Topics:
- Convolutional Neural Networks
  - Pooling layers
  - Fully-connected layers as convolutions
- Backprop in conv layers [Derived in notes]
- Toeplitz matrices and convolutions = matrix-mult

Dhruv Batra
Georgia Tech
Administrativia

• HW2 Reminder
  – Due: 09/23, 11:59pm

• Project Teams
  – https://gtvault-my.sharepoint.com/:x:/g/personal/dbatra8_gatech_edu/EY4_65XOzWtOkXSSz2WgpoUBY8ux2gY9PsRzR6KnglFEQ?e=4tnKWl
  – Project Title
  – 1-3 sentence project summary TL;DR
  – Team member names
Recap from last time
Convolutions for programmers

\[ y[a, c] = \sum_{a=0}^{k_x-1} \sum_{b=0}^{k_y-1} x[a+a', c+b'] w[a', b'] \]
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

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)

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

32x32x3 image
5x5x3 filter
convolve (slide) over all spatial locations

feature map
activation map

Slide Credit: Fei-Fei Li, Justin Johnson, Serena Yeung, CS 231n
Consider a second, green filter.

Convolution Layer

Convolve (slide) over all spatial locations.

32x32x3 image

5x5x3 filter

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!
Plan for Today

- Convolutional Neural Networks
  - Stride, padding
  - 1x1 convolutions
  - Backprop in conv layers [Derived in notes]
  - Pooling layers
  - Fully-connected layers as convolutions
  - Toeplitz matrices and convolutions = matrix-mult
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

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
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
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. \(N = 7, F = 3\):

- \(\text{stride 1} \Rightarrow (7 - 3)/1 + 1 = 5\)
- \(\text{stride 2} \Rightarrow (7 - 3)/2 + 1 = 3\)
- \(\text{stride 3} \Rightarrow (7 - 3)/3 + 1 = 2.33\)
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.
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:)

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

\[ N \times N \rightarrow \frac{3 \times 3}{1} \times \frac{N}{1} \]

\[ \bar{N} = N + 2 \cdot \text{pad} \]

\[ \text{pad} = \frac{F-1}{2} \]

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

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 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
Examples time:

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

10 5x5 filters with stride 1, pad 2

Output volume size: ?

\[
\frac{M \times M \times C_2}{32 \times 32 \times 10}
\]
Examples time:

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

Output volume size:
\[(32+2\times2-5)/1+1 = 32\] spatially, so \textbf{32x32x10}

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
\((5 \times 5 \times 3)\)

Number of parameters in this layer?

\[
\begin{align*}
    k_1 \times k_2 & \quad C_1 \quad C_2 \\
    \left(\frac{5 \times 5 \times 3}{+1}\right) \times 10 &= 76 \times 10 = 760
\end{align*}
\]
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)
=> \(76 \times 10 = 760\)
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:

- \( K \) = (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)
  - \( F = 1, S = 1, P = 0 \)

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.
Plan for Today

- Convolutional Neural Networks
  - Stride, padding
  - 1x1 convolutions
  - Backprop in conv layers [Derived in notes]
  - Pooling layers
  - Fully-connected layers as convolutions
  - Toeplitz matrices and convolutions = matrix-mult
Can we have 1x1 filters?

\[ y[l, c] = \sum_{a=0}^{k-1} \sum_{b=0}^{k-1} x[l+a, c+b] \omega[l, c] \]

\[ y[l, c] = x[l, c] \omega[l, c] \]
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

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

• Convolutional Neural Networks
  – Stride, padding
  – 1x1 convolutions
  – Backprop in conv layers [Derived in notes]
  – Pooling layers
  – Fully-connected layers as convolutions
  – Toeplitz matrices and convolutions = matrix-mult
Any DAG of differentiable modules is allowed!
Key Computation: **Forward-Prop**
Key Computation: Back-Prop

\[
\frac{\partial L}{\partial X} = \left\{ \frac{\partial Z}{\partial X}, \frac{\partial Z}{\partial \theta} \right\}
\]

\[\frac{\partial L}{\partial Z} \]

\[\frac{\partial L}{\partial \theta} \]
Backprop in Convolutional Layers

• Notes

  - [https://www.cc.gatech.edu/classes/AY2021/cs7643_fall/slides/L11_cnns_backprop_notes.pdf](https://www.cc.gatech.edu/classes/AY2021/cs7643_fall/slides/L11_cnns_backprop_notes.pdf)
Backprop in Convolutional Layers

\[ \frac{\partial L}{\partial w[a', b']} = \sum \text{pixel dependency of } y[p] \text{ on } w[a', b'] \]

Input: \[ \frac{\partial L}{\partial y[p]} \]
Output: \[ \frac{\partial L}{\partial w} \]

Component of the Jacobian

\[ \frac{\partial L}{\partial w[a', b']} = \sum \sum \frac{\partial L}{\partial y[a, c]} \frac{\partial y[a, c]}{\partial w[a', b']} \]

\[ = x[a + d, c + b'] \]

\[ y[0, 0] \]

\[ x \]

\[ \frac{\partial y[a, c]}{\partial w[a', b']} = \sum_{a=0}^{k-1} \sum_{b=0}^{k-1} x[a + d, c + b'] \]

