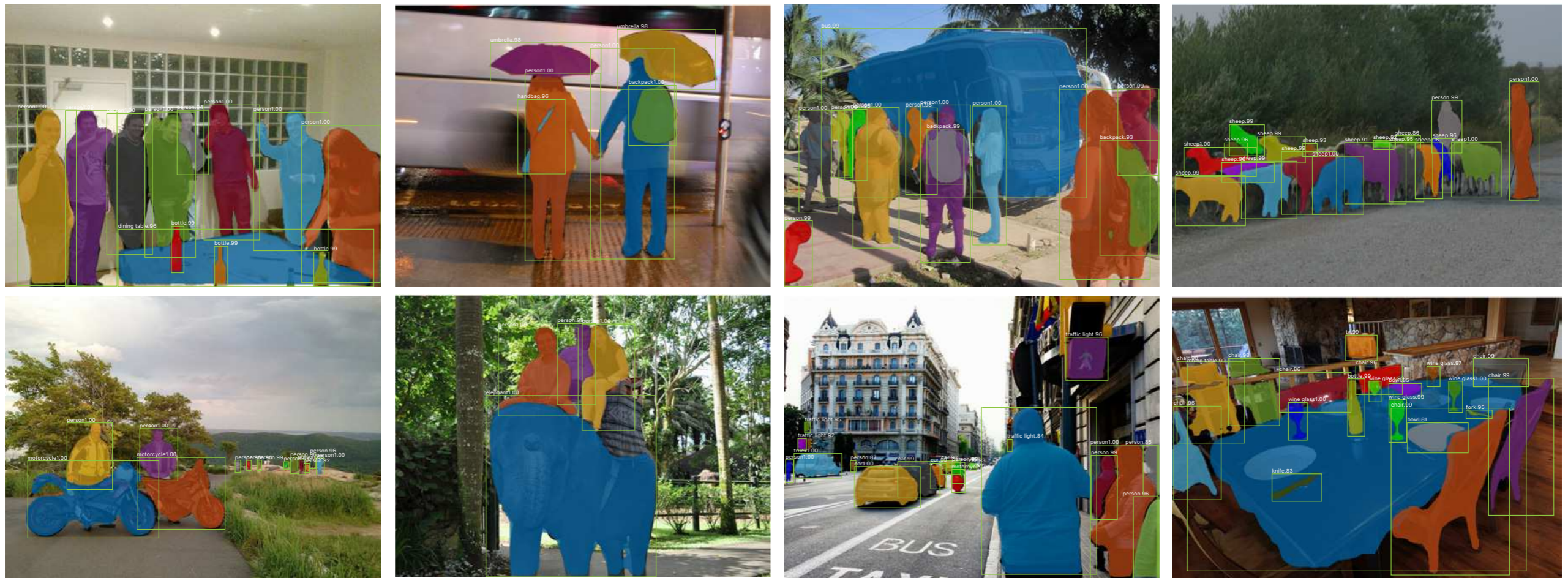
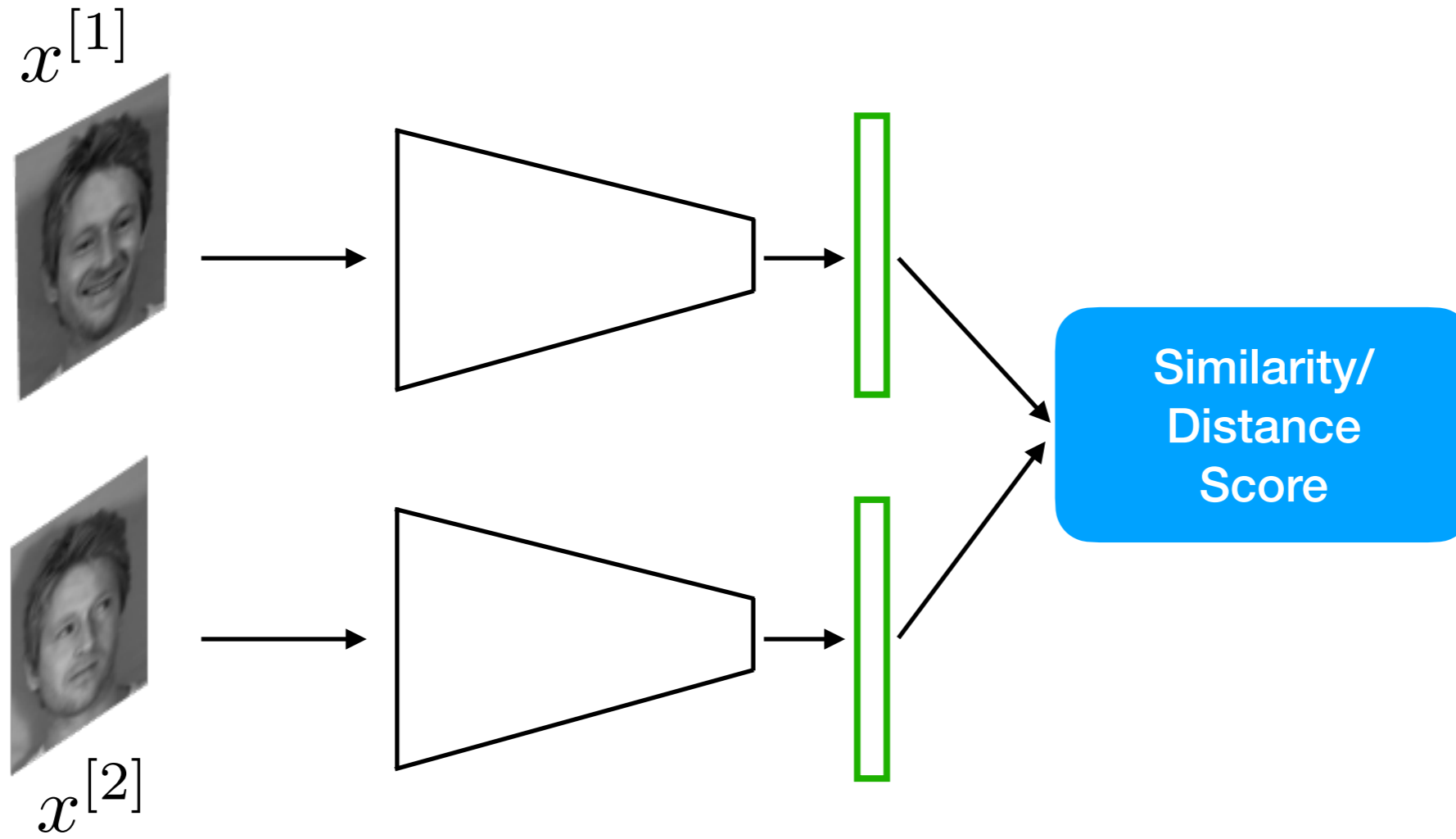


Figure 2. **Mask R-CNN** results on the COCO test set. These results are based on ResNet-101 [15], achieving a *mask AP* of 35.7 and running at 5 fps. Masks are shown in color, and bounding box, category, and confidences are also shown.

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



Siamese neural network



# Lecture Overview

1. Image Classification
2. Convolutional Neural Network Basics
3. CNN Architectures
4. What a CNN Can See
5. CNNs in PyTorch

# Why Image Classification is Hard

Different lighting, contrast, viewpoints, etc.

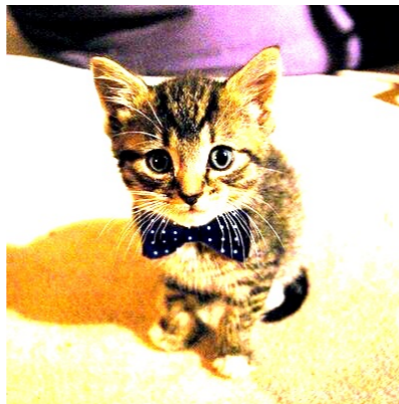
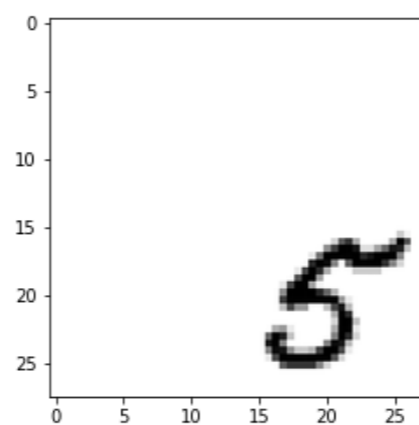
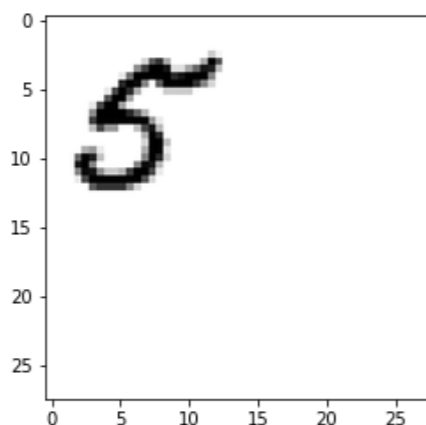


Image Source: [twitter.com/%2Fcats&psig=AOvVaw30\\_o-PCM-K21DiMAJQimQ4&ust=1553887775741551](https://twitter.com/%2Fcats&psig=AOvVaw30_o-PCM-K21DiMAJQimQ4&ust=1553887775741551)

Image Source: [https://www.123rf.com/photo\\_76714328\\_side-view-of-tabby-cat-face-over-white.html](https://www.123rf.com/photo_76714328_side-view-of-tabby-cat-face-over-white.html)

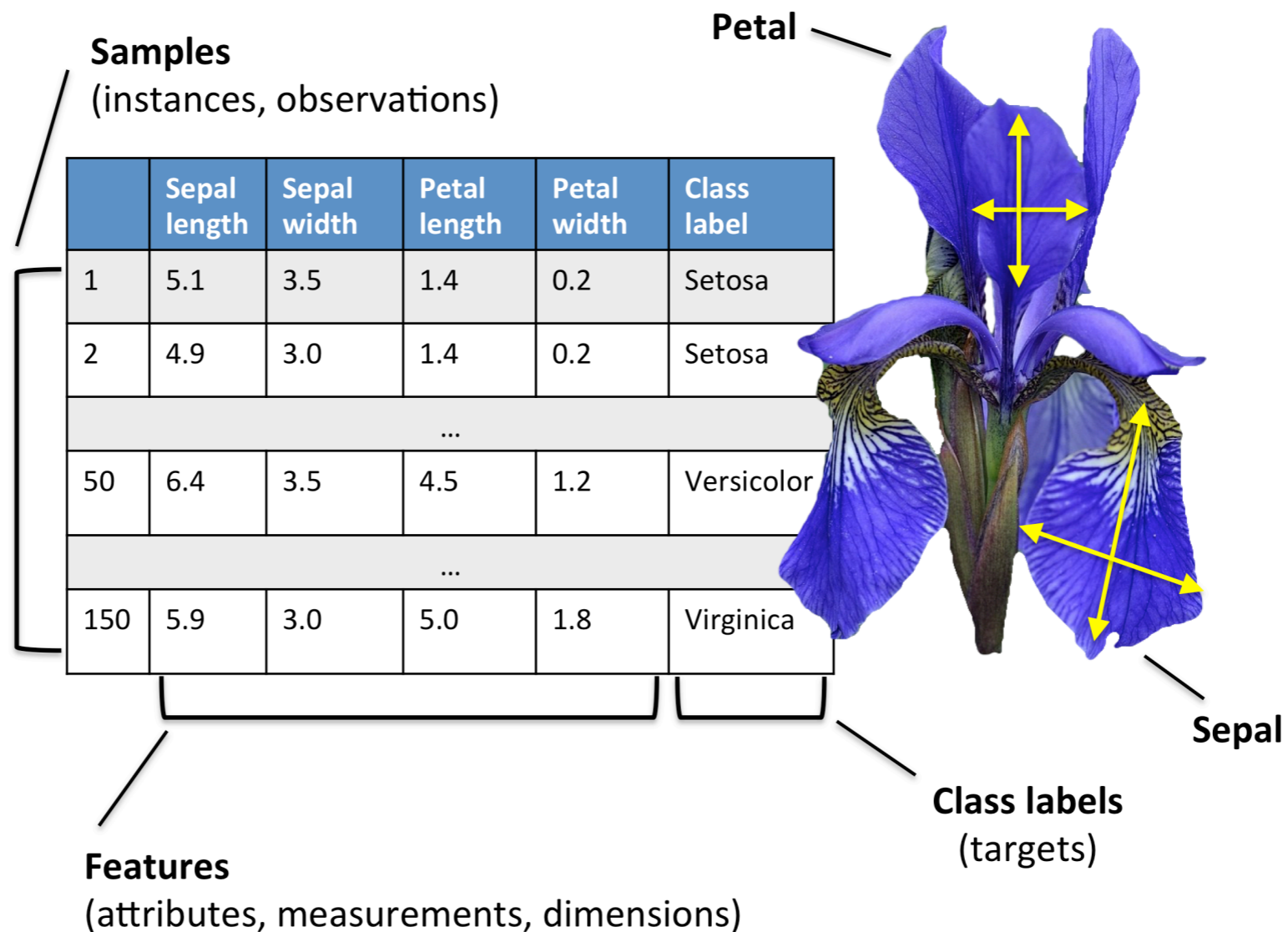
Or even simple translation



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

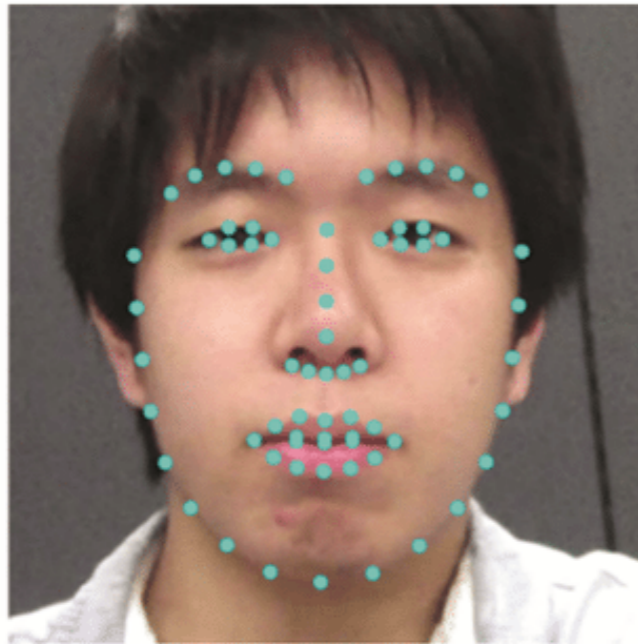
# Traditional Approaches

a) Use hand-engineered features



# 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>

# Lecture Overview

1. Image Classification
- 2. Convolutional Neural Network Basics**
3. CNN Architectures
4. What a CNN Can See
5. CNNs in PyTorch

# 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

# Convolutional Neural Networks

PROC. OF THE IEEE, NOVEMBER 1998

7

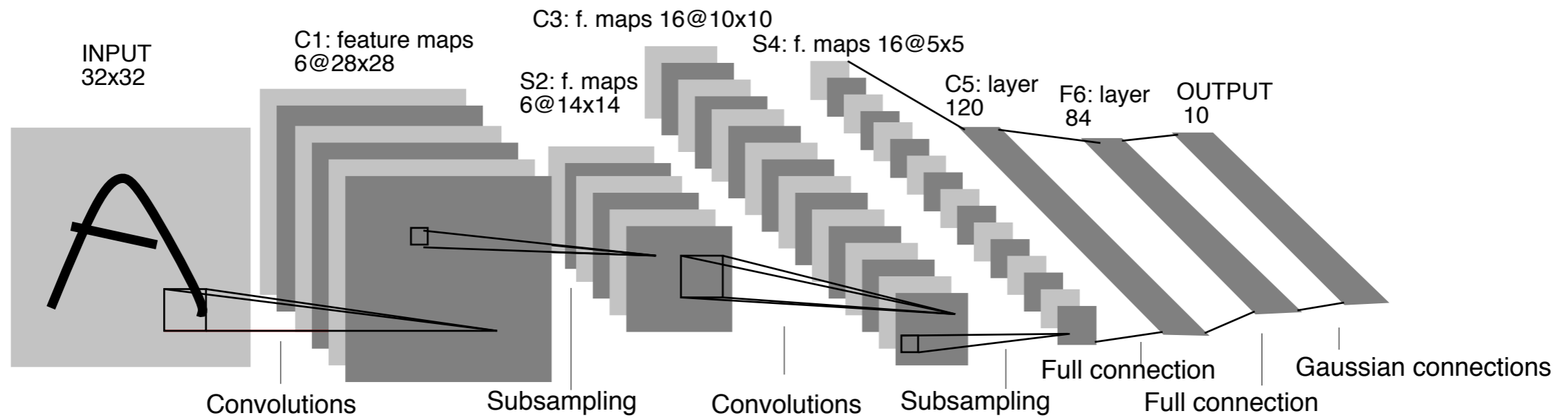


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

## Pooling

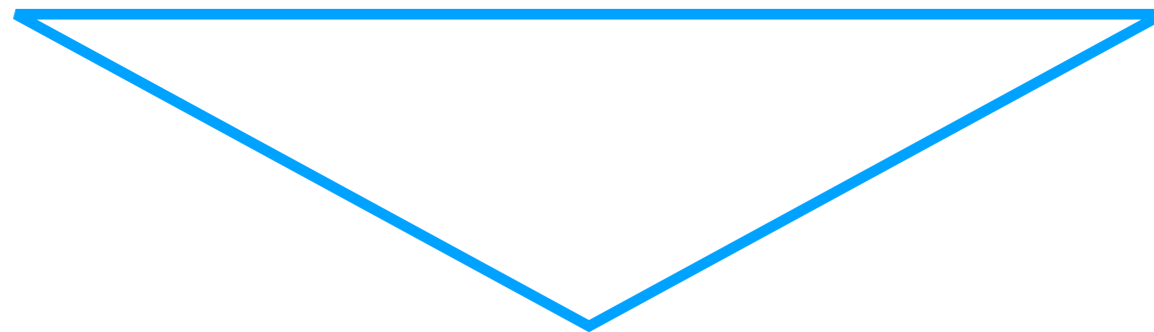
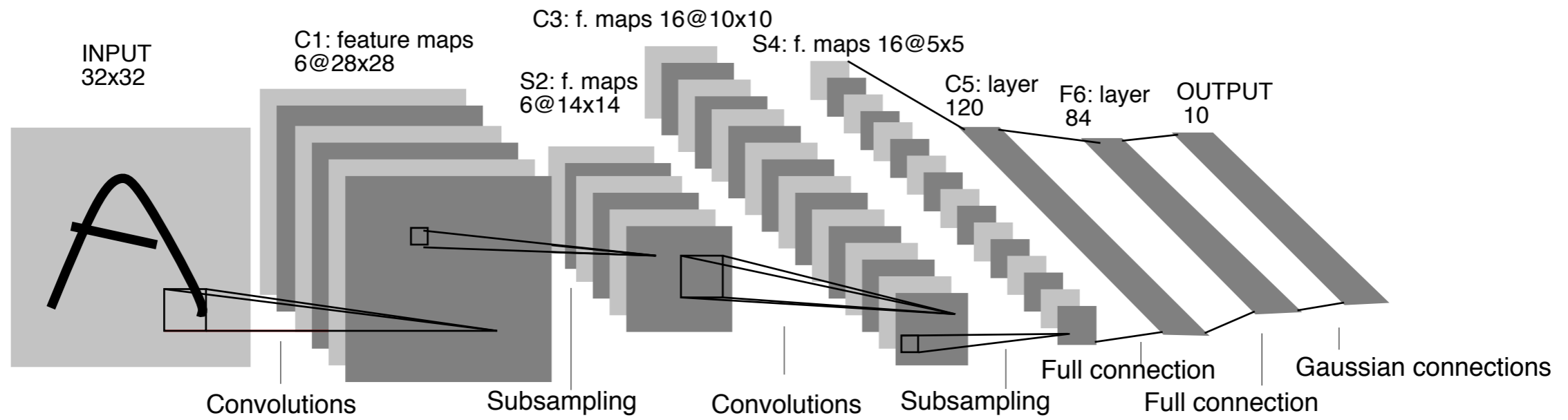
Yann LeCun, Léon Bottou, Yoshua Bengio and Patrick Haffner: [Gradient Based Learning Applied to Document Recognition](#), Proceedings of IEEE, 86(11):2278–2324, 1998.



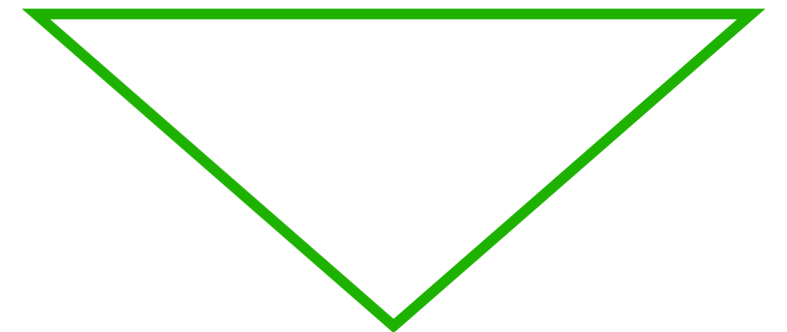
# Hidden Layers

PROC. OF THE IEEE, NOVEMBER 1998

7



"Automatic feature extractor"



"Regular classifier"

