Lecture 15: Convolutional Neural Networks

Mar 13, 2023
CIS 4190/5190
Spring 2023
Course Progress

Till now: foundational algorithms applicable to large classes of machine learning problems.

Going forward: applications to specific types of data and specific types of problems.

• New Types of Data: Grids (e.g. Images), Sequences (e.g. Language)
• New Types of Problems: Making Sequences of Decisions (e.g. Robotics), Recommendation Systems
• Ethics
Other Types of Data

• Until now, the $i^{\text{th}}$ sample in our dataset was either naturally a vector $x_i$ or we converted it into one.

• What if our data samples were more naturally expressed in a different structure?
  - $x_i$ is a “grid”: e.g. images
  - $x_i$ is a “sequence”: e.g. text
  - $x_i$ is a “graph”: e.g. protein structure
Neural Networks Specialized to Grid Data

• We will study a class of neural networks called convolutions that specialize to properties often present in *grid* data, particularly images.

Spectrogram encoding of audio

Digital image
Images as 2D Arrays

What we see

What a computer sees

Computer vision:

How to extract meaning out of these 2D arrays?

Note: for color images, a stack of (typically 3) 2D arrays, each called a “channel”.

Source: S. Narasimhan, S. Lazebnik
Color Images Are 3D Arrays with 2 Spatial Dimensions

We will see: convenient to deal with the *spatial dimensions* separately, and there are still only two of those.
What Info can be Extracted from Images?

Source: S. Lazebnik
What Info can be Extracted from Images?

Source: S. Lazebnik
What Info can be Extracted from Images?

- **geometric information**
  - building
  - roof
  - door
  - person
  - trashcan
  - ground

- **semantic information**
  - tree
  - sky
  - chimney
  - building
  - window
  - car
  - Outdoor scene
  - City
  - European

Source: S. Lazebnik
Vision is Deceptively Hard!

In the 1960s, Marvin Minsky assigned a couple of undergrads to spend the summer programming a computer to use a camera to identify objects in a scene. He figured they'd have the problem solved by the end of the summer.

Half a century later, we're still working on it.
Vision often involves making educated guesses.

“This is not a pipe”

The Treachery of Images – Rene Magritte
ML in Computer Vision

The very old: 1960’s - Mid 1990’s
Image $\rightarrow$ hand-def. features $\rightarrow$ hand-def. classifier

The old: Mid 1990’s – 2012
Image $\rightarrow$ hand-def. features $\rightarrow$ learned classifier
What Should Good Visual Representations Do?

Image  →  ?  →  $D$-length feature $x$
What Should Good Visual Representations Do?

What is a “good” feature space?

Good features make useful tasks easy to perform.
What Should Good Visual Representations Do?

How should we produce such good features?

Image

D-length feature $x$

ML model

“How Dog”
Visual Features Before Deep Learning
Most Feature Extraction Frameworks Pre-2012

• Step 1: Focus on “interest points” rather than all pixels
  ▪ E.g. corner points, “difference of gaussians”, or even a uniform grid
• Step 2: Compute features at interest points.
  ▪ E.g. “SIFT”, “HOG”, “SURF”, “GIST”, etc.
• Step 3: Convert to fixed-dimensional feature vector by measuring statistics of the features such as histograms
  ▪ E.g. “Bag of Words”, “Spatial Pyramids”, etc.

See libraries like VLFeat and OpenCV

Use your favorite ML model now!
Successes of ML for Vision Pre-2012

Viola-Jones face detector (with AdaBoost!) ~2000

Deformable Parts Model object detection (with SVMs!) ~2010

GIST Scene retrieval (with nearest neighbors!) ~2006

https://github.com/alexdemartos/ViolaAndJones
ML in Computer Vision

The very old: 1960’s - Mid 1990’s
Image $\rightarrow$ hand-def. features $\rightarrow$ hand-def. classifier

The old: Mid 1990’s – 2012
Image $\rightarrow$ hand-def. features $\rightarrow$ learned classifier

The new: 2012 – ?
Image $\rightarrow$ jointly learned features + classifier with “deep” multi-layer neural networks
Representation Learning for Images

Convolutional Neural Networks
What is Different Now?

