95-865 Unstructured Data Analytics
Lecture 12: Wrap up neural net basics; image analysis with convolutional neural nets (also called CNNs or convnets)

Slides by George H. Chen
Handwritten Digit Recognition

Demo
Architecting Neural Nets

• Basic building block that is often repeated: linear layer followed by nonlinear activation

• Without nonlinear activation, two consecutive linear layers is mathematically equivalent to having a single linear layer!

• How to select # of nodes in a layer, or # of layers?
  • These are hyperparameters! Infinite possibilities!
  • Choose between different hyperparameter settings by using the strategy from last lecture (choose based on validation accuracy)
    • Very expensive in practice!
      (Active area of research: neural architecture search)

• Much more common in practice: modify existing architectures that are known to work well
  (e.g., ResNet for image classification/object recognition)
PyTorch Examples

This page lists various PyTorch examples that you can use to learn and experiment with PyTorch.

1. **Image Classification using Vision Transformer**
   - This example shows how to train a Vision Transformer from scratch on the CIFAR10 database.
   - [GO TO EXAMPLE](#)

2. **Image Classification Using ConvNets**
   - This example demonstrates how to run image classification with Convolutional Neural Networks ConvNets on the MNIST database.
   - [GO TO EXAMPLE](#)

3. **Measuring Similarity using Siamese Network**
   - This example demonstrates how to measure similarity between two images using Siamese.

4. **Word-level Language Modeling using RNN and Transformer**
   - This example demonstrates how to train a multi-
Find a Massive Collection of Models at the Model Zoo
Learning a neural net amounts to "curve fitting"

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

Given \textit{input}, learn a computer program that computes \textit{output}.

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

Multinomial logistic regression:

- The only things that we are learning are \( W \) and \( b \)!

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

Multinomial logistic regression:

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

Multilayer perceptron:

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

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

Learning a neural net: learning a simple computer program that maps inputs (raw feature vectors) to outputs (predictions)
Complexity of a Neural Net?

Increasing number of layers (depth) makes neural net more “complex”

⟹ Learn computer program that has more lines of code

Earlier: MLP had more parameters than logistic regression

Upcoming: we’ll see an example where a deeper network has fewer parameters than a shallower one
Accounting for image structure: convolutional neural nets (CNNs or convnets)
Convolution

filter

Slide by Phillip Isola
Convolution

Input image

Filter (also called "kernel")
Convolution

Input image

Filter (also called “kernel”)
Convolution

Take dot product!

Input image

Output image
Convolution

Take dot product!

Input image

Output image
Convolution

Take dot product!

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

Input image

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

Output image
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!

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

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

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

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

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

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<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>
<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>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>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>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>

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<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>
<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>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>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>

**Input image**

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

**Output image**

\[
\begin{array}{cccc}
0 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 0 \\
\end{array}
\]
Convolution

Input image

Output image
### Convolution

#### Input image

<p>| | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<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>
<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>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>

#### Output image

\[
\begin{pmatrix}
  0 & 1 & 3 & 1 & 0 \\
  1 & 1 & 1 & 3 & 3 \\
  0 & 0 & -2 & -4 & -4 \\
  1 & 1 & 1 & 3 & 3 \\
  0 & 1 & 3 & 1 & 0 \\
\end{pmatrix}
\]

\[
\begin{pmatrix}
  -1 & -1 & -1 \\
  2 & 2 & 2 \\
  -1 & -1 & -1 \\
\end{pmatrix}
\]

\[
\begin{pmatrix}
  0 & 1 & 3 & 1 & 0 \\
  1 & 1 & 1 & 3 & 3 \\
  0 & 0 & -2 & -4 & -4 \\
  1 & 1 & 1 & 3 & 3 \\
  0 & 1 & 3 & 1 & 0 \\
\end{pmatrix}
\]
Convolution

Very commonly used for:

- Blurring an image

![Blurring example]

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

\( \ast \)

![Blurred image]

- Finding edges

![Finding edges example]

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

\( \ast \)

![Edges image]

(this example finds horizontal edges)

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

- **Conv2d layer**: Images from: http://aishack.in/tutorials/image-convolution-examples/

  - Convolve with each filter
  - Filters & biases (1 bias number per filter) are unknown and are learned!

- **Activation layer (such as ReLU)**: Add bias, apply activation

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

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

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

Input
- shape: 1 (# channels), height, width

Conv2d
- 3 kernels, each size 3x3
- ReLU activation

Stack output
- images into a single “output feature map”
- shape: 3, height-2, width-2
Convolution Layer

Input shape:
1 (# channels),
height, width

Conv2d (k kernels each size 3x3),
ReLU activation

Stack output images into a single “output feature map”

shape:
k,
height-2,
width-2

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

Input

- shape: $d$ (# channels)
- height, width

Conv2d

- $(k$ kernels, each size $d \times 3 \times 3$), ReLU activation

Stack output images into a single “output feature map”

- shape: $k$, height-2, width-2

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

Input shape: 
- \(d\) (# channels)
- height, width

Conv2d
- \(k\) kernels, each size \(d \times 3 \times 3\), ReLU activation

Stack output images into a single "output feature map"
- shape: 
  - \(k\), height-2, width-2

Each filter:
- \(d\)
Pooling

- Produces smaller image summarizing original larger image

- To produce this smaller image, need to aggregate or “pool” together information

- If “object” in input image shifts by a little bit, want output to stay about the same
Max Pooling

Called “2-by-2” max pooling since this green box is 2 rows by 2 columns

Take maximum value

Input image

Output image

3-by-4 max pooling would mean that the green box is 3 rows by 4 columns, etc
Max Pooling