# Hidden Layers

PROC. OF THE IEEE, NOVEMBER 1998

7

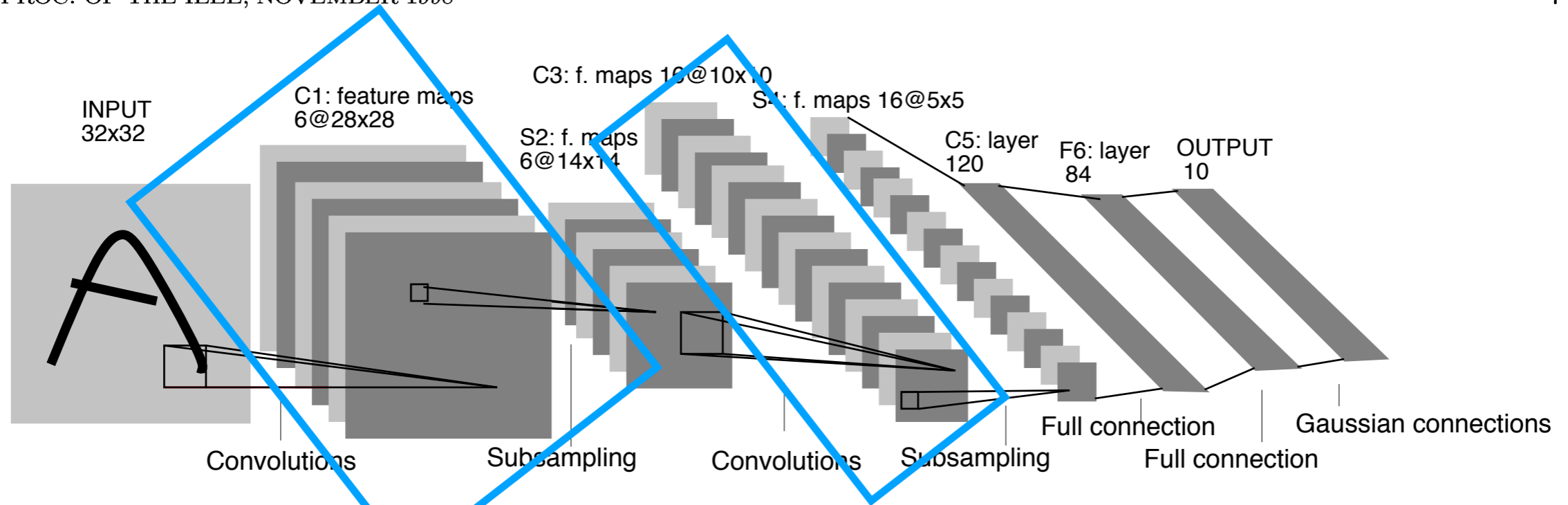


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

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

PROC. OF THE IEEE, NOVEMBER 1998

7

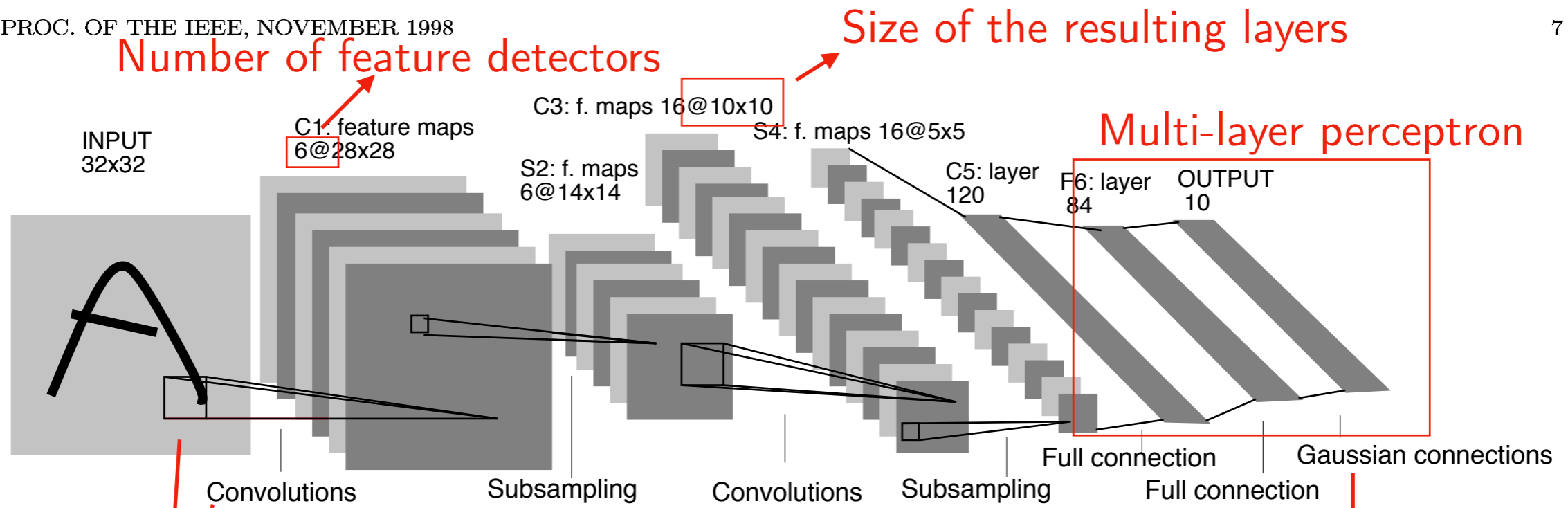


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

nowadays called "pooling"

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

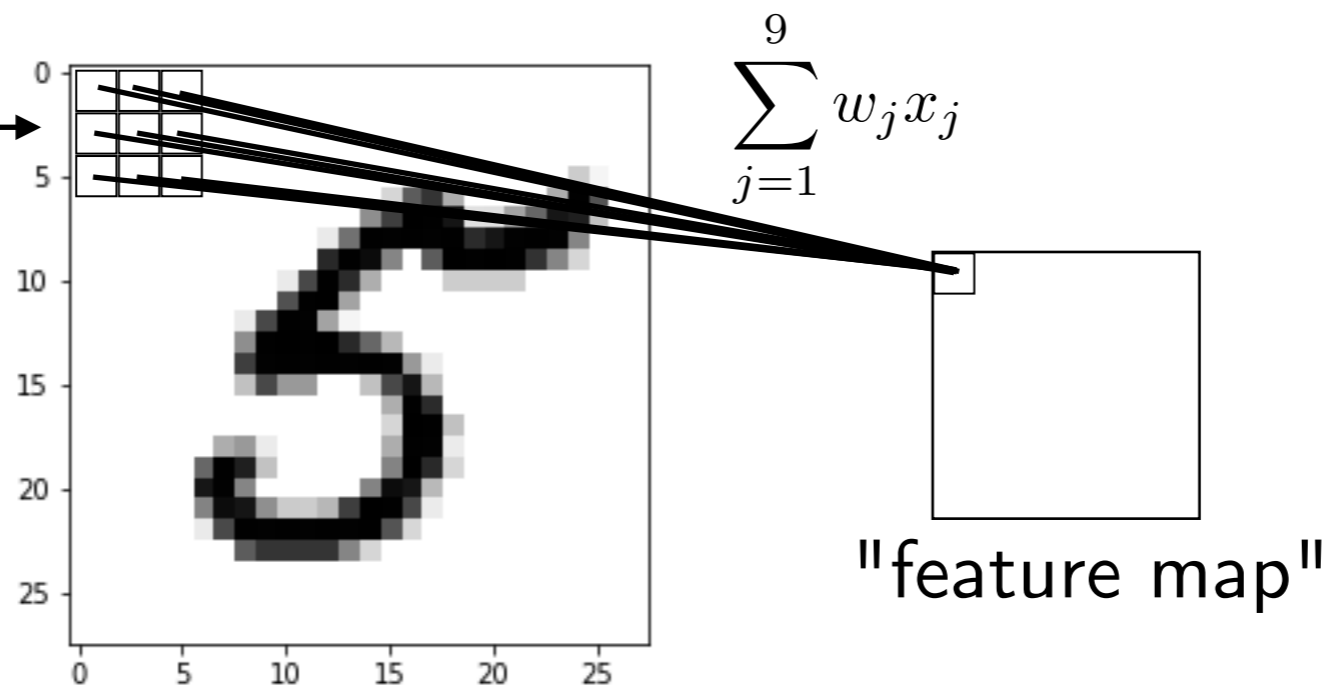
basically a fully-connected  
layer + MSE loss  
(nowadays better to use  
fc-layer + softmax  
+ cross entropy)

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

# Weight Sharing

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

The pixels are referred to as "receptive field"

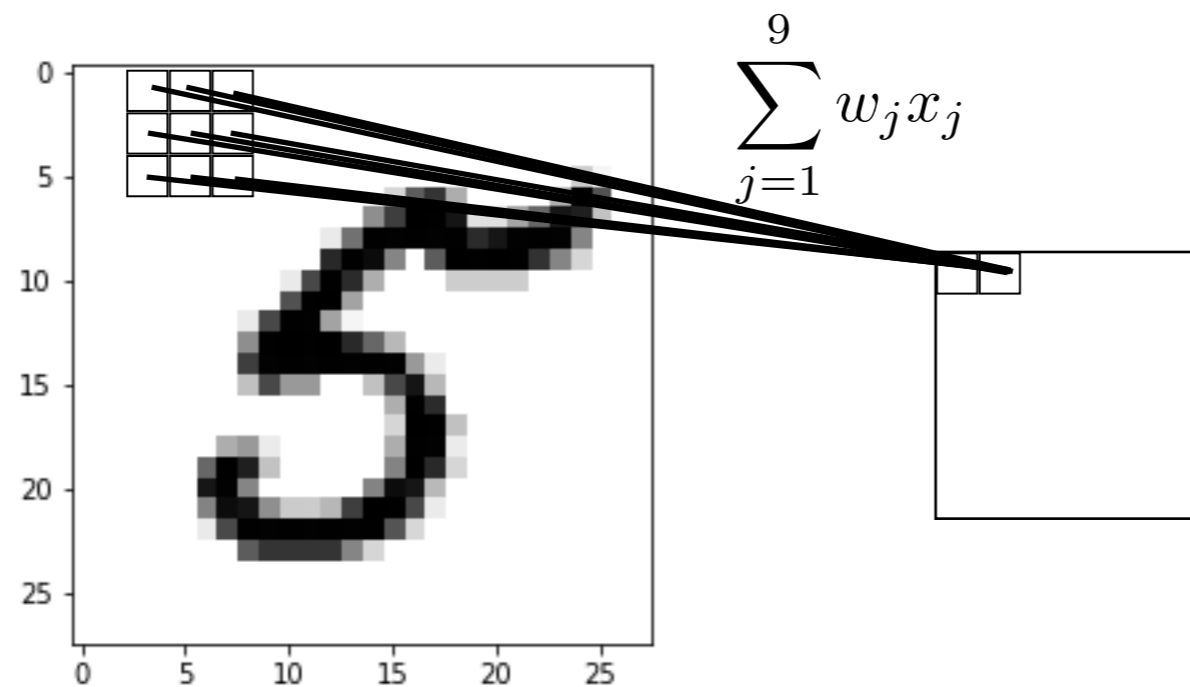


Rationale: A feature detector that works well in one region may also work well in another region

Plus, it is a nice reduction in parameters to fit

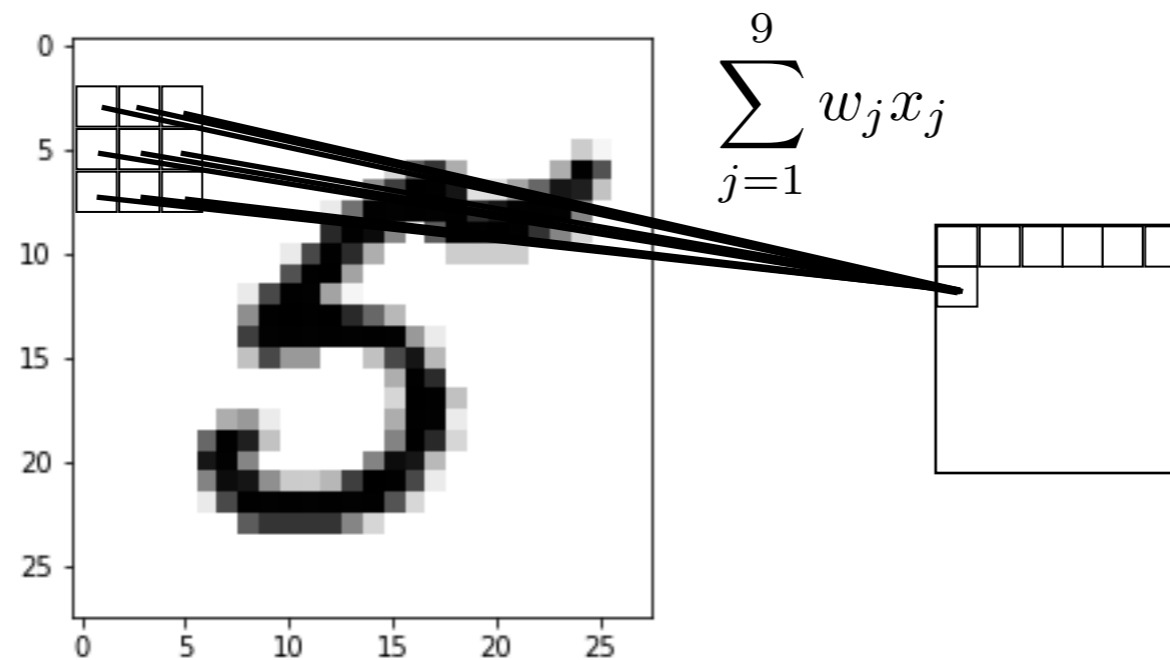
# Weight Sharing

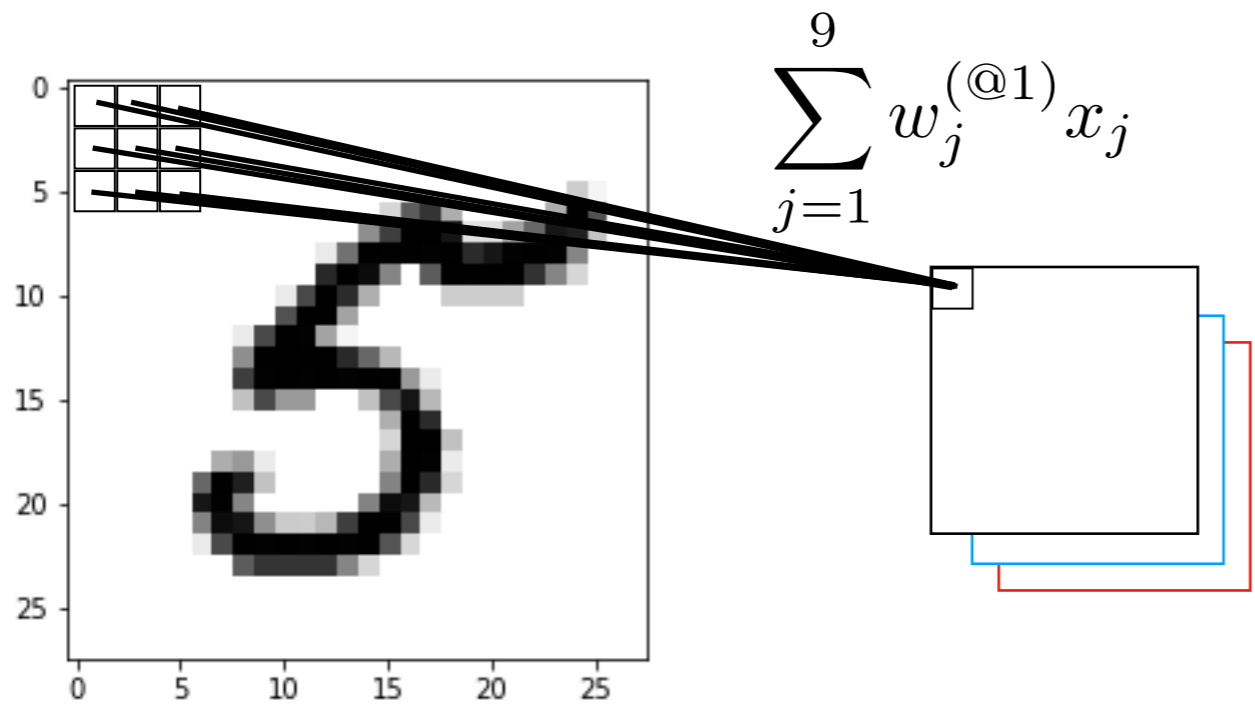
A "feature detector" (kernel) slides over the inputs to generate a feature map



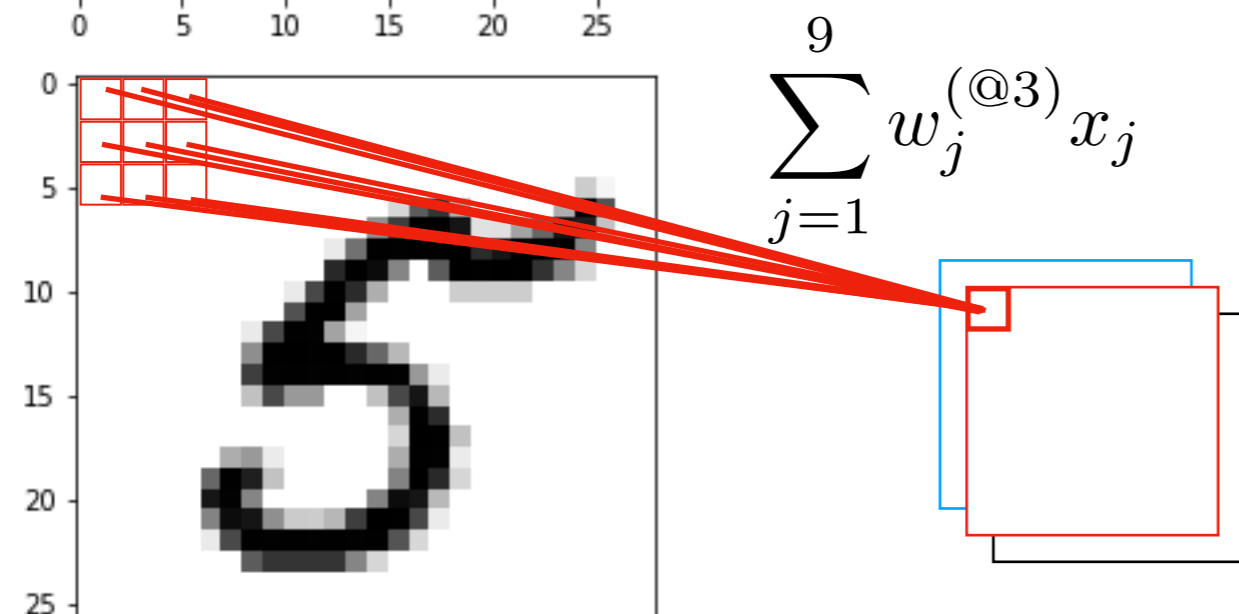
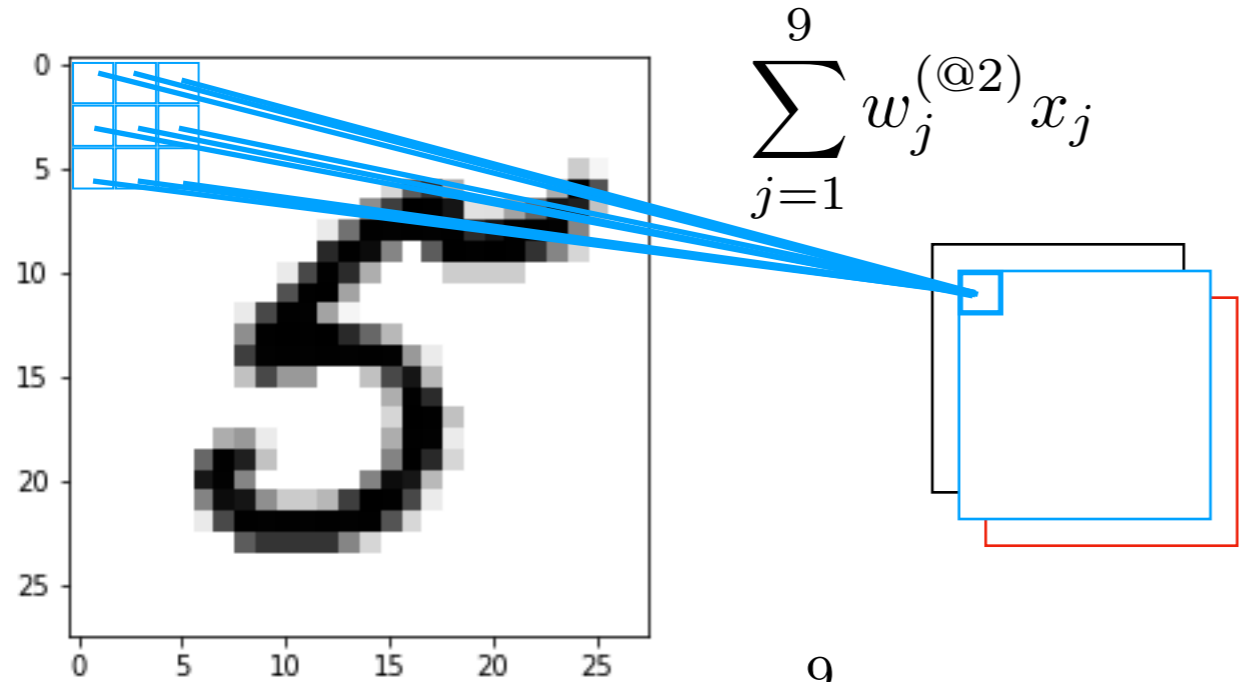
# Weight Sharing

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





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



# Size Before and After Convolutions

Feature map size:

$$O = \frac{W - K + 2P}{S} + 1$$

input width

kernel width

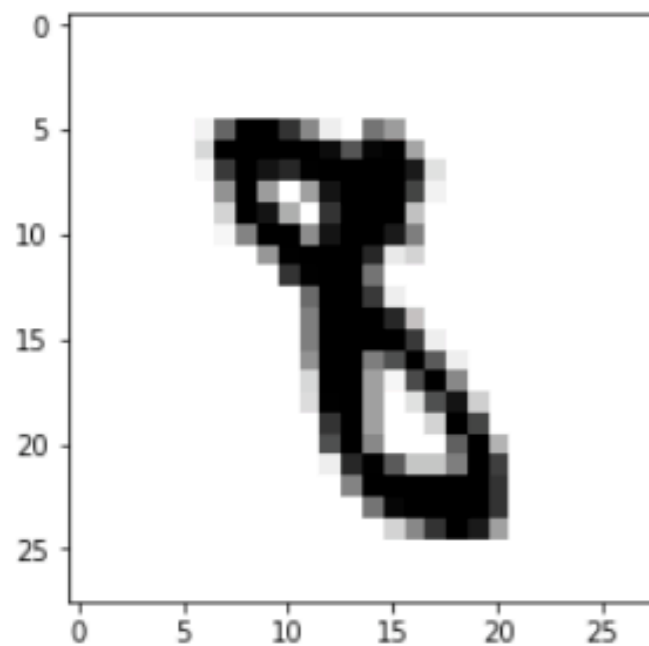
padding

stride

output width



# Kernel Dimensions and Trainable Parameters



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

```
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()
```

