Lecture 5: Convolutional Neural Networks
Administrative

Assignment 1 due Wednesday April 18, 11:59pm
Assignment 2 will also be released Wednesday
Last time: Neural Networks

Linear score function:

2-layer Neural Network

\[ f = Wx \]
\[ f = W_2 \max(0, W_1 x) \]
Next: Convolutional Neural Networks

Illustration of LeCun et al. 1998 from CS231n 2017 Lecture 1
A bit of history...

The **Mark I Perceptron** machine was the first implementation of the perceptron algorithm.

The machine was connected to a camera that used 20×20 cadmium sulfide photocells to produce a 400-pixel image.

Recognized letters of the alphabet

**update rule:**

\[
    w_i(t + 1) = w_i(t) + \alpha (d_j - y_j(t)) x_{j,i}
\]

Frank Rosenblatt, ~1957: Perceptron

This image by Rocky Acosta is licensed under CC-BY 3.0
Widrow and Hoff, ~1960: Adaline/Madaline

These figures are reproduced from Widrow 1960, Stanford Electronics Laboratories Technical Report with permission from Stanford University Special Collections.
A bit of history...

\[
\frac{\partial E_p}{\partial w_{ji}} = \frac{\partial E_p}{\partial o_{pj}} \frac{\partial o_{pj}}{\partial w_{ji}}
\]

Illustration of Rumelhart et al., 1986 by Lane McIntosh, copyright CS231n 2017

Rumelhart et al., 1986: First time back-propagation became popular
A bit of history...

[Reinvigorated research in Deep Learning by Hinton and Salakhutdinov 2006]

Illustration of Hinton and Salakhutdinov 2006 by Lane McIntosh, copyright CS231n 2017
First strong results

**Acoustic Modeling using Deep Belief Networks**  
Abdel-rahman Mohamed, George Dahl, Geoffrey Hinton, 2010

**Context-Dependent Pre-trained Deep Neural Networks for Large Vocabulary Speech Recognition**  
George Dahl, Dong Yu, Li Deng, Alex Acero, 2012

**Imagenet classification with deep convolutional neural networks**  

A bit of history:

Hubel & Wiesel, 1959
RECEPTIVE FIELDS OF SINGLE NEURONES IN THE CAT'S STRIATE CORTEX

1962
RECEPTIVE FIELDS, BINOCULAR INTERACTION AND FUNCTIONAL ARCHITECTURE IN THE CAT'S VISUAL CORTEX

1968...
A bit of history

Topographical mapping in the cortex:

nearby cells in cortex represent
nearby regions in the visual field
Hierarchical organization

Simple cells: Response to light orientation

Complex cells: Response to light orientation and movement

Hypercomplex cells: Response to movement with an end point

Illustration of hierarchical organization in early visual pathways by Lane McIntosh, copyright CS231n 2017
A bit of history:

Neocognitron

[Fukushima 1980]

“sandwich” architecture (SCSCSC...)  
simple cells: modifiable parameters  
complex cells: perform pooling
A bit of history:
Gradient-based learning applied to document recognition
[LeCun, Bottou, Bengio, Haffner 1998]
A bit of history:
ImageNet Classification with Deep Convolutional Neural Networks
[Krizhevsky, Sutskever, Hinton, 2012]

Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.

“AlexNet”
Fast-forward to today: ConvNets are everywhere

Classification


Retrieval
Fast-forward to today: ConvNets are everywhere

Detected

Segmentation

Figures copyright Shaoqing Ren, Kaiming He, Ross Girshick, Jian Sun, 2015. Reproduced with permission.

[Faster R-CNN: Ren, He, Girshick, Sun 2015]


[Farabet et al., 2012]
Fast-forward to today: ConvNets are everywhere

NVIDIA Tesla line
(These are the GPUs on rye01.stanford.edu)

Note that for embedded systems a typical setup would involve NVIDIA Tegras, with integrated GPU and ARM-based CPU cores.
Fast-forward to today: ConvNets are everywhere

[Simonyan et al. 2014]

Figures copyright Simonyan et al., 2014. Reproduced with permission.
Fast-forward to today: ConvNets are everywhere

[Toshev, Szegedy 2014]

Images are examples of pose estimation, not actually from Toshev & Szegedy 2014. Copyright Lane McIntosh.

[Guo et al. 2014]

Fast-forward to today: ConvNets are everywhere

From left to right: public domain by NASA, usage permitted by ESA/Hubble, public domain by NASA, and public domain.

Photos by Lane McIntosh. Copyright CS231n 2017.

Figure copyright Levy et al. 2016. Reproduced with permission.
Whale recognition, Kaggle Challenge

Mnih and Hinton, 2010

This image by Christin Khan is in the public domain and originally came from the U.S. NOAA.

