Convolutional Neural Networks
Lecture in Neural Networks course, LTAT.02.001

Ardi Tampuu

Tartu, March 2021
Disclaimer

These slides are adapted from Stanford University course CS231 on "Convolutional Neural Networks for Visual Recognition"
So far you have seen that there are shallow and deep networks.

You have seen that deep networks can be trained using backpropagation.

You have heard that deep networks can learn more complex things than shallow ones.
Shallow network weights (Practice 2: Softmax):

Product between input and weights gives higher value if the image is similar to weights → weights look like average of each class.
Seen so far

Deep network:

- can capture more complex patterns (not just pixelwise similarity)
- can hopefully recognize any color of car, viewpoint variations, ..
Desired data of artificial vision system:

- can recognize any color of the object, different view angles, variations in shape
- can recognize anywhere on the image
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...

Fei-Fei Li, Ranjay Krishna, Danfei Xu

Lecture 5 - 42

April 21, 2019
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 Lone 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]

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

"AlexNet"
Fast-forward to today: ConvNets are everywhere

Fei-Fei Li, Ranjay Krishna, Danfei Xu
*Lecture 5 - 48* April 21, 2019
Fast-forward to today: ConvNets are everywhere

Detection

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

[Guo et al. 2014]

Fast-forward to today: ConvNets are everywhere

[Levy et al. 2016]

Figure copyright Levy et al. 2016. Reproduced with permission.

[Dieleman et al. 2014]

From left to right: public domain by NASA, usage permitted by [NASA logo, public domain by NASA, and public domain].

[Serbanet et al. 2011]

[Cauesan et al.]
Whale recognition, Kaggle Challenge

Mnih and Hinton, 2010
Convolutional Neural Networks
Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1

\[ W x \]

Weights: 10 x 3072
Fully Connected Layer

32x32x3 image -> stretch to 3072 x 1 (one long column)

\[
\text{dim}(W) = (10, 3072)
\]

Product dims: (10,3072) x (3072,1) -> (10,1)
Fully Connected Layer

32x32x3 image -> stretch to $3072 \times 1$ 1 x 3072 (row)

$xW$

3072 x 10 weights

Product dims: $(1,3072) \times (3072,10)$

$\rightarrow (1,10)$

N inputs: $(N,3072) \times (3072,10)$

$\rightarrow (N,10)$
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

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 $\mathbf{w}$

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)

$\mathbf{w}^T \mathbf{x} + b$
Convolution Layer
Convolution Layer
Convolution Layer
Convolution Layer
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
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.
Preview: ConvNet is a sequence of Convolution Layers, interspersed with activation functions.
Preview

[Zeiler and Fergus 2013]

Visualization of VGG-16 by Lana McIntosh. VGG-16 architecture from [Simonyan and Zisserman 2014].
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 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 \(\Rightarrow\) \((7 - 3)/1 + 1 = 5\)
- stride 2 \(\Rightarrow\) \((7 - 3)/2 + 1 = 3\)
- stride 3 \(\Rightarrow\) \((7 - 3)/3 + 1 = 2.33\)
In practice: Common to zero pad the border

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

(recall:)

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

- 0 0 0 0 0 0
- 0
- 0
- 0
- 0

E.g. input 7x7

3x3 filter, applied with stride 1

Pad with 1 pixel border => what is the output?

7x7 output!

(recall:)

\[(N + 2P - F) / \text{stride} + 1\]
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 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: **32x32x3**
10 5x5 filters with stride 1, pad 2

Output volume size: ?
Examples time:

Input volume: $32\times32\times3$
10 5x5 filters with stride 1, pad 2

Output volume size:
$\frac{(32+2\times2-5)}{1}+1 = 32$ spatially, so
$32\times32\times10$
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 $5 \times 5$ 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)
$\Rightarrow 76 \times 10 = 760$
Convolution layer: summary

Let’s assume input is $W_1 \times H_1 \times C$
Conv layer needs 4 hyperparameters:
- Number of filters $K$
- The filter size $F$
- The stride $S$
- The zero padding $P$
This will produce an output of $W_2 \times H_2 \times K$
where:
- $W_2 = (W_1 - F + 2P)/S + 1$
- $H_2 = (H_1 - F + 2P)/S + 1$
Number of parameters: $F^2CK$ and $K$ biases
Convolution layer: summary