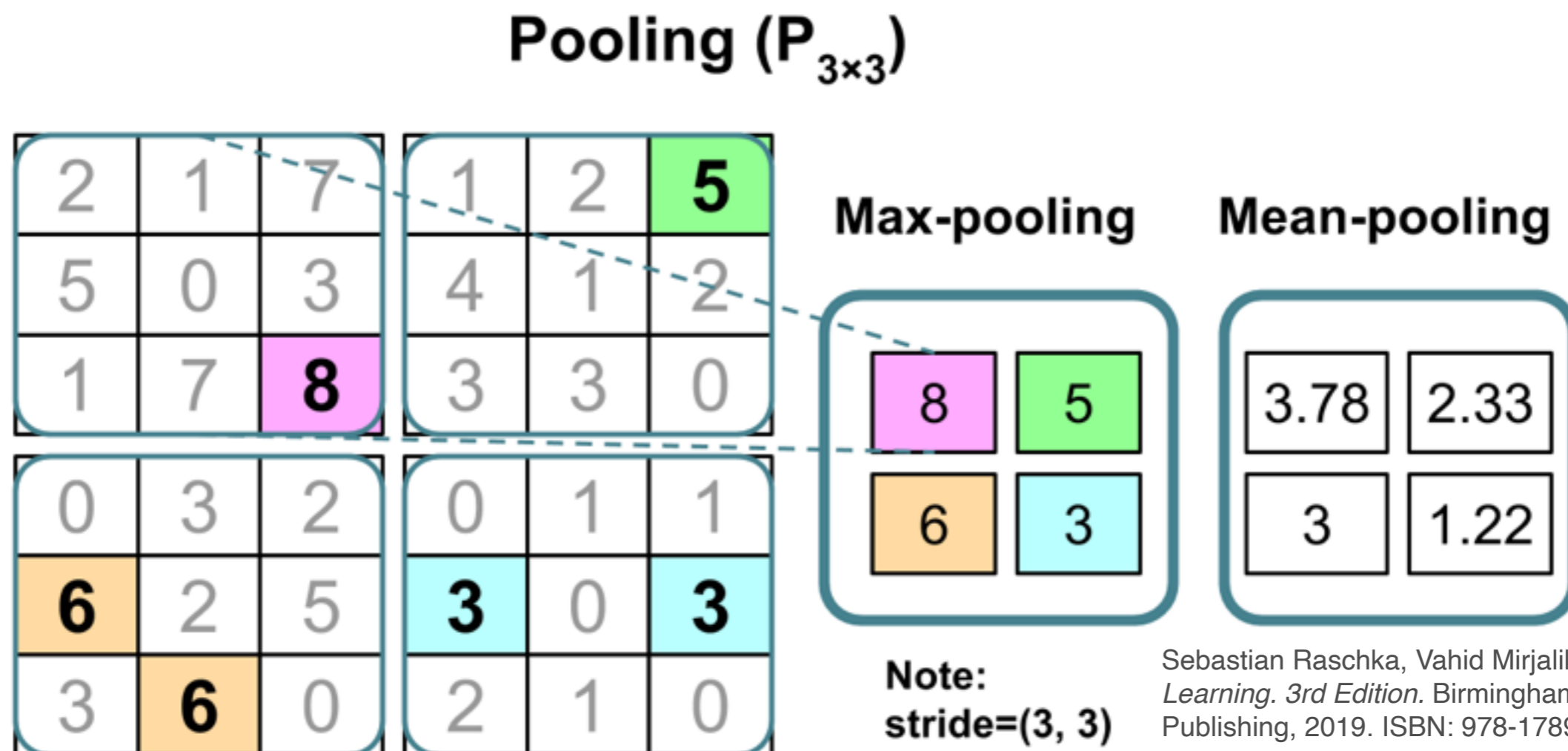
```
torch.Size([8])
```

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

# Backpropagation in CNNs

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

# Pooling Layers Can Help With Local Invariance

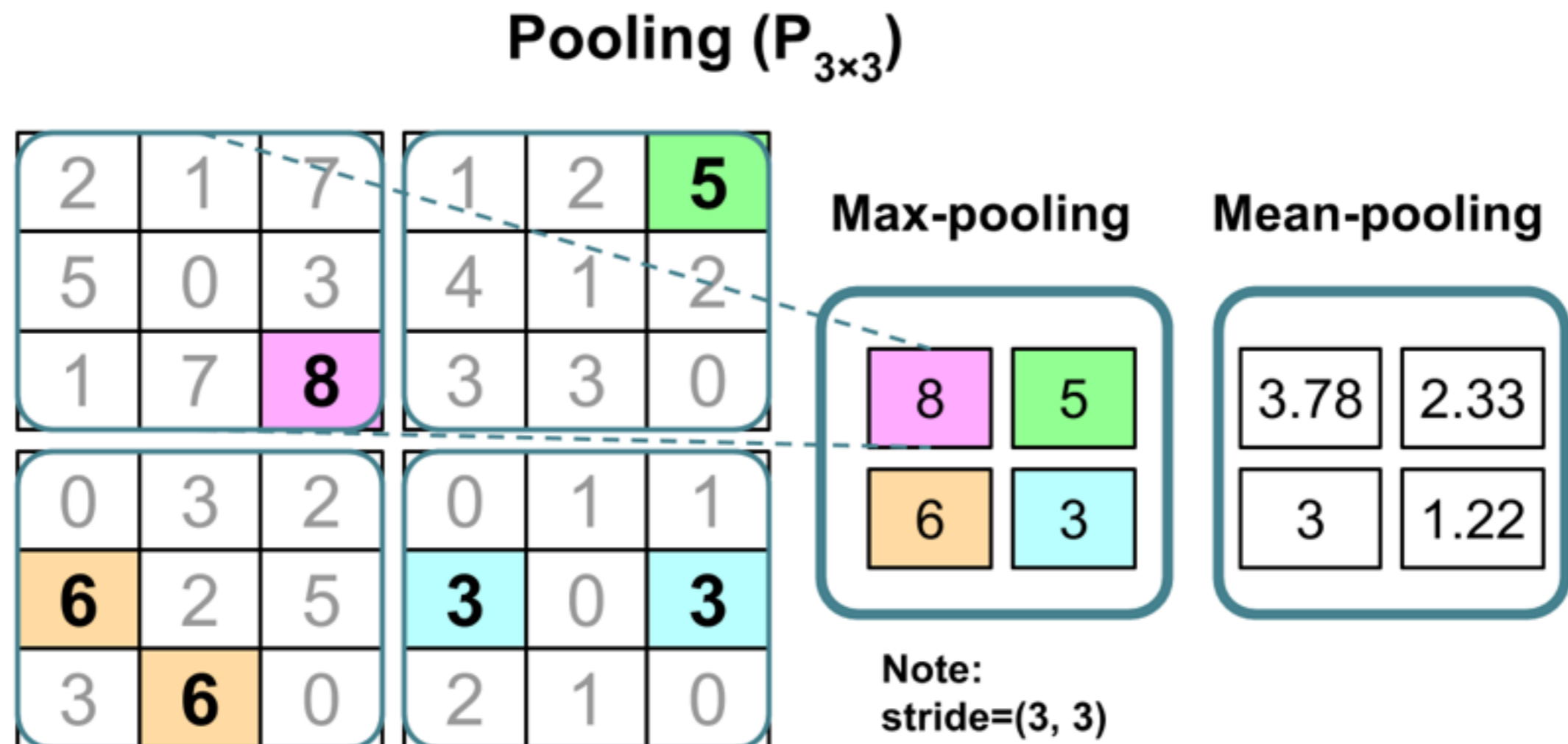


Downside: Information is lost.

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

# Lecture Overview

1. Image Classification
2. Convolutional Neural Network Basics
3. CNN Architectures
- 4. What a CNN Can See**
5. CNNs in PyTorch

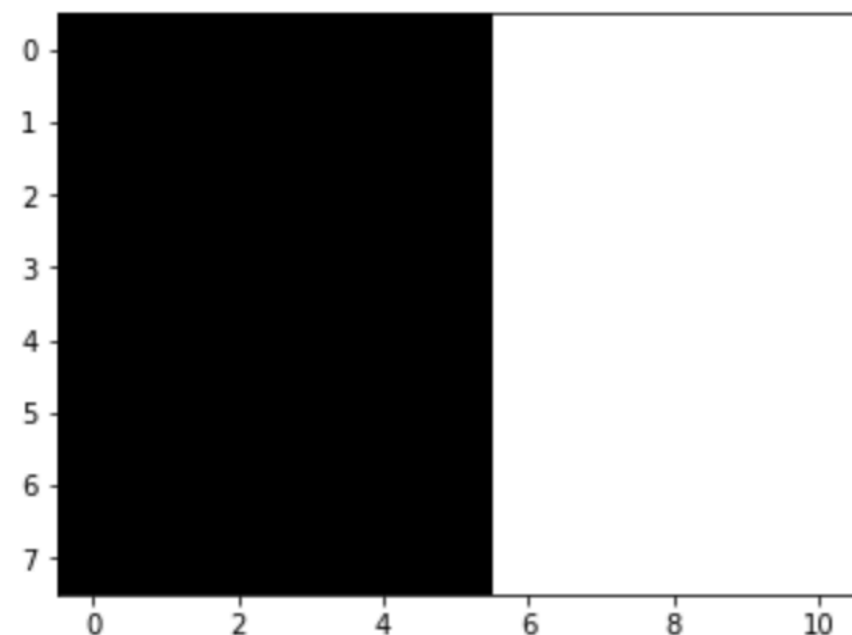
# What a CNN Can See

Simple example: vertical edge detector

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

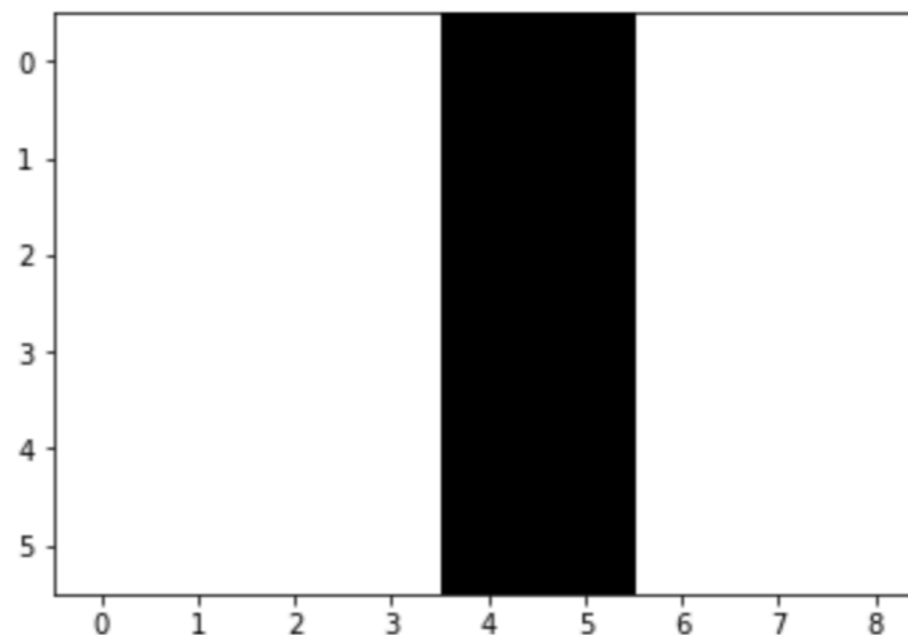
```
t = torch.tensor([  
  [0., 0., 0., 0., 0., 0., 1., 1., 1., 1., 1.],  
  [0., 0., 0., 0., 0., 0., 1., 1., 1., 1., 1.],  
  [0., 0., 0., 0., 0., 0., 1., 1., 1., 1., 1.],  
  [0., 0., 0., 0., 0., 0., 1., 1., 1., 1., 1.],  
  [0., 0., 0., 0., 0., 0., 1., 1., 1., 1., 1.],  
  [0., 0., 0., 0., 0., 0.,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

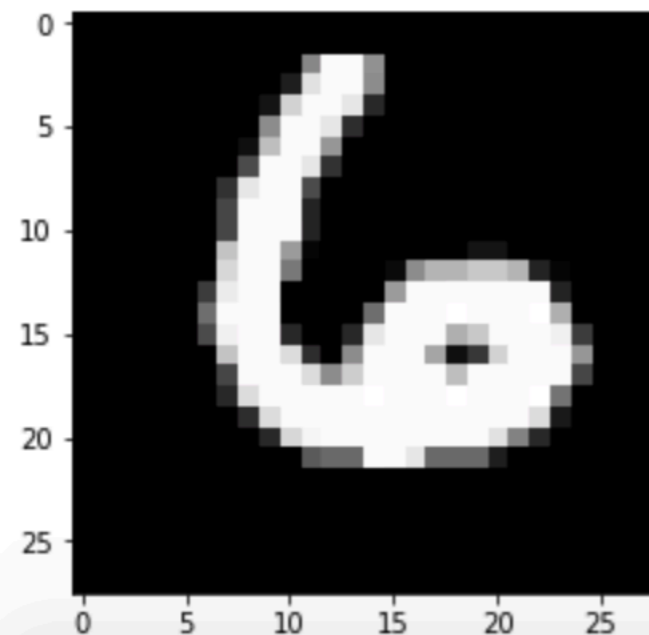
```
conv.weight.size()
```

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

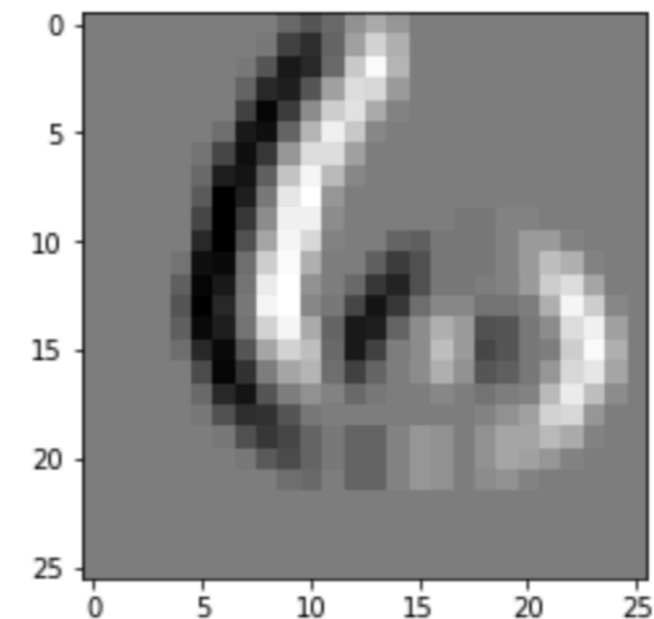
```
conv.weight[0, 0, :, :] = torch.tensor([[1, 0, -1],  
                                         [1, 0, -1],  
                                         [1, 0, -1]]).float()  
conv.bias[0] = torch.tensor([0.]).float()
```

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

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



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



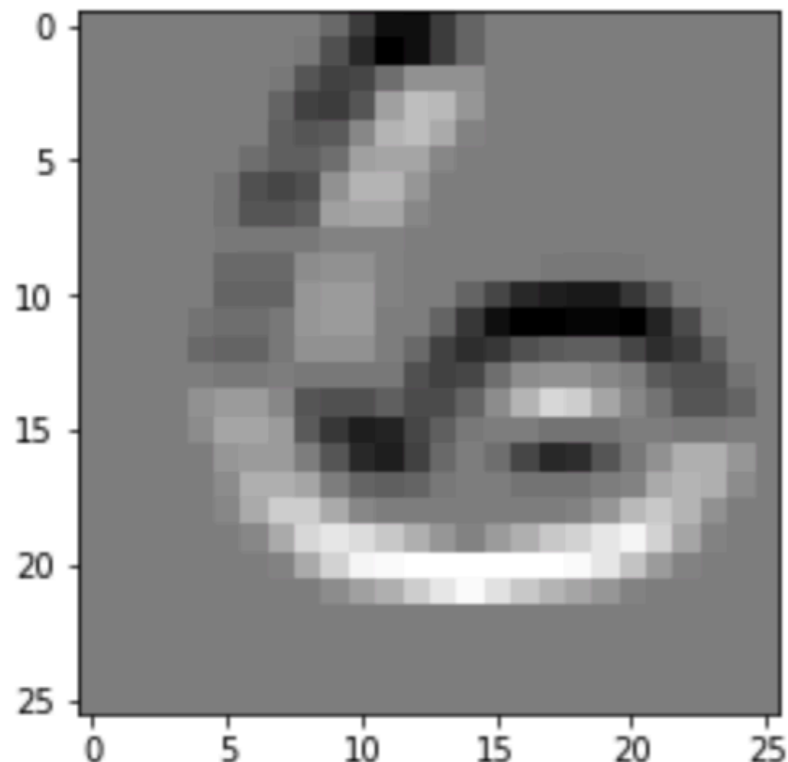
# What a CNN Can See

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');
```

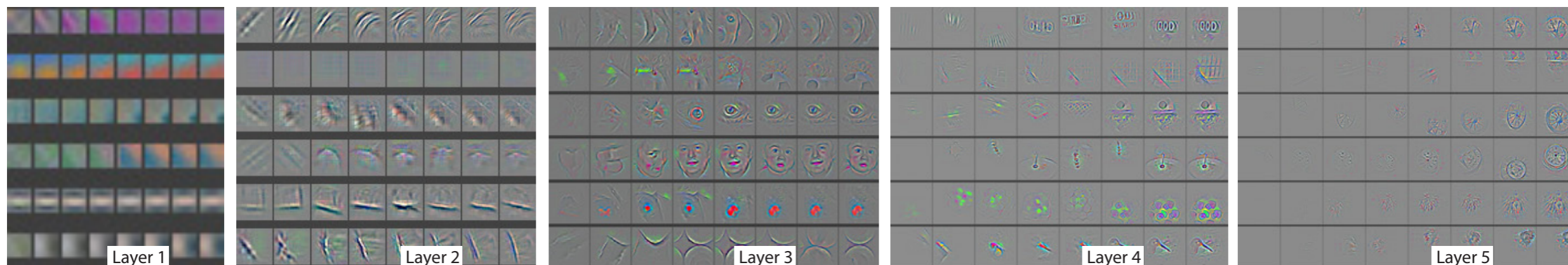


A CNN can learn whatever it finds best based on optimizing the objective (e.g., minimizing a particular loss to achieve good classification accuracy)



# 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.

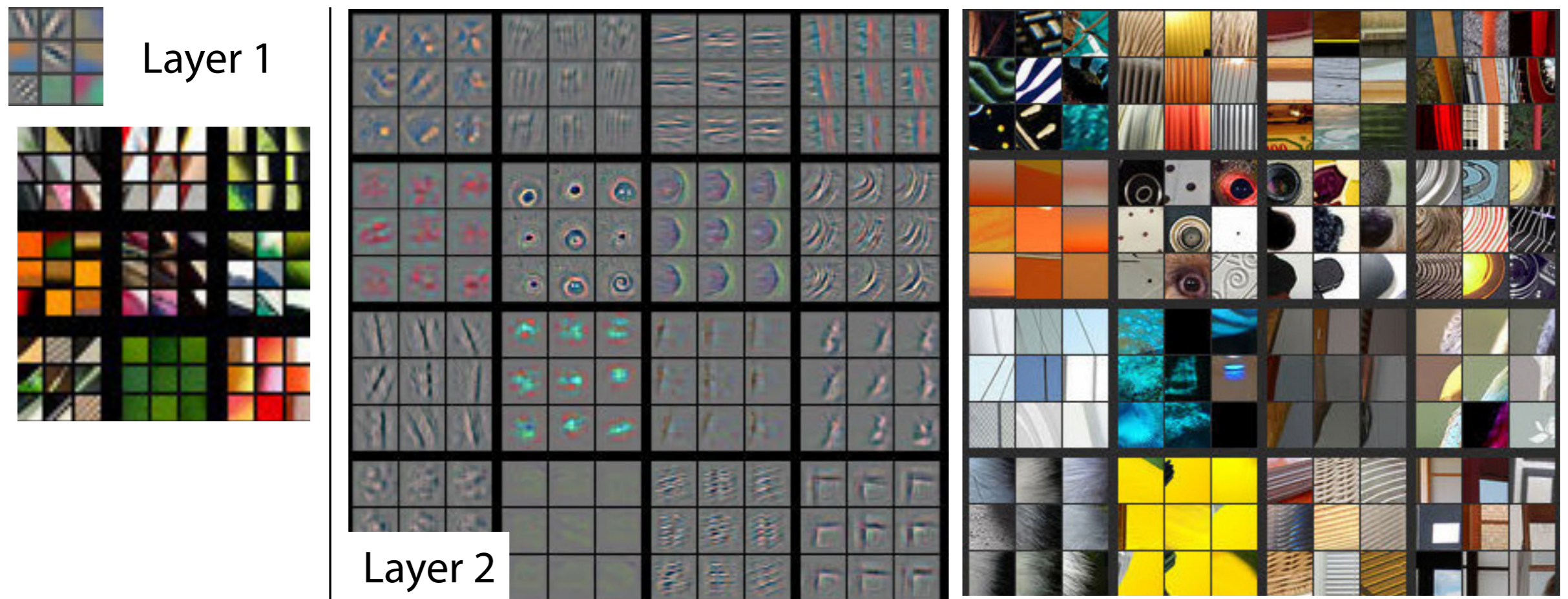
[Zeiler, M. D., & Fergus, R. \(2014, September\). Visualizing and understanding convolutional networks. In \*European conference on computer vision\* \(pp. 818-833\). Springer, Cham.](#)

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?

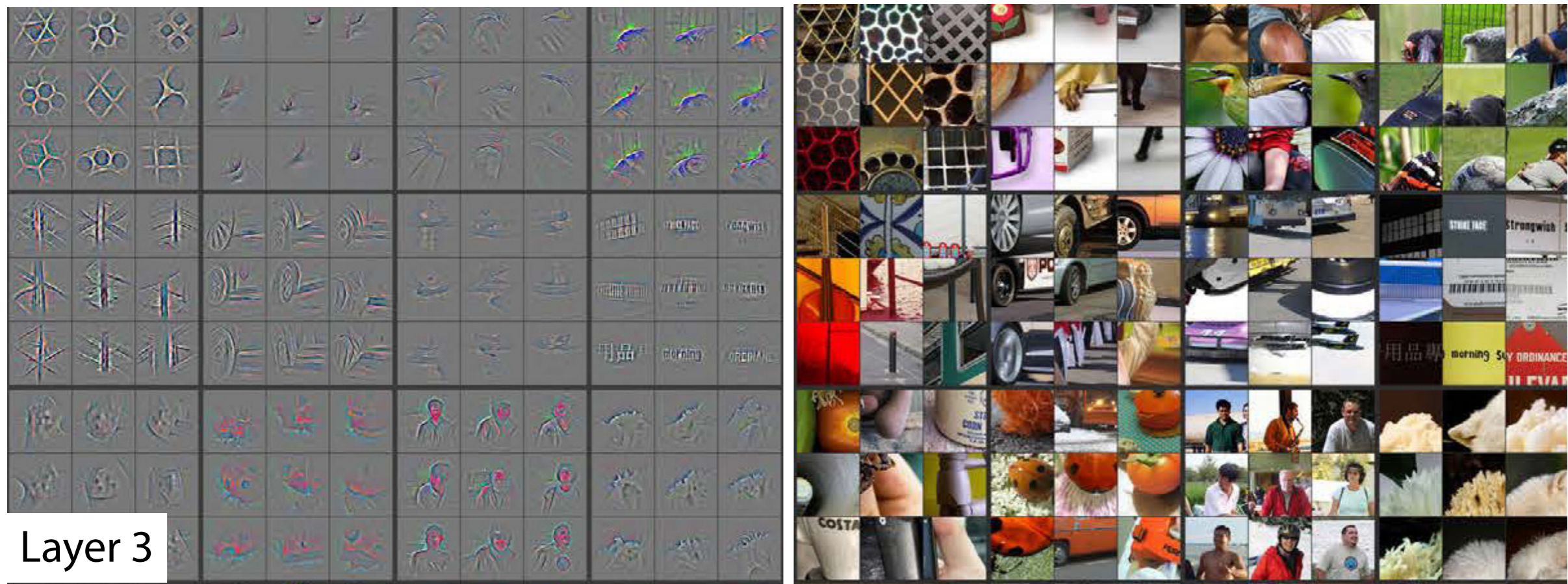
Zeiler, M. D., & Fergus, R. (2014, September). Visualizing and understanding convolutional networks. In *European conference on computer vision* (pp. 818-833). Springer, Cham.



# What a CNN Can See

Which patterns from the training set activate the feature map?

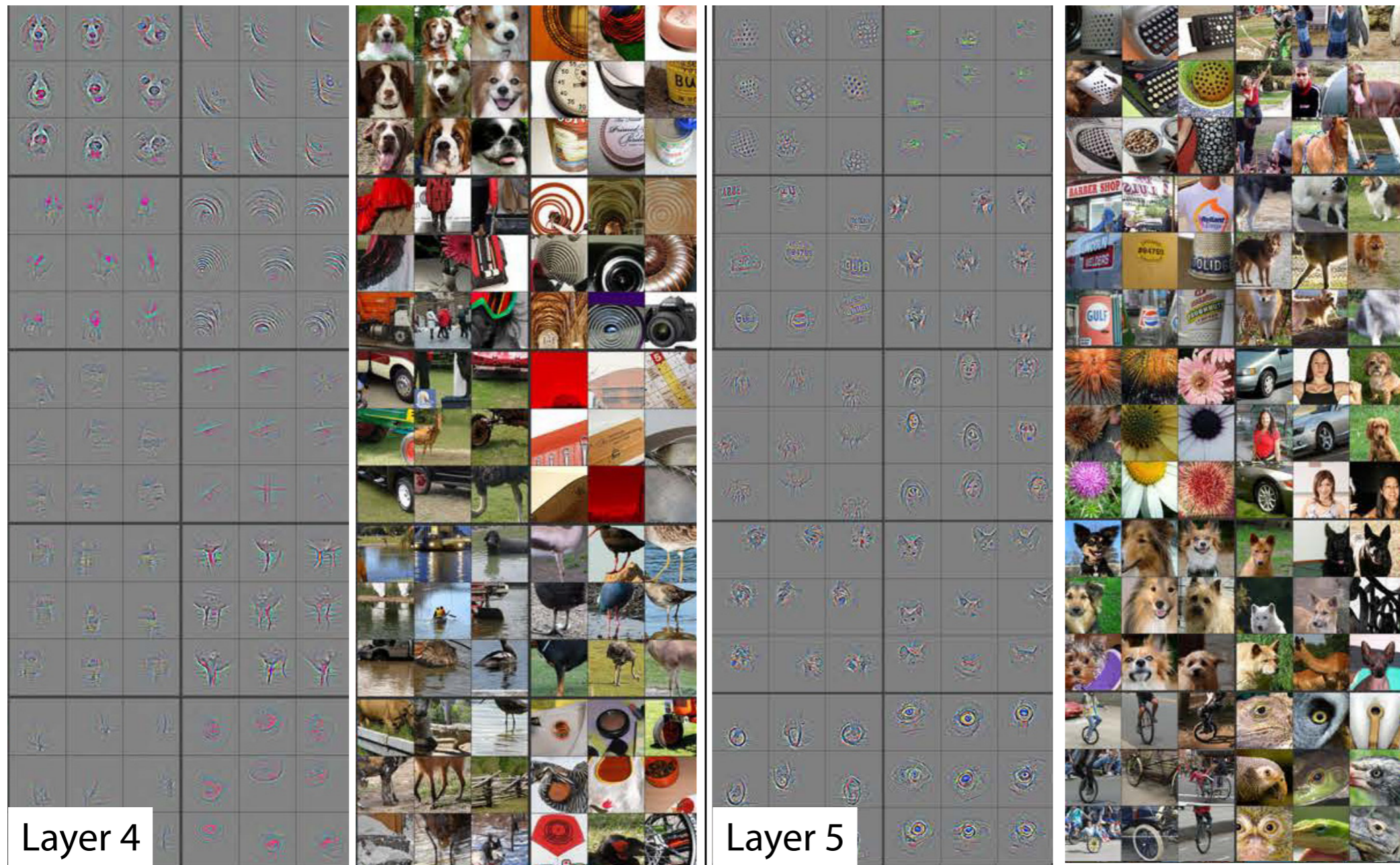
Zeiler, M. D., & Fergus, R. (2014, September). Visualizing and understanding convolutional networks. In *European conference on computer vision* (pp. 818-833). Springer, Cham.



