



# STAT 453: Introduction to Deep Learning and Generative Models

---

Ben Lengerich

Lecture 13: CNNs

October 15, 2025



# 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

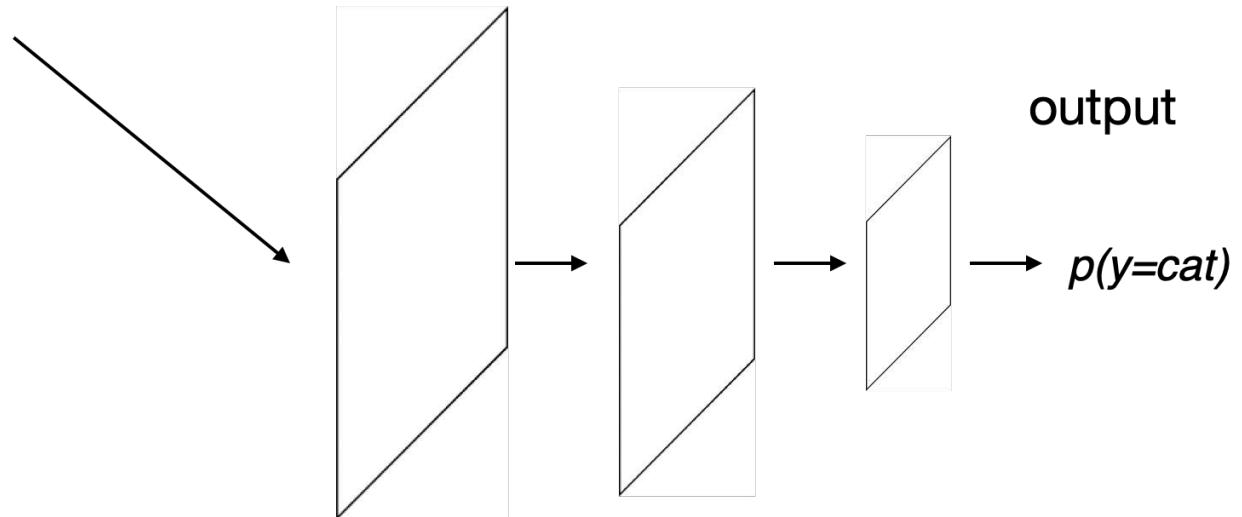
---



Image Source:  
[twitter.com%2Fcats&psig=AOvVaw30\\_o-PCM-K21DiMAJQimQ4&ust=155388775741551](https://twitter.com%2Fcats&psig=AOvVaw30_o-PCM-K21DiMAJQimQ4&ust=155388775741551)



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

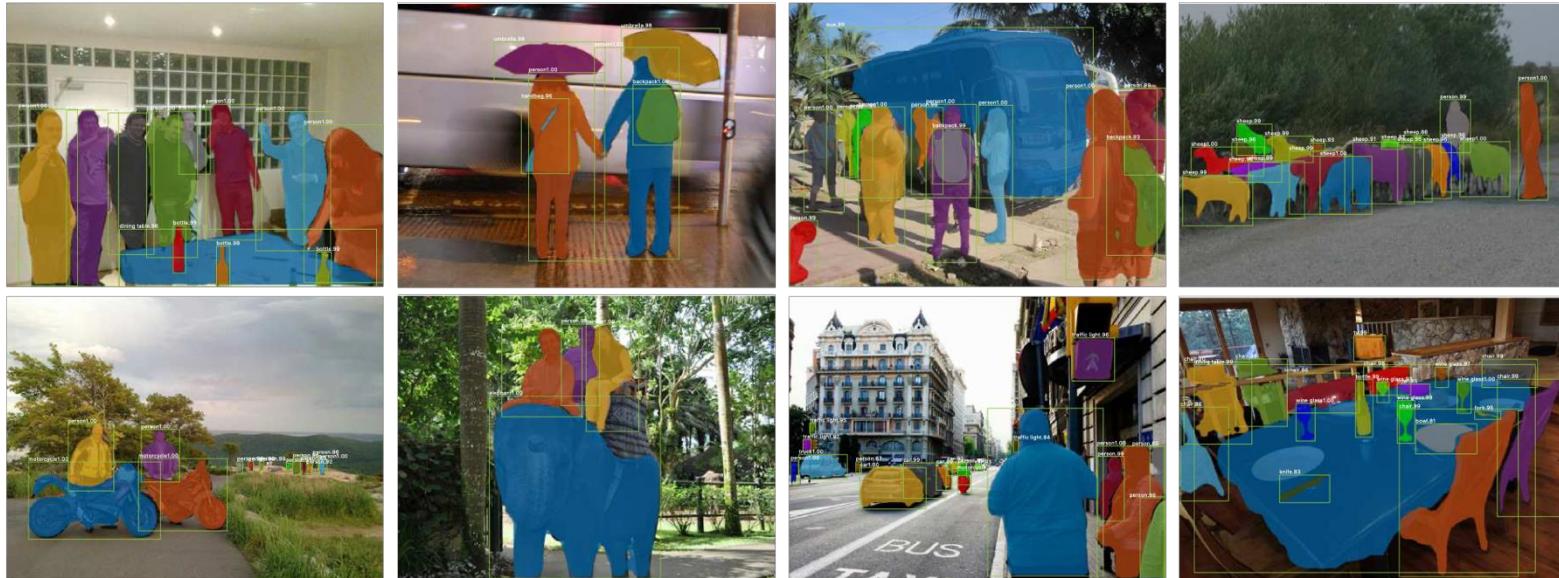


Figure 2. **Mask R-CNN** results on the COCO test set. These results are based on ResNet-101 [15], achieving a *mask AP* of 35.7 and running at 5 fps. Masks are shown in color, and bounding box, category, and confidences are also shown.

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.

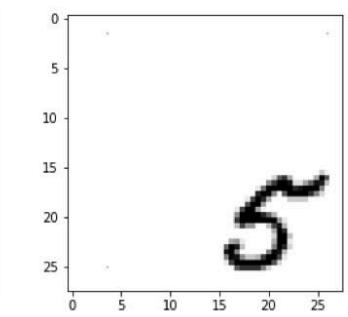
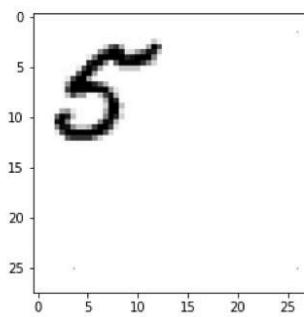


Image Source:  
[twitter.com%2Fcats&psig=AOvVaw30\\_o-PCM-K21DiMAJQimQ4&ust=155388775741551](https://twitter.com%2Fcats&psig=AOvVaw30_o-PCM-K21DiMAJQimQ4&ust=155388775741551)



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)