Let’s assume input is $W_1 \times H_1 \times C$

Conv layer needs 4 hyperparameters:

- Number of filters $K$
- The filter size $F$
- The stride $S$
- The zero padding $P$

This will produce an output of $W_2 \times H_2 \times K$

where:

- $W_2 = (W_1 - F + 2P)/S + 1$
- $H_2 = (H_1 - F + 2P)/S + 1$

Number of parameters: $F^2CK$ and $K$ biases

Common settings:

- $F = 3, S = 1, P = 1$
- $F = 5, S = 1, P = 2$
- $F = 5, S = 2, P = \,? \ (whatever \ fits)$
- $F = 1, S = 1, P = 0$
(btw, 1x1 convolution layers make perfect sense)

1x1 CONV with 32 filters

(each filter has size 1x1x64, and performs a 64-dimensional dot product)
(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 Keras

Conv2D

keras.layers.Conv2D(filters, kernel_size, strides=(1, 1), padding='valid', data_format='channels_last', d)

2D convolution layer (e.g. spatial convolution over images).

This layer creates a convolution kernel that is convolved with the layer input to produce a tensor of convolutions. If `dim_ordering` is 'th', a bias vector is created and added to the output. Finally, if `activation` is not None, it is applied to the output as well.

When using this layer as the first layer in a model, provide the argument `input_shape` (tuple of integers, does not include the batch axis, e.g. `input_shape=(128, 128, 2)` for 128x128 RGB pictures in 'channels_last' format).

Arguments

- `filters`: integer, the number of output filters in the convolution.
- `kernel_size`: An integer or tuple/list of 2 integers, specifying the height and width of the 2D convolution window. Can be a single integer to specify the same value for all spatial dimensions.
- `strides`: An integer or tuple/list of 2 integers, specifying the strides of the convolution along the height and width. Can be a single integer to specify the same value for all spatial dimensions.
- `activation`: one of `None`, `linear`, `relu`, `tanh`, etc. (default: `linear`).
- `padding`: one of `valid` or `same` (case-insensitive). Note that `same` is slightly inconsistent across backends with `strides > 1`, as described here.
- `data_format`: A string, one of `channels_last` or `channels_first`. The ordering of the dimensions in the inputs. `channels_last` corresponds to inputs with shape (batch, height, width, channels) while `channels_first` corresponds to inputs with shape (batch, channels, height, width). It defaults to the `image_data_format` value found in your Keras config file at `~/.keras/keras.json`. If you never set it, then it will be `channels_last`.

Fei-Fei Li, Ranjay Krishna, Danfei Xu

Lecture 5 - 10

April 21, 2019
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. 5x5x3 = 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

\[ W \times x \]

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)
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

Fei-Fei Li, Ranjay Krishna, Danfei Xu  Lecture 5  April 21, 2019
Convolution layer: summary

Let’s assume input is $W_1 \times H_1 \times C$
Conv layer needs 2 hyperparameters:
- The spatial extent $F$
- The stride $S$

This will produce an output of $W_2 \times H_2 \times C$ where:
- $W_2 = (W_1 - F)/S + 1$
- $H_2 = (H_1 - F)/S + 1$

Number of parameters: 0
Fully Connected Layer (FC layer)
- Contains neurons that connect to the entire input volume, as in ordinary Neural Networks
Summary

- ConvNets stack CONV, POOL, FC layers
- Trend towards smaller filters and deeper architectures
- Trend towards getting rid of POOL/FC layers (just CONV)
- Historically architectures looked like
  \[(\text{CONV-RELU}^N \cdot \text{POOL}?)^M \cdot \text{(FC-RELU)}^K, \text{SOFTMAX}\]
  where N is usually up to \sim 5, M is large, 0 \leq K \leq 2.
  - but recent advances such as ResNet/GoogLeNet have challenged this paradigm
CNN Architectures
Today: CNN Architectures

Case Studies
- AlexNet
- VGG
- GoogLeNet
- ResNet

Also....
- SENet
- Wide ResNet
- ResNeXt
- DenseNet
- MobileNets
- NASNet
- EfficientNet
Conv filters were 5x5, applied at stride 1
Subsampling (Pooling) layers were 2x2 applied at stride 2
i.e. architecture is [CONV-POOL-CONV-POOL-FC-FC]
ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