# 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.



Lecture 13

# Introduction to Convolutional Neural Networks Part 2

STAT 453: Deep Learning, Spring 2020

Sebastian Raschka

<http://stat.wisc.edu/~sraschka/teaching/stat453-ss2020/>

<https://github.com/rasbt/stat453-deep-learning-ss20/tree/master/L13-cnns-part2>

# Lecture Overview



- 1. Padding (control output size in addition to stride)**
2. Spatial Dropout and BatchNorm
3. Considerations for CNNs on GPUs
4. Common Architectures
  - LeNet-5
  - AlexNet
  - VGG-16
  - ResNet-50
  - Inception-v1
5. Transfer learning

# Padding

output size

input size

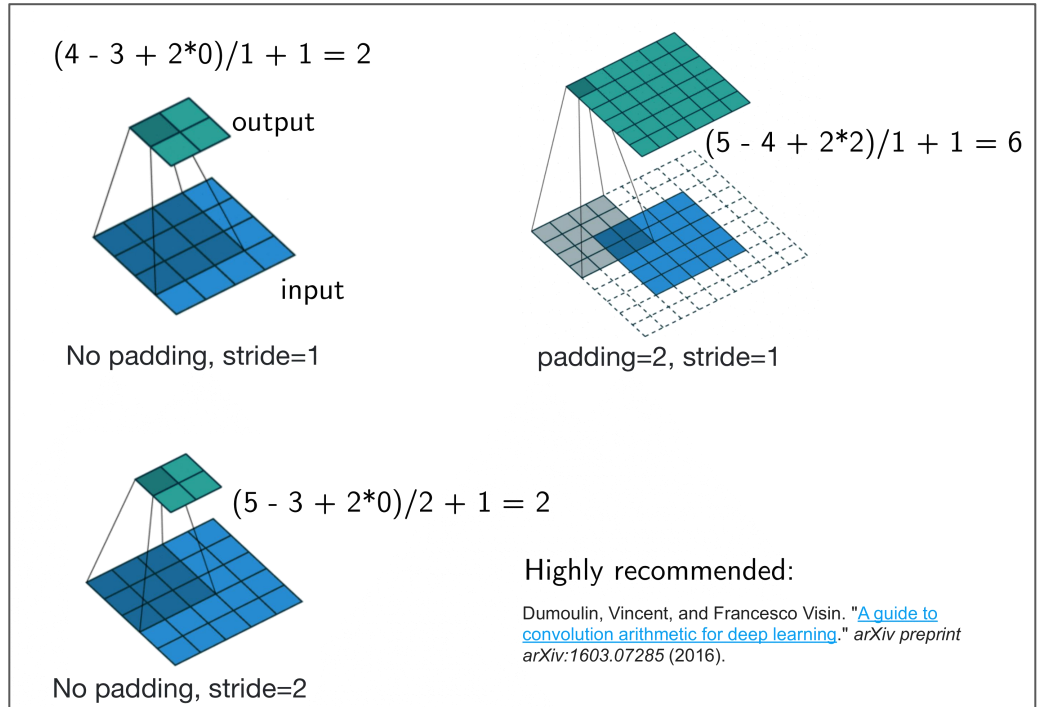
padding pixels per side

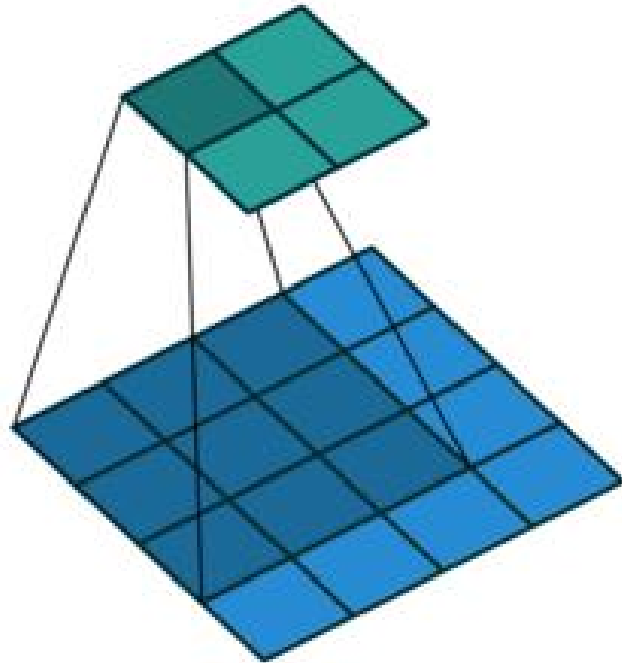
$$O = \left\lfloor \frac{i + 2p - k}{s} \right\rfloor + 1$$

"floor" function

stride

kernel size





no\_padding no\_strides





## Padding jargon

- "valid" convolution: no padding (feature map may shrink)
- "same" convolution: padding such that the output size is equal to the input size
- Common kernel size conventions:
  - 3x3, 5x5, 7x7 (sometimes 1x1 in later layers to reduce channels)

### Padding

$$o = \left\lfloor \frac{i + 2p - k}{s} \right\rfloor + 1$$

Assume you want to use a convolutional operation with stride 1 and maintain the input dimensions in the output feature map:

How much padding do you need for "same" convolution?

$$o = i + 2p - k + 1$$

$$\Leftrightarrow p = (o - i + k - 1)/2$$

$$\Leftrightarrow p = (k - 1)/2$$

Probably explains why common kernel size conventions are 3x3, 5x5, 7x7 (sometimes 1x1 in later layers to reduce channels)

# Lecture Overview



1. Padding (control output size in addition to stride)
- 2. Spatial Dropout and BatchNorm**
3. Considerations for CNNs on GPUs
4. Common Architectures
  - LeNet-5
  - AlexNet
  - VGG-16
  - ResNet-50
  - Inception-v1
5. Transfer learning

# Spatial Dropout -- Dropout2D

- Problem with regular dropout and CNNs:  
Adjacent pixels are likely highly correlated  
(thus, may not help with reducing the  
"dependency" much as originally intended by  
dropout)
- Hence, it may be better to drop entire feature maps

Idea comes from

Tompson, Jonathan, Ross Goroshin, Arjun Jain, Yann LeCun, and Christoph Bregler.  
["Efficient object localization using convolutional networks."](#) In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 648-656. 2015.

# BatchNorm 2D

In BatchNorm2d, the mean and standard deviation are computed for  $N \times H \times W$ , i.e., over the channel dimension

```
[1]: import torch.nn as nn
import torch.nn.functional as F

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

        self.cn1 = nn.Conv2d(3, 192, kernel_size=5,
                             stride=1, padding=2, bias=False)
        self.bn1 = nn.BatchNorm2d(192)

# ...

[2]: model = Model()

[3]: model.bn1.weight.size()

[3]: torch.Size([192])
```

# Lecture Overview



1. Padding (control output size in addition to stride)
2. Spatial Dropout and BatchNorm
- 3. Considerations for CNNs on GPUs**
4. Common Architectures
  - LeNet-5
  - AlexNet
  - VGG-16
  - ResNet-50
  - Inception-v1
5. Transfer learning

# Computing Convolutions on the GPU

- There are many different approaches to compute (approximate) convolution operations
- DL libraries usually use NVIDIA's CUDA & CuDNN libraries, which implement many different convolution algorithms
- These algorithms are usually more efficient than the CPU variants (convolutions on the CPU e.g., in CPU usually take up much more memory due to the algorithm choice compared to using the GPU)

If you are interested, you can find more info in:

Lavin, Andrew, and Scott Gray. "Fast algorithms for convolutional neural networks." *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016.

[https://www.cv-foundation.org/openaccess/content\\_cvpr\\_2016/papers/](https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/)

[Lavin\\_Fast\\_Algorithms\\_for\\_CVPR\\_2016\\_paper.pdf](#)

# Computing Convolutions on the GPU

- CuDNN is more geared towards engineers & speed rather than scientists and is unfortunately not deterministic/reproducible by default
- I.e., it determines which convolution algorithm to choose during run-time automatically, based on predicted speeds given the data flow
- For reproducibility and consistent results, I recommend setting the deterministic flag (speed is about the same, often even a bit faster, sometimes a bit slower)