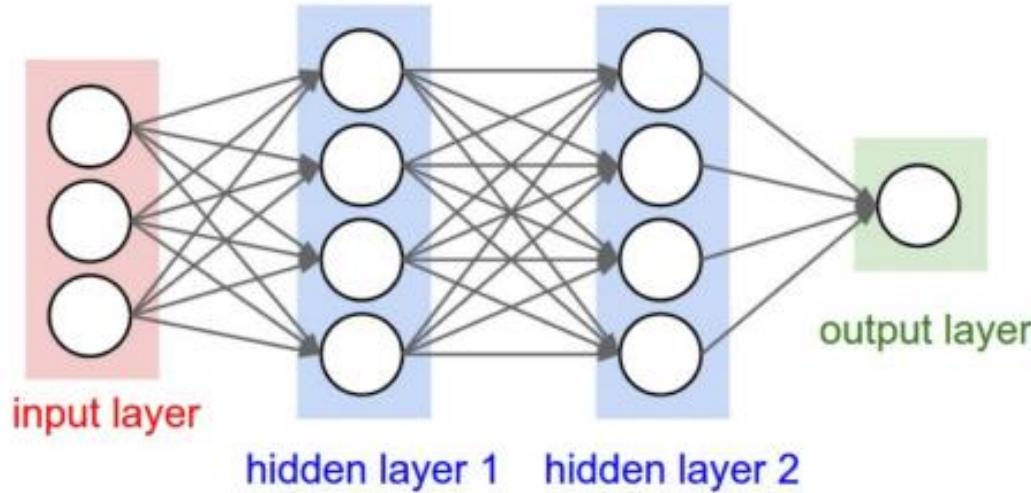
Or even simple translation



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

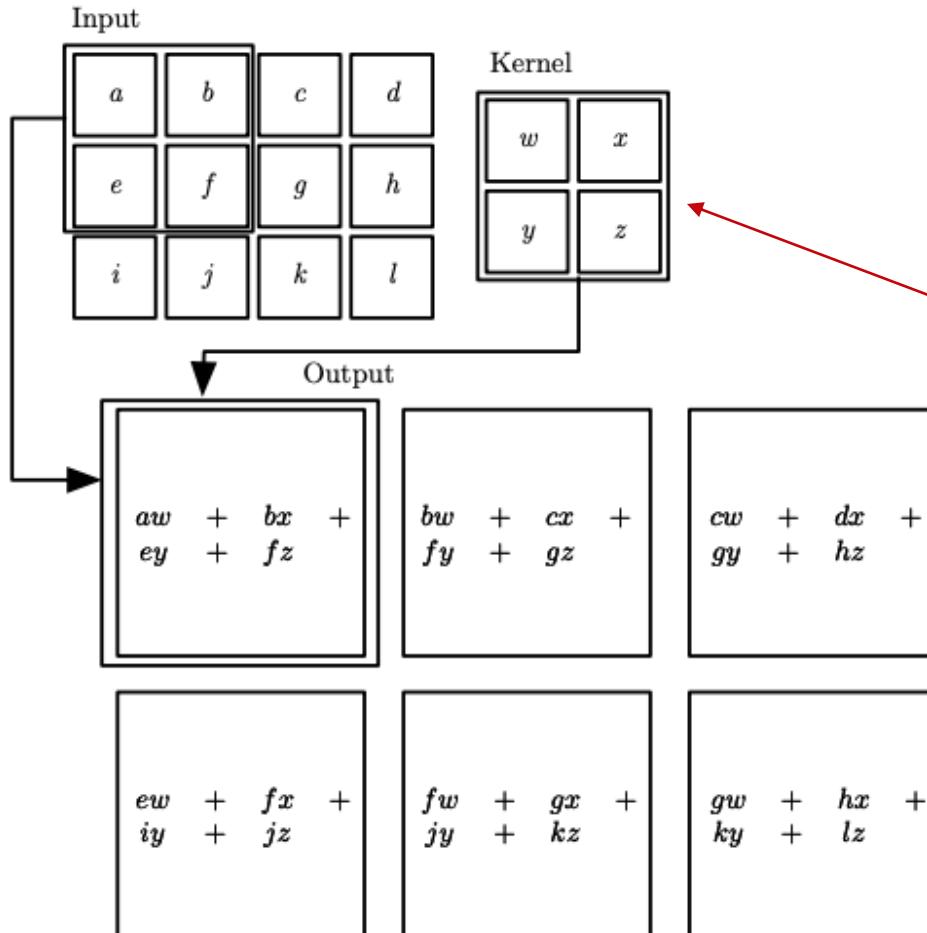
# Convolutional Neural Networks [LeCun 1989]

---

- 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)!

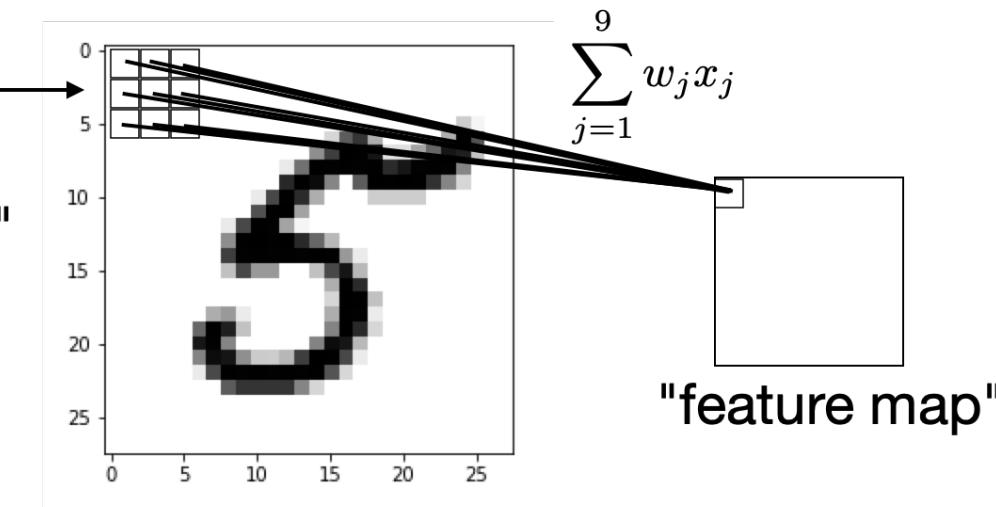
Fig. Goodfellow et al. 2016

# Alternative visualization of kernels

---

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

The pixels are  
referred to  
as "receptive field"

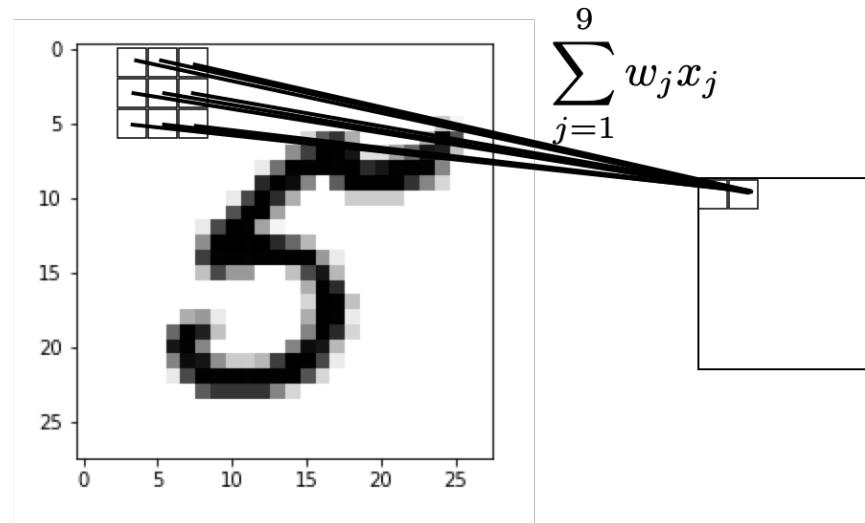


A feature detector that works well in one region may also work well in another region

