STAT 992: Foundation Models for Biomedical Data

Ben Lengerich

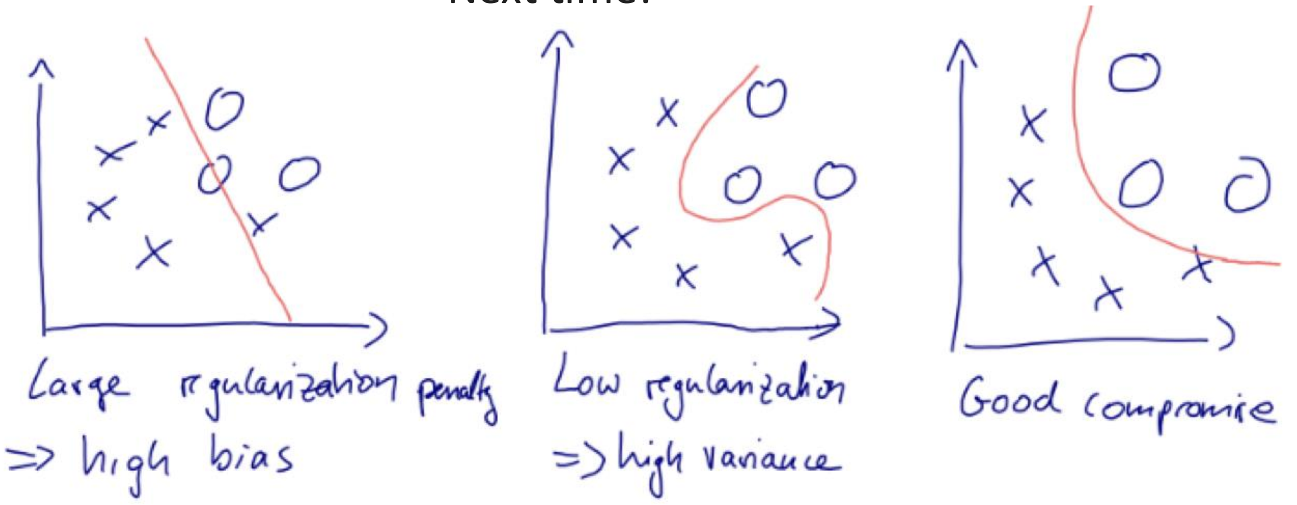
Lecture 04: Initialization and CNNs

Feb 02, 2026

Where we are…

- Good news: We can solve non-linear problems!
- Bad news: Our multilayer neural networks have lots of parameters and it's easy to overfit the data…

Next time:

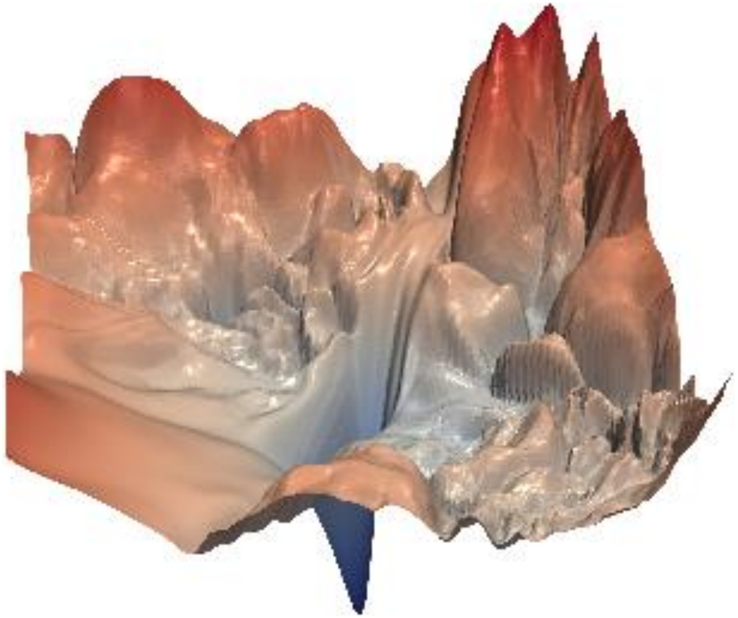


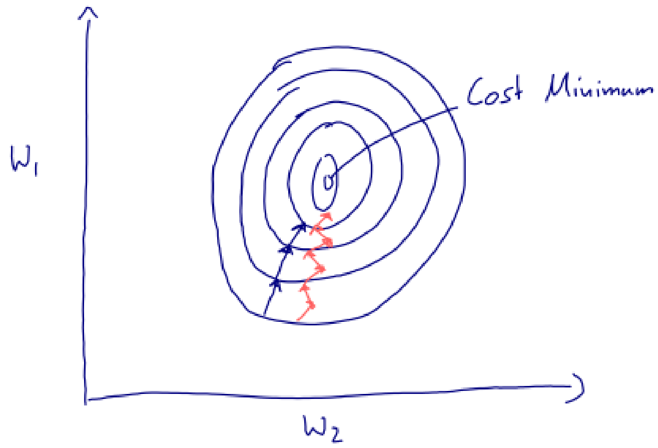
Last time…


Improving generalization
Dataset
Collecting more data
Data augmentation
Label smoothing
Leveraging unlabeled data
Semi-supervised
Self-supervised
Leveraging related data
Meta-learning
Transfer learning
Architecture setup
Weight initialization strategies
Activation functions
Residual layers
Knowledge distillation
Normalization
Input standardization
BatchNorm and variants
Weight standardization
Gradient centralization
Training loop
Adaptive learning rates
Auxiliary losses
Gradient clipping
Regularization
L2 (/L1) regularization
Early stopping
Dropout

Last time…

Complicated Loss Landscape → Improvements to Optimization





Mini-batch training

Momentum

Adaptive Training

$$w_{i,j} := w_{i,j} - \eta \frac{m_t}{\sqrt{r} + \epsilon}$$

Initialization

Weight initialization

- Recall: Can't initialize all weights to 0 (symmetry problem)
- But we want weights to be relatively small.
  - Traditionally, we can initialize weights by sampling from a random uniform distribution in range [0, 1], or better, [-0.5, 0.5]
  - Or, we could sample from a Gaussian distribution with mean 0 and small variance (e.g., 0.1 or 0.01)

Xavier Initialization

Method:

- **Step 1:** Initialize weights from Gaussian or uniform distribution
- **Step 2:** Scale the weights proportional to the number of inputs to the layer
  - For the first hidden layer, that is the number of features in the dataset; for the second hidden layer, that is the number of units in the 1st hidden layer, etc.

Xavier Glorot and Yoshua Bengio. "Understanding the difficulty of training deep feedforward neural networks." *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. 2010.

Xavier Initialization

<u>Rationale behind this scaling:</u>
Variance of the sample (between data points, not variance of the mean) linearly increases as the sample size increases (variance of the sum of independent variables is the sum of the variances); square root for standard deviation

$$\mathrm{Var}\left(z_j^{(l)}\right) = \mathrm{Var}\left(\sum_{j=1}^{m_{l-1}} W_{jk}^{(l)} a_k^{(l-1)}\right)$$

$$= \sum_{j=1}^{m^{(l-1)}} \mathrm{Var}\left[W_{jk}^{(l)} a_k^{(l-1)}\right] = \sum_{i=1}^{m^{(l-1)}} \mathrm{Var}\left[W_{jk}^{(l)}\right] \mathrm{Var}\left[a_k^{(l-1)}\right]$$

$$= \sum_{j=1}^{m^{(l-1)}} \mathrm{Var}\left[W^{(l)}\right] \mathrm{Var}\left[a^{(l-1)}\right] = m^{(l-1)} \mathrm{Var}\left[W^{(l)}\right] \mathrm{Var}\left[a^{(l-1)}\right]$$

He Initialization

- Assuming activations with mean 0, which is reasonable, Xavier Initialization assumes a derivative of 1 for the activation function (which is reasonable for tanH)
- For ReLU, the **activations are not centered at zero**
- He initialization takes this into account
- The result is that we add a scaling factor of $\sqrt{2}$

$$\mathbf{W}^{(l)} := \mathbf{W}^{(l)} \cdot \sqrt{\frac{2}{m^{(l-1)}}}$$

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification." In *Proceedings of the IEEE international conference on computer vision*, pp. 1026-1034. 2015.

Convolutional Neural Networks (CNNs)

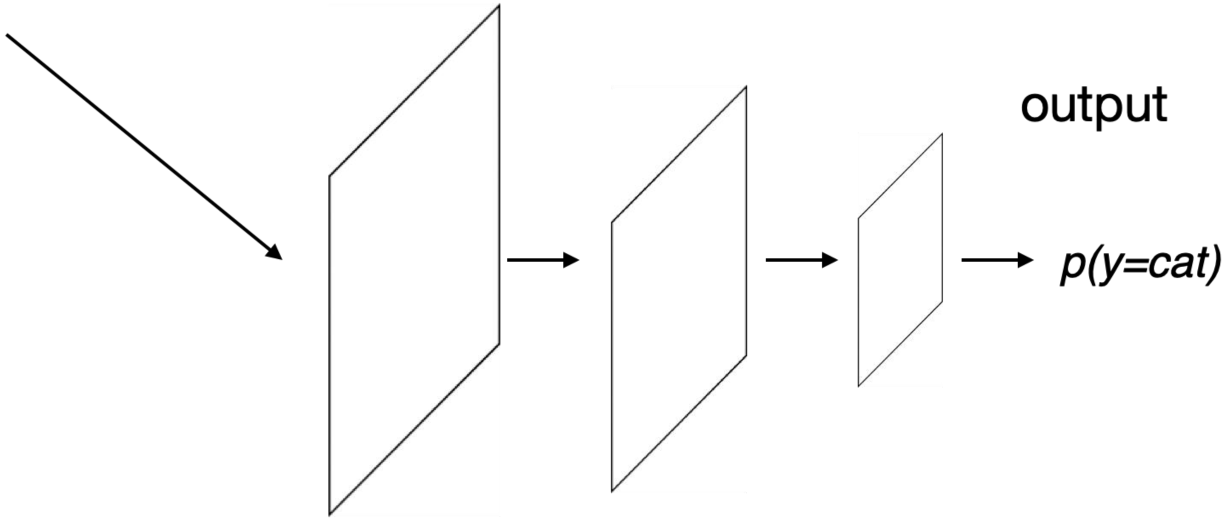
Today: CNNs

1. **What CNNs Can Do**
2. Image Classification
3. Convolutional Neural Network Basics
4. Cross-Correlation vs Convolution
5. CNNs & Backpropagation
6. CNNs in PyTorch