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

if torch.cuda.is_available():
    torch.backends.cudnn.deterministic = True
```

# Lecture Overview

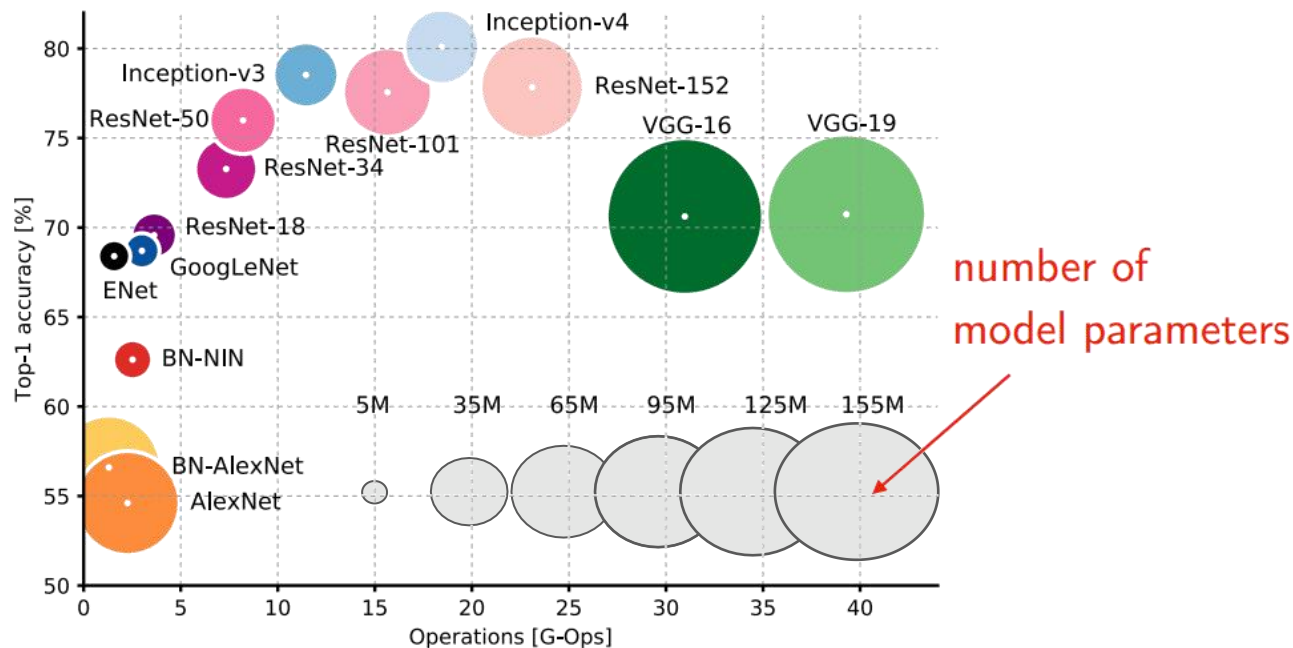


1. Padding (control output size in addition to stride)
2. Spatial Dropout and BatchNorm
3. Considerations for CNNs on GPUs
- 4. Common Architectures**
  - LeNet-5
  - AlexNet
  - VGG-16
  - ResNet-50
  - Inception-v1
5. Transfer learning

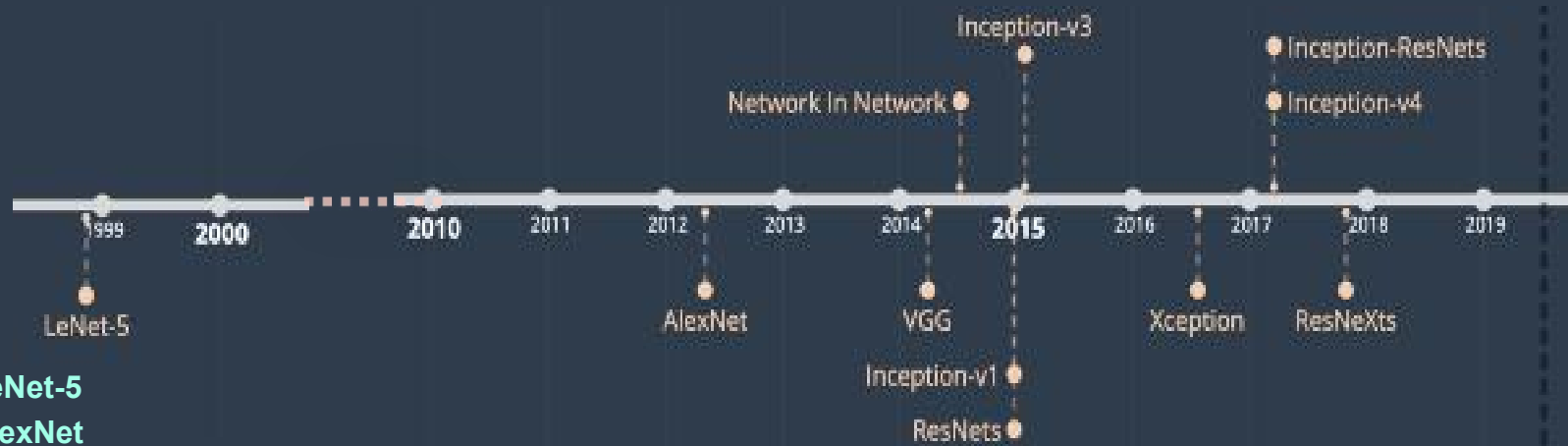


# Common Architectures Revisited

We will discuss some additional common CNN architectures since the field evolved quite a bit since 2012 ...



# CNNs Architectures Illustrated



1. LeNet-5
2. AlexNet
3. VGG-16
4. ResNet-50
5. Inception-v1
6. Inception-v3
7. Xception
8. Inception-v4
9. Inception-ResNets
10. ResNeXt-50

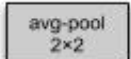
Source: <https://towardsdatascience.com/illustrated-10-cnn-architectures-95d78ace614d>

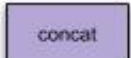



# Legend

## Layers

 Convolutional operations, in **red**

 Pooling operations, in **grey**

 Merge operations eg. concat, add in **purple**


 Dense layer, **blue**


## Activation Functions

 Tanh

 ReLU

## Other Functions

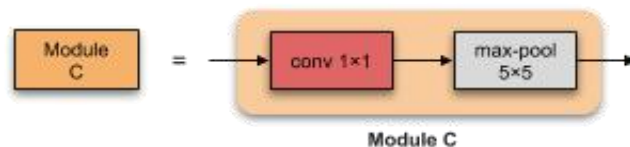
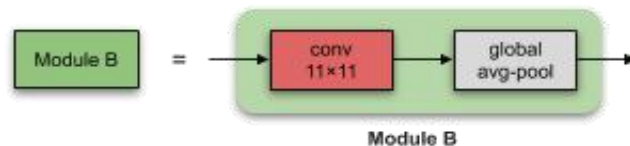
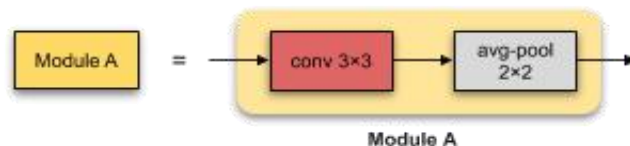
 Batch normalisation

 Softmax

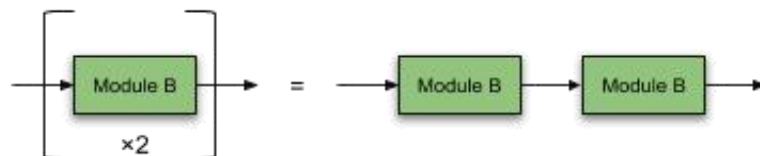
## Modules/Blocks

Modules (groups of convolutional, pooling and merge operations), in **yellow, green, or orange**.

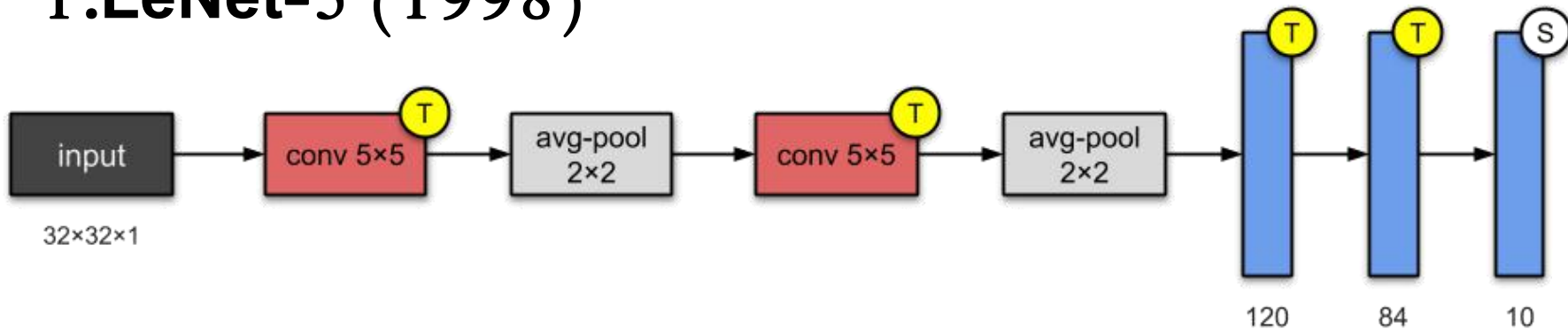
The operations that make up these modules will also be shown.



## Repeated layers or modules/blocks



# 1. LeNet-5 (1998)



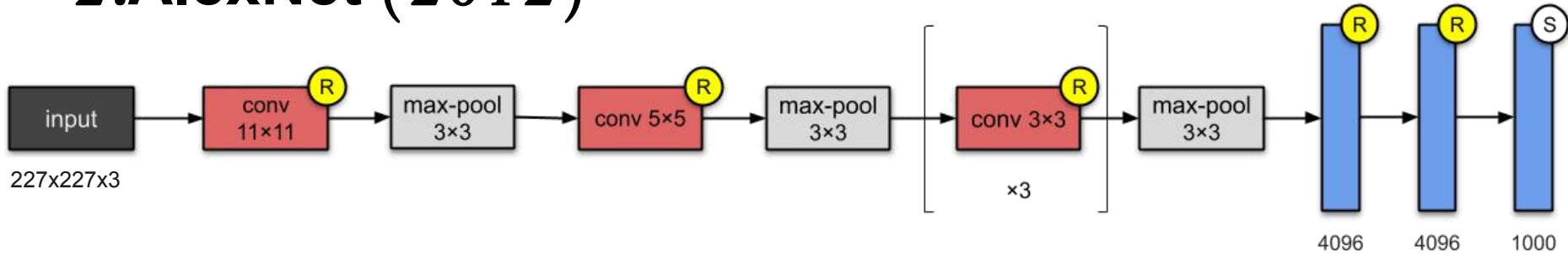
- ~60,000 parameters.
- One of the simplest architectures.
- (“5”-layers) 2 convolutional and 3 fully-connected layers.
- Sub-sampling layer and trainable weights (aka average-pooling layer)
  - (trainable weights is not current practice of designing CNNs nowadays).

[paper](#): [Gradient-Based Learning Applied to Document Recognition.](#)  
Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner.  
[Proceedings of the IEEE \(1998\).](#)

- This architecture has become the standard ‘template’: stacking convolutions and pooling layers, and ending the network with one or more fully-connected layers.



## 2. AlexNet (2012)

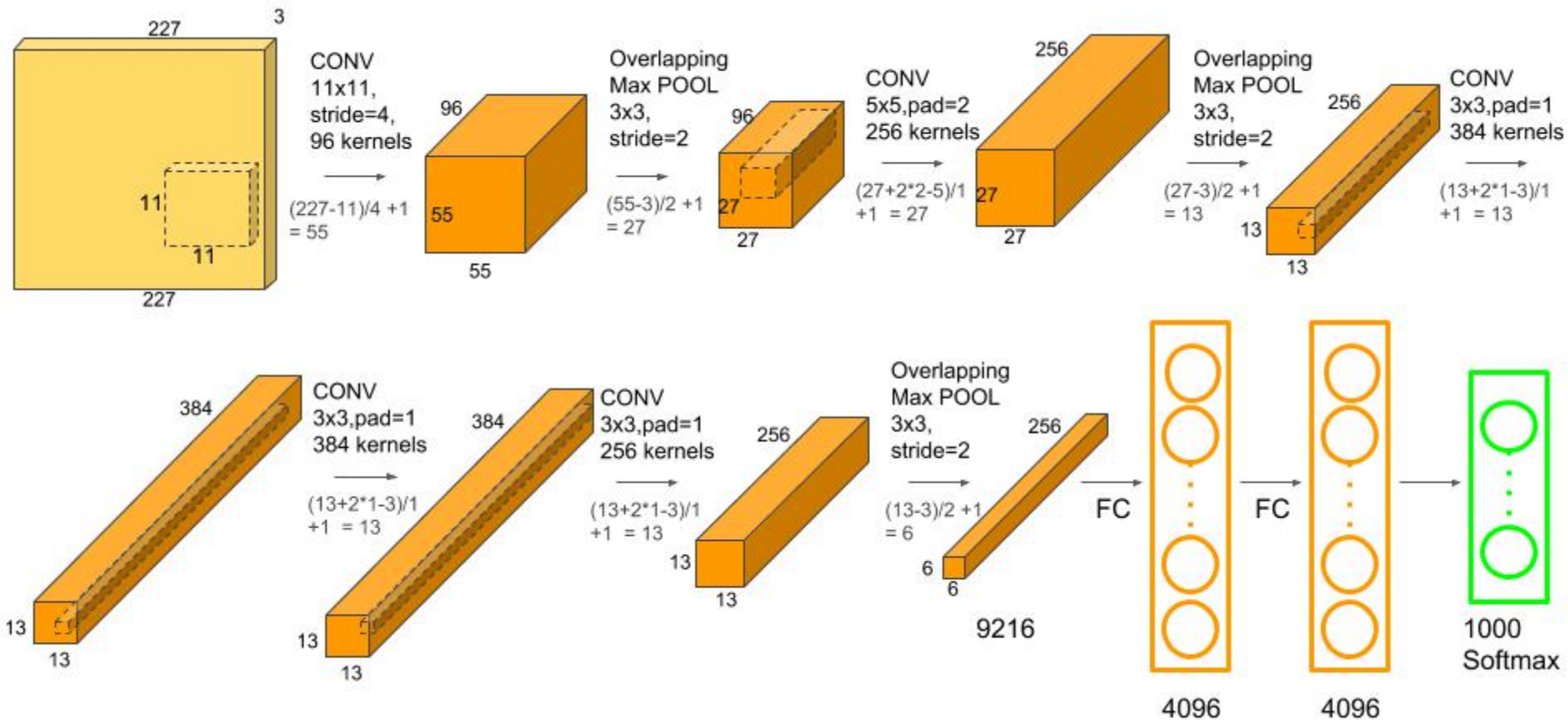


- ~60M parameters,
- 8 layers – 5 convolutional and 3 fully-connected.
- AlexNet just stacked a few more layers onto LeNet-5.
- Trained in two GPUs GTX 580 between 5 and 6 days

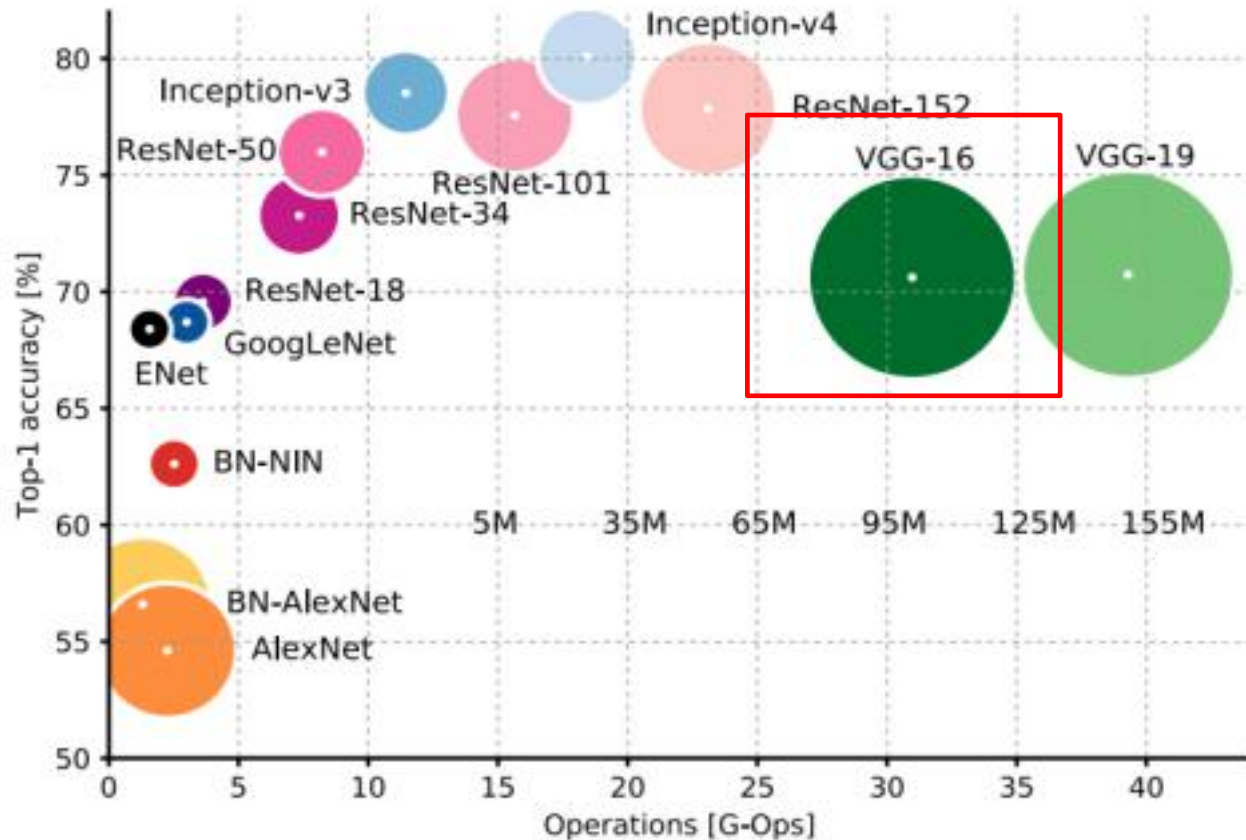
[paper: ImageNet Classification with Deep Convolutional Neural Networks.](#) Alex Krizhevsky, Ilya Sutskever, Geoffrey Hinton. University of Toronto, Canada. NeurIPS 2012

- They were the first to implement Rectified Linear Units (ReLUs) as activation functions.



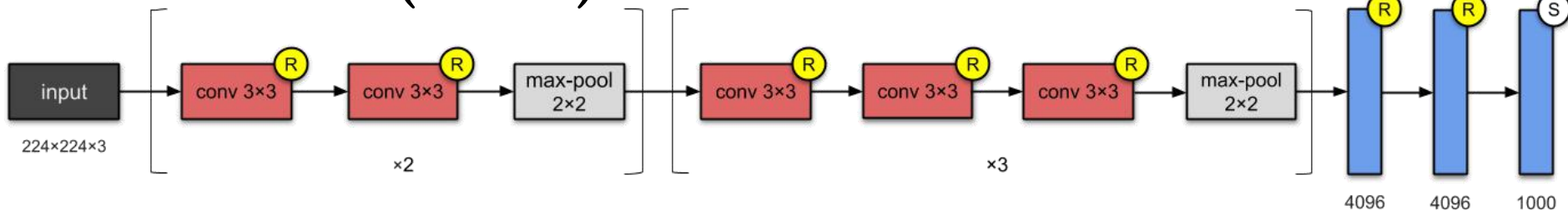


# 3.VGG-16 (2014)





# 3.VGG-16 (2014)



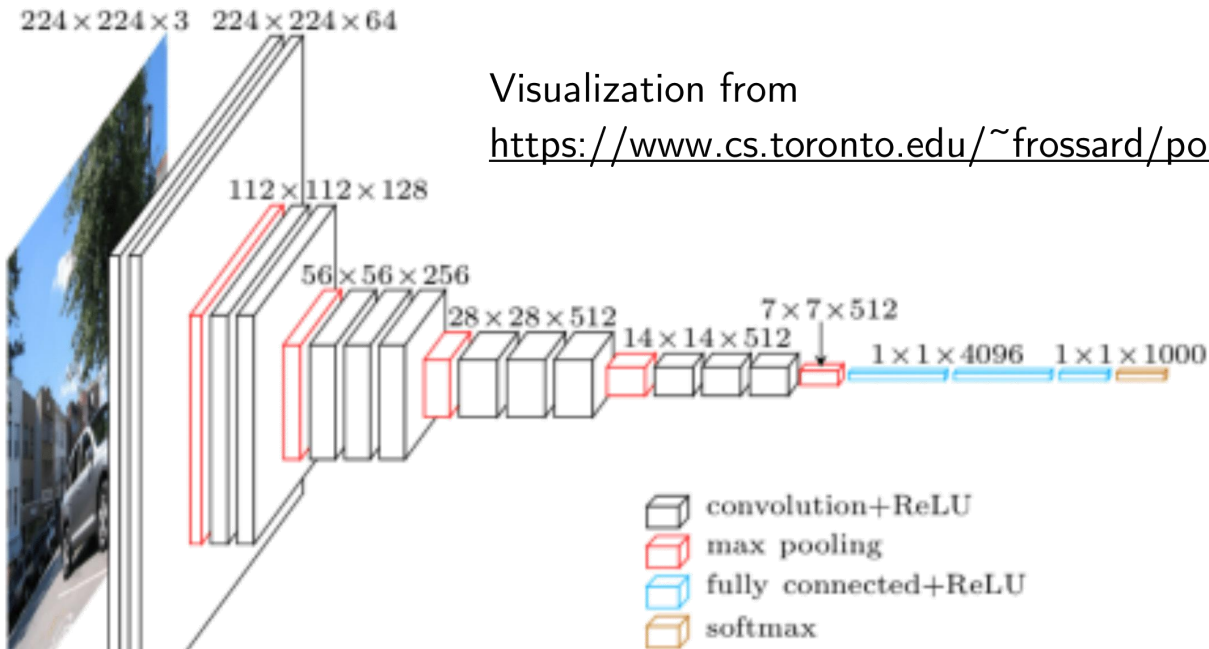
- CNNs starting to get deeper and deeper.
- Creators: Visual Geometry Group (VGG).
- 13 convolutional and 3 fully-connected layers
  - carrying ReLU tradition from AlexNet.
- Stacks more layers onto AlexNet, and smaller size filters
  - (2x2 and 3x3).
- ~138M parameters.
- ~500MB of storage space.

[paper](#): [Very Deep Convolutional Networks for Large-Scale Image Recognition](#). Karen Simonyan, Andrew Zisserman. University of Oxford, UK. arXiv preprint, 2014

- The contribution from this paper is the designing of deeper networks (roughly twice as deep as AlexNet).

Simonyan, Karen, and Andrew Zisserman. "[Very deep convolutional networks for large-scale image recognition](#)." *arXiv preprint arXiv:1409.1556* (2014).

# 3.VGG-16 (2014)



Visualization from

<https://www.cs.toronto.edu/~frossard/post/vgg16/>

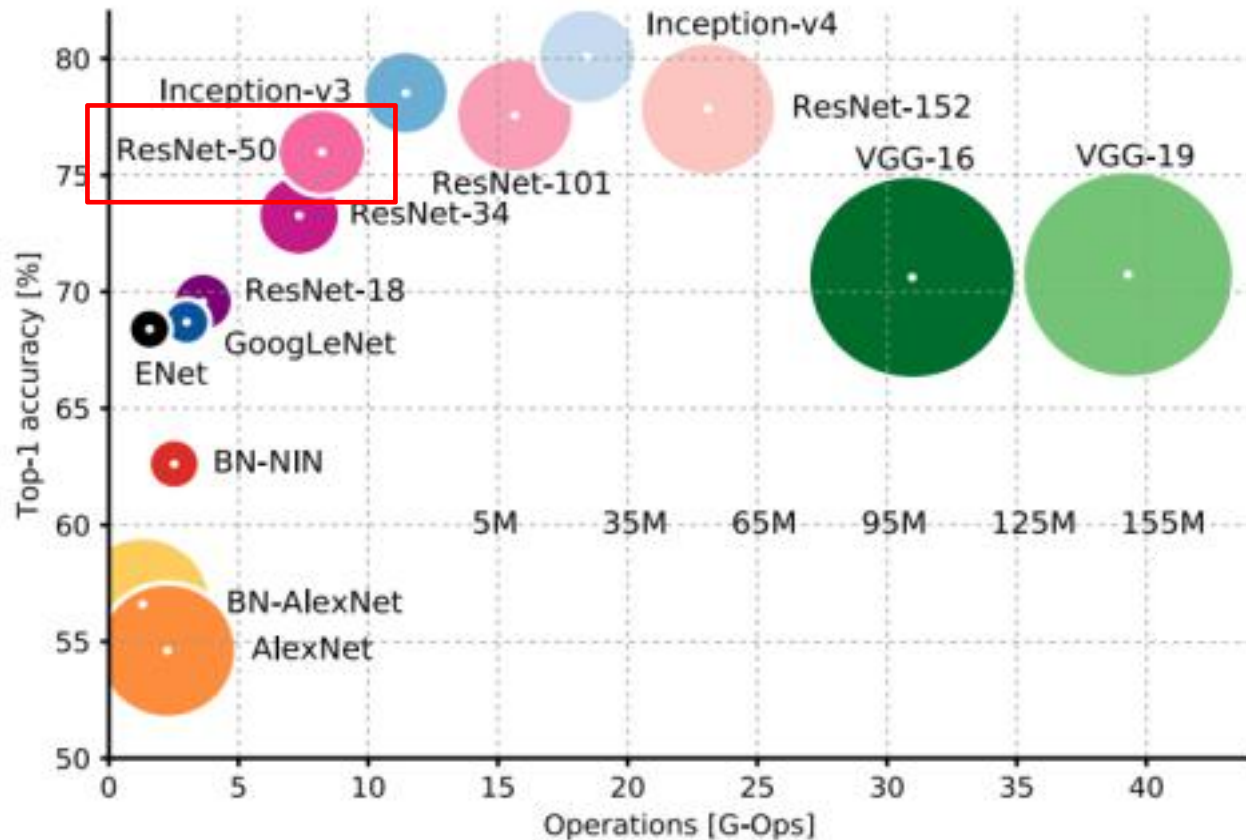
PyTorch implementation:

<https://github.com/rasbt/stat453-deep-learning-ss20/blob/master/L13-cnns-part2/code/vgg16.ipynb>

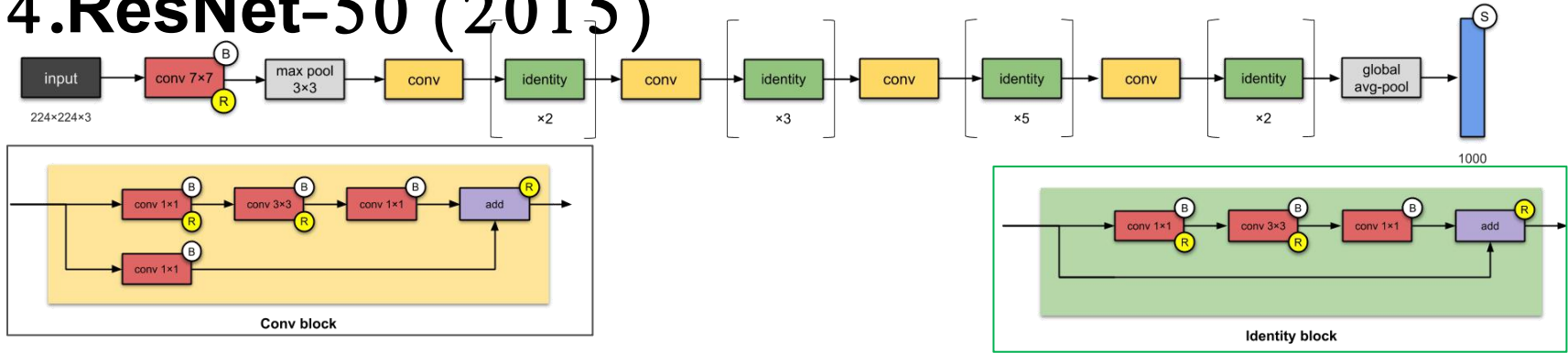
Simonyan, Karen, and Andrew Zisserman. "[Very deep convolutional networks for large-scale image recognition](#)." *arXiv preprint arXiv:1409.1556* (2014).



# 4. ResNet-50 (2015)



# 4. ResNet-50 (2015)



*“with the network depth increasing, accuracy gets saturated and then degrades rapidly.”*

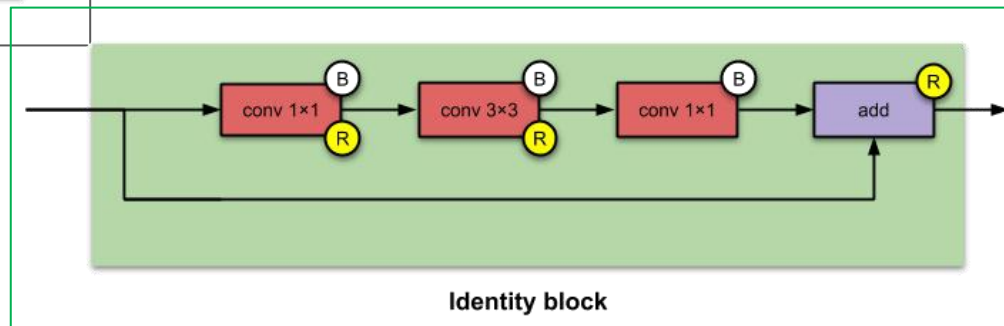
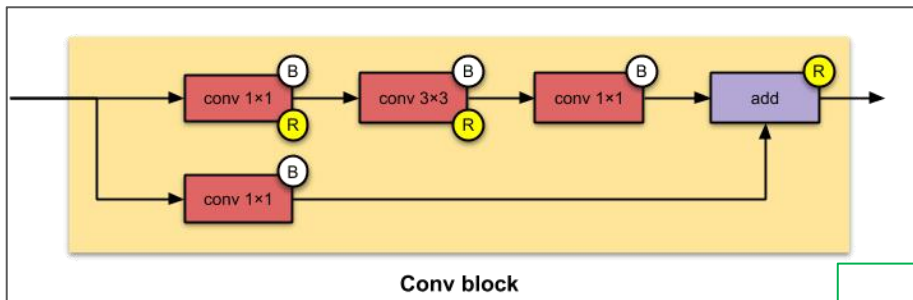
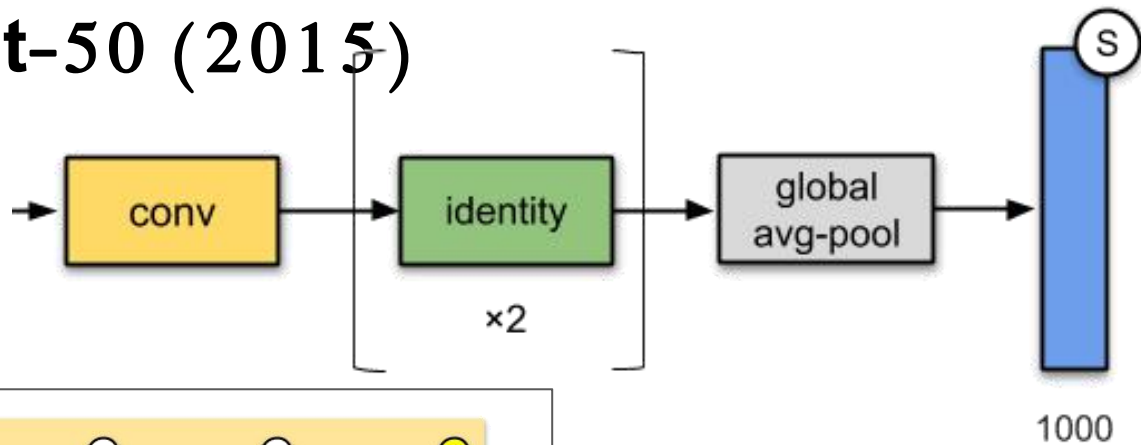
- Microsoft Research addressed this problem – using skip connections.
- Early adopters of batch normalisation
- ~26M parameters.
- The basic building block for ResNets are the conv and identity blocks.

GitHub [code](#) from keras-team.

Deep Residual Learning for Image Recognition. Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun. Microsoft. 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

- ResNet 34, 50, 101 up to 152 layers without compromising generalisation power
- 152 layers - trained in a cluster of 8 GPUs for 2 to 3 weeks
- Among the first to use batch normalisation.

# 4. ResNet-50 (2015)



# 4.ResNet-50 (2015)

PyTorch implementations of the previous slides:

<https://github.com/rasbt/stat453-deep-learning-ss20/blob/master/L13-cnns-part2/code/resnet-blocks.ipynb>

PyTorch implementations:

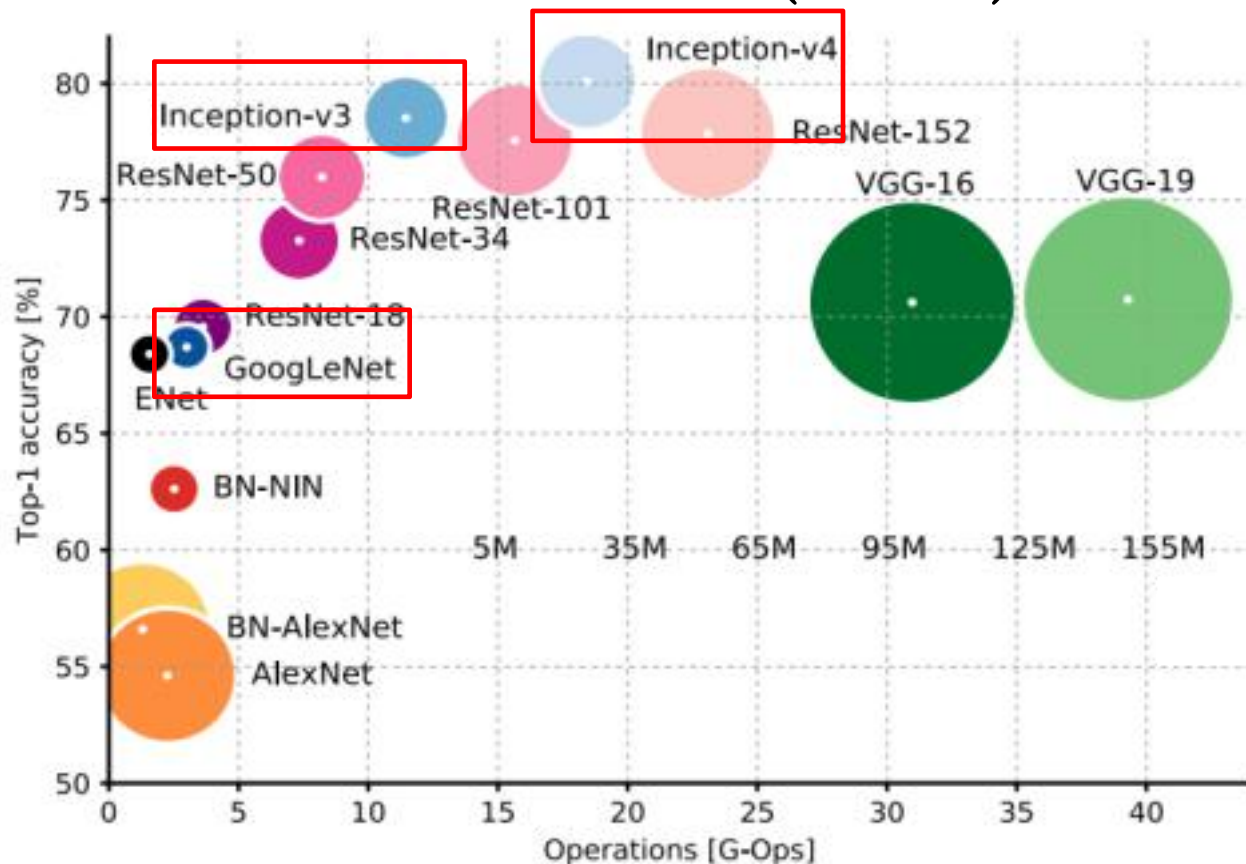
<https://github.com/rasbt/stat453-deep-learning-ss20/blob/master/L13-cnns-part2/code/resnet-34.ipynb>

<https://github.com/rasbt/stat453-deep-learning-ss20/blob/master/L13-cnns-part2/code/resnet-152.ipynb>

(Can be substantially improved with more hyperparameter tuning)



# 5. Inception-v1/ GoogLeNet (2014)



# 5. Inception-v1 / GoogLeNet (2014)

*"In this paper, we will focus on an efficient deep neural network architecture for computer vision, codenamed Inception, which derives its name from the Network in network paper by Lin et al [12] in conjunction with the famous "we need to go deeper" internet meme"*