- **2010**: Lin et al
- **2011**: Sanchez & Perronnin
- **2012**: Krizhevsky et al (AlexNet)
- **2013**: Zeiler & Fergus
- **2014**: Simonyan & Zisserman (VGG)
- **2014**: Szegedy et al (GoogleLeNet)
- **2015**: He et al (ResNet)
- **2016**: Shao et al
- **2017**: Hu et al (SENet)
- **Human**

**Layers:**
- **2010**: shallow (Unknown)
- **2011**: 8 layers
- **2012**: 8 layers
- **2013**: 8 layers
- **2014**: 19 layers
- **2014**: 22 layers
- **2015**: 152 layers
- **2016**: 152 layers
- **2017**: 152 layers
- **Human**: 5.1 layers
ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

First CNN-based winner

- **2010**: Lin et al
- **2011**: Sanchez & Perronnin
- **2012**: Krizhevsky et al (AlexNet)
- **2013**: Zeiler & Fergus
- **2014**: Simonyan & Zisserman (VGG)
- **2014**: Szegedy et al (GoogLeNet)
- **2015**: He et al (ResNet)
- **2016**: Shao et al
- **2017**: Hu et al (SENet)
- **Human**

- **Shallow**: 8 layers
- **19 layers**: 22 layers
- **152 layers**: 152 layers

Fei-Fei Li, Ranjay Krishna, Danfei Xu

Lecture 9 - 32

May 5, 2020
Case Study: AlexNet

[Krizhevsky et al. 2012]

Architecture:
CONV1
MAX POOL1
NORM1
CONV2
MAX POOL2
NORM2
CONV3
CONV4
CONV5
Max POOL3
FC6
FC7
FC8

Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.
Case Study: AlexNet

[Krizhevsky et al. 2012]

Input: 227x227x3 images

**First layer** (CONV1): 96 11x11 filters applied at stride 4

\[ W' = \frac{(W - F + 2P)}{S + 1} \]

=>

Q: what is the output volume size? Hint: \((227-11)/4+1 = 55\)
Case Study: AlexNet

[Krizhevsky et al. 2012]

Input: 227x227x3 images

**First layer (CONV1):** 96 11x11 filters applied at stride 4

=>

Output volume [55x55x96]

\[ W' = (W - F + 2P) / S + 1 \]

---

Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.
Case Study: AlexNet

[Krizhevsky et al. 2012]

Input: 227x227x3 images

**First layer (CONV1):** 96 11x11 filters applied at stride 4

=>

Output volume **[55x55x96]**

Q: What is the total number of parameters in this layer?

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

Q: what is the output volume size? Hint: (55-3)/2+1 = 27

$W' = (W - F + 2P) / S + 1$

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

\[ W' = \frac{(W - F + 2P)}{S + 1} \]

Q: what is the number of parameters in this layer?

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

Input: 227x227x3 images
After CONV1: 55x55x96
After POOL1: 27x27x96
...

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

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

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

Fei-Fei Li, Ranjay Krishna, Danfei Xu

Lecture 9 - 43

May 5, 2020
ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

- **2010**: Lin et al
- **2011**: Sanchez & Perronnin
- **2012**: Krizhevsky et al (AlexNet)
- **2013**: Zeiler & Fergus
- **2014**: Simonyan & Zisserman (VGG)
- **2014**: Szegedy et al (GoogleNet)
- **2015**: He et al (ResNet)
- **2016**: Shao et al
- **2017**: Hu et al (SENet)
- **Human**

- **First CNN-based winner**

- **2010** - Shallow
- **2011** - 8 layers
- **2012** - 8 layers
- **2013** - 19 layers
- **2014** - 22 layers
- **2015** - 152 layers
- **2016** - 152 layers
- **2017** - 152 layers

Fei-Fei Li, Ranjay Krishna, Danfei Xu
ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

- 2010: Lin et al
- 2011: Sanchez & Perronnin
- 2012: Krizhevsky et al (AlexNet)
- 2013: Zeiler & Fergus
- 2014: Simonyan & Zisserman (VGG)
- 2014: Szegedy et al (GoogLeNet)
- 2015: He et al (ResNet)
- 2016: Shao et al
- 2017: Hu et al (SENet)
- Human