The very old: 60's - Mid 90's
Image → hand-def. features → hand-def. classifier

The old: Mid 90's – 2012
Image → hand-def. features → learned classifier

The new: 2012 – ?
Image → jointly learned features + classifier

Answer: Representation learning
“Deep” Learning

• “Deep” multi-layer neural networks are representation learners.
• Every layer improves upon its preceding layer, tailoring the representation to the task.
Impact of Deep Learning in Computer Vision

ImageNet 1000-object category recognition challenge

But the neural networks you have seen so far won’t work well on images!
What’s special about images?

• Images are special. Why?
• Bad news: They are very high-dimensional, which makes all ML harder.
• Good news: We don’t have to treat images as just vectors of pixels. We know more about them, and can exploit that knowledge.
Structure in Images

• 2D image structure
  ▪ Location associations and spatial neighborhoods are meaningful
  ▪ So far, we can shuffle the features without changing the problem (e.g., $\beta^T x$)
  ▪ Not true for images!
Structure in Images

• Translation invariance
  ▪ Consider image classification (e.g., labels are cat, dog, etc.)
  ▪ **Invariance**: If we translate an image, it does not change the category label

Source: Ott et al., Learning in the machine: To share or not to share?
Structure in Images

- **Translation equivariance**
  - Consider object detection (e.g., find the position of the cat in an image)
  - **Equivariance:** If we translate an image, the object is translated similarly.

We will exploit this through image-specific operations in neural networks.
“Image”-Specific Operators/Layers

• We want to retain useful **location associations**, and exploit **translation invariance** and **equivariance**.

• Two key operations in neural networks for images:

  ![Convolution layers](https://towardsdatascience.com/types-of-convolutions-in-deep-learning-717013397f4d)
  ![Pooling layers](https://peltarion.com/static/2d_max_pooling_pa1.png)

  **Convolution layers**
  (capture equivariance)

  **Pooling layers**
  (capture invariance)
Convolutions Beyond Photographs

• Recall: convolutions try to gather useful location associations, and exploit translation invariance and equivariance.

• These properties are useful beyond just photographic images. Need not even be 2D grids.
  - E.g. detecting spikes in a time series of stock prices, or an audio stream. (1-D)
    - Also important to retain location associations
    - Local operations, invariance, equivariance.

  - Can also apply in higher dimensions. E.g. convolving over a 3D “grid” of voxels to detect objects.

https://towardsdatascience.com/types-of-convolutions-in-deep-learning-717013397f4d
https://peltarion.com/static/2d_max_pooling_pa1.png
Convolution
Convolution Filters: Template Matching over an Image

• Intuitively, convolutional filters search for local patterns that resemble the filters themselves.

• Suppose you are given a convolution filter like this. (later, we will learn filters)
Convolutional filtering in 1D

• Suppose your input $x$ is a 1-D sequence, such as a time sequence, e.g. the stock market: $x = [25000, 28000, 30000, 21000, 18000, ... ]$

• Given a “kernel” sequence, e.g. $k = [-1, 1, -1]$

• Convolution is defined by the following operation:

$$y[t] = \sum_{\tau=0}^{\lvert k \rvert - 1} k[\tau]x[t + \tau]$$

In neural networks, the weights $k$ are learned. (Plus a bias)

\[
\begin{align*}
\end{align*}
\]
Convolutional Filtering in 1D
Convolutional Filtering in 1D

No good positive match, but good *negative* match?
Convolutional Filtering in 1D

https://gitlab.com/brohrer/
Convolutional filtering in 2D

• 1-D convolution is defined by the following operation:

\[
y[t] = \sum_{\tau=0}^{\lvert k \rvert - 1} k[\tau]x[t + \tau]
\]

• With a 2-D signal \( x \) and 2-D \( h \times w \) kernel \( k \), 2-D convolution is defined by the following operation:

\[
y[s, t] = \sum_{\tau=0}^{h-1} \sum_{\gamma=0}^{w-1} k[\tau, \gamma]x[s + \tau, t + \gamma]
\]

Again, in convolutional neural networks, the weights \( k \) will be learned.
Convolutional filtering in 2D

\[
y[s, t] = \sum_{\tau=0}^{h-1} \sum_{\gamma=0}^{w-1} k[\tau, \gamma]x[s + \tau, t + \gamma]
\]

• To compute:
  - Slide kernel over image
  - Take the element-wise multiplication over the window and sum

![Image](https://towardsdatascience.com/intuitively-understanding-convolutions-for-deep-learning-1f6f42faee1)
Example: Edge Detection via Convolution