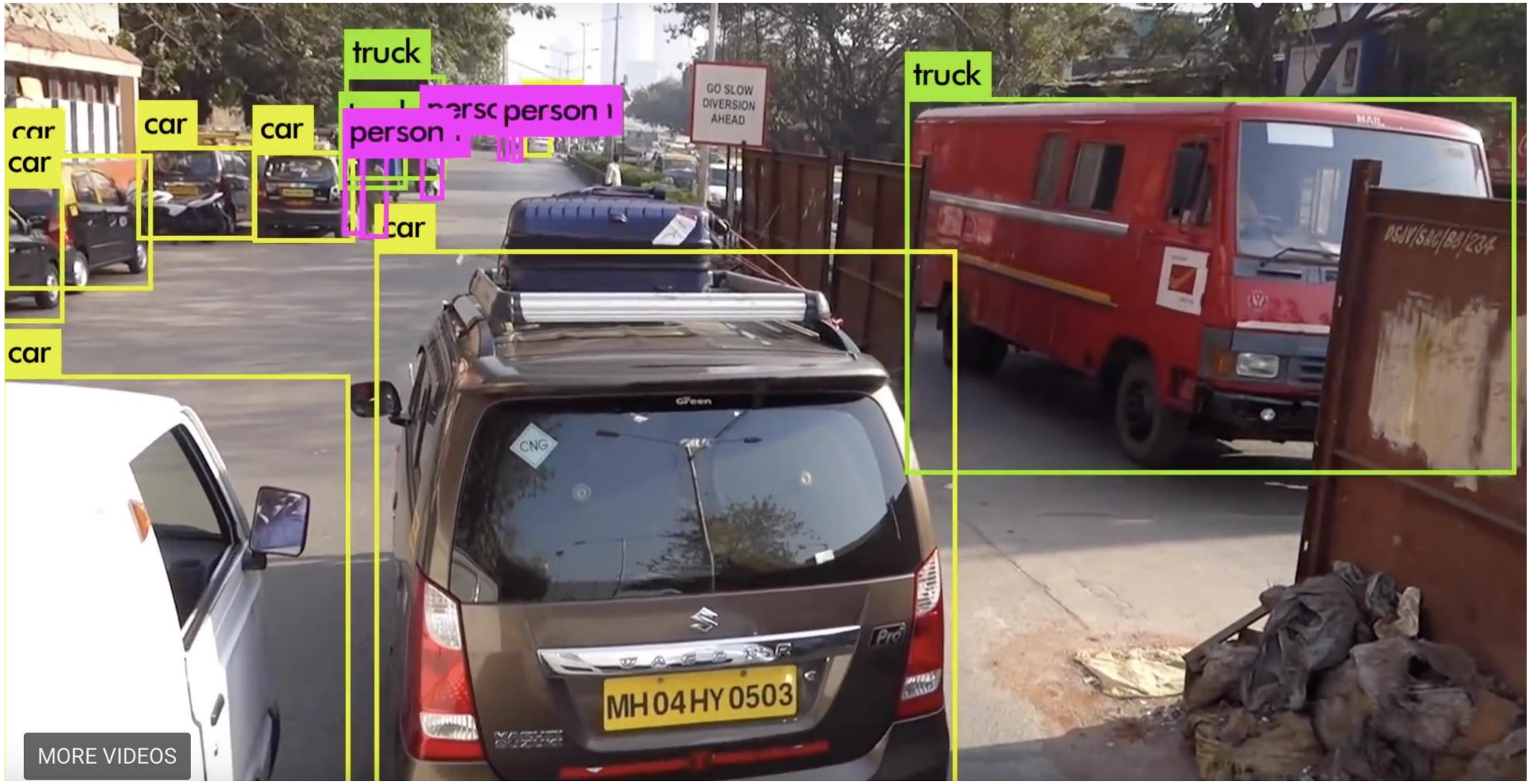
CNNs for Image Classification

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

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



CNNs for 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).

CNNs for Object Segmentation

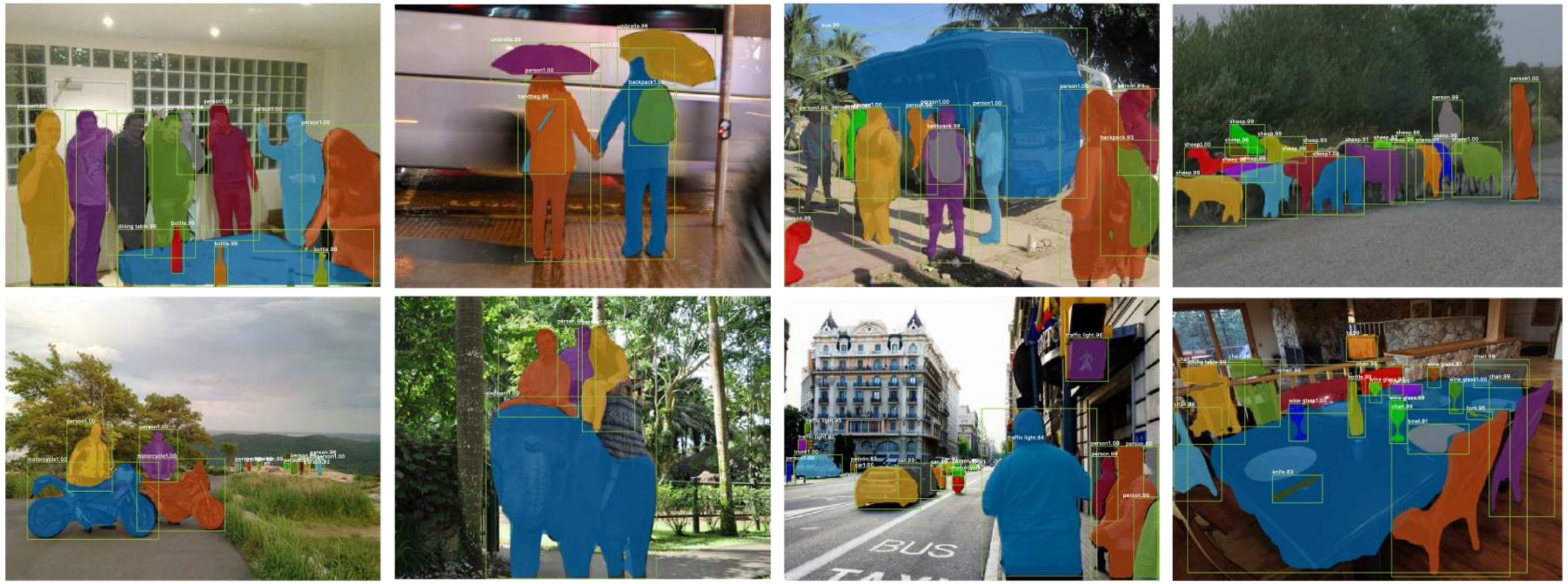


Figure 2. **Mask R-CNN** results on the COCO test set. These results are based on ResNet-101 [15], achieving a *mask* AP of 35.7 and running at 5 fps. Masks are shown in color, and bounding box, category, and confidences are also shown.

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.

Today: CNNs

1. What CNNs Can Do
2. **Image Classification**
3. Convolutional Neural Network Basics
4. Cross-Correlation vs Convolution
5. CNNs & Backpropagation
6. CNNs in PyTorch

Why images are hard

Different lighting, contrast, viewpoints, etc.

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

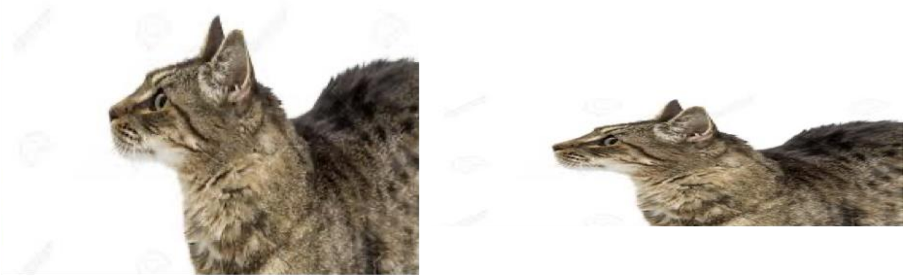
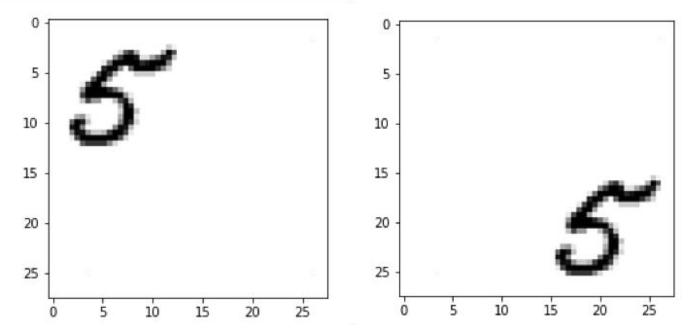