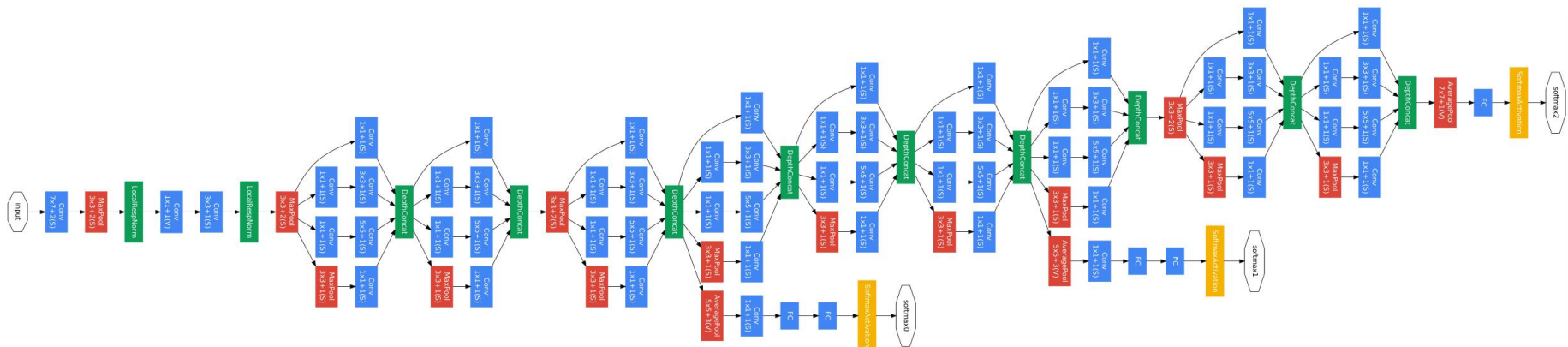


Szegedy, Christian, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. "[Going deeper with convolutions.](#)" In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1-9. 2015.



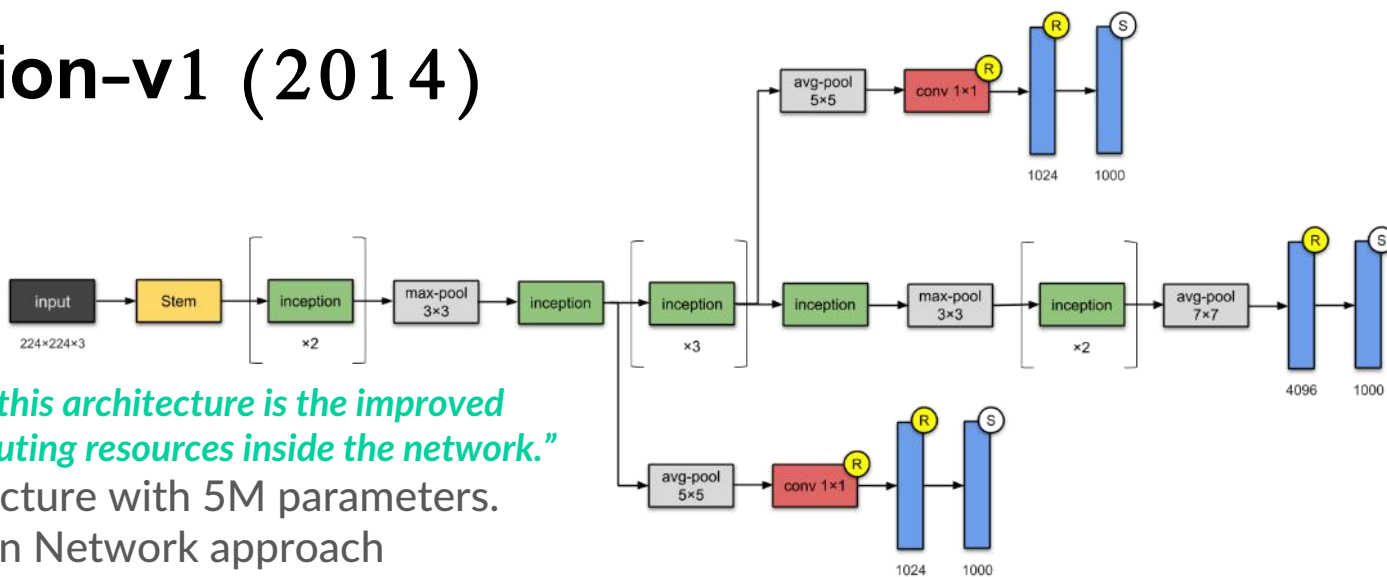
# 5. Inception-v1/ GoogLeNet (2014)

## Full architecture



Szegedy, Christian, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. "[Going deeper with convolutions.](#)" In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 1-9. 2015.

# 5. Inception-v1 (2014)



*“The main hallmark of this architecture is the improved utilisation of the computing resources inside the network.”*

- 22-layer architecture with 5M parameters.
- Used Network In Network approach
  - using ‘Inception modules’.

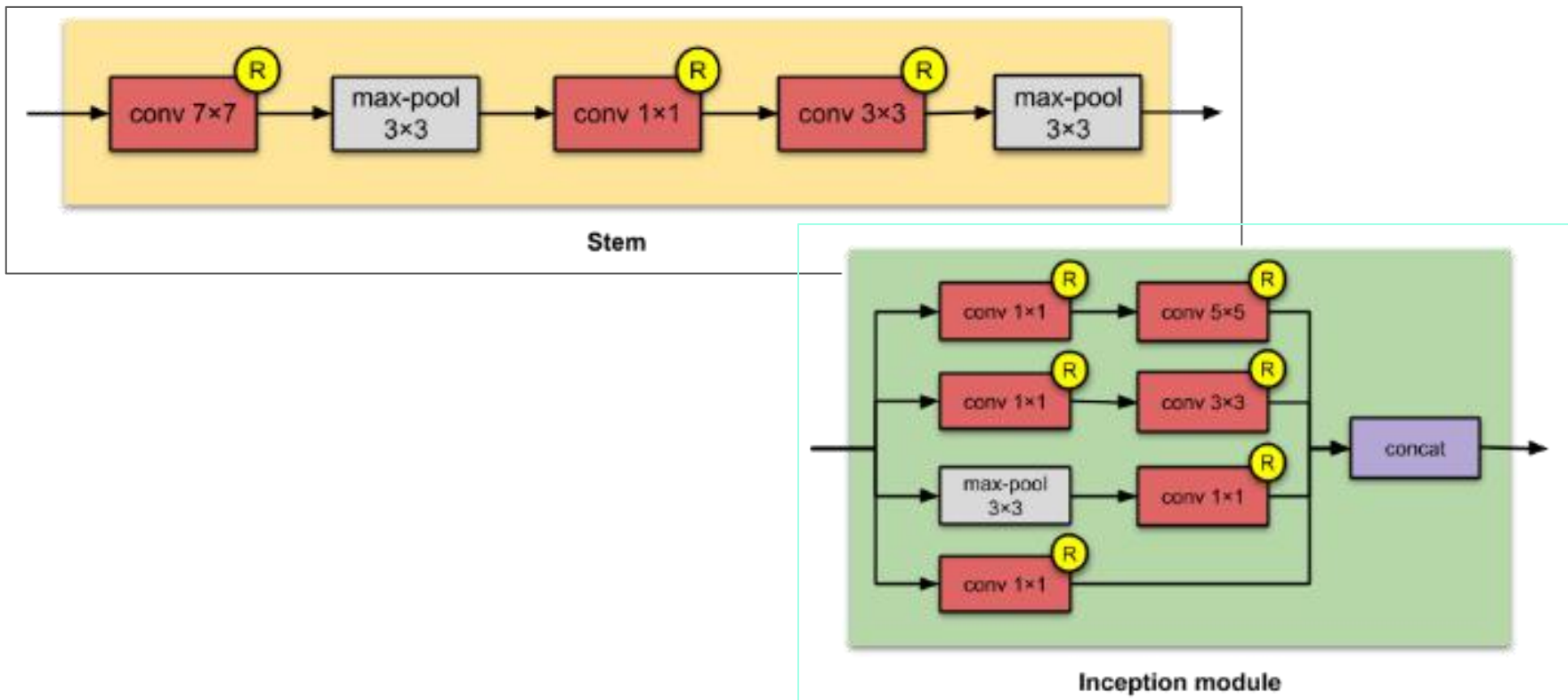
Each module presents 3 ideas:

- 1) Parallel towers of convs. with different filters, followed by concatenation
  - captures different features at 1x1, 3x3 and 5x5, ‘clustering’ them.
- 2) 1x1 convs. for dim. reduction (avoid bottlenecks).
- 3) Two auxiliary classifiers
  - discarded at inference time.

[paper](#). Going Deeper with Convolutions. Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich. Google, University of Michigan, University of North Carolina. 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)

- Dense modules/blocks instead of stacking convolutional layers.
- Curiosity: name Inception (from movie).

# 5. Inception-v1 (2014)



# Appendix: Network In Network (2014)

- Recalling from convolution:
  - A pixel is a linear combination of the weights in a filter and the current sliding window.
  - NiN proposes a mini neural network with 1 hidden layer instead of this linear combination.
- **One hidden layer network in a CNN.**
  - a.k.a MLPconv is the same as  $1 \times 1$  convolutions
  - Main feature for **Inception** architectures.

Paper: Network In Network. Min Lin, Qiang Chen, Shuicheng Yan. National University of Singapore. arXiv preprint, 2013.

- MLP convolutional layers,  $1 \times 1$  convolutions
- Global average pooling (taking average of each feature map, and feeding the resulting vector into the softmax layer)



# Lecture Overview



1. Padding (control output size in addition to stride)
2. Spatial Dropout and BatchNorm
3. Considerations for CNNs on GPUs
4. Common Architectures
  - LeNet-5
  - AlexNet
  - VGG-16
  - ResNet-50
  - Inception-v1
- 5. Transfer learning**

# Transfer Learning

- Key idea:
- ✦ Feature extraction layers may be generally useful
- ✦ Use a pre-trained model (e.g., pre-trained on ImageNet)
- ✦ Freeze the weights: Only train last layer (or last few layers)
- Related approach: Fine-tuning, train a pre-trained network on your smaller dataset

# Example 3 - Feature Extractor (I)

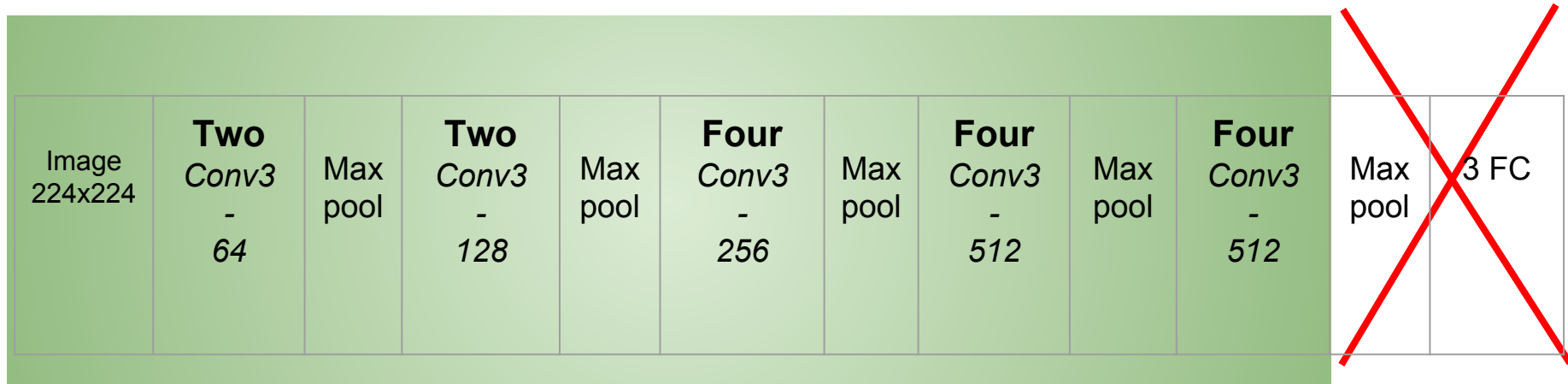
- Pre-trained VGG19 model

Image 224x224	<b>Two</b> <i>Conv3</i> - 64	Max pool	<b>Two</b> <i>Conv3</i> - 128	Max pool	<b>Four</b> <i>Conv3</i> - 256	Max pool	<b>Four</b> <i>Conv3</i> - 512	Max pool	<b>Four</b> <i>Conv3</i> - 512	Max pool	<del>3 FC</del>
------------------	---------------------------------------	-------------	--	-------------	---	-------------	---	-------------	---	-------------	-----------------



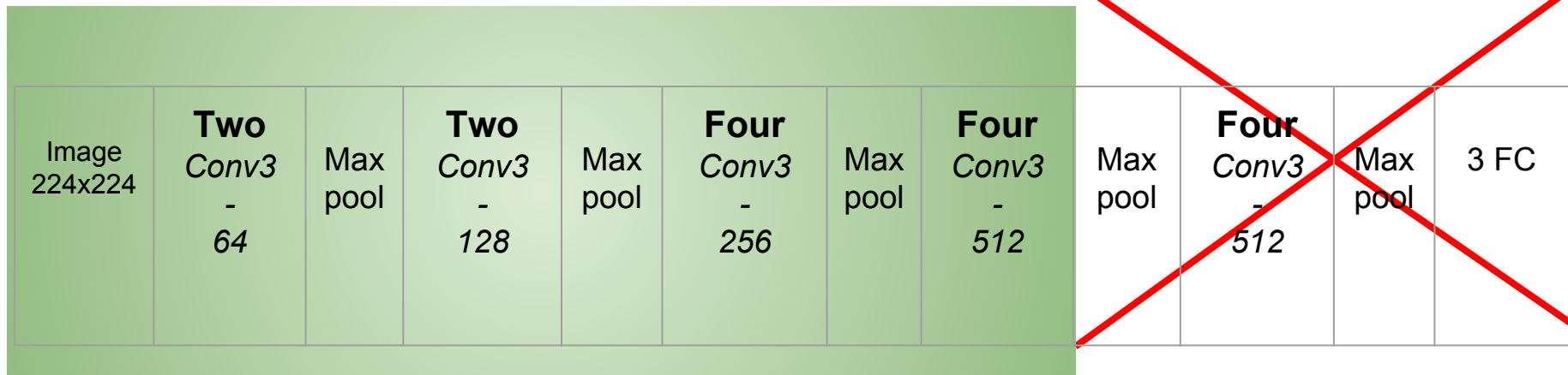
# Example 3 - Feature Extractor (II)

- Pre-trained VGG19 model



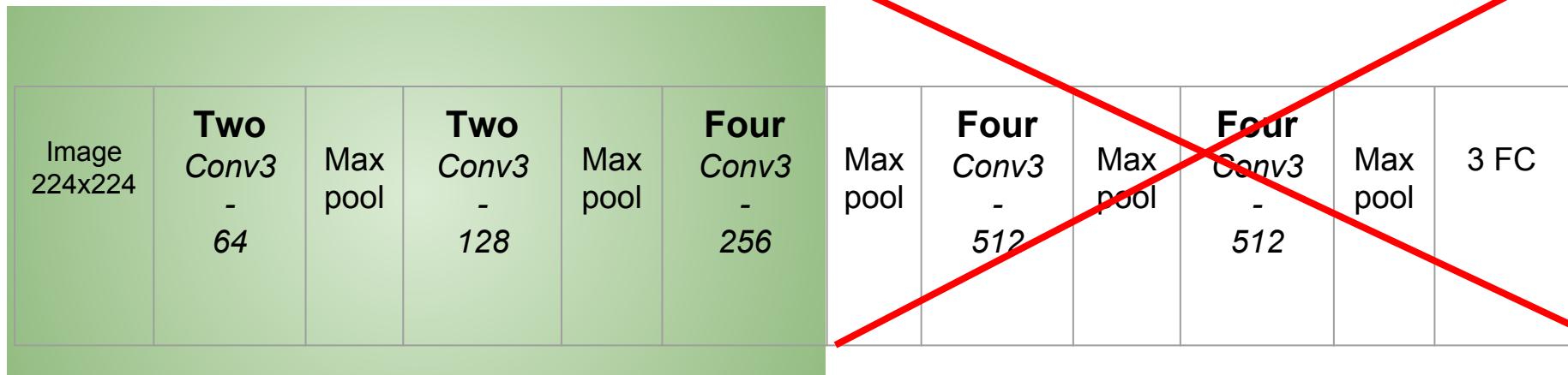
# Example 3 - Feature Extractor (III)

- Pre-trained VGG19 model



# Example 3 - Feature Extractor (IV)

- Pre-trained VGG19 model



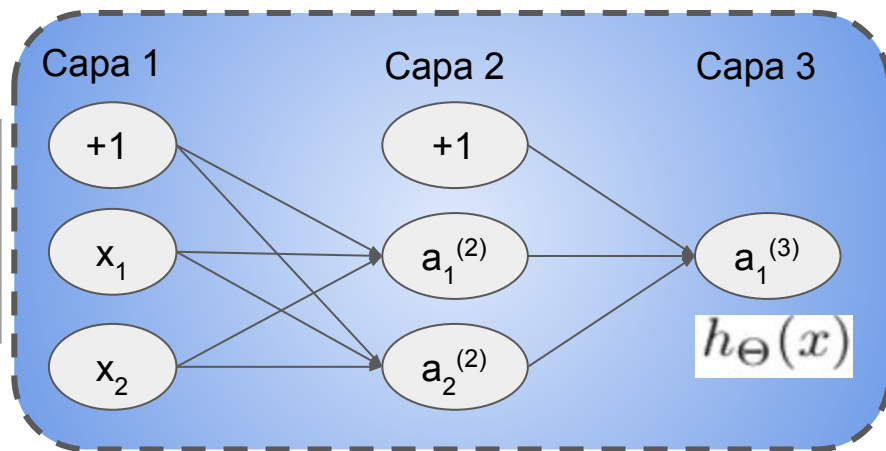
Etc .....

# Example 3 - Feature Extractor (V)

- Pre-trained VGG19 model

Image 224x224	<b>Two</b> <i>Conv3</i> - 64	Max pool	<b>Two</b> <i>Conv3</i> - 128	Max pool	<b>Four</b> <i>Conv3</i> - 256
------------------	---------------------------------------	-------------	--	-------------	---

Etc .....



# Which Layers to Replace & Train

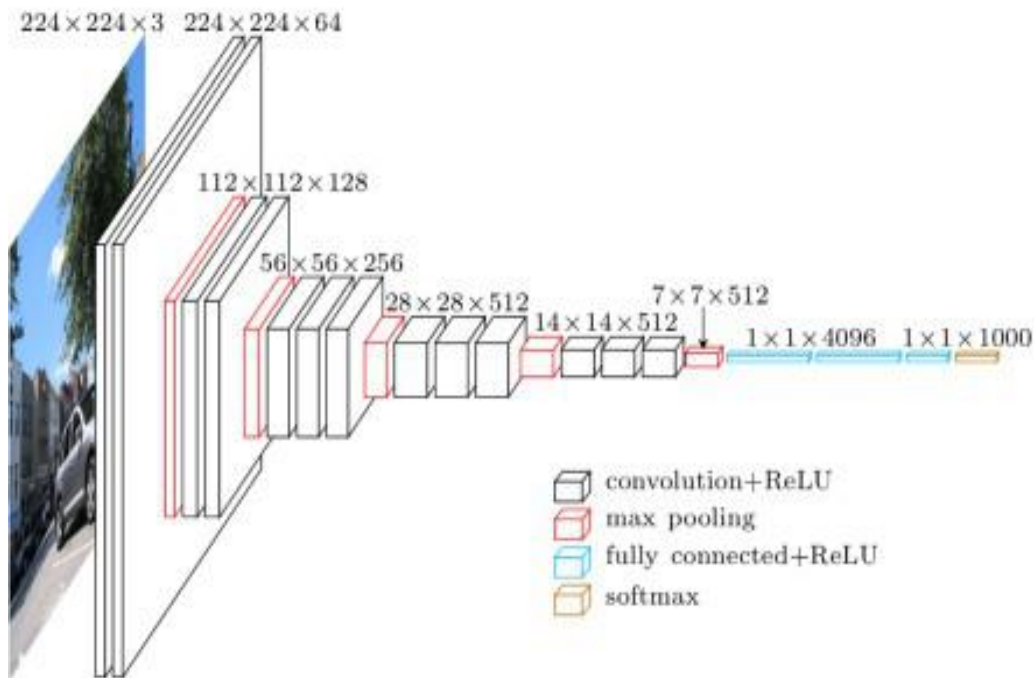
Karpathy, A., Toderici, G., Shetty, S., Leung, T., Sukthankar, R., & Fei-Fei, L. (2014). [Large-scale video classification with convolutional neural networks](#). In Proceedings of the IEEE conference on Computer Vision and Pattern Recognition (pp. 1725-1732).

<b>Model</b>	<b>3-fold Accuracy</b>
Soomro et al [22]	43.9%
Feature Histograms + Neural Net	59.0%
Train from scratch	41.3%
Fine-tune top layer	64.1%
Fine-tune top 3 layers	<b>65.4%</b>
Fine-tune all layers	62.2%

Table 3: Results on UCF-101 for various Transfer Learning approaches using the Slow Fusion network.

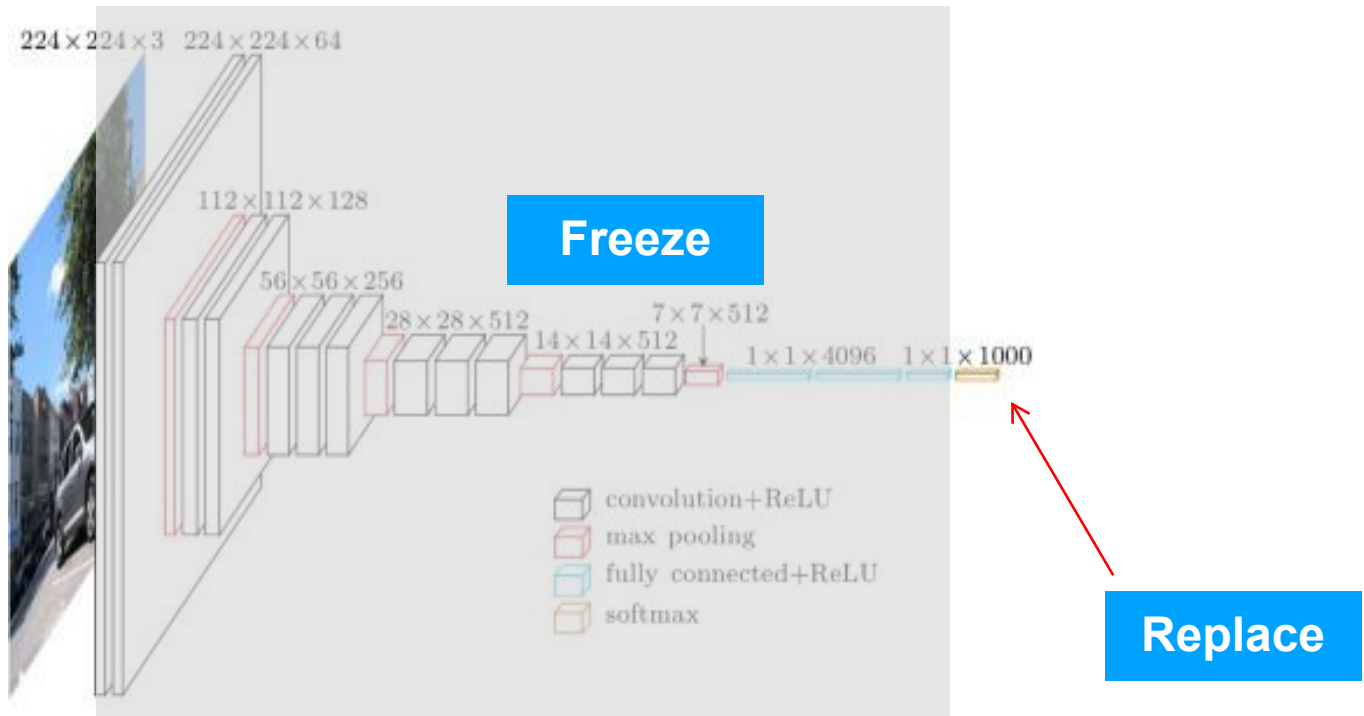
# Transfer Learning

PyTorch implementation: <https://github.com/rasbt/stat453-deep-learning-ss20/blob/master/L13-cnns-part2/code/vgg16-transferlearning.ipynb>



# Transfer Learning

PyTorch implementation: <https://github.com/rasbt/stat453-deep-learning-ss20/blob/master/L13-cnns-part2/code/vgg16-transferlearning.ipynb>



# Transfer Learning

<https://pytorch.org/docs/stable/torchvision/models.html>

## TORCHVISION.MODELS

The models subpackage contains definitions of models for addressing different tasks, including: image classification, pixelwise semantic segmentation, object detection, instance segmentation, person keypoint detection and video classification.

### Classification

The models subpackage contains definitions for the following model architectures for image classification:

- AlexNet
- VGG
- ResNet
- SqueezeNet
- DenseNet
- Inception v3
- GoogLeNet
- ShuffleNet v2
- MobileNet v2
- ResNeXt
- Wide ResNet
- MNASNet



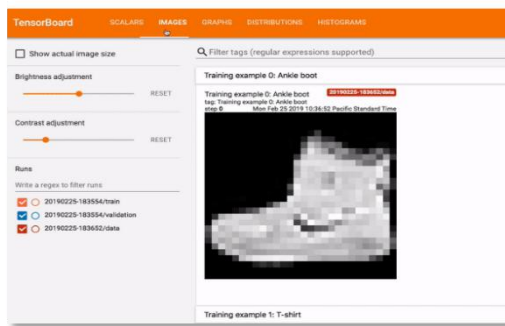
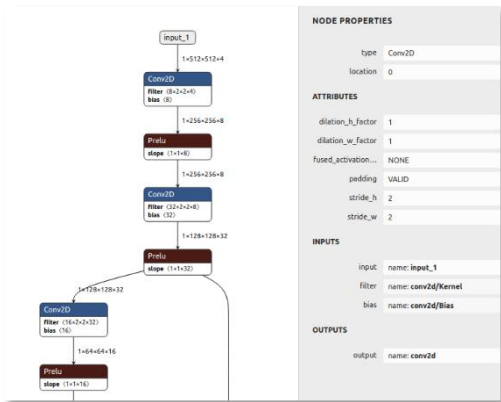
# Extra: Useful tools to visualize DL architectures

- [Netron](#)
- [Tensorboard](#)
- [PyTorchViz](#)
- [plot\\_model](#) API by Keras

