Convolutional Neural Networks

Danna Gurari
University of Colorado Boulder
Spring 2024

https://dannagurari.colorado.edu/course/neural-networks-and-deep-learning-spring-2024/
Review

• Last class:
  • Regularization
  • Parameter norm penalty
  • Early stopping
  • Dataset augmentation
  • Dropout
  • Batch normalization
  • Programming tutorial

• Assignments (Canvas):
  • Problem set 2 due Wednesday

• Questions?
Today’s Topics

• Neural Networks for Spatial Data

• History of Convolutional Neural Networks (CNNs)

• CNNs – Convolutional Layers

• CNNs – Pooling Layers

• Programming tutorial
Today’s Topics

• Neural Networks for Spatial Data

• History of Convolutional Neural Networks (CNNs)

• CNNs – Convolutional Layers

• CNNs – Pooling Layers

• Programming tutorial
What is Spatial Data?

- Data where the order matters; e.g.,

- Images
- Audio (spectrogram)
- Text (word embeddings)
- Video
Today’s Topics

• Neural Networks for Spatial Data

• History of Convolutional Neural Networks (CNNs)

• CNNs – Convolutional Layers

• CNNs – Pooling Layers

• Programming tutorial
1847

Gradient descent

1945

First programmable machine

1950

Turing test

1956

AI

1957

Perception

1959

Machine learning

1980

Neuroscientific experiments by Hubel & Weisel to understand how mammalian vision system works

1986

Wave 3: rise of “deep learning”

Nobel Prize in Physiology and Medicine to Hubel and Weisel

1989

Universal approximation paper

1990

Gradient descent

2012

Neural networks with effective learning strategy

Gradient descent
Motivation: How Vision System Works

Image Source: https://braintour.harvard.edu/archives/portfolio-items/hubel-and-wiesel
Motivation: How Vision System Works

Experiment Set-up:

Key Finding: initial neurons responded strongly only when light was shown in certain orientations

V1 physiology: direction selectivity


https://www.cns.nyu.edu/~david/courses/perception/lecturenotes/V1/lgn-V1.html
Motivation: How Vision System Works

Key Idea: cells are organized as a hierarchy of feature detectors, with higher level features responding to patterns of activation in lower level cells.

Source: https://bruceoutdoors.files.wordpress.com/2017/08/hubel.jpg
Motivation: How Vision System Works

https://neuwritesd.files.wordpress.com/2015/10/visual_stream_small.png
Historical Context: Key Ingredients

- **1847**: Gradient descent
- **1945**: First programmable machine
- **1950**: Turing test
- **1956-1959**: Machine learning
- **1957**: Perceptron
- **1959**: AI
- **1960**: Wave 3: rise of “deep learning”
- **1980**: Neural networks with effective learning strategy
- **1986**: Universal approximation paper
- **1989**: Nobel Prize in Physiology and Medicine to Hubel and Weisel
- **1986**: Wave 3: rise of “deep learning”

Additional notes:
- **1986**: Wave 3: rise of “deep learning”
- **1989**: Nobel Prize in Physiology and Medicine to Hubel and Weisel
- **1989**: Neocognitron: convolutional layers and downsampling layers
- **1989**: Neural networks with effective learning strategy
- **1986**: Universal approximation paper

Neuroscientific experiments to understand how mammalian vision system works.
Neocognitron: Key Ingredients

“In this paper, we discuss how to synthesize a neural network model in order to endow it an ability of pattern recognition like a human being... the network acquires a similar structure to the hierarchy model of the visual nervous system proposed by Hubel and Wiesel.”


http://personalpage.flsi.or.jp/fukushima/index-e.html
Neocognitron: Key Ingredients

Cascade of simple and complex cells:

Fig. 1. Correspondence between the hierarchy model by Hubel and Wiesel, and the neural network of the neocognitron

Fig. 2. Schematic diagram illustrating the interconnections between layers in the neocognitron

Fukushima, 1980.
Neocognitron: Key Ingredients

Simple cells extract local features using a sliding filter:

Fig. 1. Correspondence between the hierarchy model by Hubel and Wiesel, and the neural network of the neocognitron.