# Alternative visualization of kernels

---

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

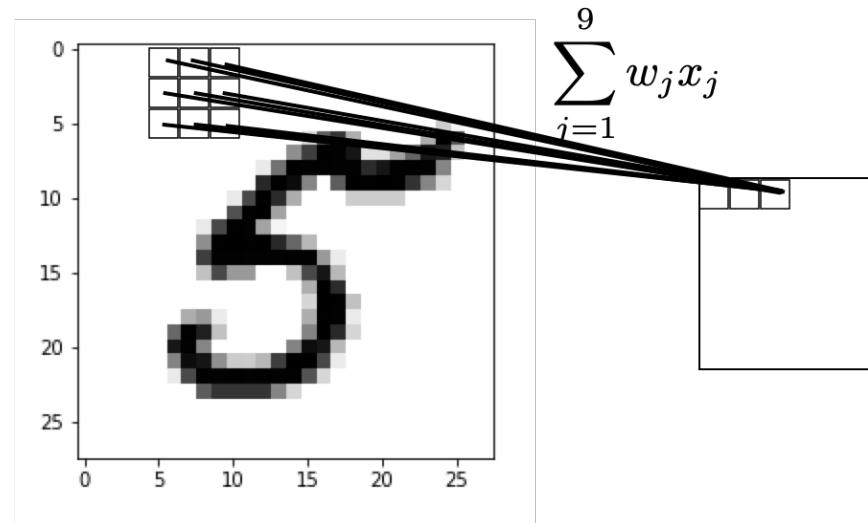


A feature detector that works well in one region may also work well in another region

# Alternative visualization of kernels

---

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

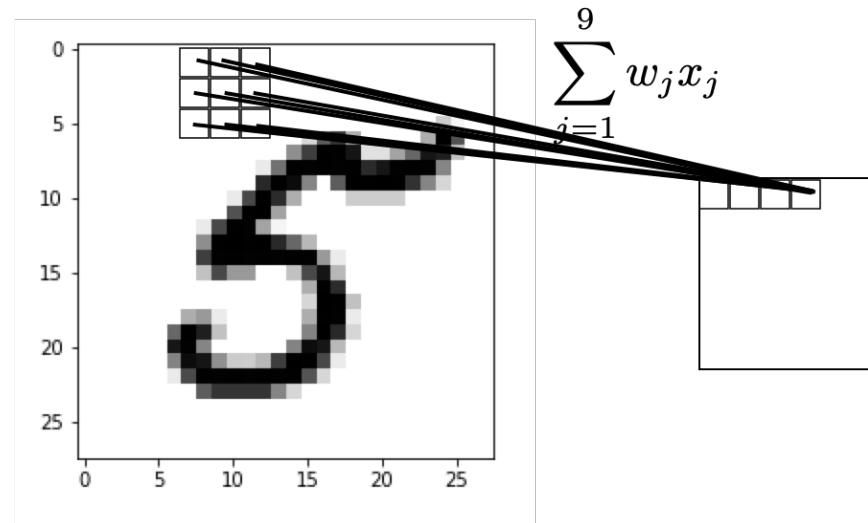


A feature detector that works well in one region may also work well in another region

# Alternative visualization of kernels

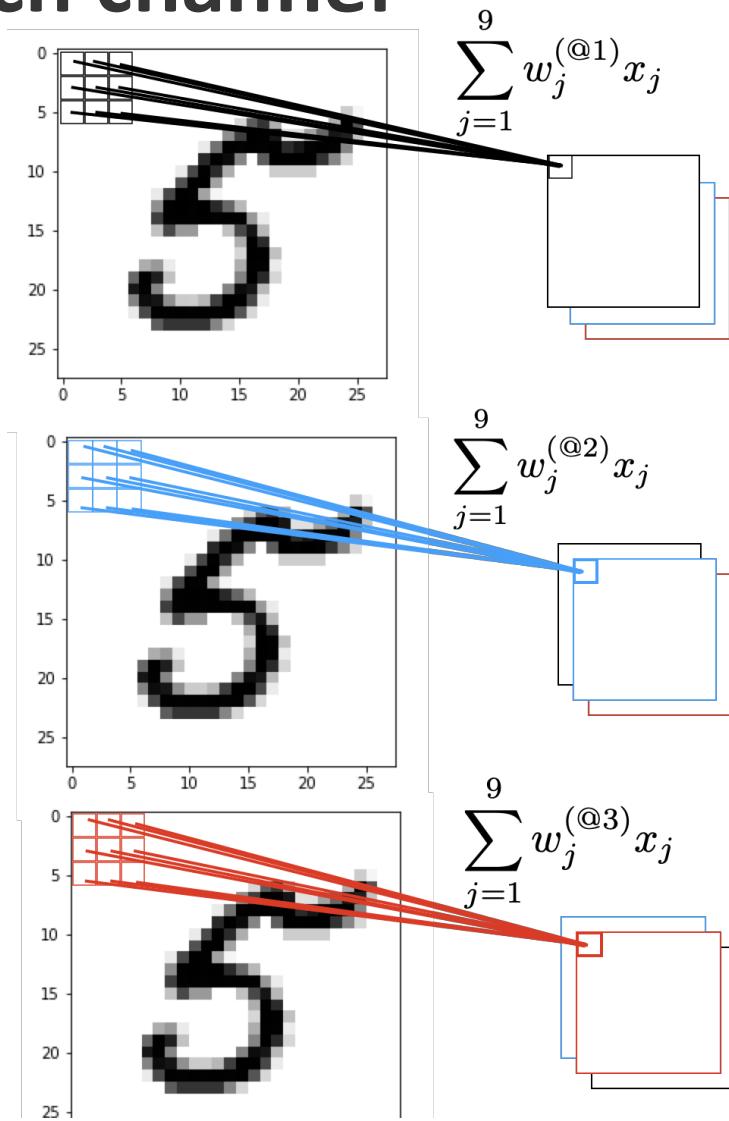
---

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

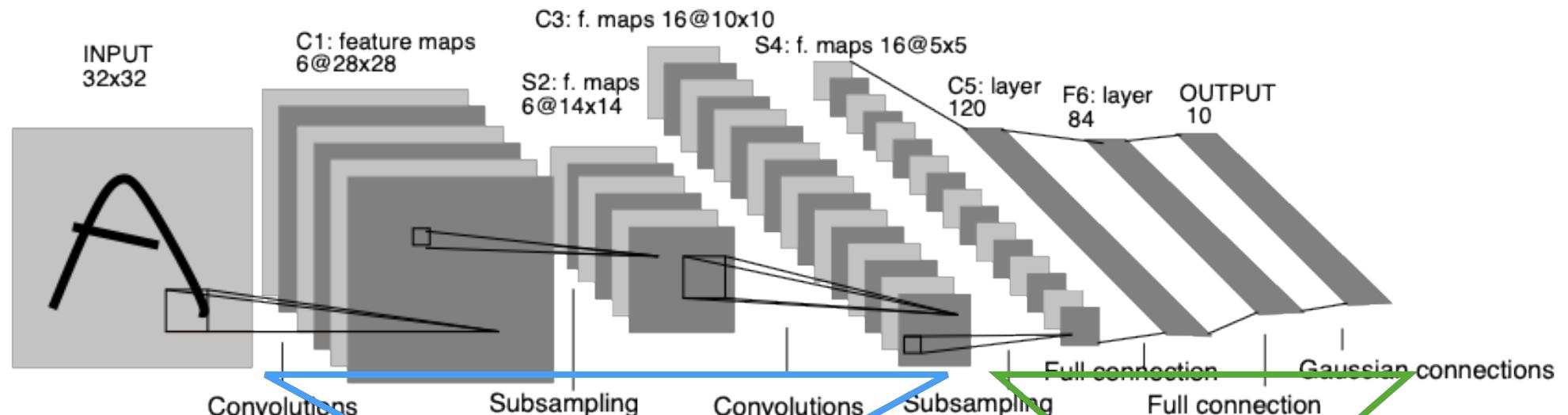


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

"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]