ZFNet: Improved hyperparameters over AlexNet

- 152 layers
- 152 layers
- 152 layers

- 19 layers
- 22 layers
ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

- 2010: 28.2
  - Lin et al
- 2011: 25.8
  - Sanchez & Perronnin
- 2012: 16.4
  - Krizhevsky et al (AlexNet)
- 2013: 11.7
  - Zeiler & Fergus
- 2014:
  - 7.3: Simonyan & Zisserman (VGG)
  - 6.7: Szegedy et al (GoogLeNet)
- 2015: 3.6
  - He et al (ResNet)
- 2016: 3
  - Shao et al
- 2017: 2.3
  - Hu et al (SENet)
- Human: 5.1

Deeper Networks:
- 19 layers (2014)
- 22 layers (2014)

Fei-Fei Li, Ranjay Krishna, Danfei Xu  Lecture 9 - 50  May 5, 2020
Small filters, Deeper networks

8 layers (AlexNet)
-> 16 - 19 layers (VGG16Net)

Only 3x3 CONV stride 1, pad 1
and 2x2 MAX POOL stride 2

11.7% top 5 error in ILSVRC’13 (ZFNet)
-> 7.3% top 5 error in ILSVRC’14
Case Study: VGGNet

[Simonyan and Zisserman, 2014]

Q: Why use smaller filters? (3x3 conv)
Case Study: VGGNet

[Simonyan and Zisserman, 2014]

Q: Why use smaller filters? (3x3 conv)

Stack of three 3x3 conv (stride 1) layers has same **effective receptive field** as one 7x7 conv layer

Q: What is the effective receptive field of three 3x3 conv (stride 1) layers?
Case Study: VGGNet

[Simonyan and Zisserman, 2014]

Q: What is the effective receptive field of three 3x3 conv (stride 1) layers?

Fei-Fei Li, Ranjay Krishna, Danfei Xu
Lecture 9 - 54
May 5, 2020
Case Study: VGGNet
[Simonyan and Zisserman, 2014]

Q: What is the effective receptive field of three 3x3 conv (stride 1) layers?
Case Study: VGGNet
[Simonyan and Zisserman, 2014]

Q: What is the effective receptive field of three 3x3 conv (stride 1) layers?
Case Study: VGGNet

[Simonyan and Zisserman, 2014]

Q: What is the effective receptive field of three 3x3 conv (stride 1) layers?
Q: What is the effective receptive field of three 3x3 conv (stride 1) layers?
Case Study: VGGNet
[Simonyan and Zisserman, 2014]

Q: Why use smaller filters? (3x3 conv)

Stack of three 3x3 conv (stride 1) layers has same **effective receptive field** as one 7x7 conv layer

[7x7]
Case Study: VGGNet

[Simonyan and Zisserman, 2014]

Q: Why use smaller filters? (3x3 conv)

Stack of three 3x3 conv (stride 1) layers has same **effective receptive field** as one 7x7 conv layer

But deeper, more non-linearities

And fewer parameters: 3 \* (3^2C^2) vs. 7^2C^2 for C channels per layer

Fei-Fei Li, Ranjay Krishna, Danfei Xu

Lecture 9 - 60

May 5, 2020
INPUT: [224x224x3] memory: 224*224*3 = 150K params: 0 (not counting biases)
CONV3-64: [224x224x64] memory: 224*224*64 = 3.2M params: (3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64 = 3.2M params: (3*3)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64 = 800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128 = 1.6M params: (3*3)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128 = 1.6M params: (3*3)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128 = 400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256 = 800K params: (3*3)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256 = 800K params: (3*3)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256 = 800K params: (3*3)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256 = 200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512 = 400K params: (3*3)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512 = 400K params: (3*3)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512 = 400K params: (3*3)*512 = 2,359,296
POOL2: [14x14x512] memory: 14*14*512 = 100K params: 0
CONV3-512: [14x14x512] memory: 14*14*512 = 100K params: (3*3)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512 = 100K params: (3*3)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512 = 100K params: (3*3)*512 = 2,359,296
POOL2: [7x7x512] memory: 7*7*512 = 25K params: 0
FC: [1x1x4096] memory: 4096 params: 7*7*512*4096 = 102,760,448
FC: [1x1x4096] memory: 4096 params: 4096*4096 = 16,777,216
FC: [1x1x1000] memory: 1000 params: 4096*1000 = 4,096,000

