Topics:

- Convolutional Neural Networks
  - Pooling layers
  - Fully-connected layers as convolutions
  - Toeplitz matrices and convolutions = matrix-mult
  - Backprop in conv layers

Dhruv Batra
Georgia Tech
Administrativia

- **HW1 Reminder**
  - Due: 10/02, 11:55pm

- **Project Idea: ICLR19 Reproducibility Challenge**
  - [https://docs.google.com/spreadsheets/d/1BipWLvvWb7Fu6OSDd-uOCF1Lr_4drKOCRVDhxm_eSHc/edit#gid=0](https://docs.google.com/spreadsheets/d/1BipWLvvWb7Fu6OSDd-uOCF1Lr_4drKOCRVDhxm_eSHc/edit#gid=0)
Recap from last time
Convolutional Neural Networks
(without the brain stuff)
Convolutional Neural Networks

Input layer

(S1) 4 feature maps

(C1) 4 feature maps

(S2) 6 feature maps

(C2) 6 feature maps

C1: feature maps 6@28x28

Subsampling

S2: feature maps 6@14x14

S4: feature maps 16@5x5

C5: layer 120

C3: feature maps 16@10x10

Full connection

Gaussian connections

Output

F6: layer 84

Full connection

Input 32x32

Convolutions

Subsampling

Convolutions

Subsampling

Full connection

Gaussian connections

Image Credit: Yann LeCun, Kevin Murphy
Convolutions for programmers

\[ y[x, c] = \begin{cases} \sum_{k=-1}^{K_1-1} \sum_{k=-1}^{K_2-1} x[(x+a, c+b)] w[a, b] \\ a = 0 \quad b = 0 \end{cases} \]
FC vs Conv Layer

\[ h_i^l = \sum_{j=1}^{l-1} h_j^{l-1} \cdot w_{ij} + b_i \]

Scalar product

\[ h_c^l = \sum_{j=1}^{l-1} \left[ h_j^{l-1} \ast w_{ij} \right] + b_c \]

\[ h_i^l [r,c] = \sum_{j=1}^{k_j} \sum_{a=0}^{k_j-1} \sum_{b=0}^{k_j-1} h_j^l [r+a, c+b] \cdot w_j^{(c,c)} \]

\[ W \in \mathbb{R}^{k_1 \times k_2 \times c_1 \times c_2} \]
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*5*3 = 75-dimensional dot product + bias)

\[ w^T x + b \]
Convolution Layer

32x32x3 image

5x5x3 filter

convolve (slide) over all spatial locations

activation map

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

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

Input Matrix

Kernel Matrix

Figure Credit: https://petewarden.com/2015/04/20/why-gemm-is-at-the-heart-of-deep-learning/
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!
Preview: ConvNet is a sequence of Convolutional Layers, interspersed with activation functions.
example 5x5 filters (32 total)

one filter => one activation map
Visualizing Learned Filters

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

(C) Dhruv Batra

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

Figure Credit: [Zeiler & Fergus ECCV14]
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
A closer look at spatial dimensions:

7x7 input (spatially)
assume 3x3 filter

stride = 1
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
Output size:
\[(N - F) \times \frac{N - F}{\text{stride}} + 1\]

E.g. \(N = 7, F = 3\):
- \(\text{stride 1} \Rightarrow \frac{7 - 3}{1} + 1 = 5\)
- \(\text{stride 2} \Rightarrow \frac{7 - 3}{2} + 1 = 3\)
- \(\text{stride 3} \Rightarrow \frac{7 - 3}{3} + 1 = 2.33\)
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?

(recall:)
\[
\frac{N - F}{\text{stride}} + 1
\]

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

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

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

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

Output volume size: ?

\[
\left(\frac{32}{N^4}\right) \times \left(\frac{32}{N^2}\right) \times \left(\frac{10}{C_2}\right)
\]
Examples time:

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

Output volume size:
\[
(32 + 2 \times 2 - 5)/1 + 1 = 32 \text{ spatially, so } 32\times32\times10
\]
Examples time:

Input volume: \(32 \times 32 \times 3\)

10 5x5 filters with stride 1, pad 2

Number of parameters in this layer?

\[ (5 \times 5 \times 3 + 1) \times 10 = 760 \]
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:

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

---

*Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n*
Plan for Today

• Convolutional Neural Networks
  – 1x1 convolutions
  – Pooling layers
  – Fully-connected layers as convolutions
  – Backprop in conv layers
  – Toeplitz matrices and convolutions = matrix-mult
  – Dilated/a-trous convolutions
Can we have 1x1 filters?

\[
y[r,c] = \sum_{a=0}^{k-1} \sum_{b=0}^{k-1} x[r+a, c+b] \cdot w[a, b]
\]
(btw, 1x1 convolution layers make perfect sense)

1x1 CONV with 32 filters
(each filter has size 1x1x64, and performs a 64-dimensional dot product)
Reminder: Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1

Each neuron looks at the full input volume

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**
Let us assume filter is an “eye” detector.

