94-775/95-865 Lecture 11: Image Analysis With Convolutional Neural Nets

George Chen
Quiz Results

95-865 Spring 2019 Quiz Score Histogram

Mean 56.1, std dev 14.2, max 95.5
Questionnaire Results

How many hours did you take (roughly) to complete homework 1?

Hours

Raw count
How many hours did you take (roughly) to complete homework 2?
Questionnaire Results

• Some people want to see more demos

• Some people want to see more math

• Some people want to see more algorithms

A mini is quite short—can’t have more of everything…
Announcements

• Start HW3 (takes something like 50% longer than HW2)

• Yes, AWS takes a while to get used to

• Quiz regrades: due Monday 11:59pm

• Some past final exams have been posted (an additional one in recitation this Friday)
Deep Learning

- Inspired by biological neural nets but otherwise not the same at all (biological neural nets do not work like deep nets)
- Learns a layered representation
  - Tries to get rid of manual feature engineering
  - Need to design constraints for what features are learned to account for structure in data (e.g., images, text, …)

“clown fish”
Learning a neural net amounts to curve fitting

We’re just estimating a function
Neural Net as Function Approximation

Given input, learn a computer program that computes output

Single-layer neural net example:

```python
def f(input):
    output = softmax(np.dot(input, W) + b)
    return output
```

We are fixing what the function $f$ looks like in code and are only adjusting $W$ and $b$!!!
Neural Net as Function Approximation

Given input, learn a computer program that computes output.

Single-layer neural net example:

\[
\text{output} = \text{softmax}(\text{np.dot(input, W) + b})
\]

Two-layer neural net example:

\[
\text{layer1_output} = \text{relu}(\text{np.dot(input, W1) + b1})
\]

\[
\text{output} = \text{softmax}(\text{np.dot(layer1_output, W2) + b2})
\]

Learning a neural net: learning a simple computer program that maps inputs (raw feature vectors) to outputs (predictions)
Architecting Neural Nets

• Increasing number of layers (depth) makes neural net more complex
  • Can approximate more functions
  • More parameters needed
    • More training data may be needed

• Designing neural net architectures is a bit of an art
  • How to select the number of neurons for intermediate layers?
  • Very common in practice: modify existing architectures that are known to work well (e.g., VGG-16 for computer vision/image processing)
Image analysis with Convolutional Neural Nets (CNNs, also called convnets)
Convolution

Slide by Phillip Isola
Convolution

Input image

Filter
(also called “kernel”)
Convolution

Input image

Filter (also called “kernel”)

```
0 0 0
0 1 0
0 0 0
```
Convolution

Take dot product!

Input image

Output image
Convolution

Take dot product!

Input image

Output image
Convolution

Take dot product!

Input image

Output image
**Convolution**

Take dot product!

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

Input image

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

Output image

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

## Take dot product!

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

Input image

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

Output image

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

Take dot product!

Input image

Output image
Convolution

Take dot product!

Input image

Output image
Convolution

Input image

Output image

Note: output image is smaller than input image
If you want output size to be same as input, pad 0’s to input
Convolution

Input image

Output image

Note: output image is smaller than input image
If you want output size to be same as input, pad 0’s to input
# Convolution

**Input image**

<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>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</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>1</td>
<td>1</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>

**Convolution Filter**

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

**Output image**

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

Input image

Output image
Convolution

![Convolution Diagram](image-url)

**Input image**

**Output image**
Convolution

Very commonly used for:

- Blurring an image

\[
\begin{array}{ccc}
\frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\
\frac{1}{9} & \frac{1}{9} & \frac{1}{9} \\
\frac{1}{9} & \frac{1}{9} & \frac{1}{9}
\end{array}
\]

- Finding edges

\[
\begin{array}{ccc}
-1 & -1 & -1 \\
2 & 2 & 2 \\
-1 & -1 & -1
\end{array}
\]

(\text{this example finds horizontal edges})

Images from: http://aishack.in/tutorials/image-convolution-examples/
Convolution Layer

Images from: http://aishack.in/tutorials/image-convolution-examples/

filters are actually unknown and are learned!

convolve with each filter

activation (e.g., ReLU)
Convolution Layer

Input image

conv2d layer
with ReLu activation
and three 3x3 kernels

Output images

Images from: http://aishack.in/tutorials/image-convolution-examples/
Convolution Layer