PROC. OF THE IEEE, NOVEMBER 1998

7

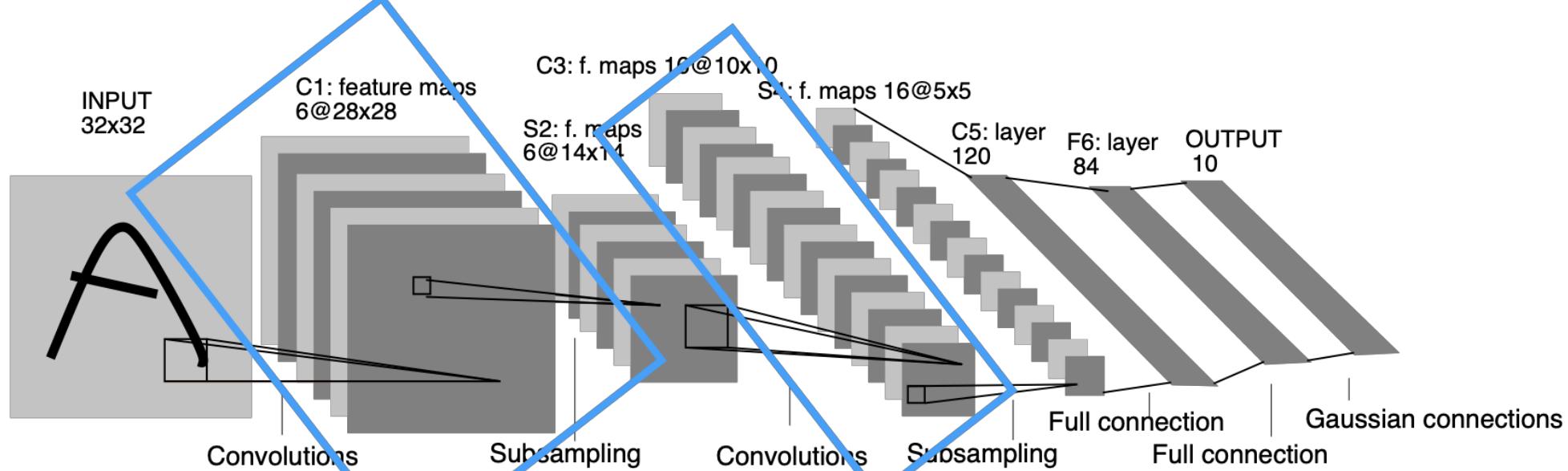


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

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 [LeCun 1989]

PROC. OF THE IEEE, NOVEMBER 1998

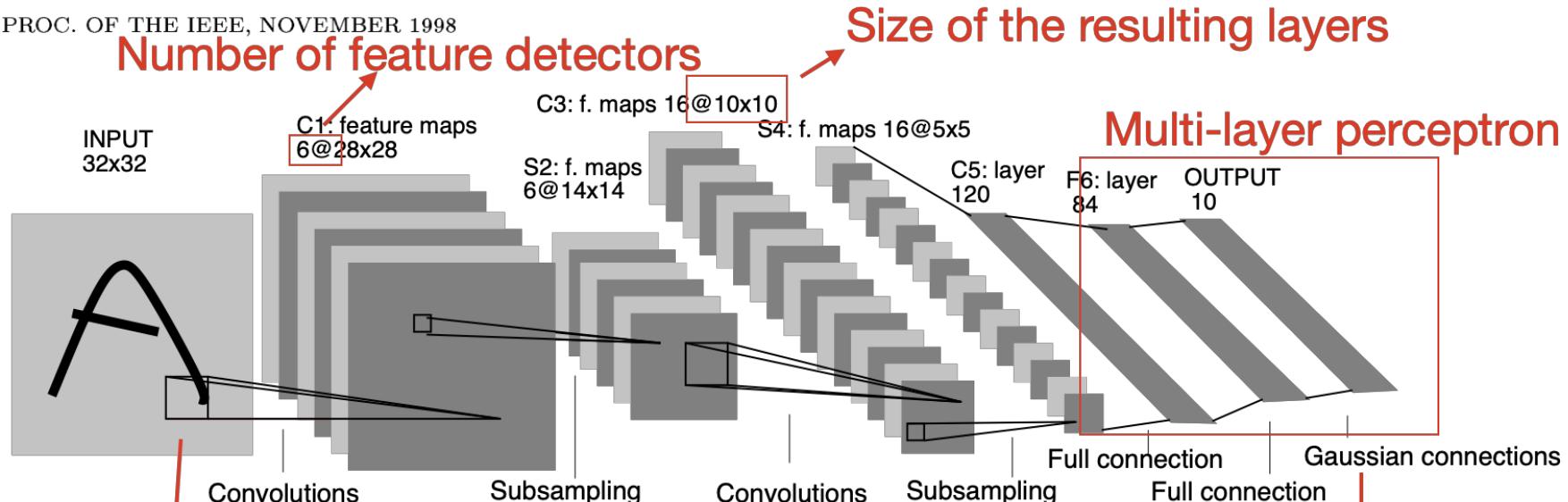
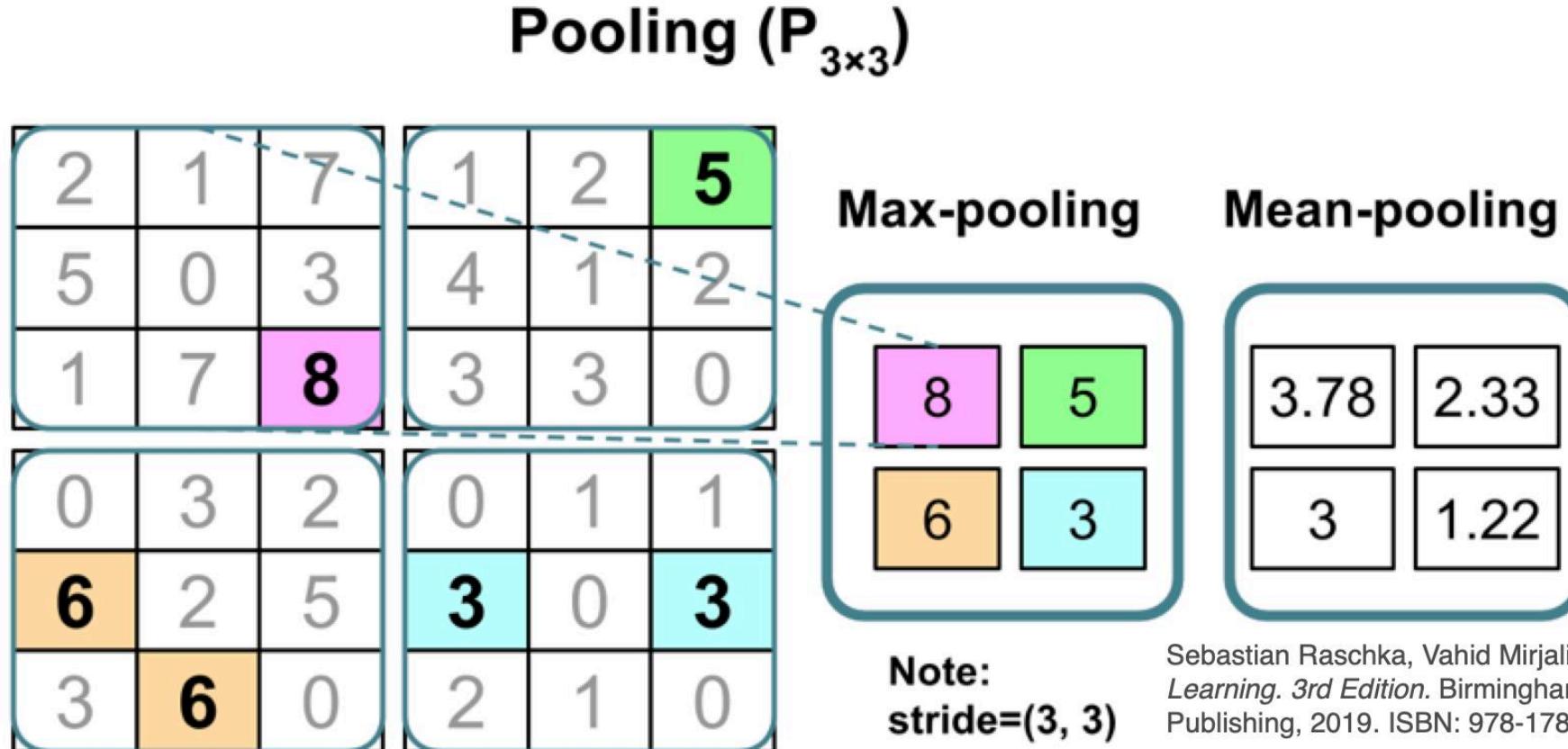