Photo and figure by Lane McIntosh; not actual example from Mnih and Hinton, 2010 paper.
Image Captioning

[Vinyals et al., 2015]
[Karpathy and Fei-Fei, 2015]

Captions generated by Justin Johnson using NeuralTalk2

No errors

- A white teddy bear sitting in the grass
- A man in a baseball uniform throwing a ball
- A woman is holding a cat in her hand
- A man riding a wave on top of a surfboard
- A cat sitting on a suitcase on the floor
- A woman standing on a beach holding a surfboard

Minor errors

- A white teddy bear sitting in the grass
- A man in a baseball uniform throwing a ball
- A woman is holding a cat in her hand
- A cat sitting on a suitcase on the floor
- A woman standing on a beach holding a surfboard

Somewhat related

- A white teddy bear sitting in the grass
- A man in a baseball uniform throwing a ball
- A woman is holding a cat in her hand
- A man standing on a beach holding a surfboard
- A cat sitting on a suitcase on the floor
- A woman standing on a beach holding a surfboard

All images are CC0 Public domain:

Captions generated by Justin Johnson using NeuralTalk2
Convolutional Neural Networks

(First without the brain stuff)
Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1

input

1 3072

\[ Wx \]

weights

10 x 3072

activation

1

10
Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1

input

1

3072

$Wx$

10 x 3072 weights

activation

1

10

1 number:
the result of taking a dot product between a row of $W$ and the input (a 3072-dimensional dot product)
Convolution Layer

32x32x3 image -> preserve spatial structure
Convolution Layer

A 32x32x3 image

A 5x5x3 filter

Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”
Convolution Layer

32x32x3 image

5x5x3 filter

Convolve the filter with the image i.e. “slide over the image spatially, computing dot products”

Filters always extend the full depth of the input volume
Convolution Layer

A 32x32x3 image

5x5x3 filter $w$

1 number: the result of taking a dot product between the filter and a small 5x5x3 chunk of the image (i.e. $5 \times 5 \times 3 = 75$-dimensional dot product + bias)

$$w^T x + b$$
Convolution Layer

32x32x3 image
5x5x3 filter

convolve (slide) over all spatial locations

activation map
Convolution Layer

Consider a second, green filter

32x32x3 image
5x5x3 filter

Convolve (slide) over all spatial locations

Activation maps

Fei-Fei Li & Justin Johnson & Serena Yeung
For example, if we had 6 5x5 filters, we’ll get 6 separate activation maps:

We stack these up to get a “new image” of size 28x28x6!
Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions.

- Convolution (CONV)
- Rectified Linear Unit (ReLU)
- Filters: e.g., 6 5x5x3 filters
Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions.
Visualization of VGG-16 by Lane McIntosh. VGG-16 architecture from [Simonyan and Zisserman 2014].
Preview

Low-level features → Mid-level features → High-level features → Linearly separable classifier

VGG-16 Conv1_1 → VGG-16 Conv3_2 → VGG-16 Conv5_3

Retinal ganglion cell receptive fields → LGN and V1 simple cells

Complex cells: Response to light orientation and movement

Hypercomplex cells: response to movement with an end point

Visual stimulus

Fei-Fei Li & Justin Johnson & Serena Yeung
We call the layer convolutional because it is related to convolution of two signals:

\[ f[x, y] \ast g[x, y] = \sum_{n_1 = -\infty}^{\infty} \sum_{n_2 = -\infty}^{\infty} f[n_1, n_2] \cdot g[x - n_1, y - n_2] \]

elementwise multiplication and sum of a filter and the signal (image)
A closer look at spatial dimensions:

32x32x3 image
5x5x3 filter

convolve (slide) over all spatial locations

activation map
A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter
A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter
A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter
A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter
A closer look at spatial dimensions:

7x7 input (spatially) 
assume 3x3 filter

=> 5x5 output
A closer look at spatial dimensions:

7x7 input (spatially) assume 3x3 filter applied with stride 2
A closer look at spatial dimensions:

7x7 input (spatially) assume 3x3 filter applied \textit{with stride 2}
A closer look at spatial dimensions:

7x7 input (spatially) assume 3x3 filter applied with stride 2 => 3x3 output!
A closer look at spatial dimensions:

7x7 input (spatially) assume 3x3 filter applied with stride 3?
A closer look at spatial dimensions:

7x7 input (spatially) assume 3x3 filter applied with stride 3?

doesn’t fit!
cannot apply 3x3 filter on 7x7 input with stride 3.
Output size:
\[(N - F) / \text{stride} + 1\]

e.g. \(N = 7, F = 3\):
- stride 1 => \((7 - 3)/1 + 1 = 5\)
- stride 2 => \((7 - 3)/2 + 1 = 3\)
- stride 3 => \((7 - 3)/3 + 1 = 2.33\)
In practice: Common to zero pad the border

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

e.g. input 7x7
3x3 filter, applied with \textbf{stride 1}
\textbf{pad with 1 pixel} border => what is the output?