(C) Dhruv Batra
Backprop in Convolutional Layers

\[
\frac{\partial L}{\partial w} = \sum_{a=0}^{k-1} \sum_{b=0}^{k-1} x[a+a', c+b'] \frac{\partial L}{\partial y[a,c]} x[y[a+a', c+b']] n[a,b]
\]

\[
y[a,c] = \sum_{a=0}^{k-1} \sum_{b=0}^{k-1} x[a+a', c+b'] n[a,b]
\]

\[
\frac{\partial L}{\partial w} = x \ast \frac{\partial L}{\partial y}
\]

(with appropriate padding)
Backprop in Convolutional Layers
Plan for Today

• Convolutional Neural Networks
  – Stride, padding
  – 1x1 convolutions
  – Backprop in conv layers [Derived in notes]
  – Pooling layers
  – Fully-connected layers as convolutions
  – Toeplitz matrices and convolutions = matrix-mult
two more layers to go: \textbf{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?
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

\[ y[x, c] = \max_{a,b \in \text{valid}} \max_{c} x[a:a+1, c:c+1] \]

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

Max-pooling:

\[ h_i^n(r, c) = \max_{\bar{r} \in N(r), \bar{c} \in N(c)} h_i^{n-1}(\bar{r}, \bar{c}) \]

Average-pooling:

\[ h_i^n(r, c) = \frac{\text{mean}_{\bar{r} \in N(r), \bar{c} \in N(c)} h_i^{n-1}(\bar{r}, \bar{c})}{\sum_{\bar{r} \in N(r), \bar{c} \in N(c)} h_i^{n-1}(\bar{r}, \bar{c})^2} \]

L2-pooling:

\[ h_i^n(r, c) = \sqrt{\sum_{\bar{r} \in N(r), \bar{c} \in N(c)} h_i^{n-1}(\bar{r}, \bar{c})^2} \]
Receptive Field

\[ h^{(l-1)} \rightarrow \text{Conv. layer} \rightarrow h^{(l)} \rightarrow \text{Pool. layer} \rightarrow h^{(l+1)} \]
Pooling Layer: Receptive Field Size

$h^{(l-1)} \rightarrow \text{Conv. layer} \rightarrow h^{(l)} \rightarrow \text{Pool. layer} \rightarrow h^{(l+1)}$
If convolutional filters are $F \times F$ and stride 1, and pooling layer has pools of size $P \times P$, then each unit in the pooling layer depends upon a patch in $h^{(l-1)}$ of size: $(P+F-1) \times (P+F-1)$.
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

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

Image Credit: Yann LeCun, Kevin Murphy
Classical View

convolution

fully connected

227 × 227
55 × 55
27 × 27
13 × 13

“tabby cat”
Ng x Ng x C₁, N small

C₂ hidden units

Fully conn. layer

Slide Credit: Marc'Aurelio Ranzato
Classical View

convolution

fully connected

227 × 227  55 × 55  27 × 27  13 × 13

“tabby cat”

Figure Credit: [Long, Shelhamer, Darrell CVPR15]
Classical View = Inefficient

1. Input image
2. Extract region proposals (~2k)
3. Compute CNN features
4. Classify regions
Classical View

convolution

fully connected

“tabby cat”

227 × 227  55 × 55  27 × 27  13 × 13
Re-interpretation

- Just squint a little!

convolution

227 × 227  55 × 55  27 × 27  13 × 13  1 × 1
(C) Dhruv Batra

Slide Credit: Marc'Aurelio Ranzato

$N \times N \times C_1$, $N$ small

$C_2$ hidden units / $1 \times 1 \times C_2$ feature maps

Fully conn. layer / Conv. layer ($C_2$ kernels of size $N \times N \times C_1$)
Re-interpretation

• Just squint a little!
“Fully Convolutional” Networks

• Can run on an image of any size!

convolution

H × W  H/4 × W/4  H/8 × W/8  H/16 × W/16  H/32 × W/32

Figure Credit: [Long, Shelhamer, Darrell CVPR15]
Benefit of this thinking

• Mathematically elegant

• Efficiency
  – Can run network on arbitrary image
  – Without multiple crops
Plan for Today

• Convolutional Neural Networks
  – Stride, padding
  – 1x1 convolutions
  – Backprop in conv layers [Derived in notes]
  – Pooling layers
  – Fully-connected layers as convolutions
  – Toeplitz matrices and convolutions = matrix-mult
Toeplitz Matrix

- Diagonals are constants

\[
\begin{bmatrix}
  a & b & c & d & e \\
  f & a & b & c & d \\
  g & f & a & b & c \\
  h & g & f & a & b \\
  i & h & g & f & a
\end{bmatrix}
\]

- \( A_{ij} = a_{i-j} \)
Why do we care?

- (Discrete) Convolution = Matrix Multiplication
  - with Toeplitz Matrices

\[
y = w \ast x
\]

\[
\begin{bmatrix}
w_k & 0 & \ldots & 0 & 0 \\
w_{k-1} & w_k & \ldots & 0 & 0 \\
w_{k-2} & w_{k-1} & \ldots & 0 & 0 \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
w_1 & \ldots & \ldots & w_k & 0 \\
0 & w_1 & \ldots & w_{k-1} & w_k \\
\vdots & \vdots & \ddots & \vdots & \vdots \\
0 & 0 & \vdots & w_1 & w_2 \\
0 & 0 & \vdots & 0 & w_1
\end{bmatrix}
\]

\[
\begin{bmatrix}
x_1 \\
x_2 \\
x_3 \\
\vdots \\
x_n
\end{bmatrix}
\]