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

"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





# 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  $P(X + Y = z)$ ?

$$\begin{aligned} P(X + Y = z) &= \int P(X = x, Y = z - x) dx \\ &= \int P_X(X = x)P_Y(Y = z - x) dx \\ &= \int P_X(x)P_Y(z - x) dx \end{aligned}$$

- This is known as a **convolution** of  $P_X$  and  $P_Y$ :

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



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

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

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

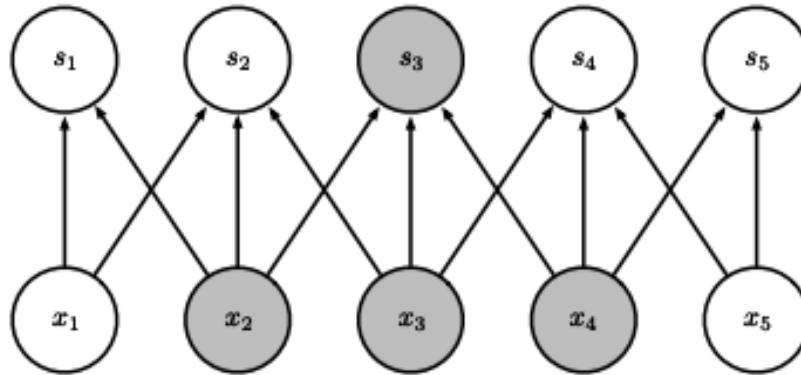
Basically, we are flipping the kernel (or the receptive field) horizontally and vertically

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

---

Sparse  
connections  
due to small  
convolution  
kernel



Dense  
connections

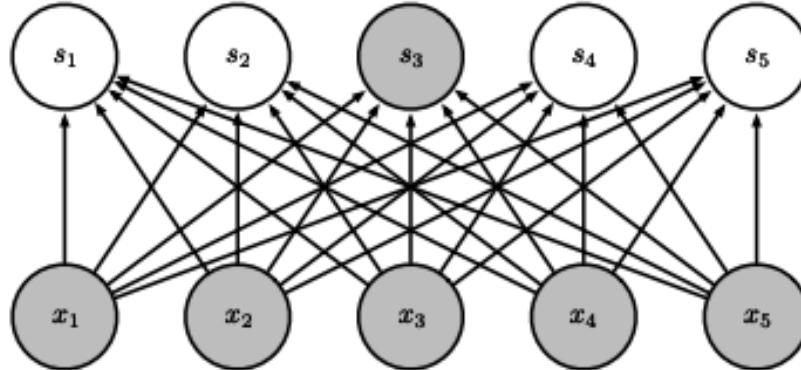


Figure 9.3

(Goodfellow 2016)

# Receptive fields grow over depth

---

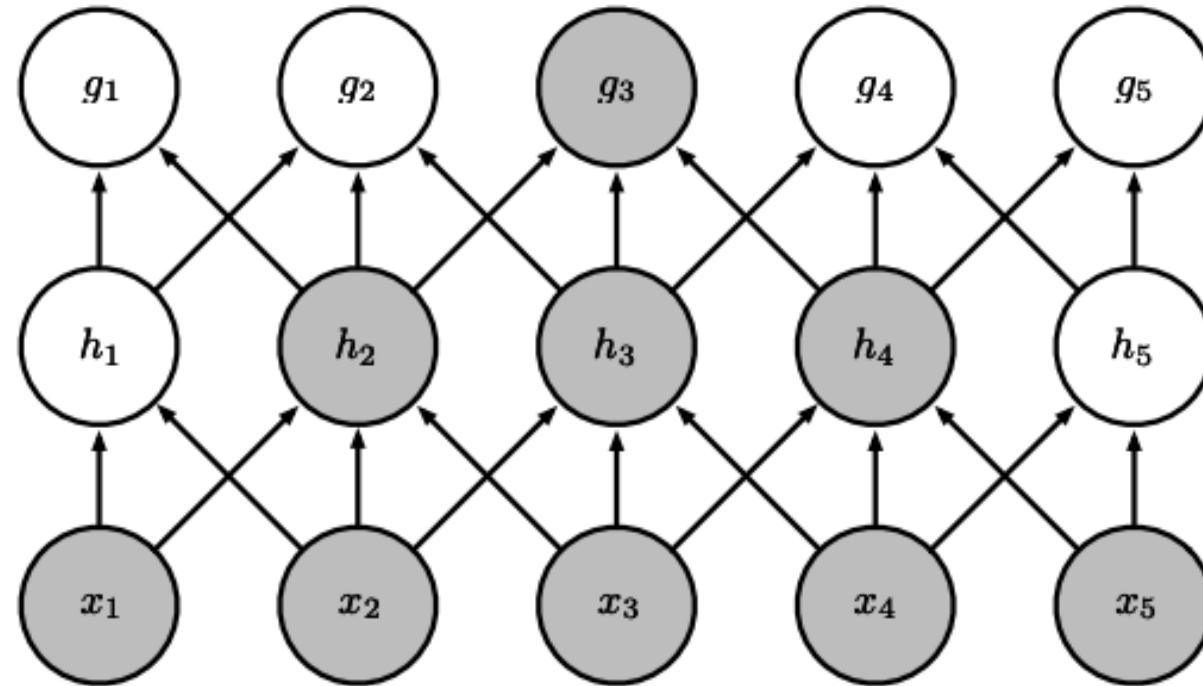
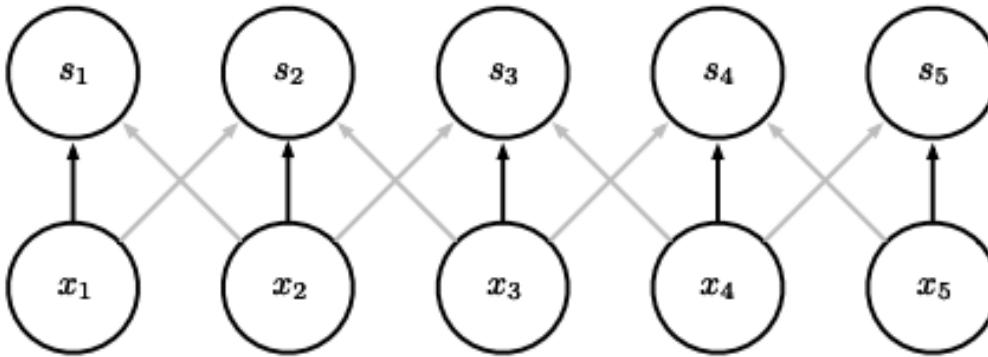


Figure 9.4

(Goodfellow 2016)