### Example Edge Detection Kernels

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

- **45 degree lines**

<table>
<thead>
<tr>
<th>135 degree lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1   -1   2</td>
</tr>
<tr>
<td>-1   2   -1</td>
</tr>
<tr>
<td>2   -1   -1</td>
</tr>
<tr>
<td>-1   -1   2</td>
</tr>
</tbody>
</table>

### Result of Convolution with Horizontal Kernel

![Result of Convolution with Horizontal Kernel](https://aishack.in/tutorials/image-convolution-examples/)
Back To Our Example

graphic credit: S. Lazebnik
output[0, 0] = \sum_{\tau=0}^{k-1} \sum_{\gamma=0}^{k-1} \text{filter}[\tau, \gamma] \cdot \text{image}[0 + \tau, 0 + \gamma]


Back To Our Example

\[ \text{output}[0,1] = \sum_{\tau=0}^{k-1} \sum_{\gamma=0}^{k-1} \text{filter}[\tau,\gamma] \cdot \text{image}[0 + \tau, 1 + \gamma] \]

graphic credit: S. Lazebnik
Back To Our Example

\[ \text{output}[0,2] = \sum_{\tau=0}^{k-1} \sum_{\gamma=0}^{k-1} \text{filter}[\tau, \gamma] \cdot \text{image}[0 + \tau, 2 + \gamma] \]
Back To Our Example

\[
\text{output}[i, j] = \sum_{\tau=0}^{k-1} \sum_{\gamma=0}^{k-1} \text{filter}[^\tau, ^\gamma] \cdot \text{image}[i + ^\tau, j + ^\gamma]
\]
Back To Our Example

\[ \text{output}[i, j] = \sum_{\tau=0}^{k-1} \sum_{\gamma=0}^{k-1} \text{filter}[\tau, \gamma] \cdot \text{image}[i + \tau, j + \gamma] \]
Back To Our Example

output\([i, j]\) = \sum_{\tau=0}^{k-1} \sum_{\gamma=0}^{k-1} \text{filter}[^\tau, \gamma] \cdot \text{image}[i + \tau, j + \gamma]
From Convolutions to Convolutional Layers
Convolutional Layer: Local Connectivity

Hence “fully connected” / “fc” layers.

- # input units (neurons): 7
- # hidden units: 3
- Number of parameters (ignoring bias)
  - Global connectivity: $3 \times 7 = 21$
  - Local connectivity: $3 \times 3 = 9$

Global connectivity

Local connectivity

Slide credit: Jia-Bin Huang
Convolutional Layer: Weight Sharing

Without weight sharing

- # input units (neurons): 7
- # hidden units: 3
- Number of parameters (ignoring bias)
  - Without weight sharing: $3 \times 3 = 9$
  - With weight sharing: $3 \times 1 = 3$

With weight sharing

Slide credit: Jia-Bin Huang
Extending convolutions

• We have just discussed the connection between normal “fully connected” layers to convolutions.

• But convolutional layers in neural networks extend this a bit more (next 3 slides):
  ▪ They can handle multiple input channels (e.g. RGB channels in color image)
  ▪ They can also handle multiple output channels
  ▪ They can modify the inputs to maintain desired activation sizes
Convolutional Layer with >1 input “channels” / “maps”

Single input channel

Multiple input channels

Filter weights

Outgoing layer

Incoming layer

Channel 1

Channel 2

Slide credit: Jia-Bin Huang
Convolutional Layer with >1 output “channels” / “maps”

Single output map

Multiple output maps

Filter weights

Slide credit: Jia-Bin Huang
Convolutional Layer Summary

- Local connectivity
- Weight sharing
- Handling multiple input/output channels
- Retains location associations

Image credit: A. Karpathy

Slide credit: Jia-Bin Huang
Stride

Filter

```
1 0
0 0.5
```

Input

Stride X

Output with stride 1

Output with stride 2

```
0 0 0 0 0 0
0 1 0 0.5 0.5 0
0 0 0.5 1 0 0
0 0 1 0.5 1 0
0 1 0.5 0.5 1 0
0 0 0 0 0 0
```

```
0.5 0 0.25 0.25 0
0 1.25 0.5 0.5 0.5
0 0.5 0.75 1.5 0
0.5 0.25 1.25 1 1
0. 1 0.5 0.5 1
```

```
0.50 0.25 0.00
0.00 0.75 0.00
0.00 0.50 1.00
```