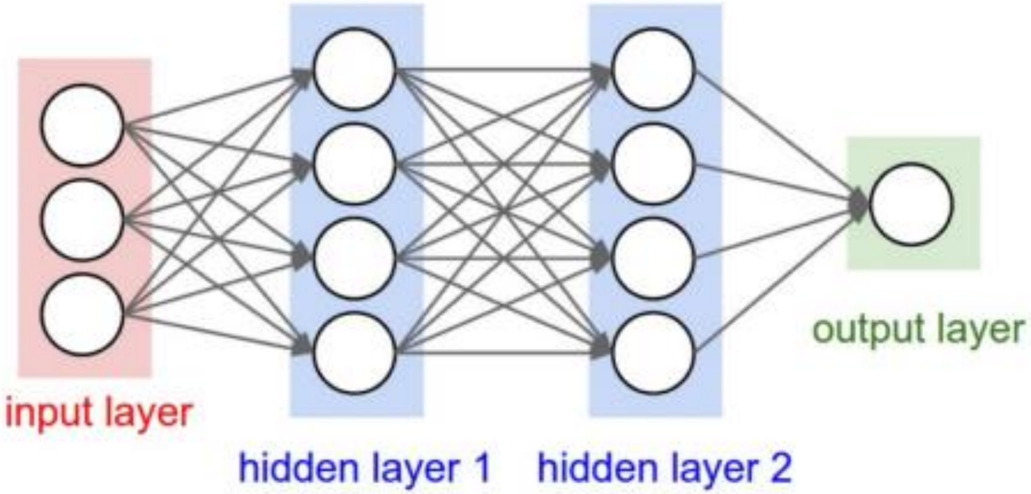
Image Source: https://www.123rf.com/photo_76714328_side-view-of-tabby-cat-face-over-white.html

Or even simple translation



Do deep fully-connected nets solve this?

Full connectivity is a problem for large inputs



- 3x200x200 images imply **120,000** weights per neuron in first hidden layer

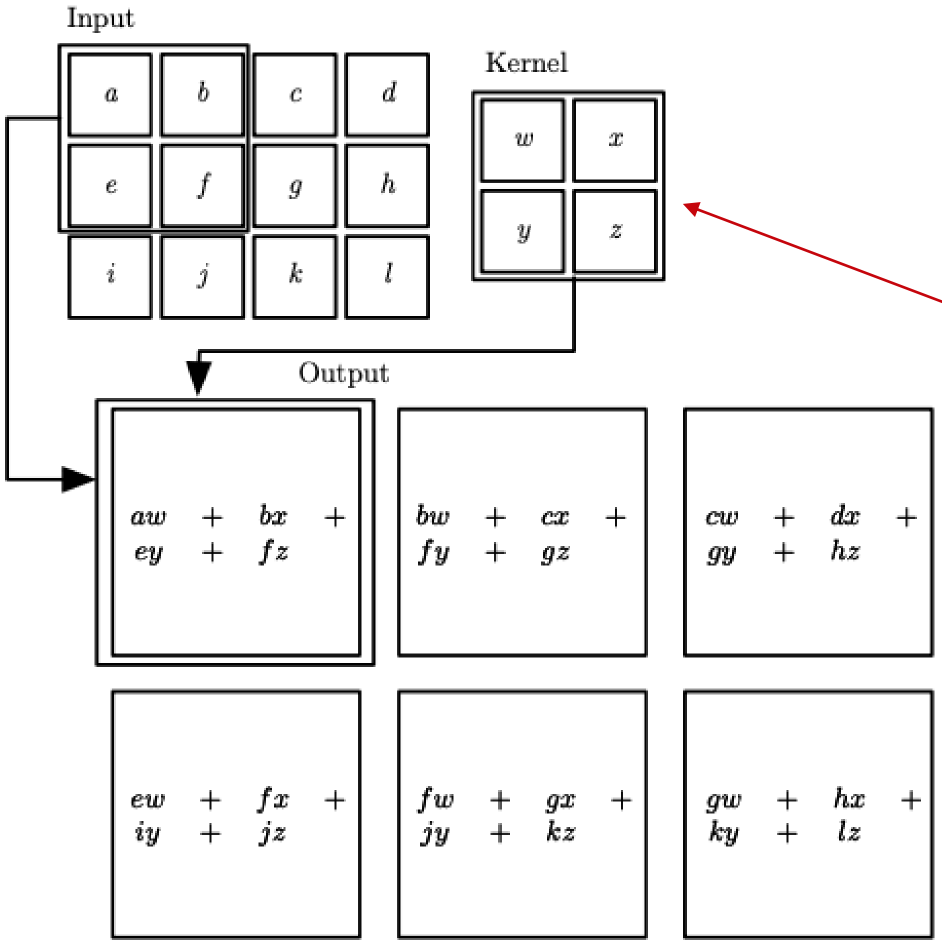
Today: CNNs

1. What CNNs Can Do
2. Image Classification
3. **Convolutional Neural Network Basics**
4. Cross-Correlation vs Convolution
5. CNNs & Backpropagation
6. CNNs in PyTorch

Convolutional Neural Networks [LeCun 1989]

- Let’s share parameters.
- Instead of learning position-specific weights, learn weights defined for **relative positions**
  - Learn “filters” that are reused across the image
  - Generalize across spatial translation of input
- Key idea:
  - Replace matrix multiplication in neural networks with a convolution
- Later, we will see that this can work for any graph-structured data, not just images.

Weight sharing in kernels



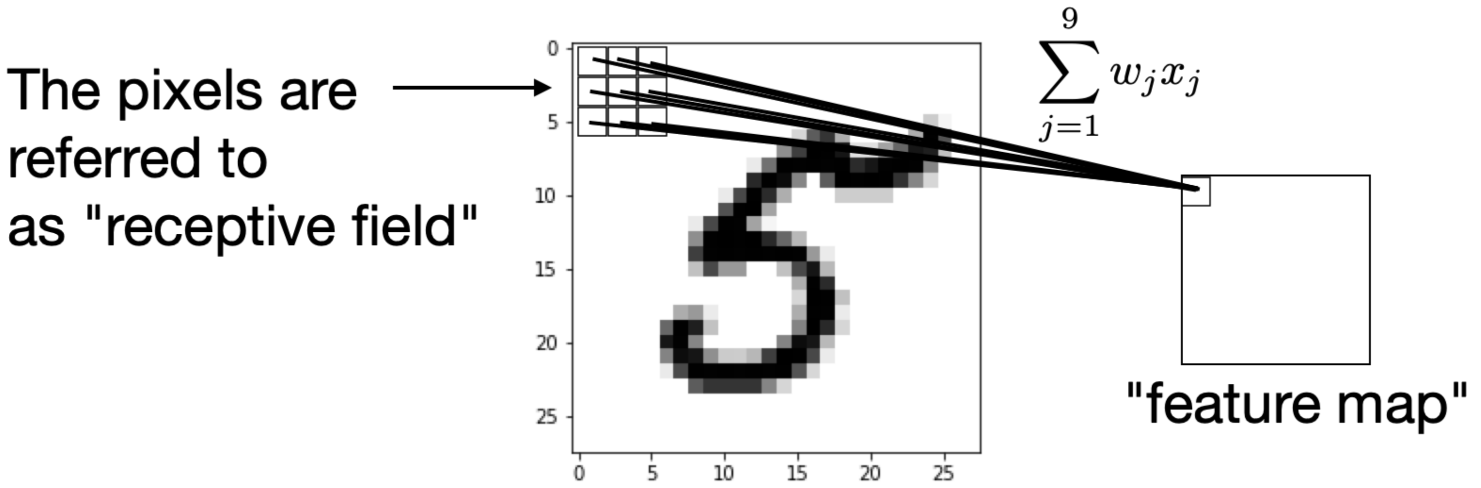
Sliding filters (kernels)

Reused weights (small)!

Fig. Goodfellow et al. 2016

Alternative visualization of kernels

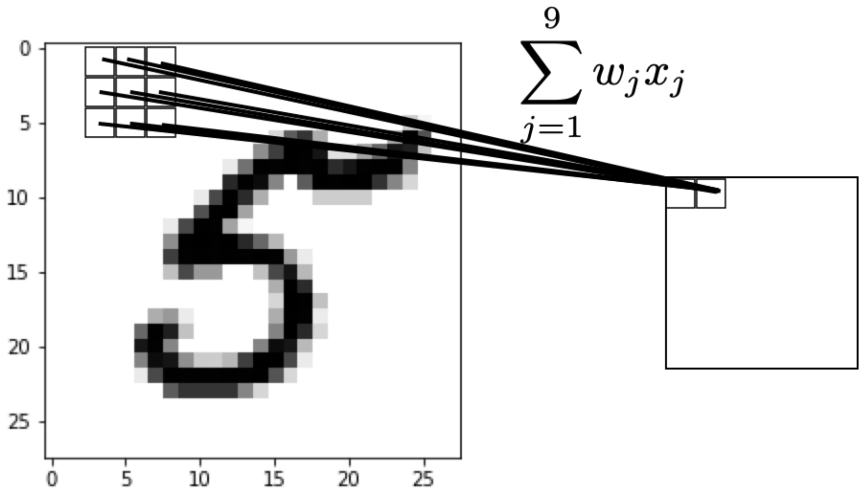
A "feature detector" (filter, kernel) slides over the inputs to generate a feature map