(recall:)
\[
\frac{N - F}{\text{stride}} + 1
\]
In practice: Common to zero pad the border

E.g. input 7x7 3x3 filter, applied with stride 1
Pad with 1 pixel border => what is the output?

7x7 output!
In practice: Common to zero pad the border

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

e.g. input 7x7
3x3 filter, applied with stride 1
pad with 1 pixel border => what is the output?

7x7 output!
In general, common to see CONV layers with stride 1, filters of size FxF, and zero-padding with (F-1)/2. (will preserve size spatially)
e.g. F = 3 => zero pad with 1
    F = 5 => zero pad with 2
    F = 7 => zero pad with 3
Remember back to…
E.g. 32x32 input convolved repeatedly with 5x5 filters shrinks volumes spatially! (32 -> 28 -> 24 ...). Shrinking too fast is not good, doesn’t work well.
Examples time:

Input volume: \(32 \times 32 \times 3\)
10 5x5 filters with stride 1, pad 2

Output volume size: ?
Examples time:

Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size:
\[(32+2\times2-5)/1+1 = 32\] spatially, so
**32x32x10**
Examples time:

Input volume: **32x32x3**
10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?
Examples time:

Input volume: \(32 \times 32 \times 3\)
10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?
each filter has \(5 \times 5 \times 3 + 1 = 76\) params (+1 for bias)
\[\Rightarrow 76 \times 10 = 760\]
Summary. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters $K$,
  - their spatial extent $F$,
  - the stride $S$,
  - the amount of zero padding $P$.
- Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  - $W_2 = (W_1 - F + 2P)/S + 1$
  - $H_2 = (H_1 - F + 2P)/S + 1$ (i.e. width and height are computed equally by symmetry)
  - $D_2 = K$
- With parameter sharing, it introduces $F \cdot F \cdot D_1$ weights per filter, for a total of $(F \cdot F \cdot D_1) \cdot K$ weights and $K$ biases.
- In the output volume, the $d$-th depth slice (of size $W_2 \times H_2$) is the result of performing a valid convolution of the $d$-th filter over the input volume with a stride of $S$, and then offset by $d$-th bias.
Common settings:

- $K = \text{(powers of 2, e.g. 32, 64, 128, 512)}$
- $F = 3, S = 1, P = 1$
- $F = 5, S = 1, P = 2$
- $F = 5, S = 2, P = ? \text{ (whatever fits)}$
- $F = 1, S = 1, P = 0$
(btw, 1x1 convolution layers make perfect sense)

1x1 CONV with 32 filters
(each filter has size 1x1x64, and performs a 64-dimensional dot product)
Example: CONV layer in Torch

**SpatialConvolution**

```python
module = nn.SpatialConvolution(nInputPlane, nOutputPlane, kW, kH, [dW], [dH], [padW], [padH])
```

Applies a 2D convolution over an input image composed of several input planes. The input tensor in `forward(input)` is expected to be a 3D tensor `(nInputPlane x height x width)`.

The parameters are the following:

- `nInputPlane`: The number of expected input planes in the image given into `forward()`.
- `nOutputPlane`: The number of output planes the convolution layer will produce.
- `kW`: The kernel width of the convolution
- `kH`: The kernel height of the convolution
- `dW`: The step of the convolution in the width dimension. Default is 1.
- `dH`: The step of the convolution in the height dimension. Default is 1.
- `padW`: The additional zeros added per width to the input planes. Default is 0, a good number is `(kW-1)/2`.
- `padH`: The additional zeros added per height to the input planes. Default is padW, a good number is `(kH-1)/2`.

Note that depending on the size of your kernel, several (of the last) columns or rows of the input image might be lost. It is up to the user to add proper padding in images.

If the input image is a 3D tensor `nInputPlane x height x width`, the output image size will be `nOutputPlane x oheight x owidth` where

```plaintext
owidth = floor((width + 2*padW - kW) / dW + 1)
oheight = floor((height + 2*padH - kH) / dH + 1)
```

---

**Summary**. To summarize, the Conv Layer:

- Accepts a volume of size $W_1 \times H_1 \times D_1$
- Requires four hyperparameters:
  - Number of filters $K$,
  - their spatial extent $F$,
  - the stride $S$,
  - the amount of zero padding $P$.  