# Parameter sharing

Convolution  
shares the same  
parameters  
across all spatial  
locations



Traditional  
matrix  
multiplication  
does not share  
any parameters

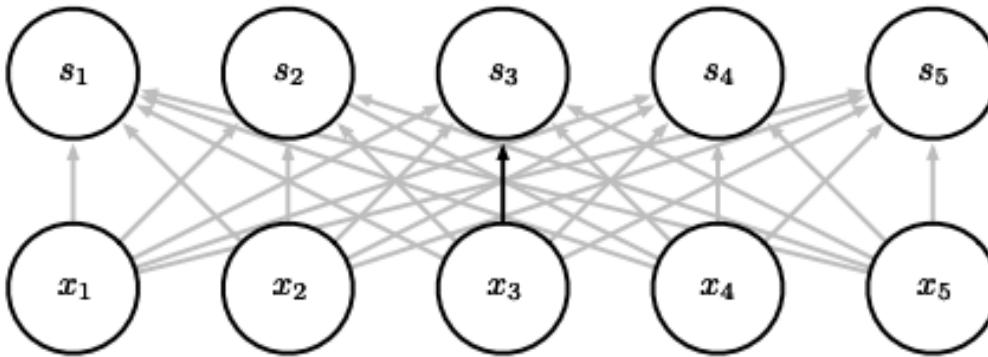


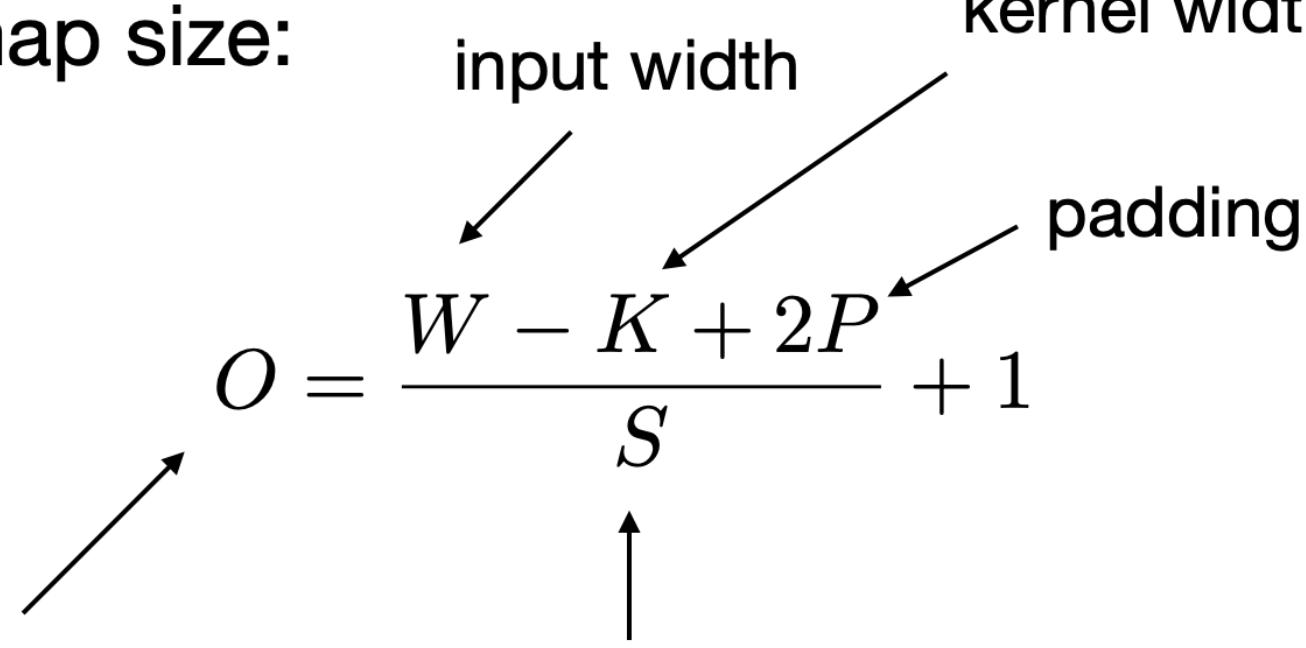
Figure 9.5

(Goodfellow 2016)

# Impact of convolutions on size

---

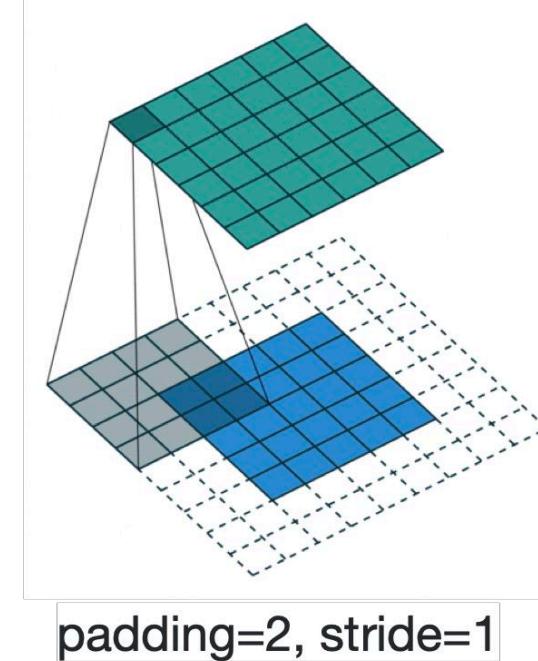
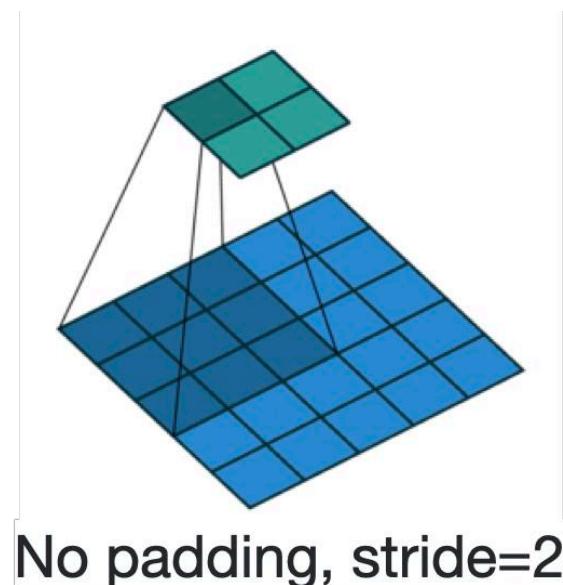
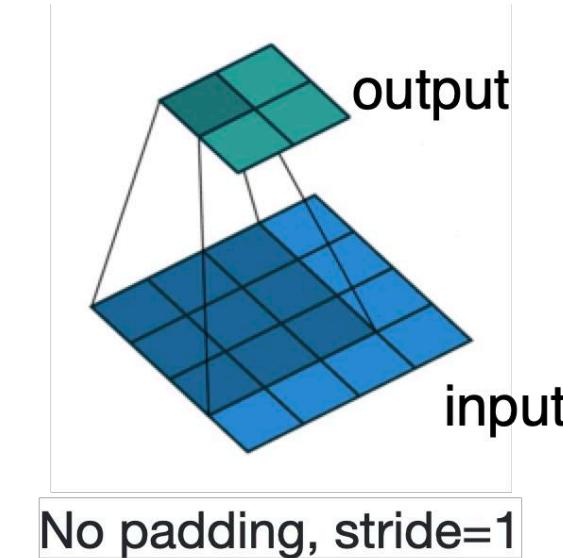
Feature map size:

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


The diagram illustrates the components of the convolution formula. The output width  $O$  is shown at the bottom left. Above it, the formula  $O = \frac{W - K + 2P}{S} + 1$  is displayed. Four arrows point from labels to the formula: an arrow from 'input width' points to the term  $W$ ; an arrow from 'kernel width' points to the term  $K$ ; an arrow from 'padding' points to the term  $2P$ ; and an arrow from 'stride' points to the term  $S$ .

