Basic CNNs

CAP 5415, FALL 2019.
Credits

• This lecture contains public domain materials from lectures by
  • Fei-Fei Li & Justin Johnson & Serena Yeung
  • Shubhendu Trivedi & Risi Kondor
  • Vicky Kalogeiton
A quote from a famous scientist...
Origin of CNNs?

- CNNs have been around for a long time.
- For example, Fukushima’s NeoCognitron (1980) introduced an architecture that is essentially what modern CNNs are
  - It was ahead of its time!
  - Learning mechanism was different than backpropagation
- Since then many researchers have contributed to the development of the field
Convolution

- Convolve image with kernel having weights $w$ (learned by backpropagation)
Convolution

$w^T x$
Convolutions

- What is the number of parameters?
Convolution Layer

32x32x3 image

5x5x3 filter

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

32x32x3 image

5x5x3 filter

Filters always extend the full depth of the input volume

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

activation map

32
32
3

28
1
28
Convolution Layer

32x32x3 image
5x5x3 filter

convolve (slide) over all spatial locations

consider a second, green filter

activation maps
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!
Stride

Strided convolution

Downsampling

Convolution
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

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

\[
\begin{align*}
\text{stride 1} & \Rightarrow (7 - 3)/1 + 1 = 5 \\
\text{stride 2} & \Rightarrow (7 - 3)/2 + 1 = 3 \\
\text{stride 3} & \Rightarrow (7 - 3)/3 + 1 = 2.33 \\
\end{align*}
\]
Zero-Padding: common to 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>
<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?

(recall:)
(N - F) / stride + 1
Zero-Padding: common to 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 $5 \times 5$ filters with stride 1, pad 2

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

- 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.
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:

- $F = 3$, $S = 1$, $P = 1$
- $F = 5$, $S = 1$, $P = 2$
- $F = 5$, $S = 2$, $P = ?$ (whatever fits)
- $F = 1$, $S = 1$, $P = 0$
Activation Functions

Computes \( f(x) = \max(0, x) \)

- Does not saturate (in +region)
- Very computationally efficient
- Converges much faster than sigmoid/tanh in practice (e.g. 6x)

ReLU
(Rectified Linear Unit)

- Not zero-centered output
- ReLU units can “die”
Activation Functions

- Does not saturate
- Computationally efficient
- Converges much faster than sigmoid/tanh in practice! (e.g. 6x)
- will not “die”.

Leaky ReLU

\[ f(x) = \max(0.01x, x) \]

[Mass et al., 2013] [He et al., 2015]
In practice

- Use ReLU. Be careful with your learning rates
- Try out Leaky ReLU / Maxout / ELU
- Try out tanh but don’t expect much
- Don’t use sigmoid
Pooling

max pooling

20 30
112 37

average pooling

13 8
79 20

Effect = invariance to small translations of the input
Pooling

- makes the representations smaller and more manageable
- operates over each activation map independently
Max Pooling

Single depth slice

<table>
<thead>
<tr>
<th>1</th>
<th>1</th>
<th>2</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

max pool with 2x2 filters and stride 2

<table>
<thead>
<tr>
<th>6</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>
Preprocessing data
Preprocessing data
Preprocessing data

original data

decorrelated data

whitened data
In practice: for images

e.g. consider CIFAR-10 example with [32,32,3] images

- Subtract the mean image (e.g. AlexNet)
  (mean image = [32,32,3] array)

- Subtract per-channel mean (e.g. VGGNet)
  (mean along each channel = 3 numbers)

Not common to normalize variance, to do PCA or whitening
Weights initialization

- If the weights in a network start too small, then the signal shrinks as it passes through each layer until it’s too tiny to be useful.
- If the weights in a network start too large, then the signal grows as it passes through each layer until it’s too massive to be useful.
Weights initialization

• All zero initialization

• Small random numbers

• Draw weights from a Gaussian distribution with standard deviation of $\sqrt{2/n}$, where $n$ is the number of outputs to the neuron
Batch normalization

Initialization of NNs by explicitly forcing the activations throughout the network to take on a unit Gaussian distribution at the beginning of the training.

Normalization is a simple differentiable operation
Batch normalization

Usually inserted after Fully Connected and/or Convolutional layers, and before nonlinearity.
Batch normalization

- Improves gradient flow through the network
- Allows higher learning rates
- Reduces the strong dependence on initialization
- Acts as a form of regularization in a funny way, and slightly reduces the need for dropout
Case Study: AlexNet  
[Krizhevsky et al. 2012]

Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4  
=>
Output volume [55x55x96]  
Parameters: (11*11*3)*96 = 35K
Case Study: AlexNet

[Krizhevsky et al. 2012]

Input: 227x227x3 images
After CONV1: 55x55x96

Second layer (POOL1): 3x3 filters applied at stride 2
Output volume: 27x27x96
Parameters: 0!
Case Study: AlexNet
[Krizhevsky et al. 2012]

Full (simplified) AlexNet architecture:
[227x227x3] INPUT
[55x55x96] CONV1: 96 11x11 filters at stride 4, pad 0
[27x27x96] MAX POOL1: 3x3 filters at stride 2
[27x27x96] NORM1: Normalization layer
[27x27x256] CONV2: 256 5x5 filters at stride 1, pad 2
[13x13x256] MAX POOL2: 3x3 filters at stride 2
[13x13x256] NORM2: Normalization layer
[13x13x384] CONV3: 384 3x3 filters at stride 1, pad 1
[13x13x384] CONV4: 384 3x3 filters at stride 1, pad 1
[13x13x256] CONV5: 256 3x3 filters at stride 1, pad 1
[6x6x256] MAX POOL3: 3x3 filters at stride 2
[4096] FC6: 4096 neurons
[4096] FC7: 4096 neurons
[1000] FC8: 1000 neurons (class scores)

Details/Retrospectives:
- first use of ReLU
- used Norm layers (not common anymore)
- heavy data augmentation
- dropout 0.5
- batch size 128
- SGD Momentum 0.9
- Learning rate 1e-2, reduced by 10 manually when val accuracy plateaus
- L2 weight decay 5e-4
- 7 CNN ensemble: 18.2% -> 15.4%
Layer 1 filters

This and the next few illustrations are from Rob Fergus
Layer 2 Patches

Layer 2: Top-9 Patches

- Patches from validation images that give maximal activation of a given feature map
Layer 2 Patches

Layer 2: Top-9 Patches
Layer 3 Patches

Layer 3: Top-9 Patches
Layer 4 Patches

Layer 4: Top-9 Patches
Layer 4 Patches

Layer 4: Top-9 Patches
Evolution of Filters
Evolution of Filters
Credits

- This lecture contains public domain materials from lectures by
  - Fei-Fei Li & Justin Johnson & Serena Yeung
  - Shubhendu Trivedi & Risi Kondor
  - Vicky Kalogeiton