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

Alternative visualization of kernels

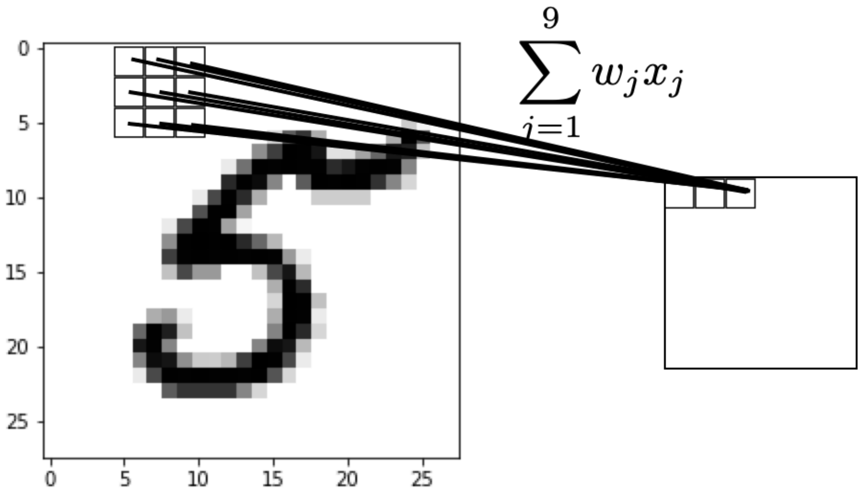
A "feature detector" (filter, kernel) slides over the inputs to generate a feature map



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

Alternative visualization of kernels

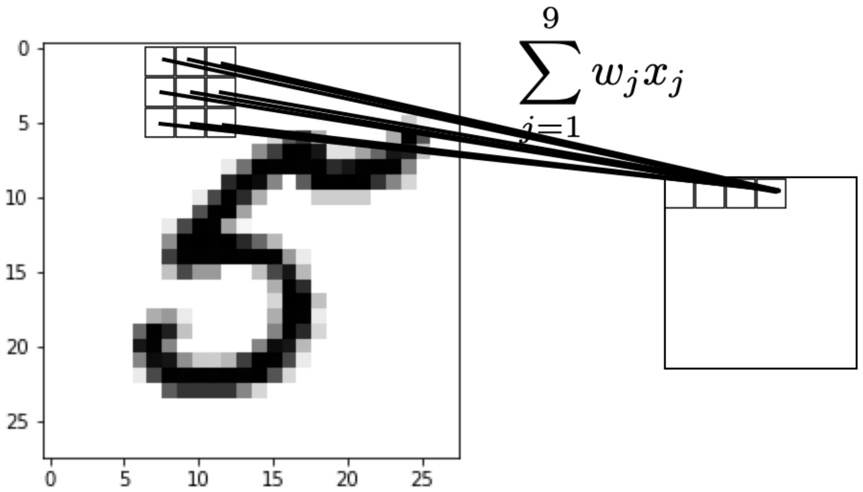
A "feature detector" (filter, kernel) slides over the inputs to generate a feature map



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

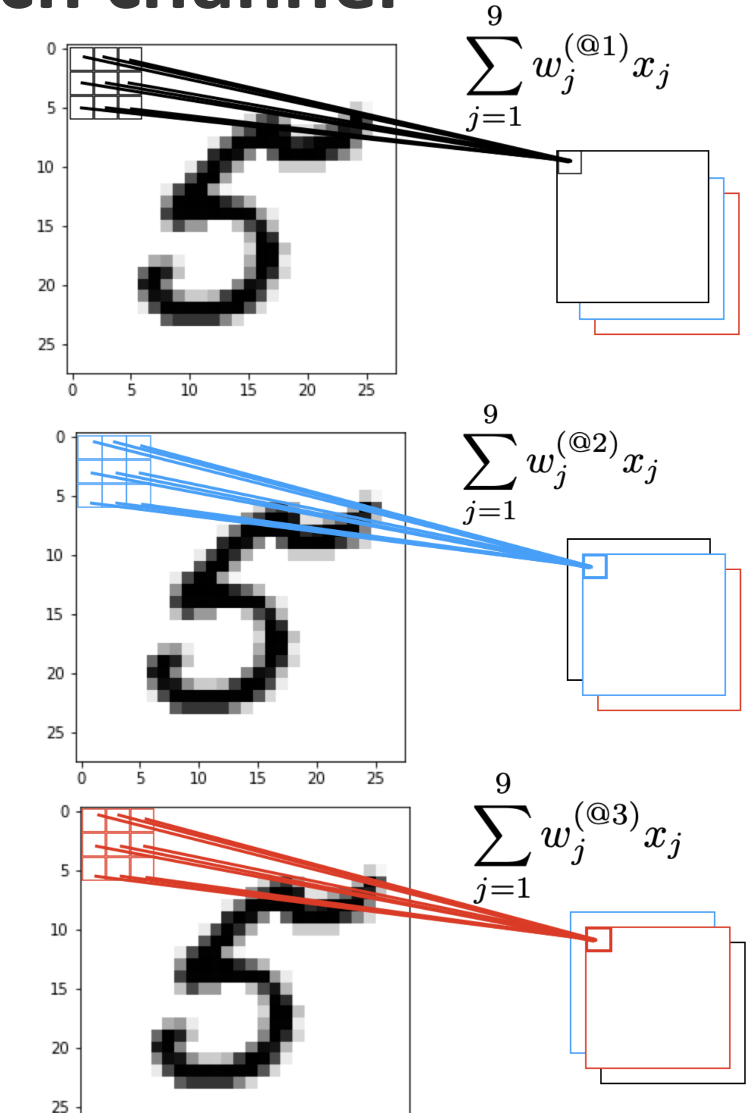
Alternative visualization of kernels

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



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

Kernels for each channel



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

Q: *Do you see sparse connectivity & weight sharing?*

Convolutional Neural Networks [LeCun 1989]

PROC. OF THE IEEE, NOVEMBER 1998 7

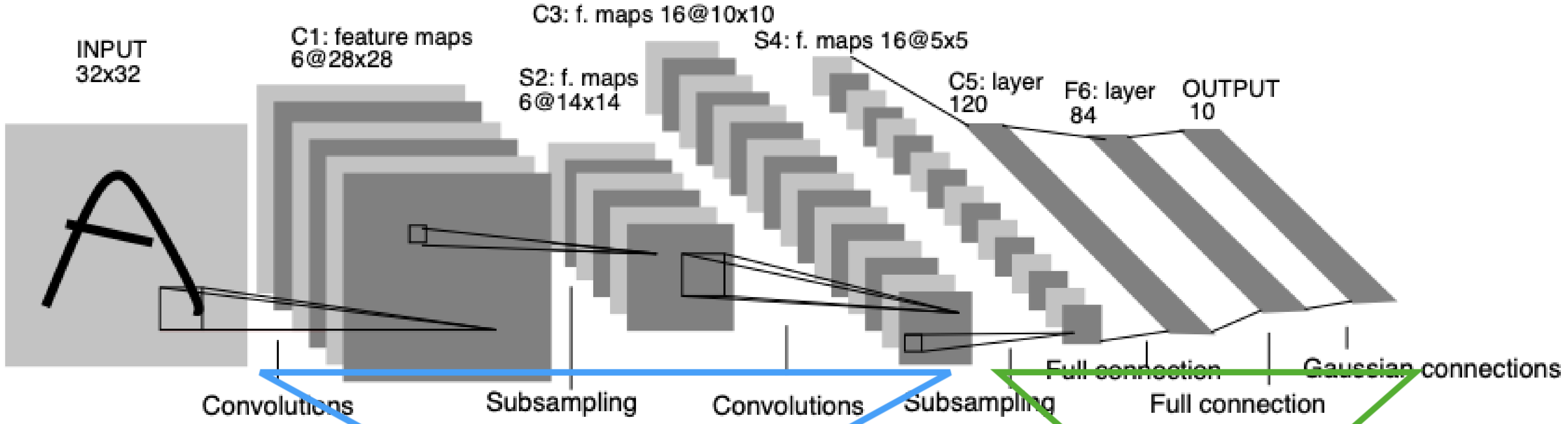
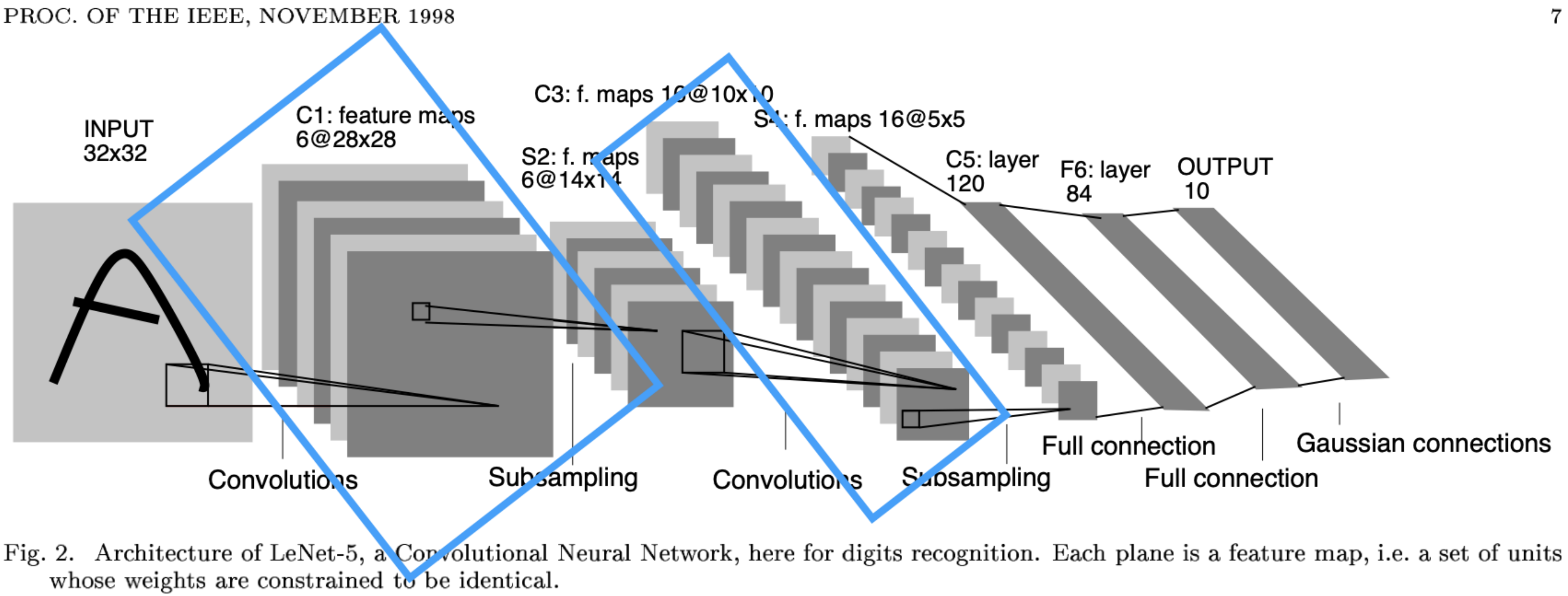