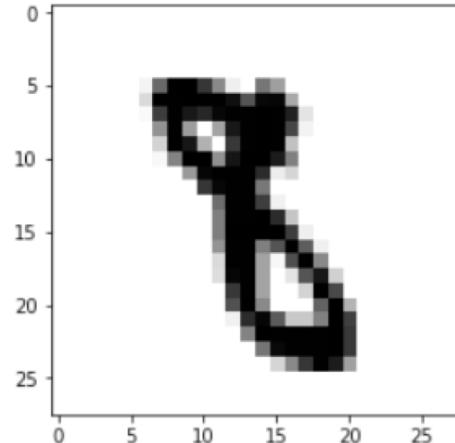
# Padding

---



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])
```

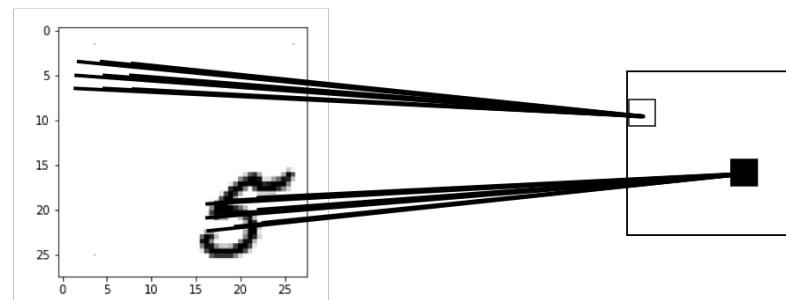
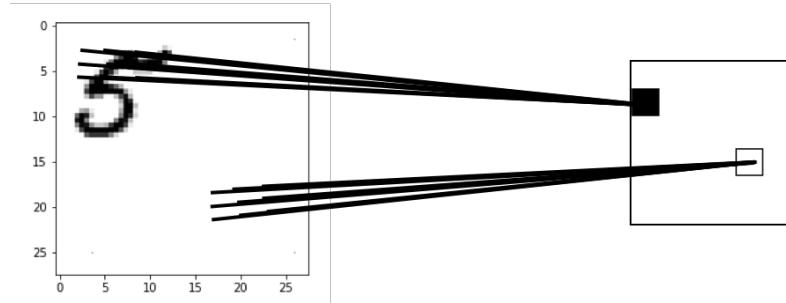
For a grayscale image with a 5x5 feature detector (kernel), we have the following dimensions (number of parameters to learn)

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

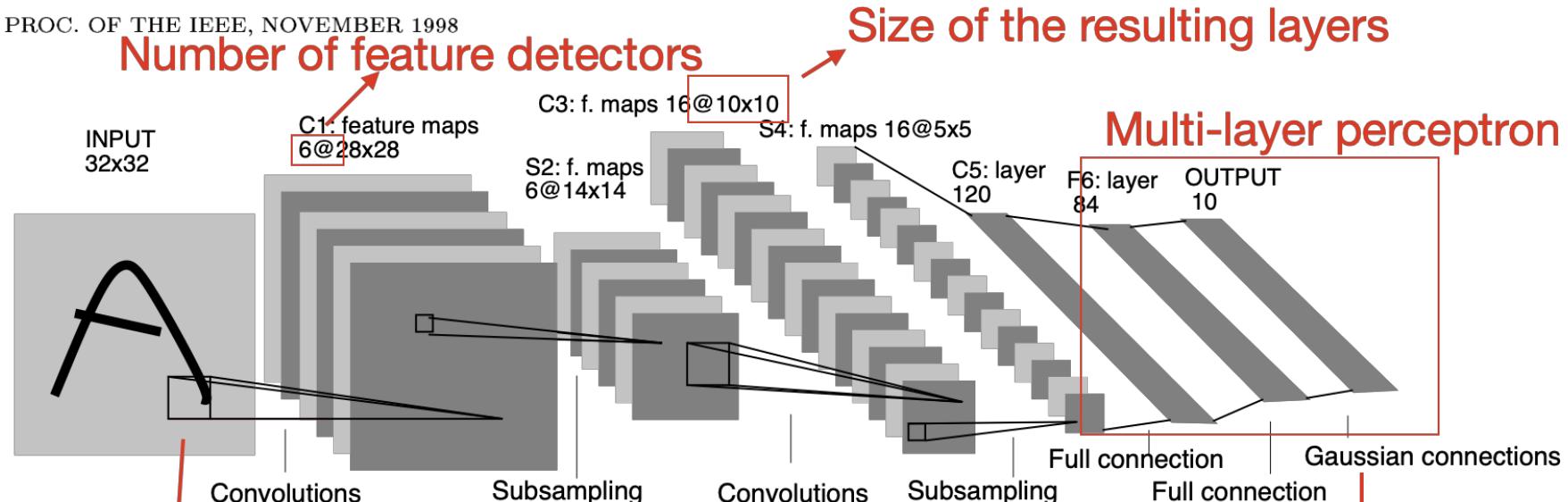


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

"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)



# 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

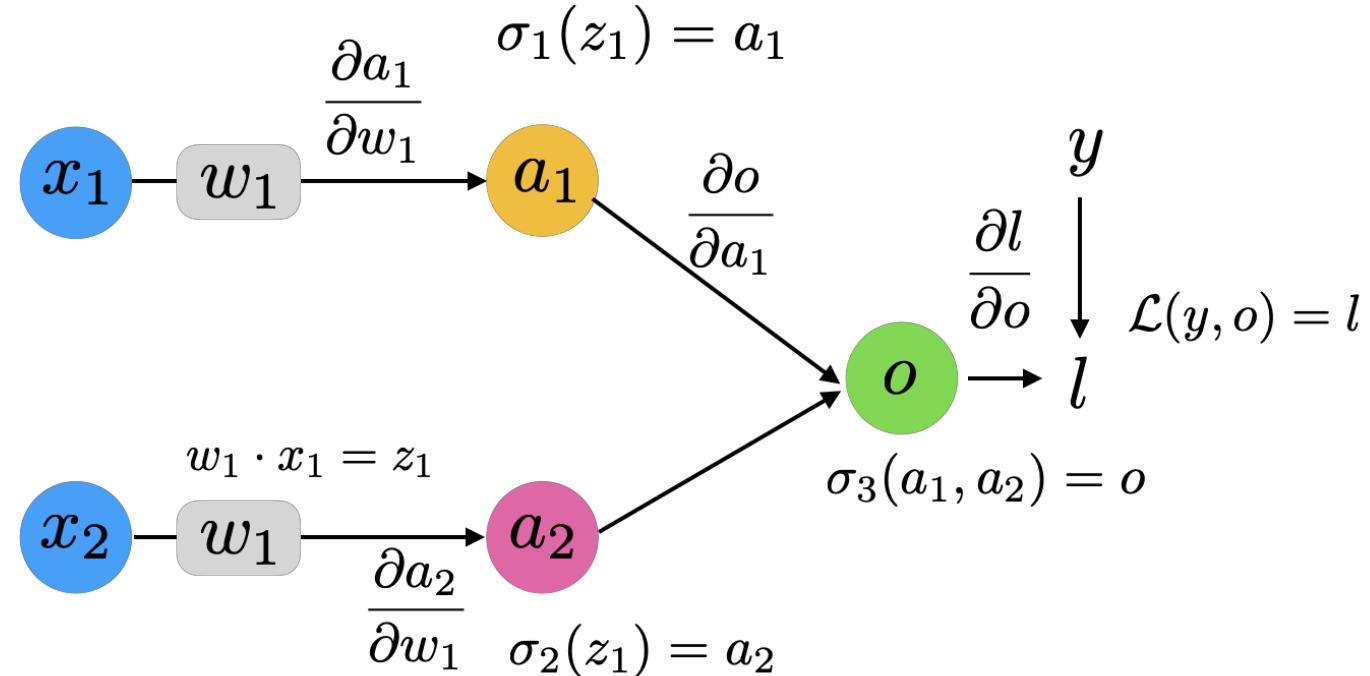


# Backpropagation in CNNs

---

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

# Recall: Weight sharing in computation graphs



Upper path

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

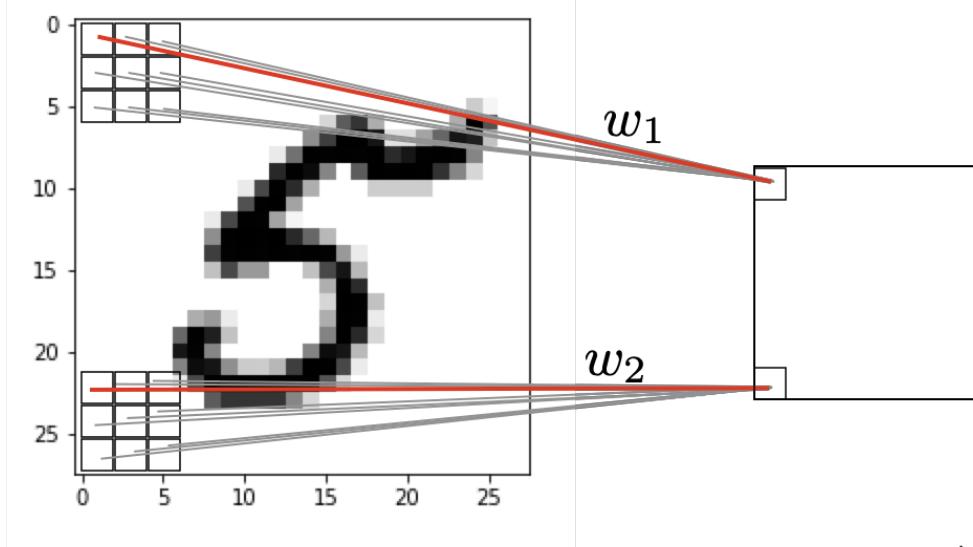
Lower path

# Backpropagation in CNNs

---

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

Due to weight sharing:  $w_1 = w_2$



weight update:

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

Optional averaging



# 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

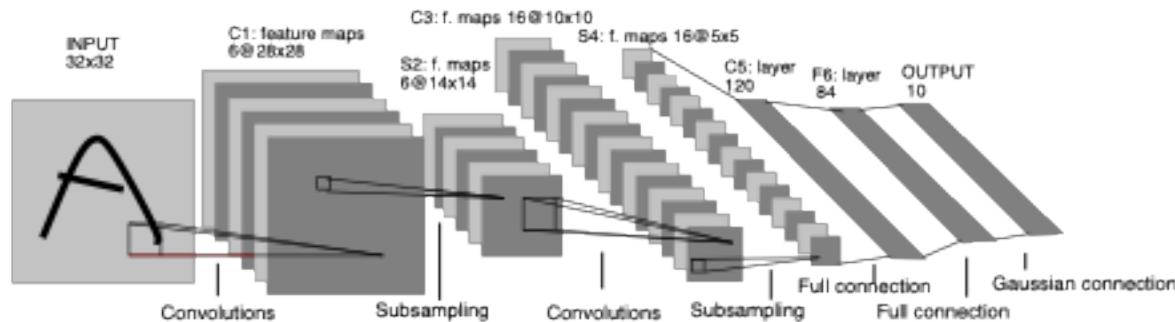


Fig. 2. Architecture of LeNet-5, a Convolutional Neural Network, here for digits recognition. Each plane is a feature map, i.e. a set of units whose weights are constrained to be identical.

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