```
model = nn.Sequential()
model.add_module('W0', nn.Linear(8, 16))
model.add_module('tanh', nn.Tanh())
model.add_module('W1', nn.Linear(16, 1))

x = Variable(torch.randn(1,8))
y = model(x)

make_dot(y.mean(), params=dict(model.named_parameters()))
```



## Model plotting

`plot_model` function

```
tf.keras.utils.plot_model(
    model,
    to_file="model.png",
    show_shapes=False,
    show_dtype=False,
    show_layer_names=True,
    rankdir="TB",
    expand_nested=False,
    dpi=96,
)
```

Lecture 13

# Introduction to Convolutional Neural Networks Part 3

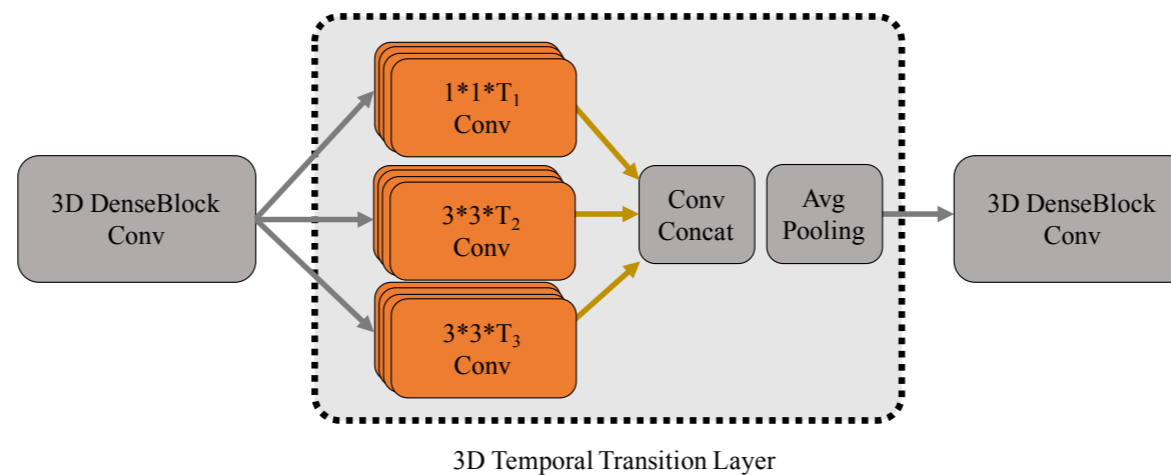
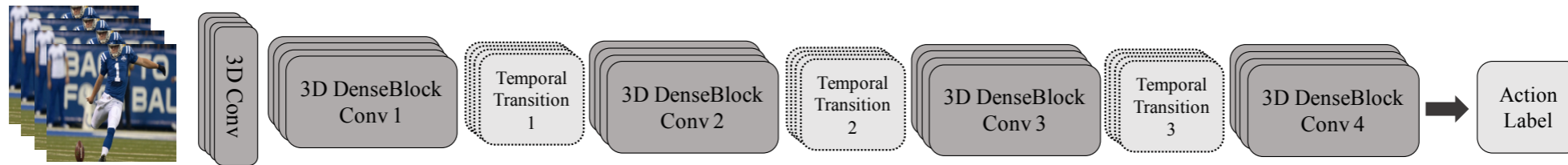
STAT 479: Deep Learning, Spring 2019

Sebastian Raschka

<http://stat.wisc.edu/~sraschka/teaching/stat479-ss2019/>

# Additional Concepts to Wrap Up the Intro to Convolutional Neural Networks

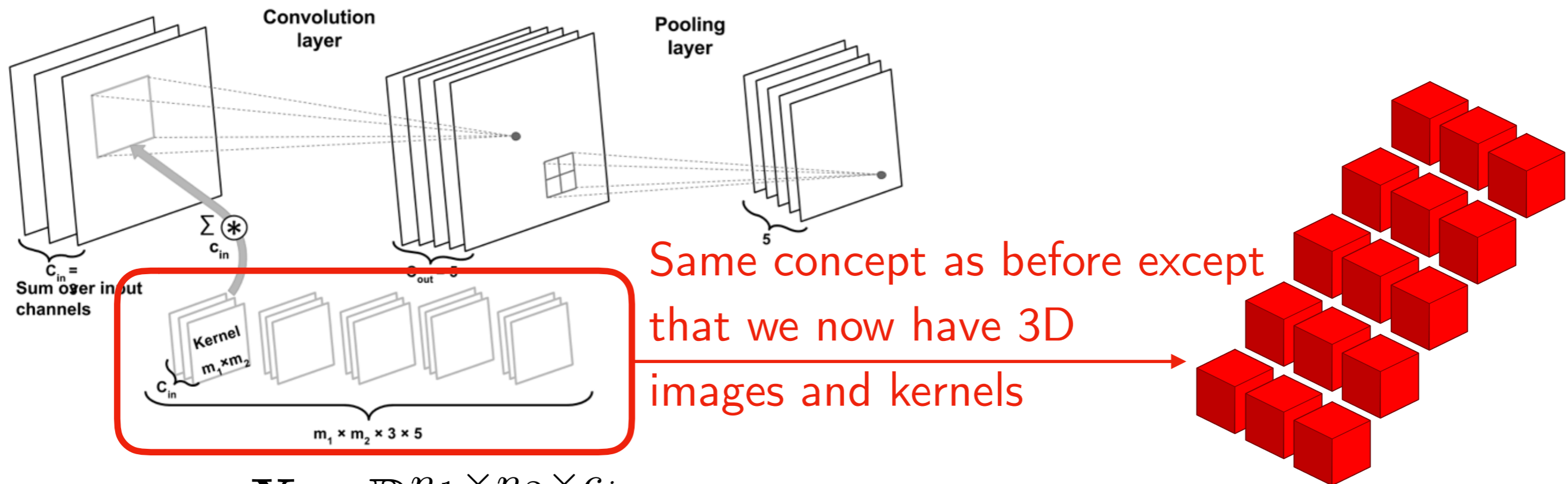
# ConvNets and 3D Inputs



Diba, Ali, Mohsen Fayyaz, Vivek Sharma, Amir Hossein Karami, Mohammad Mahdi Arzani, Rahman Yousefzadeh, and Luc Van Gool. "[Temporal 3d convnets: New architecture and transfer learning for video classification.](#)" *arXiv preprint arXiv: 1711.08200* (2017).

Also very popular for Medical Imaging (MRI, CT scans ...)

# ConvNets and 3D Inputs

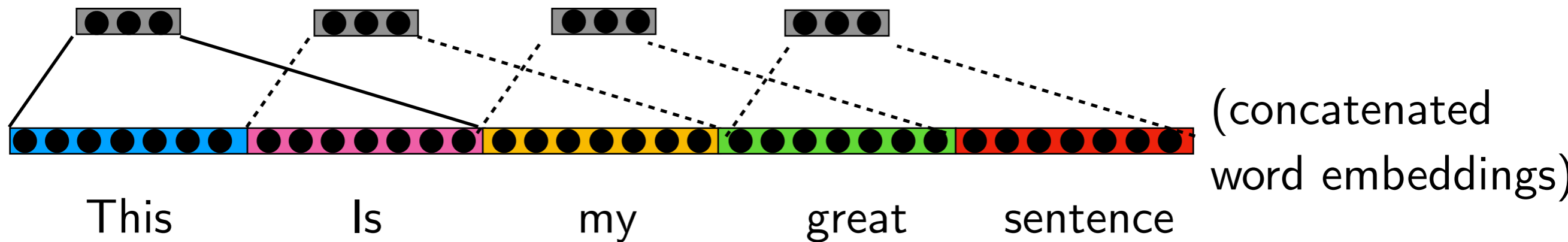


$$\mathbf{X} \in \mathbb{R}^{n_1 \times n_2 \times c_{in}}$$

$$\mathbf{W} \in \mathbb{R}^{m_1 \times m_2 \times c_{in} \times c_{out}} \quad \mathbf{b} \in \mathbb{R}^{c_{out}}$$

# ConvNets for Text with 1D Convolutions

We can think of text as image with width 1



<https://pytorch.org/docs/stable/nn.html#conv1d>

Conv1d