Fig. 2. Schematic diagram illustrating the interconnections between layers in the neocognitron.
Neocognitron: Key Ingredients

Complex cells fire when any part of the local region is the desired pattern.

Fig. 1. Correspondence between the hierarchy model by Hubel and Wiesel, and the neural network of the neocognitron.

Fig. 2. Schematic diagram illustrating the interconnections between layers in the neocognitron.
Neocognitron: Key Ingredients

1. Convolutional layers

- modifiable synapses
- unmodifiable synapses

2. Pooling Layers

Note: modern networks similarly alternate between these two types of layers!

Fig. 1. Correspondence between the hierarchy model by Hubel and Wiesel, and the neural network of the neocognitron

Fig. 2. Schematic diagram illustrating the interconnections between layers in the neocognitron

Fukushima, 1980.
Today’s Topics

• Neural Networks for Spatial Data

• History of Convolutional Neural Networks (CNNs)

• CNNs – Convolutional Layers

• CNNs – Pooling Layers

• Programming Tutorial
Motivation: Fully-Connected Layers Are Limited

Each node provides input to each node in the next layer

• Assume 3-layer model with 100 nodes, 100 nodes, and then 2 nodes
  • e.g., how many weights are in a 640x480 grayscale image?
    • $640 \times 480 \times 100 + 100 \times 100 + 100 \times 2 = 30,730,200$
  • e.g., how many weights are in a 3.1 Megapixel grayscale image (2048X1536)?
    • $2048 \times 1536 \times 100 + 100 \times 100 + 100 \times 2 = 314,583,000$
Motivation: Fully-Connected Layers Are Limited

Issue: many model parameters in fully connected networks

- Assume 3-layer model with 100 nodes, 100 nodes, and then 2 nodes
  - e.g., how many weights are in a 640x480 grayscale image?
    - 640x480x100 + 100x100 + 100x2 = 30,730,200
  - e.g., how many weights are in a 3.1 Megapixel grayscale image (2048X1536)?
    - 2048x1536x100 + 100x100 + 100x2 = 314,583,000
Motivation: Fully-Connected Layers Are Limited

Many model parameters...
- increases chance of overfitting
- requires more training data
- increases memory/storage requirements

• Assume 3-layer model with 100 nodes, 100 nodes, and then 2 nodes
  • e.g., how many weights are in a 640x480 grayscale image?
    • 640x480x100 + 100x100 + 100x2 = 30,730,200
  • e.g., how many weights are in a 3.1 Megapixel grayscale image (2048X1536)?
    • 2048x1536x100 + 100x100 + 100x2 = 314,583,000
Key Ingredient 1: Convolutional Layers

Rather than have each node provide input to each node in the next layer...

Each node receives input only from a small neighborhood in previous layer (and there is parameter sharing)

Figure Source: https://qph.fs.quoracdn.net/main-qimg-2e1f0071ca9878f7719ed0ea8aeb386d
Fully-Connected vs Convolutional Layers

Fully-connected:

Convolutional:

Convolutional layers dramatically reduce number of model parameters!

Figure Source: https://qph.fs.quoracdn.net/main-qimg-2e1f0071ca9878f7719ed0ea8aeb386d
Key Ingredient 1: Convolutional Layers

INPUT \star FILTER = \text{ReLU} \{ + b \}
Recall: Image Representation (8-bit Grayscale)

https://ai.stanford.edu/~syyeung/cvweb/tutorial1.html
Key Ingredient 1: Convolutional Layers

\[
\text{INPUT} \ast \text{FILTER} = \text{ReLU} \left\{ \text{FILTER} + b \right\}
\]
Convolution: Applies Linear Filter (e.g., 2D)

- Compute a **function of local neighborhood** for each location in matrix
- A **filter** specifies the function for how to combine neighbors’ values

2D Filtering

Matrix:

Slides filter over the matrix and computes dot products

Filtered Result:

https://people.eecs.berkeley.edu/~jrs/189/lec/cnn.pdf
2D Filtering

Matrix:

Filtered Result:

Slides filter over the matrix and computes dot products

https://people.eecs.berkeley.edu/~jrs/189/lec/cnn.pdf
2D Filtering

Slides filter over the matrix and computes dot products

https://people.eecs.berkeley.edu/~jrs/189/lec/cnn.pdf
2D Filtering

Matrix:

Filtered Result:

Slides filter over the matrix and computes dot products

https://people.eecs.berkeley.edu/~jrs/189/lec/cnn.pdf
2D Filtering: Toy Example

Dot Product = \(1 \times 1 + 1 \times 0 + 1 \times 1 + 0 \times 0 + 1 \times 1 + 1 \times 0 + 0 \times 1 + 0 \times 1 + 0 \times 0 + 0 \times 0 + 1 \times 1\)

Dot Product = 4
2D Filtering: Toy Example

Input

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Filter

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Feature Map

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>
2D Filtering: Toy Example

Input

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Filter

<table>
<thead>
<tr>
<th>1</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Feature Map

<table>
<thead>
<tr>
<th>4</th>
<th>3</th>
<th>?</th>
</tr>
</thead>
<tbody>
<tr>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>
2D Filtering: Toy Example

Input

```
1 1 1 0 0
0 1 1 1 0
0 0 1 1 1
0 0 1 1 0
0 1 1 0 0
```

Filter

```
1 0 1
0 1 0
1 0 1
```

Feature Map

```
4 3 4
? ? ?
? ? ?
```
### 2D Filtering: Toy Example

In this example, we have an input feature map and a filter. The feature map is obtained by element-wise multiplication of the input with the filter, followed by summing the result.

**Input:**

```
<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
```

**Filter:**

```
<table>
<thead>
<tr>
<th>1</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
```

**Feature Map:**

```
<table>
<thead>
<tr>
<th>4</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>
```
2D Filtering: Toy Example

Input

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Filter

<table>
<thead>
<tr>
<th>1</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Feature Map

<table>
<thead>
<tr>
<th>4</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
<td>?</td>
</tr>
<tr>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>
2D Filtering: Toy Example

Input

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Filter

<table>
<thead>
<tr>
<th>1</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Feature Map

<table>
<thead>
<tr>
<th>4</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>?</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>
2D Filtering: Toy Example

Input

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Filter

<table>
<thead>
<tr>
<th>1</th>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Feature Map

<table>
<thead>
<tr>
<th>4</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>
2D Filtering: Toy Example

Input

```
1 1 1 0 0
0 1 1 1 0
0 0 1 1 1
0 0 1 1 0
0 1 1 0 0
```

Filter

```
1 0 1
0 1 0
1 0 1
```

Feature Map

```
4 3 4
2 4 3
2 3 ?
```
2D Filtering: Toy Example

Input

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Filter

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Feature Map

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>3</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>
Convolutional Layer

- Many neural network libraries use “convolution” interchangeably with “cross correlation”; for mathematicians, these are technically different
- Examples in these slides show the “cross-correlation” function

Convolutional Layer: Parameters to Learn

Convolutional Layer: Parameters to Learn

• For shown example, how many weights must be learned?
  • 4 (red, blue, yellow, and green values)

• If we instead used a fully connected layer, how many weights would need to be learned?
  • 36 (9 turquoise nodes x 4 magenta nodes)
Neocognitron hard-coded filter values... filter values are learned for CNNs

Convolutional Layer: What Can Filters Do?

Input * Filter (aka – Kernel) = Feature Map

Way to Interpret Neural Network

Convolutional Layer: What Can Filters Do?
Convolutional Layer: What Can Filters Do?

- e.g.,

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>30</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>30</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>30</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>30</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>30</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>30</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>30</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Visualization of Filter

Image Credit: https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/
Convolutional Layer: What Can Filters Do?