Torch is licensed under **BSD 3-clause**.
Example: CONV layer in Caffe

```
layer {
  name: "conv1"
  type: "Convolution"
  bottom: "data"
  top: "conv1"

  # learning rate and decay multipliers for the filters
  param { lr_mult: 1 decay_mult: 1 }

  # learning rate and decay multipliers for the biases
  param { lr_mult: 2 decay_mult: 0 }

  convolution_param {
    num_output: 96 # learn 96 filters
    kernel_size: 11 # each filter is 11x11
    stride: 4 # step 4 pixels between each filter application
    weight_filler {
      type: "gaussian" # initialize the filters from a Gaussian
      std: 0.01 # distribution with stdev 0.01 (default mean: 0)
    }
    bias_filler {
      type: "constant" # initialize the biases to zero (0)
      value: 0
    }
  }
}
```
The brain/neuron view of CONV Layer

32x32x3 image
5x5x3 filter

1 number:
the result of taking a dot product between
the filter and this part of the image
(i.e. 5*5*3 = 75-dimensional dot product)
The brain/neuron view of CONV Layer

32x32x3 image
5x5x3 filter

1 number: the result of taking a dot product between the filter and this part of the image (i.e. 5*5*3 = 75-dimensional dot product)

It's just a neuron with local connectivity...
The brain/neuron view of CONV Layer

An activation map is a 28x28 sheet of neuron outputs:
1. Each is connected to a small region in the input
2. All of them share parameters

“5x5 filter” -> “5x5 receptive field for each neuron”
The brain/neuron view of CONV Layer

E.g. with 5 filters, CONV layer consists of neurons arranged in a 3D grid (28x28x5)

There will be 5 different neurons all looking at the same region in the input volume
Reminder: Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1

Each neuron looks at the full input volume

- **Input**: 3072
- **Weights**: 10 x 3072
- **Activation**: 1

1 number: the result of taking a dot product between a row of W and the input (a 3072-dimensional dot product)
two more layers to go: POOL/FC
Pooling layer
- makes the representations smaller and more manageable
- operates over each activation map independently:
MAX POOLING

Single depth slice

max pool with 2x2 filters and stride 2
• Accepts a volume of size $W_1 \times H_1 \times D_1$
• Requires three hyperparameters:
  ◦ their spatial extent $F$,
  ◦ the stride $S$,
• Produces a volume of size $W_2 \times H_2 \times D_2$ where:
  ◦ $W_2 = (W_1 - F')/S + 1$
  ◦ $H_2 = (H_1 - F')/S + 1$
  ◦ $D_2 = D_1$
• Introduces zero parameters since it computes a fixed function of the input
• Note that it is not common to use zero-padding for Pooling layers
Common settings:

\[ F = 2, \quad S = 2 \]
\[ F = 3, \quad S = 2 \]

- Accepts a volume of size \( W_1 \times H_1 \times D_1 \)
- Requires three hyperparameters:
  - their spatial extent \( F \),
  - the stride \( S \),
- Produces a volume of size \( W_2 \times H_2 \times D_2 \) where:
  - \( W_2 = (W_1 - F)/S + 1 \)
  - \( H_2 = (H_1 - F)/S + 1 \)
  - \( D_2 = D_1 \)
- Introduces zero parameters since it computes a fixed function of the input
- Note that it is not common to use zero-padding for Pooling layers
Fully Connected Layer (FC layer)
- Contains neurons that connect to the entire input volume, as in ordinary Neural Networks
ConvNetJS CIFAR-10 demo

This demo trains a Convolutional Neural Network on the CIFAR-10 dataset in your browser, with nothing but Javascript. The state of the art on this dataset is about 90% accuracy and human performance is at about 94% (not perfect as the dataset can be a bit ambiguous). I used this python script to parse the original files (python version) into batches of images that can be easily loaded into page DOM with img tags.

This dataset is more difficult and it takes longer to train a network. Data augmentation includes random flipping and random image shifts by up to 2px horizontally and vertically.

By default, in this demo we’re using Adadelta which is one of per-parameter adaptive step size methods, so we don’t have to worry about changing learning rates or momentum over time. However, I still included the text fields for changing these if you’d like to play around with SGD+Momentum trainer.

Report questions/bugs/suggestions to @karpathy.

http://cs.stanford.edu/people/karpathy/convnetjs/demo/cifar10.html
Summary

- ConvNets stack CONV, POOL, FC layers
- Trend towards smaller filters and deeper architectures
- Trend towards getting rid of POOL/FC layers (just CONV)
- Typical architectures look like
  
  
  $$\left[\text{CONV-RELU}\right]^N \cdot \text{POOL?} \cdot M \cdot \text{FC-RELU}^K, \text{SOFTMAX}$$

  where $N$ is usually up to ~5, $M$ is large, $0 \leq K \leq 2$.

  - but recent advances such as ResNet/GoogLeNet challenge this paradigm