```
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)
        )

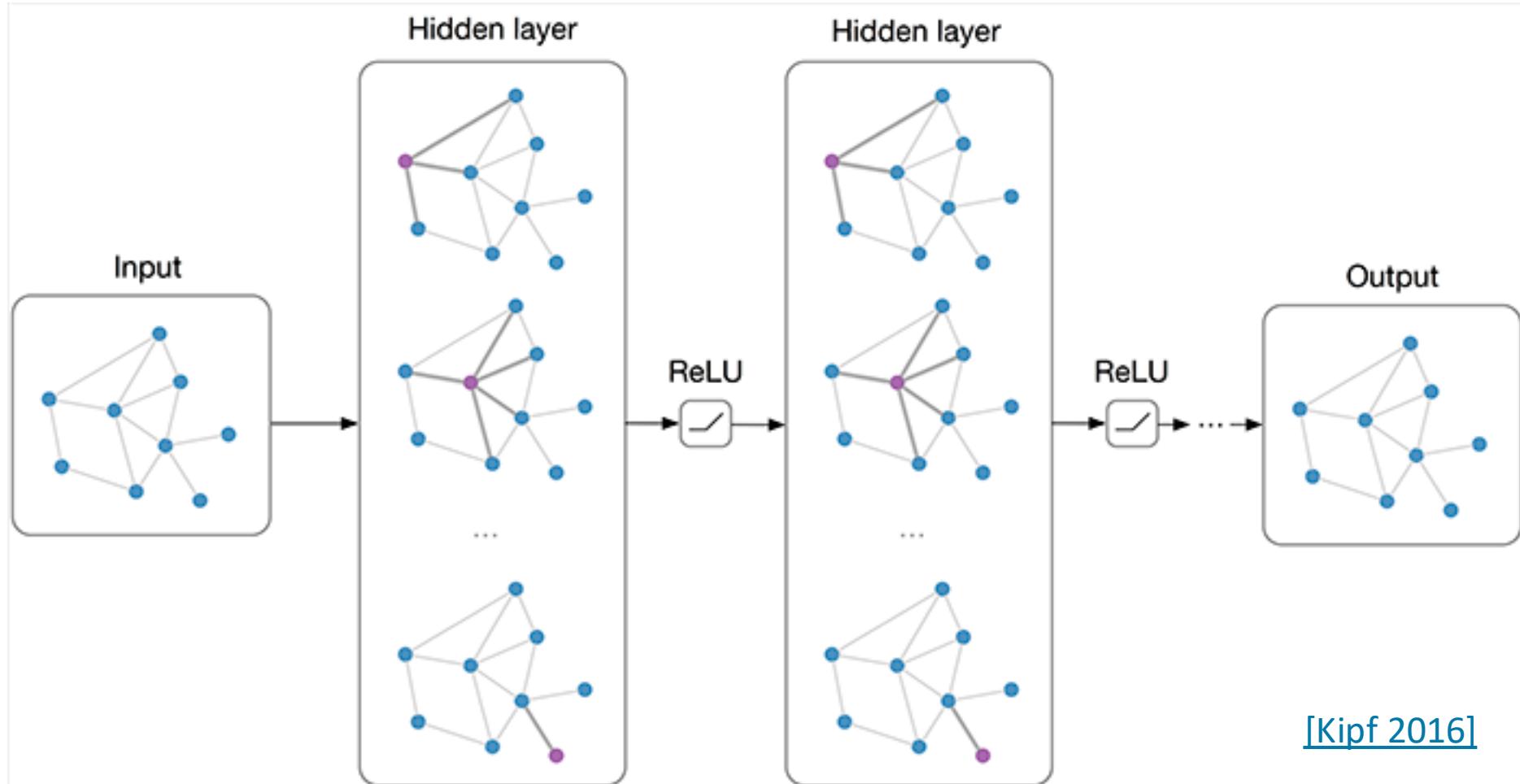
        self.classifier = nn.Sequential(
            nn.Linear(16*5*5, 120),
            nn.Tanh(),
            nn.Linear(120, 84),
            nn.Tanh(),
            nn.Linear(84, num_classes),
        )

    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



Questions?