- e.g.,

Filter Overlaid on Image

<table>
<thead>
<tr>
<th>0 0 0</th>
<th>0 0 0</th>
<th>0 0 0</th>
<th>0 0 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
<td>0 0 0</td>
</tr>
<tr>
<td>0 0 0</td>
<td>0 0 50</td>
<td>50 50</td>
<td>50 50</td>
</tr>
<tr>
<td>0 0 0</td>
<td>0 0 20</td>
<td>50 50</td>
<td>50 50</td>
</tr>
<tr>
<td>0 0 0</td>
<td>0 0 50</td>
<td>50 50</td>
<td>50 50</td>
</tr>
<tr>
<td>0 0 0</td>
<td>0 0 50</td>
<td>50 50</td>
<td>50 50</td>
</tr>
</tbody>
</table>

Filter

<table>
<thead>
<tr>
<th>0 0 0</th>
<th>0 0 0</th>
<th>0 0 30</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 0 0</td>
<td>0 0 30</td>
<td></td>
</tr>
<tr>
<td>0 0 0</td>
<td>0 0 30</td>
<td></td>
</tr>
<tr>
<td>0 0 0</td>
<td>0 0 30</td>
<td></td>
</tr>
<tr>
<td>0 0 0</td>
<td>0 0 30</td>
<td></td>
</tr>
</tbody>
</table>

Weighted Sum = ?

- Weighted Sum = (50x30) + (20x30) + (50x30) + (50x3) + (50x30)

Weighted Sum = 6600 (Large Number!!)

Image Credit: https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/
Convolutional Layer: What Can Filters Do?

• e.g.,

Image Credit: https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/
Convolutional Layer: What Can Filters Do?

• e.g.,

This Filter is a Curve Detector!

Filter Overlaid on Image (Big Response!)

Filter Overlaid on Image (Small Response!)

Image Credit: https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/
**Convolutional Layer: What Can Filters Do?**

<table>
<thead>
<tr>
<th>Filter</th>
<th>Feature Map</th>
<th>Filter</th>
<th>Feature Map</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Identity</strong></td>
<td></td>
<td><strong>Sharpen</strong></td>
<td></td>
</tr>
</tbody>
</table>
| \[
\begin{bmatrix}
0 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 0 \\
\end{bmatrix}
\] | ![Feature Map](https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/feature_map_identity.png) | \[
\begin{bmatrix}
0 & -1 & 0 \\
-1 & 5 & -1 \\
0 & -1 & 0 \\
\end{bmatrix}
\] | ![Feature Map](https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/feature_map_sharpen.png) |
| **Box blur** (normalized) | | **Gaussian blur** (approximation) | |
| \[
\begin{bmatrix}
1 & 1 & 1 \\
1 & 1 & 1 \\
1 & 1 & 1 \\
\end{bmatrix}
\] | ![Feature Map](https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/feature_map_box_blur_normalized.png) | \[
\begin{bmatrix}
1 & 2 & 1 \\
2 & 4 & 2 \\
1 & 2 & 1 \\
\end{bmatrix}
\] | ![Feature Map](https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/feature_map_gaussian_blur_approximation.png) |
| **Edge detection** | | | |
| \[
\begin{bmatrix}
1 & 0 & -1 \\
0 & 0 & 0 \\
-1 & 0 & 1 \\
\end{bmatrix}
\] | ![Feature Map](https://ujjwalkarn.me/2016/08/11/intuitive-explanation-convnets/feature_map_edge_detection.png) | |
Convolutional Layer: What Can Filters Do?

Filter: 
Sharpen

Image: 
Bell

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>-3</td>
<td>0</td>
</tr>
<tr>
<td>-3</td>
<td>21</td>
<td>-3</td>
</tr>
<tr>
<td>0</td>
<td>-3</td>
<td>0</td>
</tr>
</tbody>
</table>

Divisor: 9

The Matrix

Demo: http://beej.us/blog/data/convolution-image-processing/
Key Ingredient 1: Convolutional Layers

Can choose filters of any size to support feature learning!
Key Ingredient 1: Convolutional Layers

Filtered results are passed, with a bias term, through an activation function to create activation/feature maps.
Key Ingredient 1: Convolutional Layers

Can have multiple filters (with a unique bias parameter per filter)
Key Ingredient 1: Convolutional Layer Summary

Neural networks learn values for all filters and biases in all layers
How Filters Are Applied to Multi-Channel Inputs

e.g., RGB images

Pixel_A = [255, 0, 255]

Pixel_B = [127, 255, 0]

https://www.geeksforgeeks.org/matlab-rgb-image-representation/
How Filters Are Applied to Multi-Channel Inputs