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

"Automatic feature extractor" "Regular classifier"

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

Convolutional Neural Networks [LeCun 1989]



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 [LeCun 1989]

PROC. OF THE IEEE, NOVEMBER 1998

Number of feature detectors

Size of the resulting layers

Multi-layer perceptron

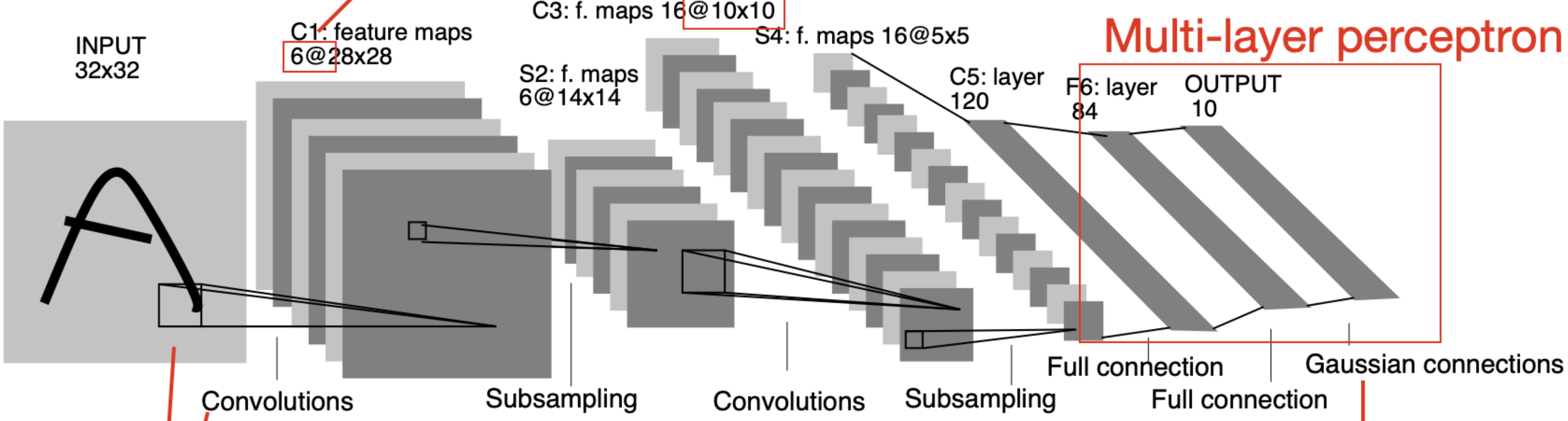


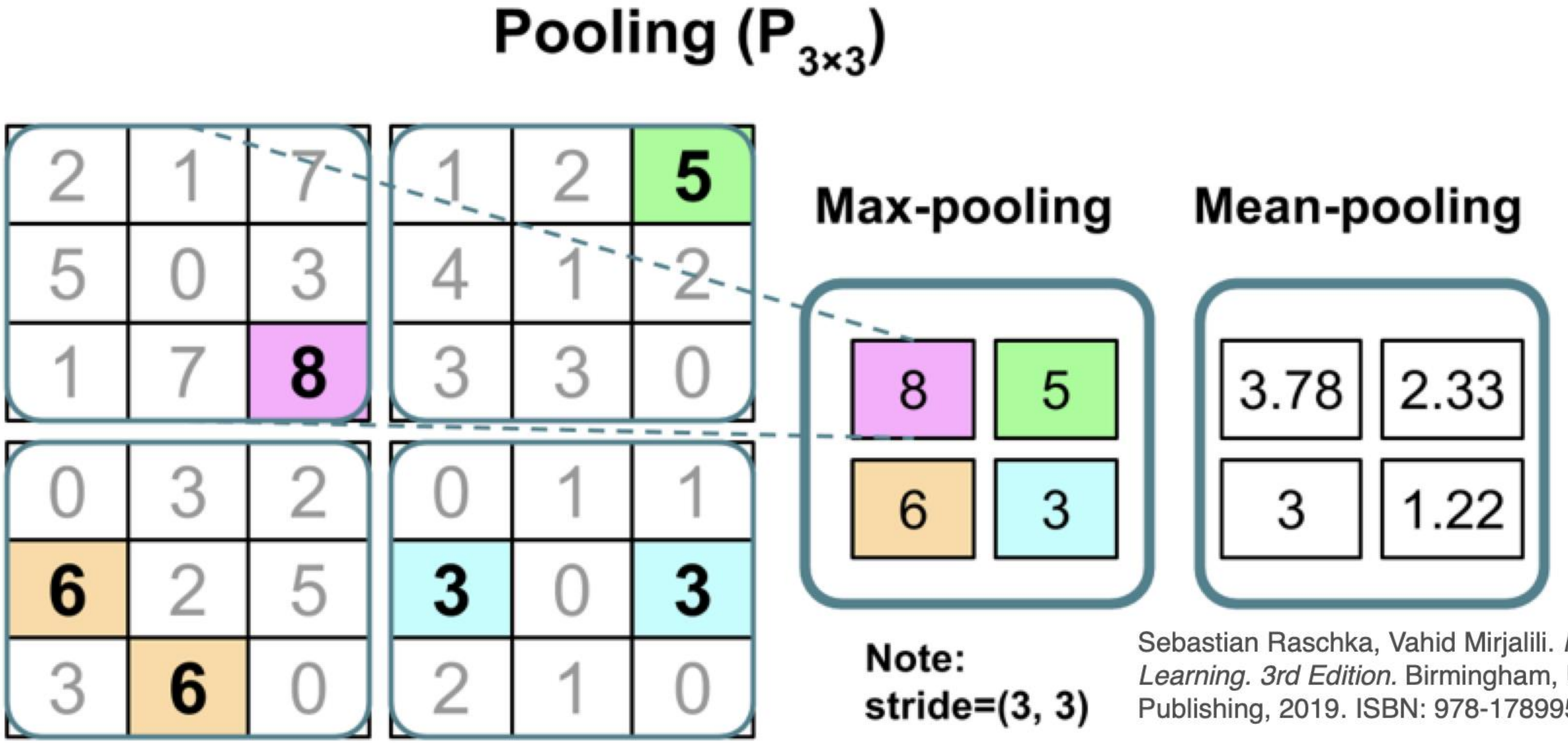
Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

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 common to use fc-layer + softmax + cross entropy)

"Pooling": lossy compression



Note: stride=(3, 3)

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

Main ideas of CNNs

- **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

Today: CNNs

1. What CNNs Can Do
2. Image Classification
3. Convolutional Neural Network Basics
4. **Cross-Correlation vs Convolution**
5. CNNs & Backpropagation
6. CNNs in PyTorch

Convolution: Adding two random variables

- Let X $\sim P_X, Y \ \sim P_Y$ be independent RVs. What's $E[X] + E[Y]$?
- What's $P(X + Y = z)$?

$$P(X + Y = z) = \int P(X = x, Y = z - x)dx$$

$$= \int P_X(X = x)P_Y(Y = z - x)dx$$

$$= \int P_X(x)P_Y(z - x)dx$$

- This is known as a **convolution** of P_X and $\mathrm{P_Y}$:

$$(P_X * P_Y)(z) = \int P_X(x)P_Y(z - x)dx$$

Convolution: Adding two random variables

- Let $X \sim P_X, Y \ \sim P_Y$ be indep. discrete RVs. What's $E[X] + E[Y]$?
- What's $P(X + Y = z)$?
- This is a **convolution** of P_X and P_Y:

$$(P_X * P_Y)(z) = \sum_x P_X(x)\, P_Y(z - x)$$

- More generally:
  - Discrete:

$$P_{X+Y}(z) = \sum_x P_{X,Y}\,(x, z - x)$$

  - Continuous:

$$f_{X+Y}(z) = \int f_{X,Y}(x, z - x) dx$$

Where's the "Convolution" in CNNs?