Called “2-by-2” max pooling since this green box is 2 rows by 2 columns

Take maximum value

3-by-4 max pooling would mean that the green box is 3 rows by 4 columns, etc
Max Pooling

Called “2-by-2” max pooling since this green box is 2 rows by 2 columns

Input image

Output image

3-by-4 max pooling would mean that the green box is 3 rows by 4 columns, etc
Max Pooling

Called “2-by-2” max pooling since this green box is 2 rows by 2 columns

3-by-4 max pooling would mean that the green box is 3 rows by 4 columns, etc
Max Pooling

Called “2-by-2” max pooling since this green box is 2 rows by 2 columns

<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>

Input image

Take maximum value

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

Output image

3-by-4 max pooling would mean that the green box is 3 rows by 4 columns, etc
Max Pooling

Called “2-by-2” max pooling since this green box is 2 rows by 2 columns

Input image

Output image

3-by-4 max pooling would mean that the green box is 3 rows by 4 columns, etc
**Max Pooling After Convolution & ReLU**

Convolution layer (1 filter, for simplicity no bias, i.e., bias = 0)

<table>
<thead>
<tr>
<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>
</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>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>
</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>

Input

<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>

Convolution layer (1 filter, for simplicity no bias, i.e., bias = 0)

<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>
Max Pooling After Convolution & ReLU

Convolution layer (1 filter, for simplicity no bias, i.e., bias = 0)

Input:

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

Convolution kernel:

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

Output after convolution:

\[
\begin{array}{cccccccc}
0 & 0 & 1 & 3 & 1 & 0 & 0 & 0 \\
1 & 1 & 1 & 3 & 3 & 3 & 3 & 3 \\
0 & 0 & -2 & -4 & -4 & 0 & 0 & 0 \\
1 & 1 & 1 & 3 & 3 & 3 & 3 & 3 \\
0 & 1 & 3 & 1 & 0 & 0 & 0 & 0 \\
0 & 1 & 3 & 1 & 0 & 0 & 0 & 0 \\
\end{array}
\]

Output after ReLU:

\[
\begin{array}{cccc}
0 & 1 & 3 & 1 \\
1 & 1 & 1 & 3 \\
1 & 1 & 1 & 3 \\
0 & 1 & 3 & 1 \\
\end{array}
\]

Output after 2-by-2 max pooling:

\[
\begin{array}{cccc}
0 & 1 & 3 & 1 \\
1 & 1 & 1 & 3 \\
1 & 1 & 1 & 3 \\
0 & 1 & 3 & 1 \\
\end{array}
\]
Max Pooling After Convolution & ReLU

Convolution layer (1 filter, for simplicity no bias, i.e., bias = 0)

Input

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

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

\[
\begin{array}{cccc}
0 & 1 & 3 & 1 & 0 \\
1 & 1 & 1 & 3 & 3 \\
0 & 0 & -2 & -4 & -4 \\
1 & 1 & 1 & 3 & 3 \\
0 & 1 & 3 & 1 & 0 \\
\end{array}
\]

Output image after ReLU

Output after 2-by-2 max pooling

1
**Max Pooling After Convolution & ReLU**

Convolution layer (1 filter, for simplicity no bias, i.e., bias = 0)

Input:

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

Convolution 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 after convolution:

<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 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>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>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 2-by-2 max pooling:

<table>
<thead>
<tr>
<th>1</th>
<th>3</th>
</tr>
</thead>
</table>

Output 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>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>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 After Convolution & ReLU

Convolution layer (1 filter, for simplicity no bias, i.e., bias = 0)

Input

<table>
<thead>
<tr>
<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>
</tr>
<tr>
<td>0</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>
</tr>
<tr>
<td>0</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>
</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 layer kernel:

| -1 | -1 | -1 |
| 2  | 2  | 2  |
| -1 | -1 | -1 |

Output 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 image after ReLU

| 0 | 1 | 3 | 1 | 0 |
| 1 | 1 | 1 | 3 | 3 |
| 0 | 0 | 0 | 0 | 0 |
| 1 | 1 | 1 | 3 | 3 |
| 0 | 1 | 3 | 1 | 0 |

Output after 2-by-2 max pooling

<table>
<thead>
<tr>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
</tr>
<tr>
<td>1</td>
</tr>
</tbody>
</table>
Max Pooling After Convolution & ReLU

Convolution layer (1 filter, for simplicity no bias, i.e., bias = 0)

Input

Output image after ReLU

Output after 2-by-2 max pooling
Max Pooling After Convolution & ReLU

Convolution layer (1 filter, for simplicity no bias, i.e., bias = 0)

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!
Small Shifts & Max Pooling

Small shift in input object of interest results in same output
Small Shifts & Max Pooling

A bigger shift in the input results in a different output
Common Building Block of CNNs

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

Training label: 6

Input

Flatten (512 nodes), ReLU

Linear (10 nodes), Softmax

Loss

Categorical cross entropy

error
Handwritten Digit Recognition

Input: Conv2d, ReLU → Max Pool 2d → Flatten → Linear (10 nodes), Softmax → Loss → error

Training label: 6

Categorical cross entropy
Handwritten Digit Recognition

Input

Conv2d, ReLU
Max Pool 2d
Conv2d, ReLU
Max Pool 2d
Flatten
Linear (10 nodes), Softmax

extract low-level visual features & aggregate
extract higher-level visual features & aggregate

non-vision-specific classifier

Loss → error

Categorical cross entropy

Training label: 6
CNNs

Demo