```
CLASS torch.nn.Conv1d(in_channels, out_channels, kernel_size, stride=1,  
padding=0, dilation=1, groups=1, bias=True)
```

[SOURCE]

Applies a 1D convolution over an input signal composed of several input planes.

# CNNs for Text (with 2D Convolutions)

Good results have also been achieved by representing a sentence as a matrix of word vectors and applying 2D convolutions (where each filter uses a different kernel size)

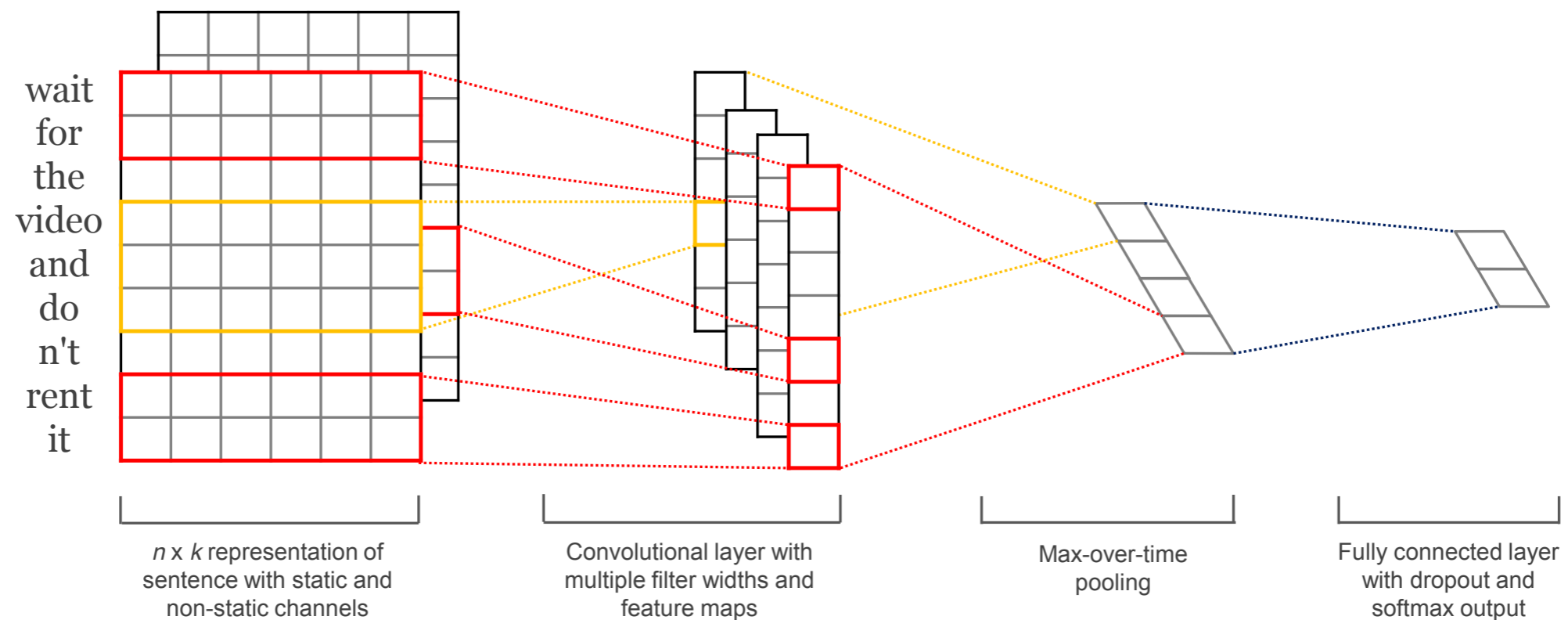


Figure 1: Model architecture with two channels for an example sentence.

Kim, Y. (2014). [Convolutional neural networks for sentence classification](#). *arXiv preprint arXiv:1408.5882*.

# Pre-Trained Models for Text

<https://modelzoo.co/model/pytorch-nlp>



# Lecture 14

# Introduction to Recurrent Neural Networks

STAT 453: Deep Learning, Spring 2020

Sebastian Raschka

<http://stat.wisc.edu/~sraschka/teaching/stat453-ss2020/>

Lecture Slides:

<https://github.com/rasbt/stat453-deep-learning-ss20/tree/master/L14-rnns>

# A Classic Approach for Text Classification: Bag-of-Words Model

"Raw" training dataset

$\mathbf{x}^{[1]}$  = "The sun is shining"

$\mathbf{x}^{[2]}$  = "The weather is sweet"

$\mathbf{x}^{[3]}$  = "The sun is shining,  
the weather is sweet, and  
one and one is two"

vocabulary = {  
  'and': 0,  
  'is': 1  
  'one': 2,  
  'shining': 3,  
  'sun': 4,  
  'sweet': 5,  
  'the': 6,  
  'two': 7,  
  'weather': 8,  
}

Training set as design matrix

$$\mathbf{X} = \begin{bmatrix} 0 & 1 & 0 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 2 & 3 & 2 & 1 & 1 & 1 & 2 & 1 & 1 \end{bmatrix}$$

$$\mathbf{y} = [0, 1, 0]$$

class labels

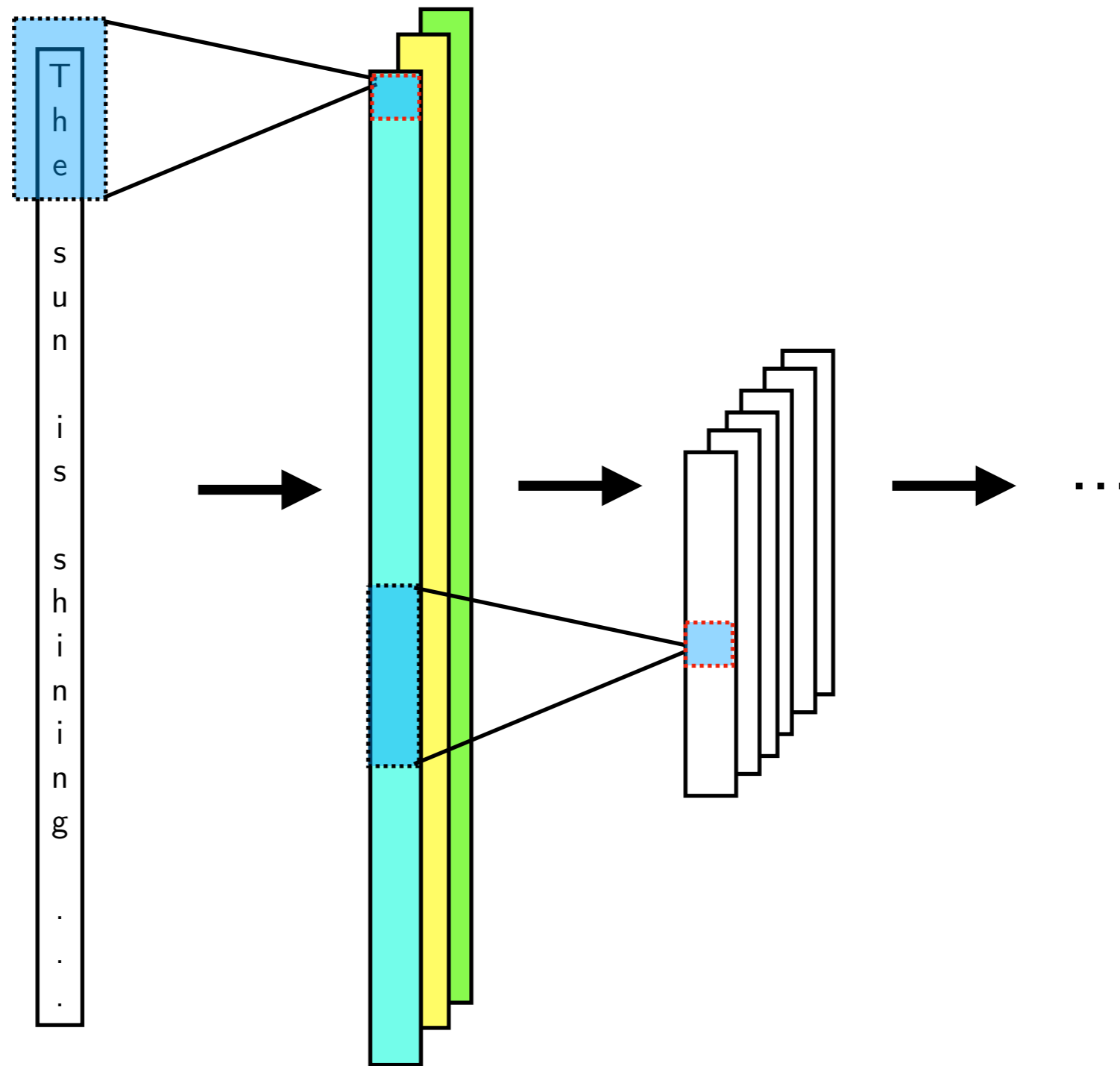
training

Classifier

(e.g., logistic regression, MLP, ...)

Ex.: <https://github.com/rasbt/python-machine-learning-book-3rd-edition/tree/master/ch08>

# 1D CNNs for text (and other sequence data)



# Lecture Overview

## **RNNs and Sequence Modeling Tasks**

Backpropagation Through Time

Long-short term memory (LSTM)

Many-to-one Word RNNs

Generating Text with Character RNNs

Attention Mechanisms and Transformers

# Sequential data is not i.i.d.

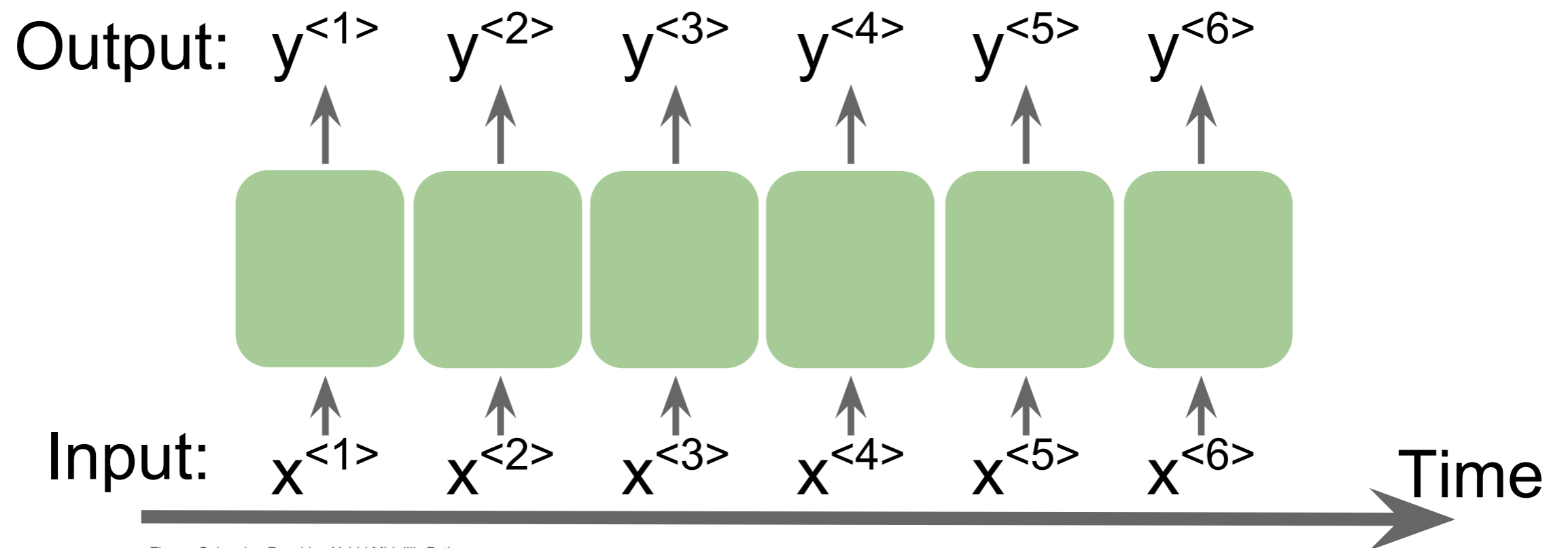
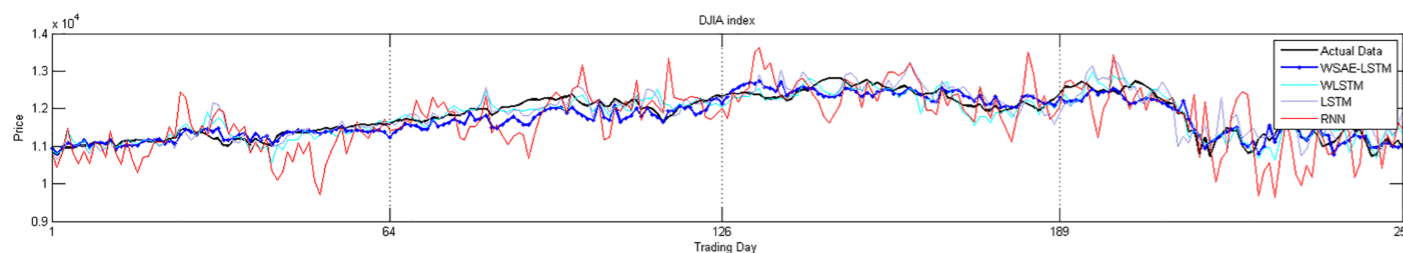


Figure: Sebastian Raschka, Vahid Mirjalili. *Python Machine Learning, 3rd Edition*. Birmingham, UK: Packt Publishing, 2019

# Applications: Working with Sequential Data

- Text classification
- Speech recognition (acoustic modeling)
- language translation
- ...

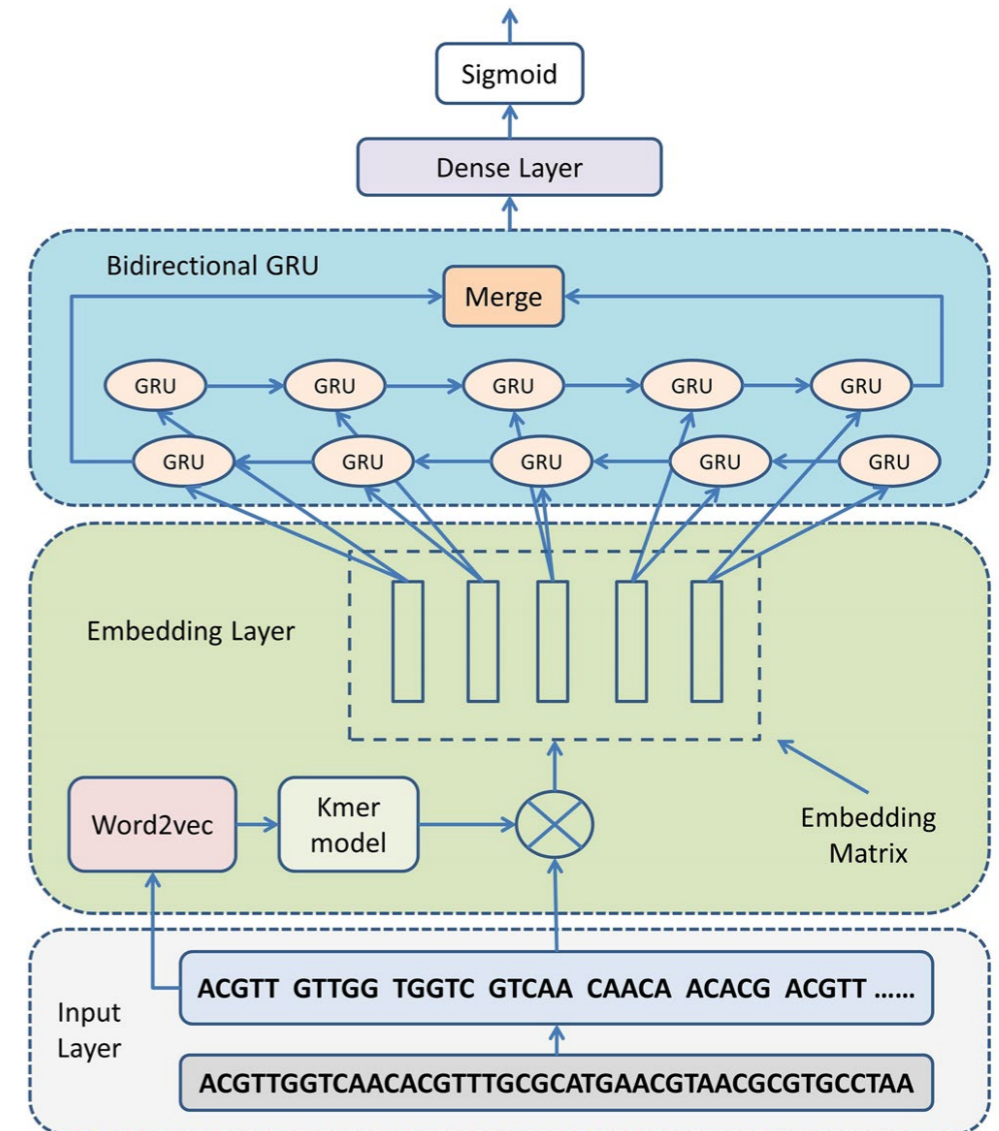
## Stock market predictions



**Fig 8.** Displays the actual data and the predicted data from the four models for each stock index in Year 1 from 2010.10.01 to 2011.09.30.

<https://doi.org/10.1371/journal.pone.0180944.g008>

Bao, Wei, Jun Yue, and Yulei Rao. "A deep learning framework for financial time series using stacked autoencoders and long-short term memory." *PloS one* 12, no. 7 (2017): e0180944.



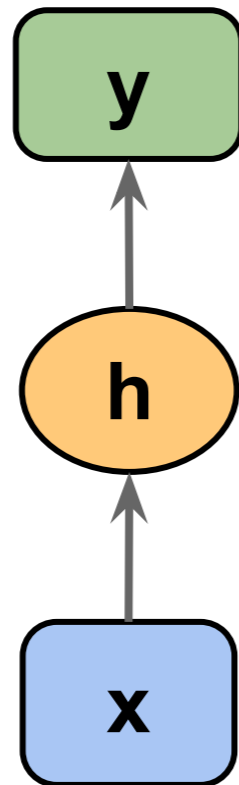
Shen, Zhen, Wenzheng Bao, and De-Shuang Huang. "Recurrent Neural Network for Predicting Transcription Factor Binding Sites." *Scientific reports* 8, no. 1 (2018): 15270.

DNA or (amino acid/protein)  
sequence modeling

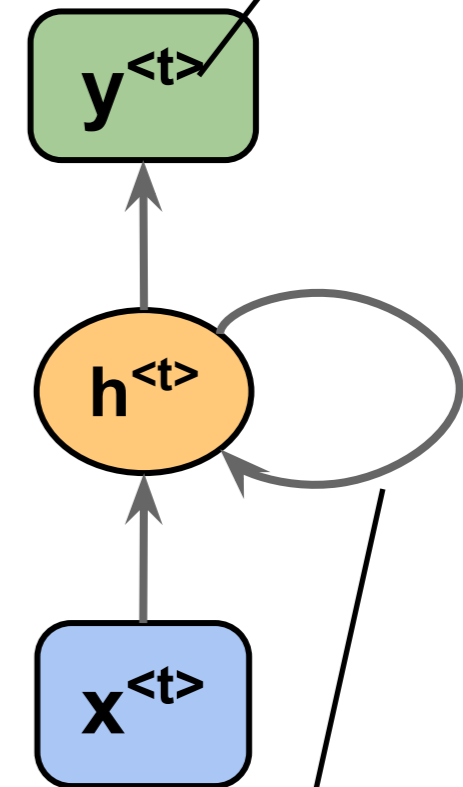
# Overview

time step  $t$

Networks we used previously: also called feedforward neural networks



Recurrent Neural Network (RNN)



Recurrent edge

Figure: Sebastian Raschka, Vahid Mirjalili. *Python Machine Learning, 3rd Edition*. Birmingham, UK: Packt Publishing, 2019

# Overview

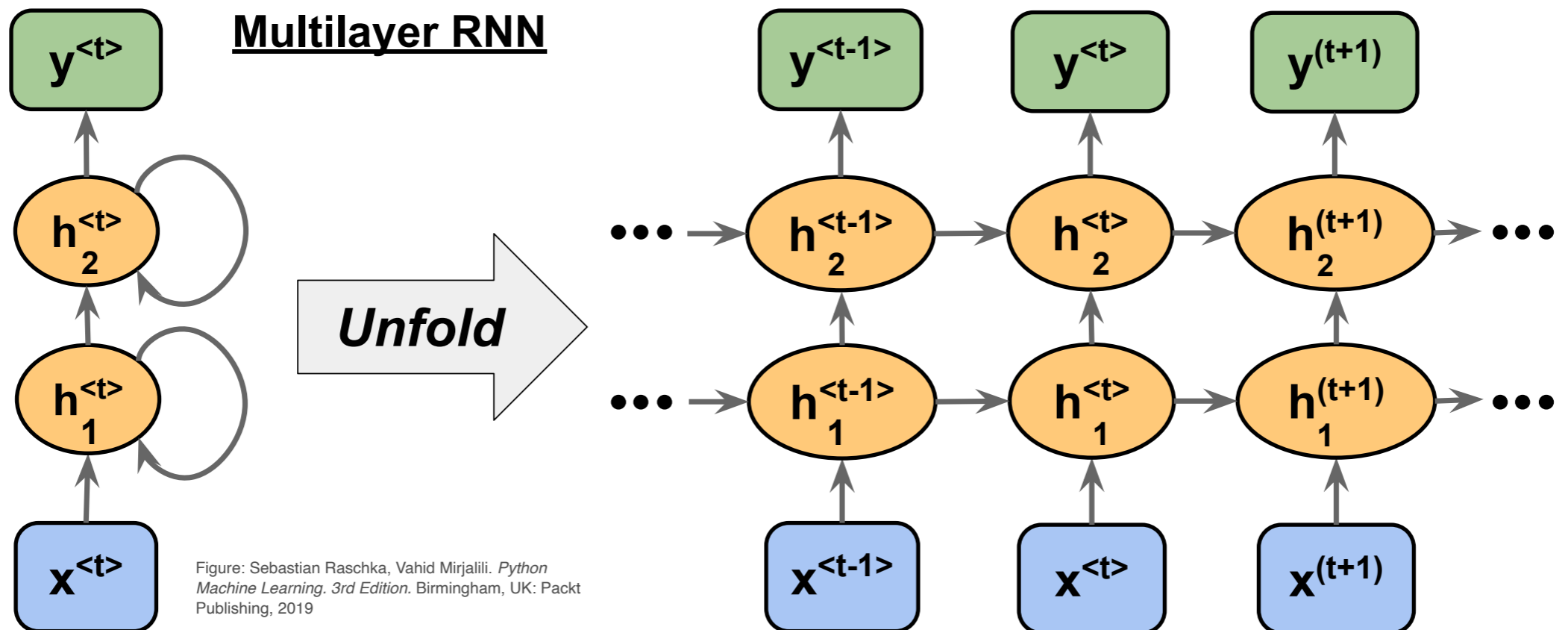
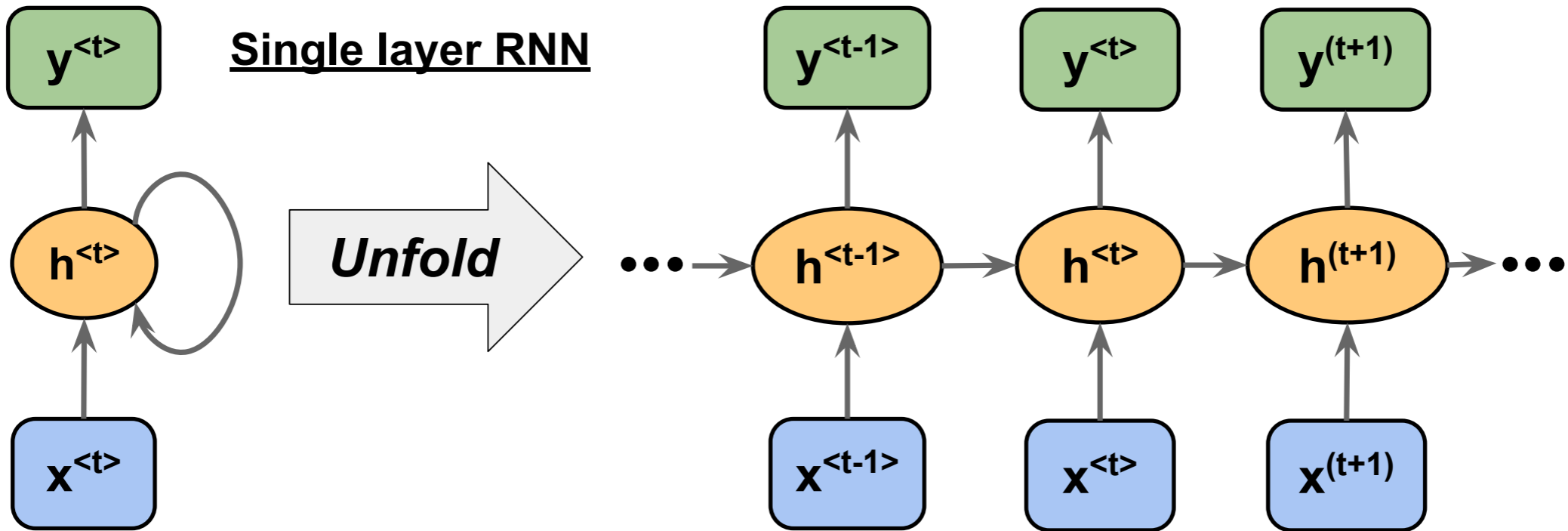


Figure: Sebastian Raschka, Vahid Mirjalili. *Python Machine Learning, 3rd Edition*. Birmingham, UK: Packt Publishing, 2019



# Different Types of Sequence Modeling Tasks

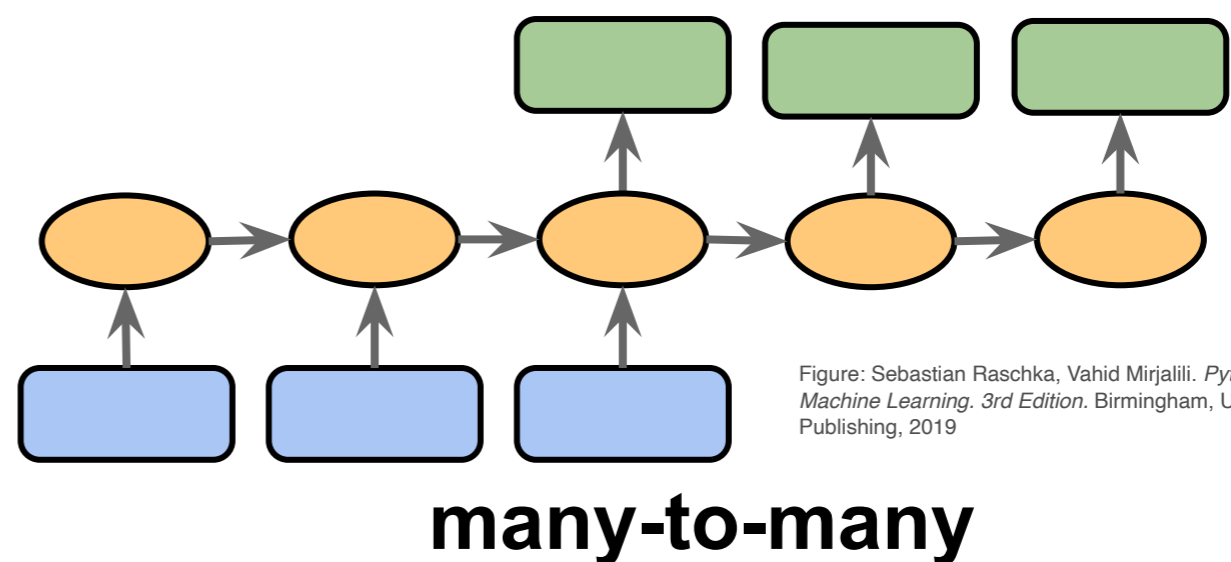
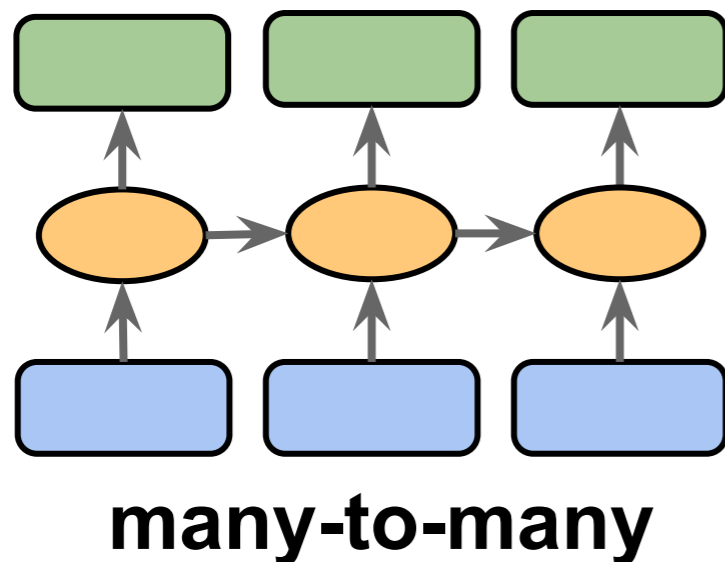
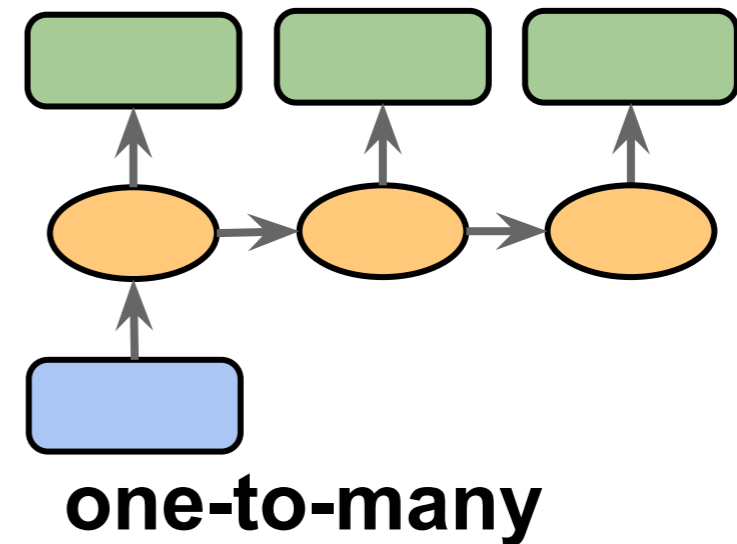
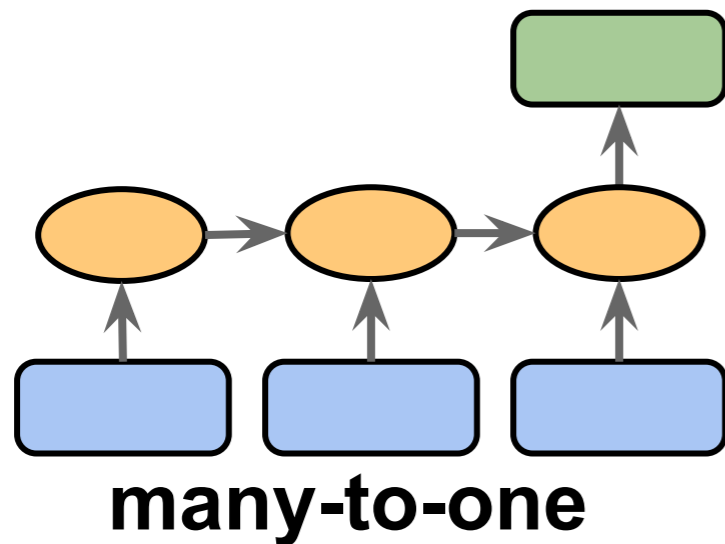
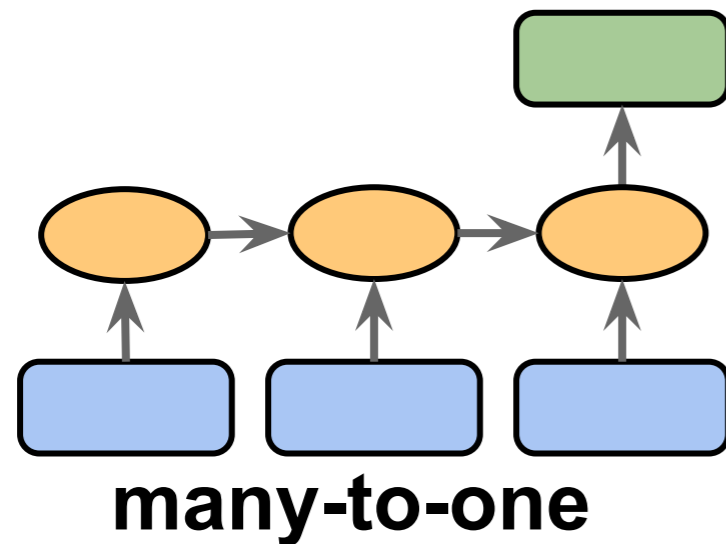


Figure: Sebastian Raschka, Vahid Mirjalili. *Python Machine Learning, 3rd Edition*. Birmingham, UK: Packt Publishing, 2019

Figure based on:

*The Unreasonable Effectiveness of Recurrent Neural Networks* by Andrej Karpathy (<http://karpathy.github.io/2015/05/21/rnn-effectiveness/>)

# Different Types of Sequence Modeling Tasks



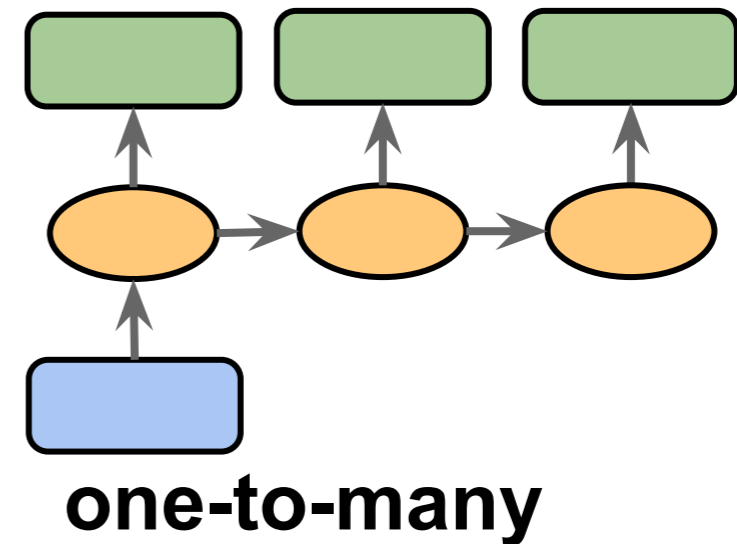
**Many-to-one:** The input data is a sequence, but the output is a fixed-size vector, not a sequence.

Ex.: sentiment analysis, the input is some text, and the output is a class label.

# Sentiment analysis



# Different Types of Sequence Modeling Tasks



**One-to-many:** Input data is in a standard format (not a sequence), the output is a sequence.

Ex.: Image captioning, where the input is an image, the output is a text description of that image

# Image captioning



(Train image 1) Caption -> The black cat sat on grass



(Train image 2) Caption -> The white cat is walking on road



(Test image) Caption -> The black cat is walking on grass



"man in black shirt is playing guitar."



"construction worker in orange safety vest is working on road."



"two young girls are playing with lego toy."

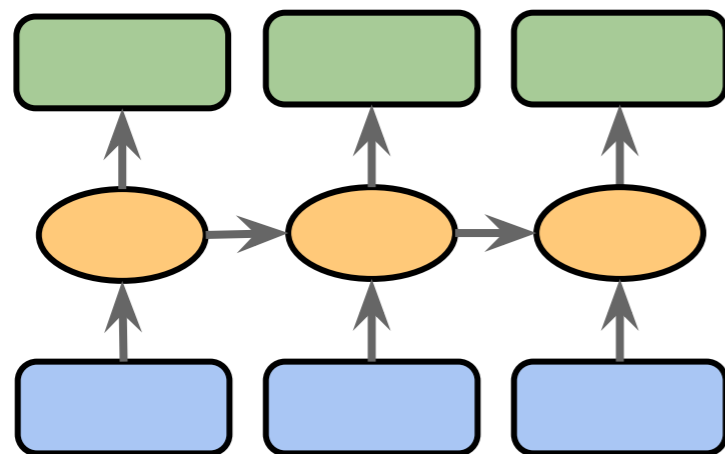
Image Captioning

# Different Types of Sequence Modeling Tasks

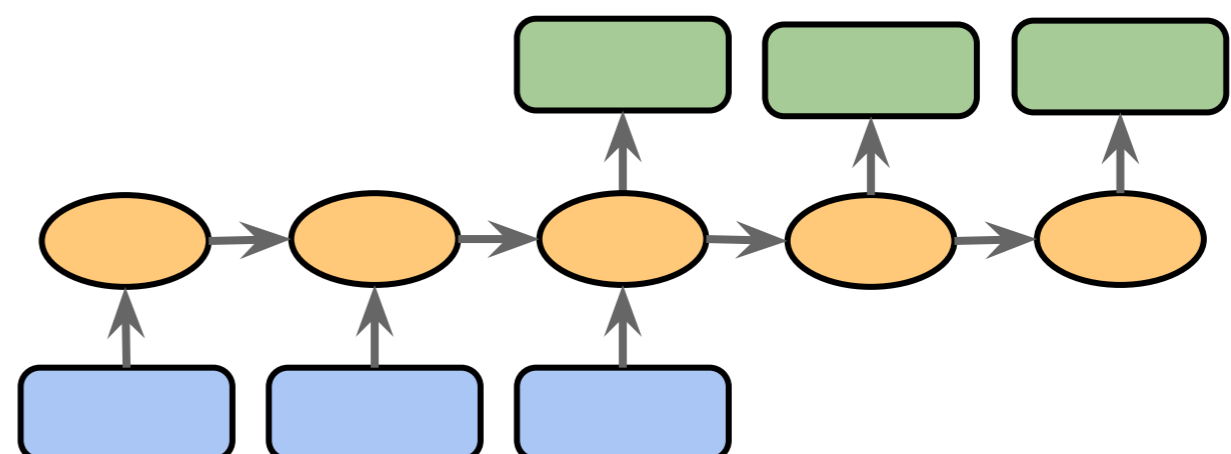
**Many-to-many:** Both inputs and outputs are sequences. Can be direct or delayed.

Ex.: Video-captioning, i.e., describing a sequence of images via text (direct).

Translating one language into another (delayed)



**many-to-many**



**many-to-many**

# Video captioning

*Video*



*Caption*

“Two boys are playing baseball in the ground”