TOTAL memory: 24M * 4 bytes ~= 96MB / image (only forward! ~2 for bwd)
TOTAL params: 138M parameters

Note:
Most memory is in early CONV
Most params are in late FC

Fei-Fei Li, Ranjay Krishna, Danfei Xu

Lecture 9 - 63
May 5, 2020
Case Study: VGGNet

[Simonyan and Zisserman, 2014]

Details:
- ILSVRC’14 2nd in classification, 1st in localization
- Similar training procedure as Krizhevsky 2012
- No Local Response Normalisation (LRN)
- Use VGG16 or VGG19 (VGG19 only slightly better, more memory)
- Use ensembles for best results
- FC7 features generalize well to other tasks
ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

Deeper Networks

- 2010: 28.2 - Lin et al
- 2011: 25.8 - Sanchez & Perronnin
- 2012: 16.4 - Krizhevsky et al (AlexNet)
- 2013: 11.7 - Zeiler & Fergus
- 2014: 7.3 - Simonyan & Zisserman (VGG)
- 2014: 6.7 - Szegedy et al (GoogLeNet)
- 2015: 3.6 - He et al (ResNet)
- 2016: 3 - Shao et al
- 2017: 2.3 - Hu et al (SENet)
- Human: 5.1

Layers:
- Shallow: 8 layers
- Deeper: 19 layers, 22 layers, 152 layers

Fei-Fei Li, Ranjay Krishna, Danfei Xu
Lecture 9 - May 5, 2020
Case Study: GoogLeNet

[ Szegedy et al., 2014 ]

Deeper networks, with computational efficiency

- ILSVRC’14 classification winner (6.7% top 5 error)
- 22 layers
- Only 5 million parameters!
  12x less than AlexNet
  27x less than VGG-16
- Efficient “Inception” module
- No FC layers

Inception module
“Inception module”: design a good local network topology (network within a network) and then stack these modules on top of each other.
Case Study: GoogLeNet

[Szegedy et al., 2014]

Apply parallel filter operations on the input from previous layer:
- Multiple receptive field sizes for convolution (1x1, 3x3, 5x5)
- Pooling operation (3x3)

Concatenate all filter outputs together channel-wise
Case Study: GoogLeNet

[Szegedy et al., 2014]

Apply parallel filter operations on the input from previous layer:
- Multiple receptive field sizes for convolution (1x1, 3x3, 5x5)
- Pooling operation (3x3)

Concatenate all filter outputs together channel-wise

Q: What is the problem with this?
[Hint: Computational complexity]
Case Study: GoogLeNet
[Szegedy et al., 2014]

Example:

Q2: What is output size after filter concatenation?

\[
28 \times 28 \times (128 + 192 + 96 + 256) = 28 \times 28 \times 672
\]

Conv Ops:
[1x1 conv, 128] 28x28x128x1x1x256
[3x3 conv, 192] 28x28x192x3x3x256
[5x5 conv, 96] 28x28x96x5x5x256
Total: 854M ops

Very expensive compute

Pooling layer also preserves feature depth, which means total depth after concatenation can only grow at every layer!
Case Study: GoogLeNet

[Szegedy et al., 2014]

Example:

Q2: What is output size after filter concatenation?

$28 \times 28 \times (128 + 192 + 96 + 256) = 529k$

Solution: “bottleneck” layers that use 1x1 convolutions to reduce feature channel size.

Naive Inception module
Case Study: GoogLeNet

[Szegedy et al., 2014]

Naive Inception module

Inception module with dimension reduction
Review: 1x1 convolutions

1x1 CONV with 32 filters

(each filter has size 1x1x64, and performs a 64-dimensional dot product)
Review: 1x1 convolutions

Alternatively, interpret it as applying the same FC layer on each input pixel.

1x1 CONV with 32 filters
(each filter has size 1x1x64, and performs a 64-dimensional dot product)
Review: 1x1 convolutions

Alternatively, interpret it as applying the same FC layer on each input pixel.

1x1 CONV with 32 filters preserves spatial dimensions, reduces depth!

