Topics:
- Convolutional Neural Networks
  - Stride, padding
  - Pooling layers
- Fully-connected layers as convolutions

Dhruv Batra
Georgia Tech
Administrativia

• HW1 Reminder
  – Due: 09/26, 11:55pm

• Project Teams Google Doc
  – [https://docs.google.com/spreadsheets/d/1ouD6ctaemV_3nb2MQHs7rUOAaW9DFLu8l5Zd3yOFs7E/edit?usp=sharing](https://docs.google.com/spreadsheets/d/1ouD6ctaemV_3nb2MQHs7rUOAaW9DFLu8l5Zd3yOFs7E/edit?usp=sharing)
  – Project Title
  – 1-3 sentence project summary TL;DR
  – Team member names
Recap from last time
Convolutional Neural Networks
(without the brain stuff)
- Spatial correlation is local
- Waste of resources + we have not enough training samples anyway..

Example: 200x200 image
40K hidden units

~2B parameters!!!

Slide Credit: Marc'Aurelio Ranzato
Locally Connected Layer

Example: 200x200 image
40K hidden units
“Filter” size: 10x10
4M parameters

Note:
This parameterization is good when input image is registered (e.g., face recognition).
Locally Connected Layer

**STATIONARITY?**
Statistics similar at all locations

![Diagram with connections and labels](image_url)

Slide Credit: Marc'Aurelio Ranzato
Share the same parameters across different locations (assuming input is stationary): Convolutions with learned kernels.
Convolutions!

math $\rightarrow$ CS $\rightarrow$ programming
Convolutions for programmers

\[ y[a, c] = \sum_{a=0}^{k_2-1} \sum_{b=0}^{k_1-1} x[a+a, c+b] w[a, b] \]
Convolution
Convolution
Convolution
Convolution
Convolution
Convolution
Convolution
Convolution
Convolution
Convolution
Convolution
Convolution
Convolution
Convolution
Convolution
Convolution
Convolution Layer

32x32x3 image -> preserve spatial structure

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Convolution Layer

32x32x3 image

5x5x3 filter

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Convolution Layer

Filters always extend the full depth of the input volume

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Convolution Layer

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)

$w^T x + b$

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Convolution Layer

32x32x3 image
5x5x3 filter

convolve (slide) over all spatial locations, computing all dot products

Feature maps
Activation map

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Convolution Layer

consider a second, green filter

32x32x3 image
5x5x3 filter

convolve (slide) over all spatial locations

activation maps

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
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!
Im2Col

Figure Credit: https://petewarden.com/2015/04/20/why-gemm-is-at-the-heart-of-deep-learning/
GEMM

Input Matrix

Kernel Matrix

(C) Dhruv Batra  Figure Credit: https://petewarden.com/2015/04/20/why-gemm-is-at-the-heart-of-deep-learning/
Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions.
**Preview:** ConvNet is a sequence of Convolutional Layers, interspersed with activation functions.
Plan for Today

• Convolutional Neural Networks
  – Features learned by CNN layers
    – Stride, padding
    – 1x1 convolutions
    – Pooling layers
    – Fully-connected layers as convolutions
Convolutional Neural Networks
Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
example 5x5 filters
(32 total)

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Visualizing Learned Filters

Figure Credit: [Zeiler & Fergus ECCV14]
Visualizing Learned Filters

Layer 3

Figure Credit: [Zeiler & Fergus ECCV14]
Visualizing Learned Filters

Figure Credit: [Zeiler & Fergus ECCV14]
We can learn image features now!

Feature visualization of convolutional net trained on ImageNet from [Zeiler & Fergus 2013]

Slide Credit: Marc'Aurelio Ranzato, Yann LeCun
Plan for Today

• Convolutional Neural Networks
  – Features learned by CNN layers
  – Stride, padding
  – 1x1 convolutions
  – Pooling layers
  – Fully-connected layers as convolutions
A closer look at spatial dimensions:

- **32x32x3 image**
- **5x5x3 filter**
- **Convolve (slide) over all spatial locations**

**Activation map**

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
A closer look at spatial dimensions:

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

7x7 input (spatially)
assume 3x3 filter

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
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 with stride 2

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
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.

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Output size:
\[(\text{N} - \text{F}) / \text{stride} + 1\]

e.g. N = 7, F = 3:
- stride 1 => \((7 - 3)/1 + 1 = 5\)
- stride 2 => \((7 - 3)/2 + 1 = 3\)
- stride 3 => \((7 - 3)/3 + 1 = 2.33\)

\[\frac{7-3}{2} + 1\]

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
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.

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
In practice: Common to zero pad the border

- e.g. input $7 \times 7$
- $3 \times 3$ filter, applied with stride 1
- pad with 1 pixel border => what is the output?

\[
N = \left\lfloor \frac{N + 2 \text{pad} - F + 1}{\text{stride}} \right\rfloor
\]

(recall:)

\[
(N - F) / \text{stride} + 1
\]

\[
\frac{q - 3 + 1}{1} = 6 + 1 = 7
\]
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

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-paddings 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
Examples time:

Input volume: \textbf{32x32x3}
10, 5x5 filters with stride 1, pad 2

Output volume size: ?

\[
\left(\frac{32 \times 4 - 5}{1} + 1\right) \quad M \times M \times \sqrt{C_2}^{10}
\]
Examples time:

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

Output volume size:
$(32+2 \times 2-5)/1+1 = 32$ spatially, so $32 \times 32 \times 10$

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Examples time: (5 \times 5 \times 3)_{10}

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

Number of parameters in this layer?

\((5 \times 5 \times 3 + 1) \times 10\)

\(75\)

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Examples time:

Input volume: $32 \times 32 \times 3$
10 $5 \times 5$ 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) => $76 \times 10 = 760$
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:

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

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.
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](https://opensource.org/licenses/BSD-3-Clause).
Plan for Today

• Convolutional Neural Networks
  – Features learned by CNN layers
  – Stride, padding
  – 1x1 convolutions
  – Pooling layers
  – Fully-connected layers as convolutions
  – Backprop in conv layers
Can we have 1x1 filters?

\[ y[r,c] = \sum_{a=0}^{k_1-1} \sum_{b=0}^{k_2-1} x[r+a, c+b] w[a,b] \]
1x1 convolution layers make perfect sense

1x1 CONV with 32 filters

(each filter has size 1x1x64, and performs a 64-dimensional dot product)
Fully Connected Layer as 1x1 Conv

32x32x3 image -> stretch to 3072 x 1

10 x 3072 weights

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

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n