Number of channels in a filter matches that of the input

Convolutional Layers Stacked

Can then stack a sequence of convolution layers; e.g.,

[Diagram shows a sequence of convolutional layers with dimensions and number of filters indicated.]
Convolutional Layers Stacked

Can then stack a sequence of convolution layers, which leads to identifying patterns in increasingly larger regions of the input (e.g., pixel) space:

https://www.deeplearningbook.org/contents/convnets.html
Convolutional Layers Stacked

Can then stack a sequence of convolution layers, which leads to identifying patterns in increasingly larger regions of the input (e.g., pixel) space and mimicking vision system:

Higher level features are constructed by combining lower level features
Problem #1: Input Shrinks

Why do the dimensions shrink with each convolutional layer?

Information is lost around boundary of the input!
Solution: Control Output Size with **Padding**

- **Padding**: add values at the boundaries

Image Credit: https://software.intel.com/en-us/node/586159
Problem #2: Computation Expensive

Many computations to slide filter over every point in the matrix and compute dot products

https://people.eecs.berkeley.edu/~jrs/189/lec/cnn.pdf
Idea: Reduce Computations with Stride

- **Stride**: how many steps taken spatially before applying a filter
  - e.g., 2x2

![Diagram of Image, Filter, and Feature Map](http://deeplearning.net/software/theano/tutorial/conv_arithmetic.html)
Convolutional Layers:
Parameters vs Hyperparameters

• Parameters
  • Weights
  • Biases

• Hyperparameters:
  • Number of filters, including height and width of each
  • Padding type
  • Strides
Today’s Topics

• Neural Networks for Spatial Data

• History of Convolutional Neural Networks (CNNs)

• CNNs – Convolutional Layers

• CNNs – Pooling Layers

• Programming Tutorial
Pooling Layer: Summarizes Neighborhood

- **Max-pooling**: partitions input into a set of non-overlapping rectangles and outputs the maximum value for each chunk.

![Diagram](http://cs231n.github.io/convolutional-networks/#pool)
Pooling Layer: Summarizes Neighborhood

- **Max-pooling**: partitions input into a set of non-overlapping rectangles and outputs the maximum value for each chunk
Pooling Layer: Summarizes Neighborhood

- **Max-pooling**: partitions input into a set of non-overlapping rectangles and outputs the maximum value for each chunk.
Pooling Layer

• Resilient to small translations

• e.g.,
  • Input: all values change (shift right)
  • Output: only half the values change

https://www.deeplearningbook.org/contents/convnets.html
Pooling Layer: Summarizes Neighborhood

- **Max-pooling**: partitions input into a set of non-overlapping rectangles and outputs the maximum value for each chunk
- **Average-pooling**: partitions input into a set of non-overlapping rectangles and outputs the average value for each chunk

http://cs231n.github.io/convolutional-networks/#pool
Pooling Layer: Summarizes Neighborhood

- **Max-pooling**: partitions input into a set of non-overlapping rectangles and outputs the maximum value for each chunk.
- **Average-pooling**: partitions input into a set of non-overlapping rectangles and outputs the average value for each chunk.

![Image of pooling operation](http://cs231n.github.io/convolutional-networks/#pool)
Pooling Layer: Summarizes Neighborhood

• **Max-pooling**: partitions input into a set of non-overlapping rectangles and outputs the maximum value for each chunk

• **Average-pooling**: partitions input into a set of non-overlapping rectangles and outputs the average value for each chunk

• And many more pooling options
  • E.g., listed here https://pytorch.org/docs/stable/......tml#pooling-layers
Pooling for Multi-Channel Input

Pooling is applied to each input channel separately

Pooling Layer: Benefits

• Builds in invariance to translations of the input

• Reduces memory requirements

• Reduces computational requirements
Today’s Topics

• Neural Networks for Spatial Data

• History of Convolutional Neural Networks (CNNs)

• CNNs – Convolutional Layers

• CNNs – Pooling Layers

• Programming Tutorial
Today’s Topics

- Neural Networks for Spatial Data
- History of Convolutional Neural Networks (CNNs)
- CNNs – Convolutional Layers
- CNNs – Pooling Layers
- Programming Tutorial
The End