Q.: how can we make the detection robust to the exact location of the eye?
Pooling Layer

By “pooling” (e.g., taking max) filter responses at different locations we gain robustness to the exact spatial location of features.
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

\[ y[r, c] = \max \max_{a,b} \ \mathcal{C}[a+c, b+d] \]
<table>
<thead>
<tr>
<th>1</th>
<th>3</th>
<th>2</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>4</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td>5</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>
Pooling Layer: Examples

Max-pooling:

$$h^n_i(r, c) = \max_{\bar{r} \in N(r), \bar{c} \in N(c)} h^{n-1}_i(\bar{r}, \bar{c})$$

Average-pooling:

$$h^n_i(r, c) = \frac{1}{\max_{\bar{r} \in N(r), \bar{c} \in N(c)} h^{n-1}_i(\bar{r}, \bar{c})} \sum_{\bar{r} \in N(r), \bar{c} \in N(c)} h^{n-1}_i(\bar{r}, \bar{c})$$

L2-pooling:

$$h^n_i(r, c) = \sqrt{\sum_{\bar{r} \in N(r), \bar{c} \in N(c)} h^{n-1}_i(\bar{r}, \bar{c})^2}$$
• 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:

- \( F = 2, S = 2 \)
- \( F = 3, S = 2 \)

- 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
If convolutional filters have size $K \times K$ and stride 1, and pooling layer has pools of size $P \times P$, then each unit in the pooling layer depends upon a patch (at the input of the preceding conv. layer) of size: $(P+K-1) \times (P+K-1)$.
If convolutional filters have size $K \times K$ and stride 1, and pooling layer has pools of size $P \times P$, then each unit in the pooling layer depends upon a patch (at the input of the preceding conv. layer) of size: $(P+K-1) \times (P+K-1)$
Plan for Today

• Convolutional Neural Networks
  – 1x1 convolutions
  – Pooling layers
  – Fully-connected layers as convolutions
  – Backprop in conv layers
Fully Connected Layer (FC layer)
- Contains neurons that connect to the entire input volume, as in ordinary Neural Networks
Classical View

 convolution

 fully connected

“tabby cat”

227 × 227

55 × 55

27 × 27

13 × 13

7 × 7

Figure Credit: [Long, Shelhamer, Darrell CVPR15]
$7 \times 7 \times 512$

Fully conn. layer

$N \times N \times C$, $N$ small

$H$ hidden units

Slide Credit: Marc'Aurelio Ranzato
Classical View = Inefficient
Classical View

Convolution → Fully Connected

227 × 227 | 55 × 55 | 27 × 27 | 13 × 13

"tabby cat"
Re-interpretation

• Just squint a little!
“Fully Convolutional” Networks

• Can run on an image of any size!
H hidden units / 1x1xH feature maps

NxNxC, N small

Fully conn. layer / Conv. layer (H kernels of size NxNxC)
Viewing fully connected layers as convolutional layers enables efficient use of convnets on bigger images (no need to slide windows but unroll network over space as needed to re-use computation).
Viewing fully connected layers as convolutional layers enables efficient use of convnets on bigger images (no need to slide windows but unroll network over space as needed to re-use computation).

Unrolling is order of magnitudes more efficient than sliding windows!
Training time

- Fixed-size images
Testing time

• Can run on an image of any size!
Benefit of this thinking

• Mathematically elegant

• Efficiency
  – Can run network on arbitrary image
  – Without multiple crops
“Fully Convolutional” Networks

- Up-sample to get segmentation maps
Plan for Today

• Convolutional Neural Networks
  – 1x1 convolutions
  – Pooling layers
  – Fully-connected layers as convolutions
  – Backprop in conv layers
Jacobians of FC-Layer

\[ \frac{\partial L}{\partial h^l} = \frac{\partial L}{\partial \hat{h}^l} \cdot \frac{\partial \hat{h}^l}{\partial h^l} \]

\[ h_i = \hat{w}_i \hat{h}^l \]

\[ h^l \in \mathbb{R}^{c_2} \]

\[ \hat{h}^l \in \mathbb{R}^{c_2} \]

\[ \mathbb{R}^C \]

\[ \mathbb{R}^C \]
Backprop in Convolutional Layers

• Notes
Backprop in Convolutional Layers

\[ \frac{\partial L}{\partial w[a', b']} = \sum \text{pixels } p \text{ output} \]

\[ \frac{\partial L}{\partial w[a', b']} = \text{ Jacobian} \]

\[ \frac{\partial L}{\partial y_{p+i}} \]

\[ \frac{\partial y_{p+i}}{\partial w[a', b']} \]

\[ y[p, c] = \sum_{a} \sum_{b} x[p+a', c+b'] w[a, b] \]

\[ y[p, c] = x[p+a, c+b] w[a, b] + x[p+a, c+] w[a, b] \]

\[ y[p, c] = x[p+a', c+b'] w[a', b'] \]

\[ \frac{\partial}{\partial w[a', b']} = \frac{\partial y[p, c]}{\partial [x]} \]

\[ \frac{\partial}{\partial w[a', b']} = x[p, c] \]