- Kernel sliding over the activation window:

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

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

Actually, this is a “cross-correlation”

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

CNNs give sparse connectivity

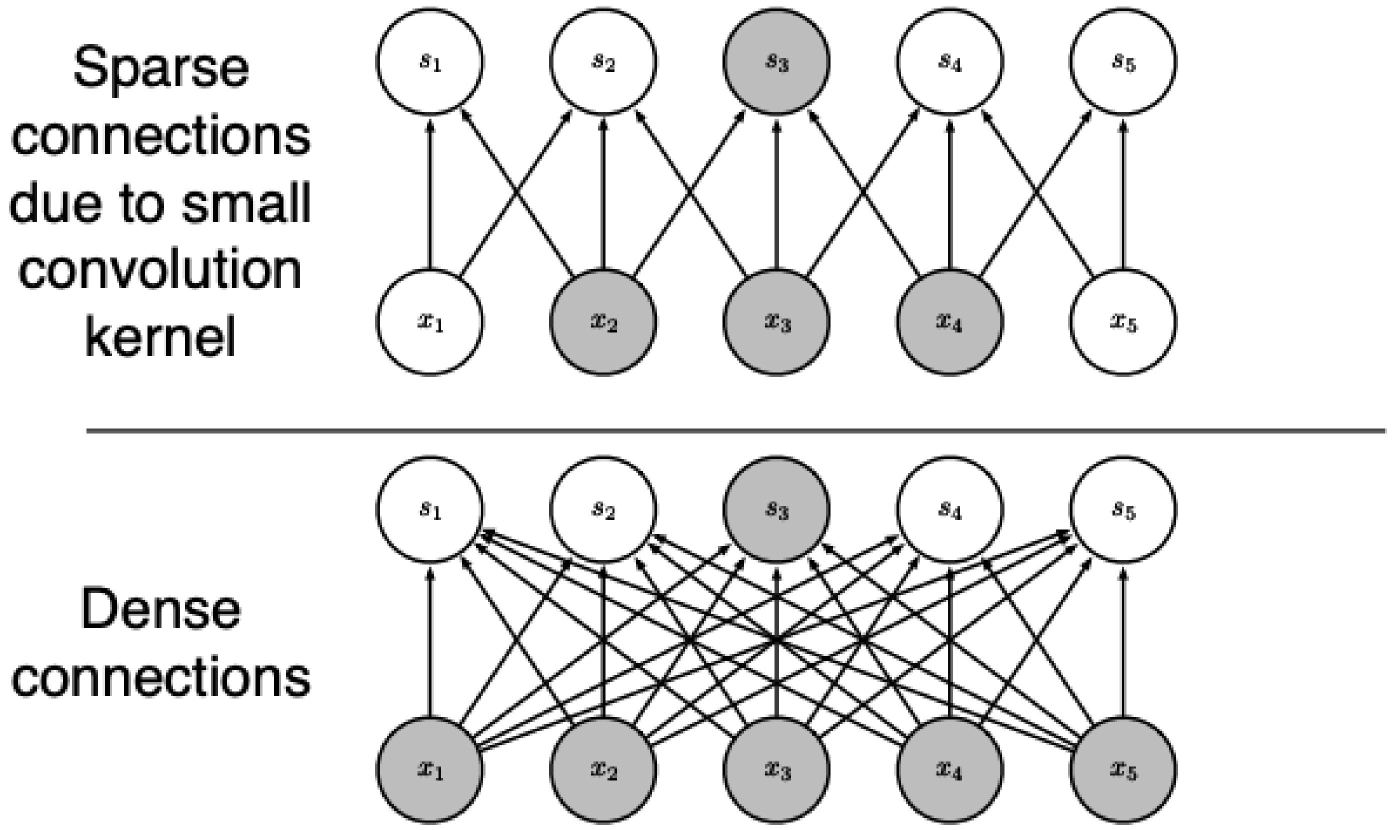


Figure 9.3

(Goodfellow 2016)

Receptive fields grow over depth

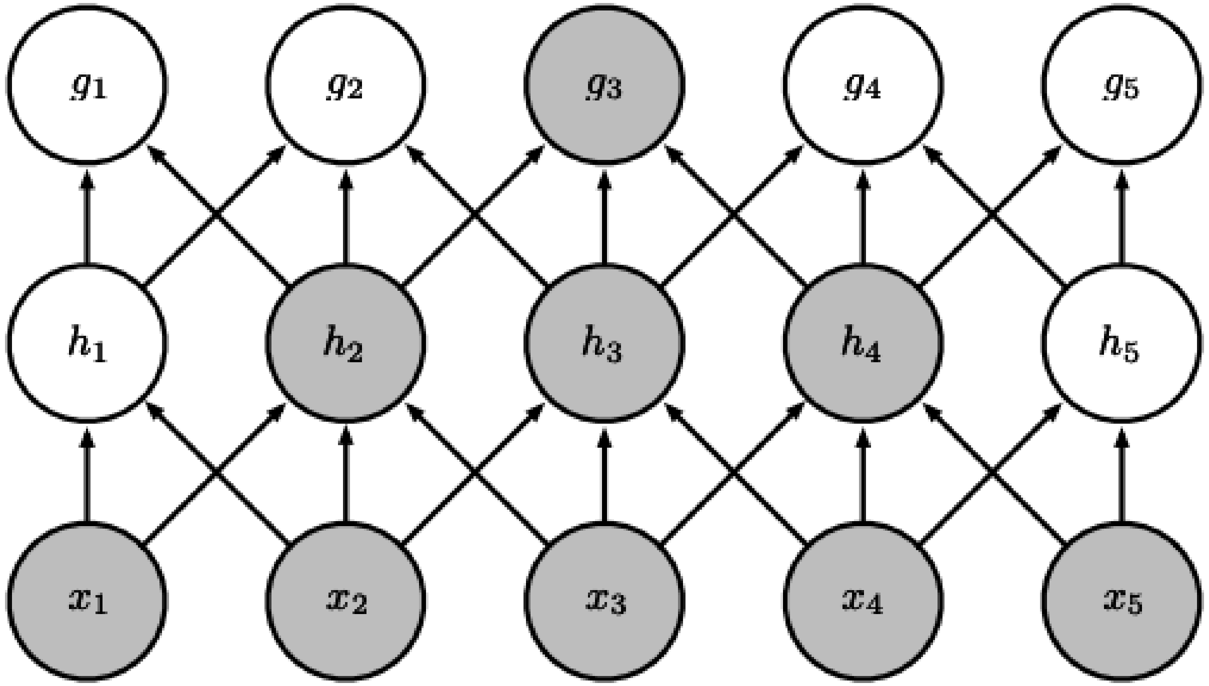


Figure 9.4

(Goodfellow 2016)

Parameter sharing

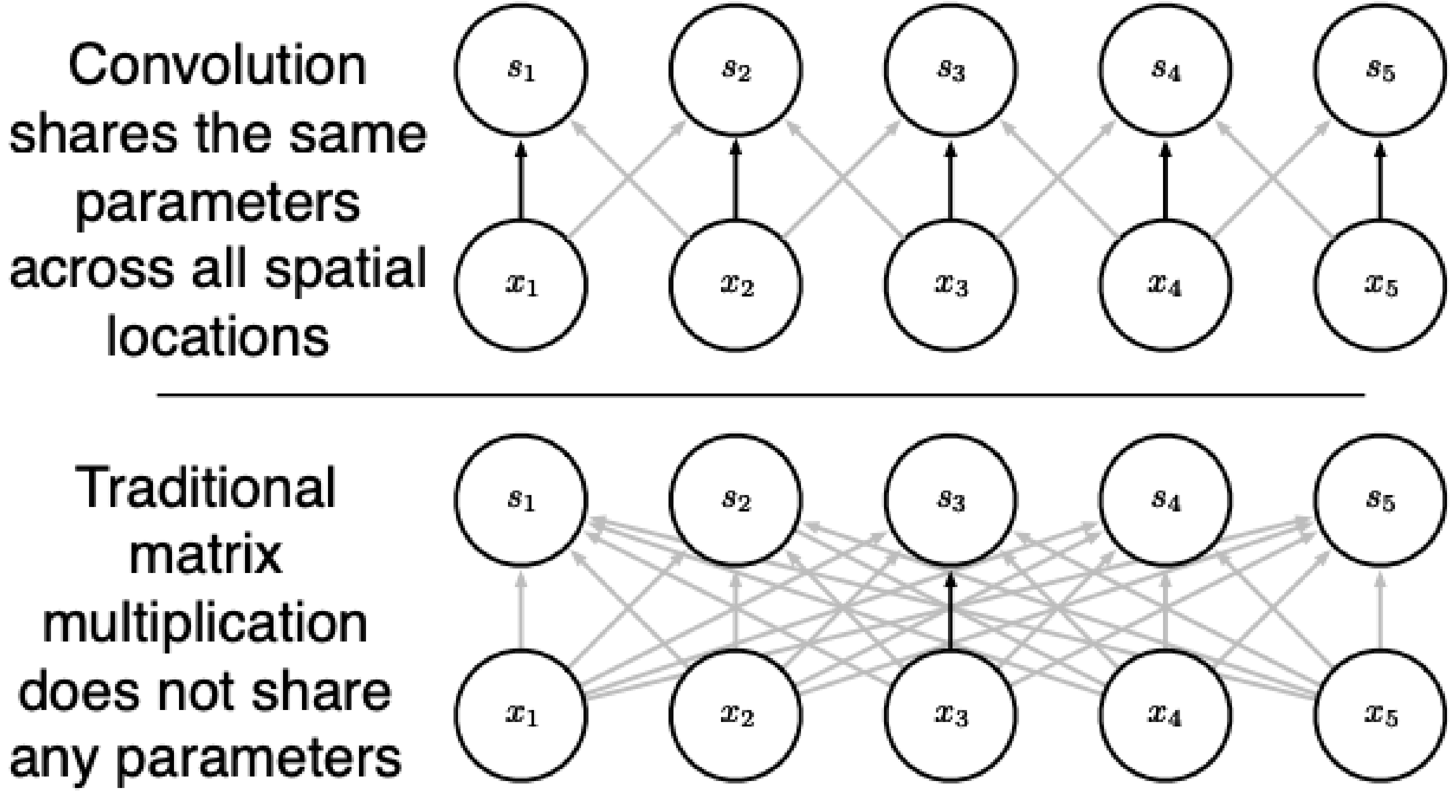


Figure 9.5

(Goodfellow 2016)