Input image
dimensions: height, width

conv2d layer
with ReLu activation
and three 3x3 kernels

Stack output images into a single “output feature map”
dimensions: height-2, width-2, number of kernels (3 in this case)

Images from: http://aishack.in/tutorials/image-convolution-examples/
Convolution Layer

Input image
dimensions: height, width

conv2d layer
with ReLu activation
and $k$ 3x3 kernels

Stack output images into a single “output feature map”
dimensions: height-2, width-2, $k$

Images from: http://aishack.in/tutorials/image-convolution-examples/
Convolution Layer

Input image

dimensions: height, width, depth $d$ (# channels)

conv2d layer
with ReLu activation
and $k$ 3x3x$d$ kernels

technical detail: there's also a bias vector

Stack output images into a single “output feature map”
dimensions: height-2, width-2, $k$

Images from: http://aishack.in/tutorials/image-convolution-examples/
Pooling

• Aggregate local information

• Produces a smaller image
  (each resulting pixel captures some “global” information)

• If object in input image shifts a little, output is the same
Max Pooling

Input image

\[
\begin{array}{cccccc}
0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 1 & 1 & 1 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 \\
0 & 1 & 1 & 1 & 1 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 \\
0 & 0 & 1 & 1 & 1 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 \\
\end{array}
\]

\[
\begin{array}{cccc}
-1 & -1 & -1 & \\
2 & 2 & 2 & \\
-1 & -1 & -1 & \\
\end{array}
\]

\[
\begin{array}{cccccc}
0 & 1 & 3 & 1 & 0 \\
1 & 1 & 1 & 3 & 3 \\
0 & 0 & -2 & -4 & -4 \\
1 & 1 & 1 & 3 & 3 \\
0 & 1 & 3 & 1 & 0 \\
\end{array}
\]
### Max Pooling

**Input image**

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

**Output image after ReLU**

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>3</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>-2</td>
<td>-4</td>
<td>-4</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

**Output after max pooling**

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>3</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>
Max Pooling

Input image

Output image after ReLU

Output after max pooling

Input image:

Output image:

Output after max pooling:
Max Pooling

Input image

Output image after ReLU

Output after max pooling
### Max Pooling

#### Input Image

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

#### Kernel

<table>
<thead>
<tr>
<th>-1</th>
<th>-1</th>
<th>-1</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
</tbody>
</table>

#### Output Image after ReLU

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>3</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>-2</td>
<td>-4</td>
<td>-4</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
</tbody>
</table>

#### Output after Max Pooling

<table>
<thead>
<tr>
<th>1</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
Max Pooling

Input image

Output after ReLU

Output after max pooling
### Max Pooling

| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

**Input image**

What numbers were involved in computing this 1?

In this example: 1 pixel in max pooling output captures information from 16 input pixels!

Example: applying max pooling again results in a single pixel that captures info from entire input image!
Max Pooling and (Slight) Shift Invariance

Small shift of object in input image results in same output.
Max Pooling and (Slight) Shift Invariance

Big shift in input can still change output
Basic Building Block of CNN’s

Input image → stack of images → output stack of smaller images

conv2d layer with ReLu activation and $k$ kernels → max pooling (applied to each image in stack)

Images from: http://aishack.in/tutorials/image-convolution-examples/
Handwritten Digit Recognition

Training label: 6

28x28 image
length 784 vector (784 input neurons)

dense layer with 512 neurons, ReLU activation

dense layer with 10 neurons, softmax activation

Loss/“error”

Popular loss function for classification (> 2 classes): categorical cross entropy

\[
\log \left( \frac{1}{\Pr(\text{digit 6})} \right)
\]

Error is averaged across training examples

Learning this neural net means learning parameters of both dense layers!
Handwritten Digit Recognition

Training label: 6

28x28 image → conv2d, ReLU → max pooling 2d → dense, softmax → Loss/“error” → error
Handwritten Digit Recognition

Training label: 6

28x28 image

extract low-level visual features & aggregate

conv2d, ReLU, max pooling

extract higher-level visual features & aggregate

conv2d, ReLU, max pooling

dense, softmax

non-vision-specific classification neural net

Loss

to error
CNN Demo
CNN’s

- Learn convolution filters for extracting simple features.

- Max pooling summarizes information and produces a *smaller* output and is invariant to small shifts in input objects.

- Can then repeat the above two layers to learn features from increasingly higher-level representations.