Impact of convolutions on size

Feature map size:

input width

kernel width

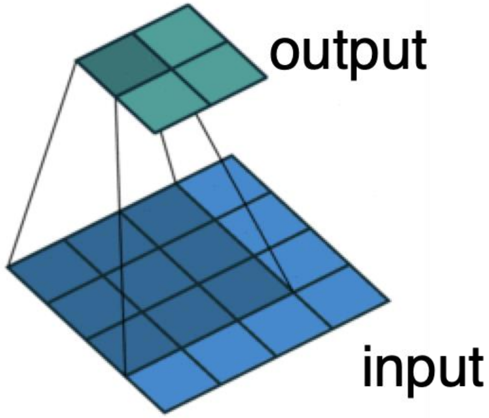
padding

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

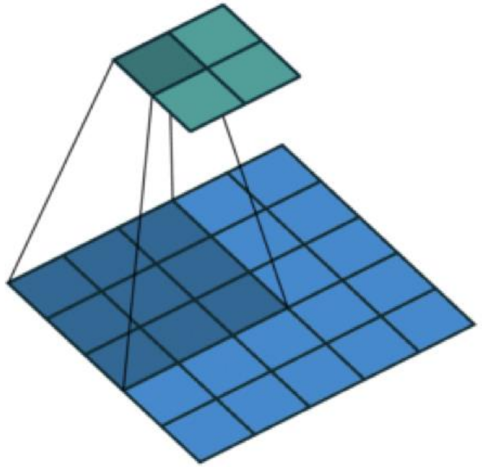
output width

stride

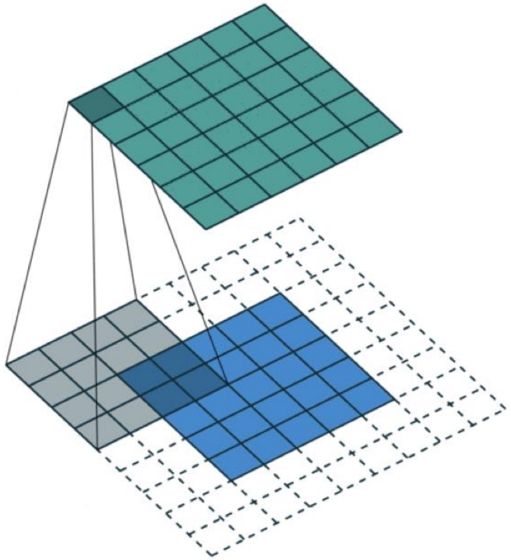
Padding



No padding, stride=1



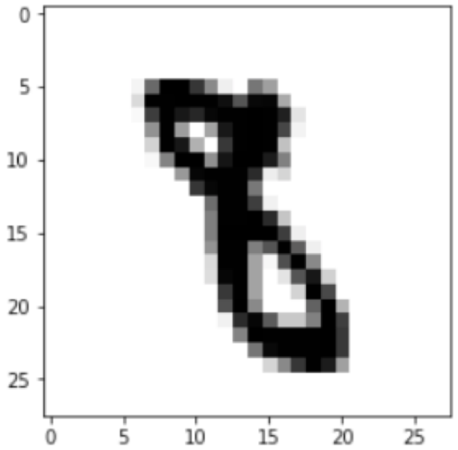
No padding, stride=2



padding=2, stride=1

Dumoulin, Vincent, and Francesco Visin. "A guide to convolution arithmetic for deep learning." arXiv preprint arXiv:1603.07285 (2016).

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()
```

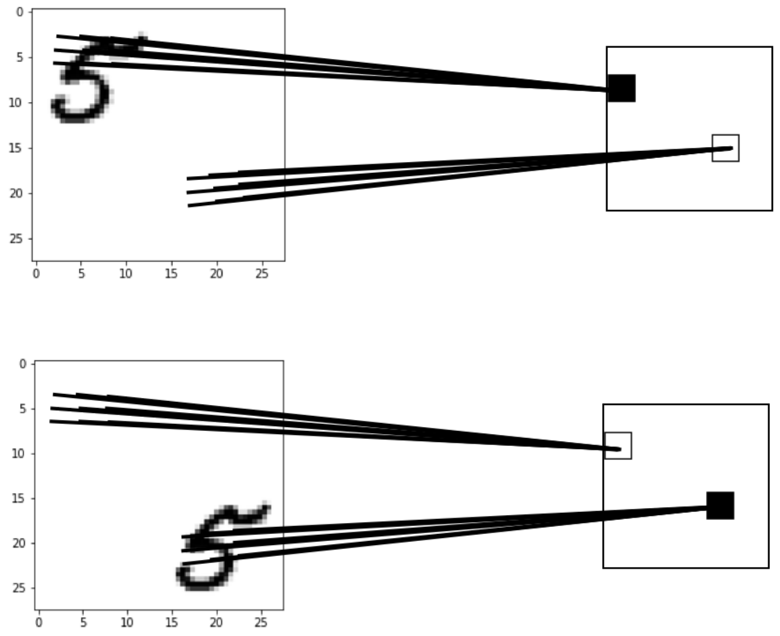
```
torch.Size([8])
```

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

What's the output size for this
28x28 image?

CNNs and Translation/Rotation/Scale Invariance

CNNs aren't really invariant to translation/rotation/scale:



The activations are still dependent on the location, etc.

Convolutional Neural Networks [LeCun 1989]

PROC. OF THE IEEE, NOVEMBER 1998

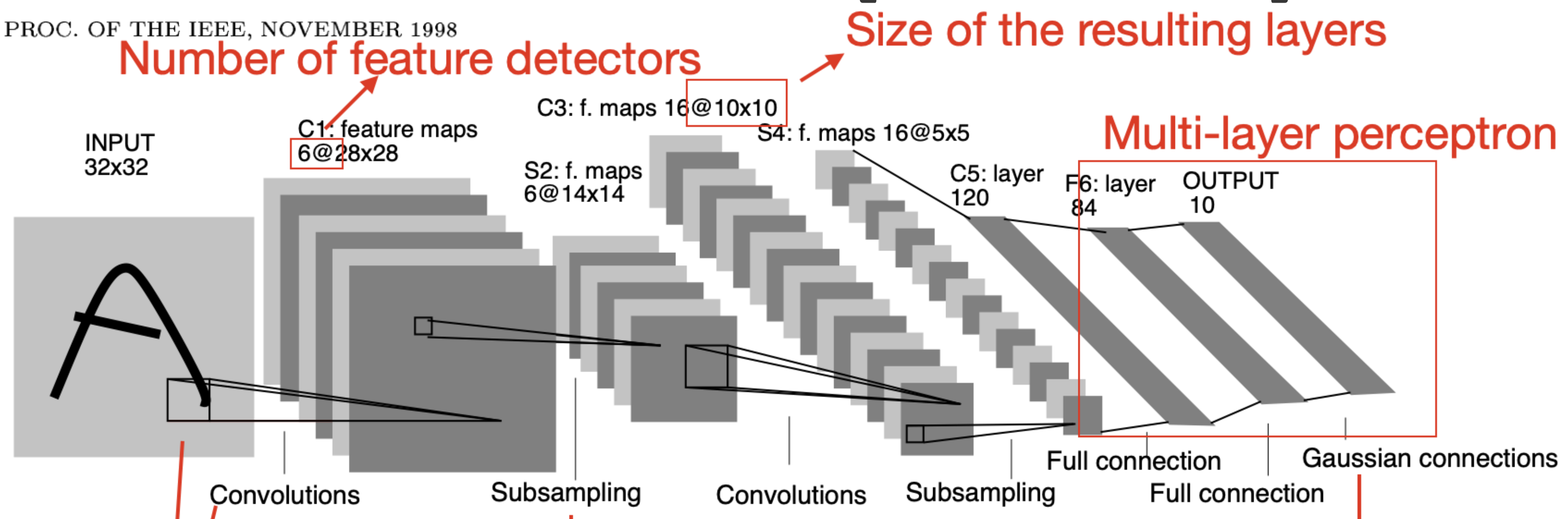


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

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"

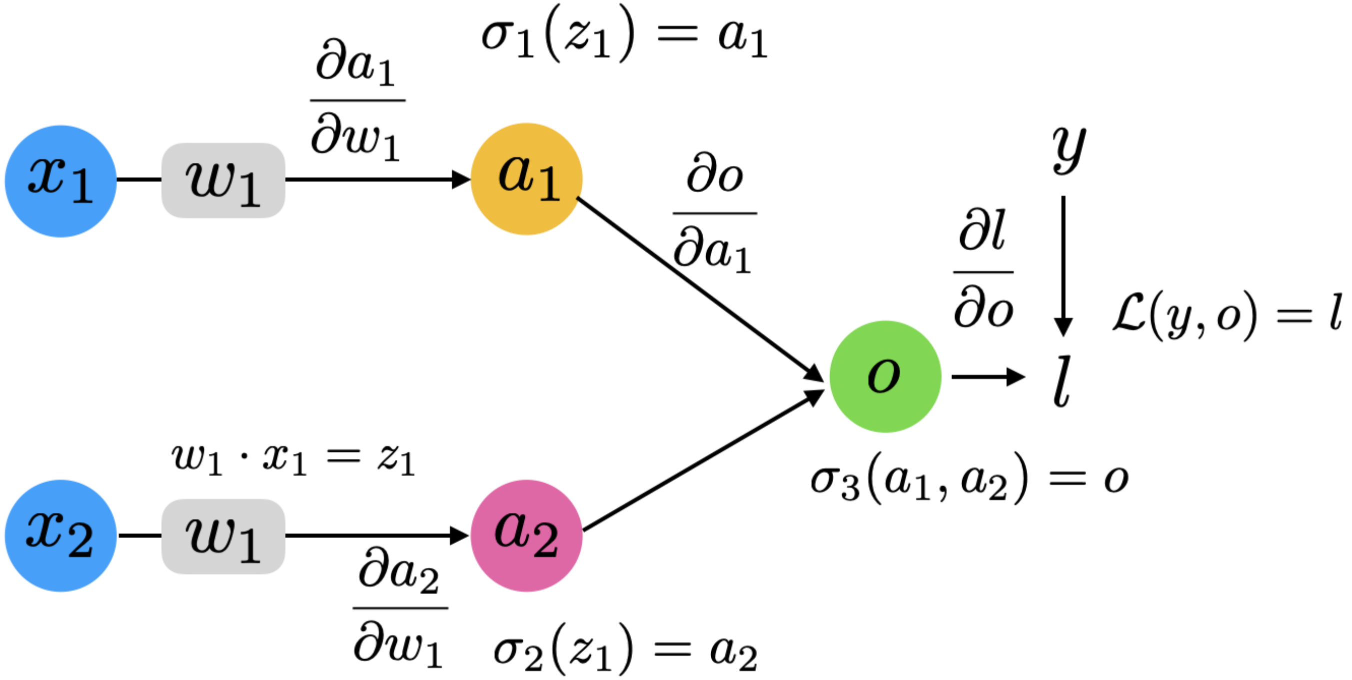
Today: CNNs

1. What CNNs Can Do
2. Image Classification
3. Convolutional Neural Network Basics
4. Cross-Correlation vs Convolution
5. **CNNs & Backpropagation**
6. CNNs in PyTorch

Backpropagation in CNNs

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

Recall: Weight sharing in computation graphs



Upper path

$$\frac{\partial l}{\partial w_1} = \frac{\partial l}{\partial o} \cdot \frac{\partial o}{\partial a_1} \cdot \frac{\partial a_1}{\partial w_1} + \frac{\partial l}{\partial o} \cdot \frac{\partial o}{\partial a_2} \cdot \frac{\partial a_2}{\partial w_1}$$

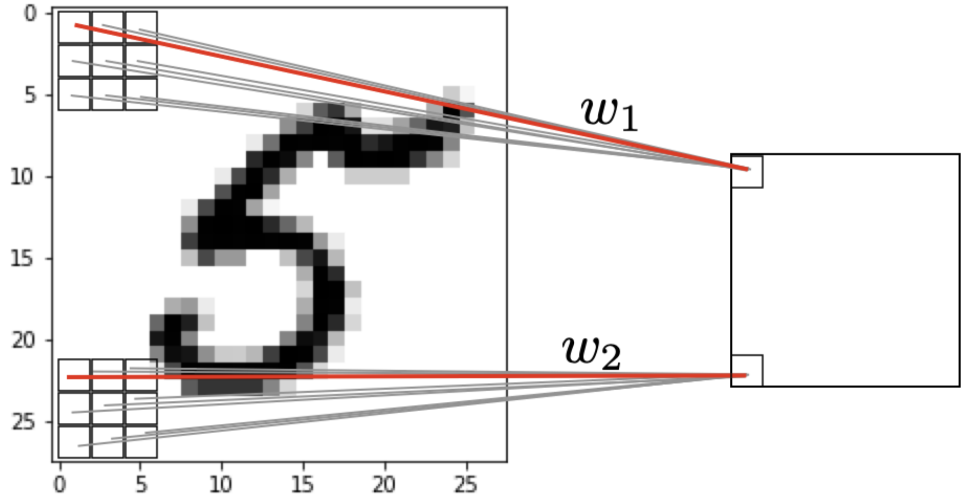
(multivariable chain rule)

Lower path

Backpropagation in CNNs

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

Due to weight sharing: $w_1 = w_2$



Optional averaging

weight update:

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

Today: CNNs

1. What CNNs Can Do
2. Image Classification
3. Convolutional Neural Network Basics
4. Cross-Correlation vs Convolution
5. CNNs & Backpropagation
6. **CNNs in PyTorch**

CNNs in PyTorch

PROC. OF THE IEEE, NOVEMBER 1998

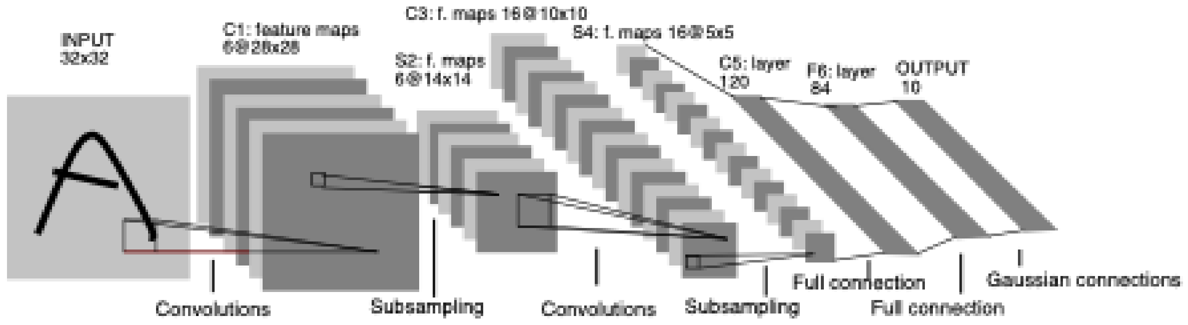


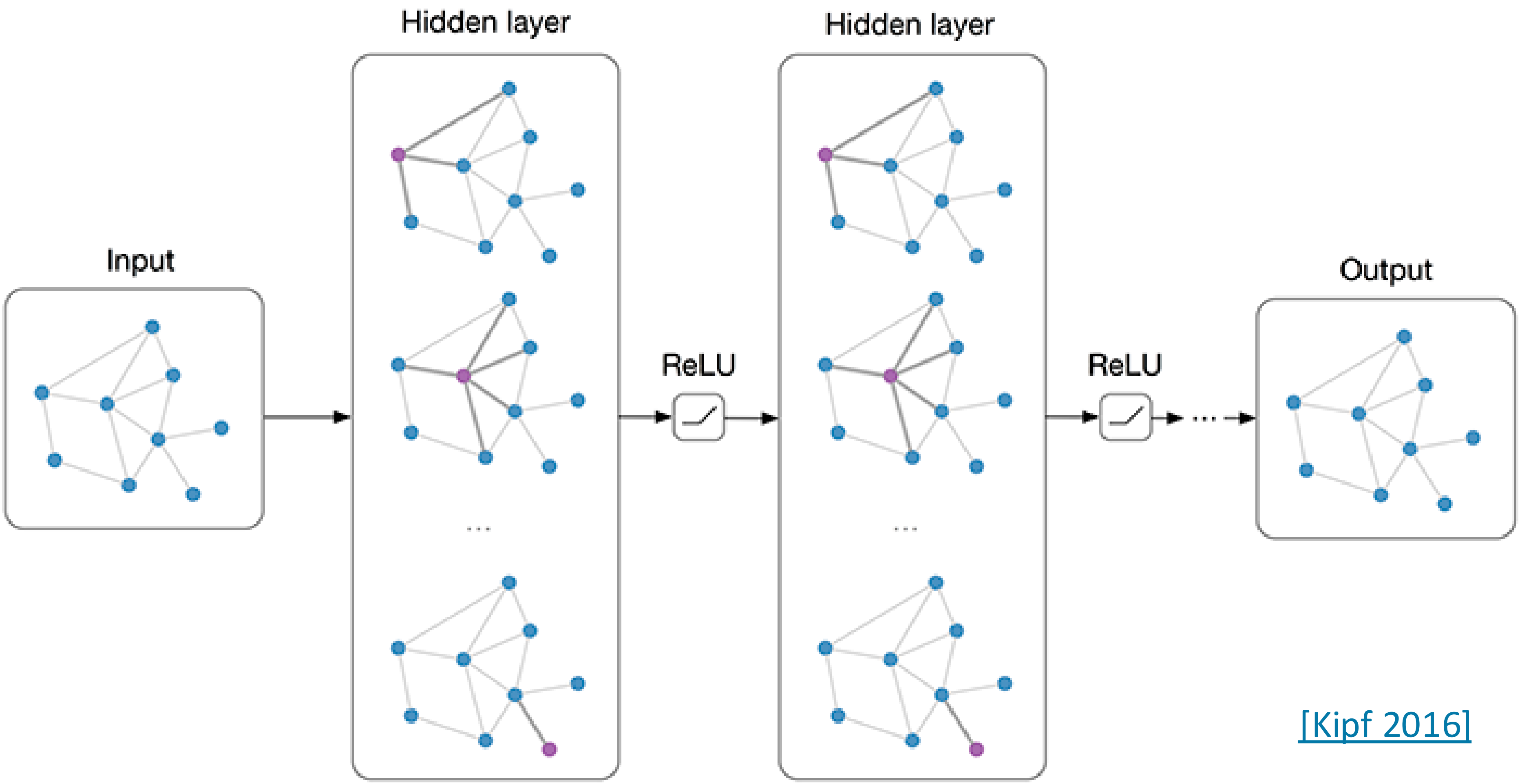
Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

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

```python
class LeNet5(nn.Module):

    def __init__(self, num_classes, grayscale=False):
        super(LeNet5, self).__init__()

        self.grayscale = grayscale
        self.num_classes = num_classes

        if self.grayscale:
            in_channels = 1
        else:
            in_channels = 3

        self.features = nn.Sequential(
            nn.Conv2d(in_channels, 6, kernel_size=5),
            nn.Tanh(),
            nn.MaxPool2d(kernel_size=2),
            nn.Conv2d(6, 16, kernel_size=5),
            nn.Tanh(),
            nn.MaxPool2d(kernel_size=2)
        )

        self.classifier = nn.Sequential(
            nn.Linear(16*5*5, 120),
            nn.Tanh(),
            nn.Linear(120, 84),
            nn.Tanh(),
            nn.Linear(84, num_classes),
        )

    def forward(self, x):
        x = self.features(x)
        x = torch.flatten(x, 1)
        logits = self.classifier(x)
        probas = F.softmax(logits, dim=1)
        return logits, probas
```

Convolutions on non-image data?

Graph Convolutional Networks



[Kipf 2016]

Questions?