Projects depth to lower dimension (combination of feature maps)
Case Study: GoogLeNet

[Szegedy et al., 2014]

Naive Inception module

Inception module with dimension reduction
Case Study: GoogLeNet

[Szegedy et al., 2014]

Using same parallel layers as naive example, and adding “1x1 conv, 64 filter” bottlenecks:

**Conv Ops:**

- [1x1 conv, 64] 28x28x64x1x1x256
- [1x1 conv, 64] 28x28x64x1x1x256
- [1x1 conv, 128] 28x28x128x1x1x256
- [3x3 conv, 192] 28x28x192x3x3x64
- [5x5 conv, 96] 28x28x96x5x5x64
- [1x1 conv, 64] 28x28x64x1x1x256

**Total:** 358M ops

Compared to 854M ops for naive version
Bottleneck can also reduce depth after pooling layer
Case Study: GoogLeNet

[Szegedy et al., 2014]

Stack Inception modules with dimension reduction on top of each other

Inception module
Case Study: GoogLeNet

[Szegedy et al., 2014]

Full GoogLeNet architecture

Auxiliary classification outputs to inject additional gradient at lower layers
(AvgPool-1x1Conv-FC-FC-Softmax)
Case Study: GoogLeNet

[ Szegedy et al., 2014 ]

Deeper networks, with computational efficiency

- 22 layers
- Efficient “Inception” module
- Avoids expensive FC layers
- 12x less params than AlexNet
- 27x less params than VGG-16
- ILSVRC’14 classification winner
  (6.7% top 5 error)
ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

“Revolution of Depth”

- **2010**: 28.2
  - Lin et al.
  - Shallow

- **2011**: 25.8
  - Sanchez & Perronnin
  - 8 layers

- **2012**: 16.4
  - Krizhevsky et al. (AlexNet)
  - 8 layers

- **2013**: 11.7
  - Zeiler & Fergus
  - 8 layers

- **2014**: 7.3
  - Simonyan & Zisserman (VGG)
  - 19 layers

- **2014**: 6.7
  - Szegedy et al. (GoogLeNet)
  - 22 layers

- **2015**: 3.6
  - He et al. (ResNet)
  - 152 layers

- **2016**: 3
  - Shao et al.
  - 152 layers

- **2017**: 2.3
  - Hu et al. (SENet)
  - 152 layers

- **Human**: 5.1

Fei-Fei Li, Ranjay Krishna, Danfei Xu

Lecture 9 - 93
May 5, 2020
Case Study: ResNet

[He et al., 2015]

Very deep networks using residual connections

- 152-layer model for ImageNet
- ILSVRC’15 classification winner (3.57% top 5 error)
- Swept all classification and detection competitions in ILSVRC’15 and COCO’15!
Case Study: ResNet

[He et al., 2015]

What happens when we continue stacking deeper layers on a “plain” convolutional neural network?
Case Study: ResNet
[He et al., 2015]

What happens when we continue stacking deeper layers on a “plain” convolutional neural network?
Case Study: ResNet

[He et al., 2015]

What happens when we continue stacking deeper layers on a “plain” convolutional neural network?

56-layer model performs worse on both training and test error

-> The deeper model performs worse, but it’s not caused by overfitting!
Case Study: ResNet

[He et al., 2015]

Fact: Deep models have more representation power (more parameters) than shallower models.

Hypothesis: the problem is an optimization problem, deeper models are harder to optimize
Case Study: ResNet

[He et al., 2015]

Solution: Use network layers to fit a residual mapping instead of directly trying to fit a desired underlying mapping

Identity mapping:
\[ H(x) = x \text{ if } F(x) = 0 \]

Use layers to fit residual
\[ F(x) = H(x) - x \]

instead of
\[ H(x) \text{ directly} \]
Case Study: ResNet

[He et al., 2015]

Full ResNet architecture:
- Stack residual blocks
- Every residual block has two 3x3 conv layers
- Periodically, double # of filters and downsample spatially using stride 2 (\(\frac{1}{2}\) in each dimension)
- Additional conv layer at the beginning (stem)
- No FC layers at the end (only FC 1000 to output classes)
- (In theory, you can train a ResNet with input image of variable sizes)
Case Study: ResNet

Total depths of 18, 34, 50, 101, or 152 layers for ImageNet
Case Study: ResNet

[He et al., 2015]

For deeper networks (ResNet-50+), use “bottleneck” layer to improve efficiency (similar to GoogLeNet)
Case Study: ResNet

[He et al., 2015]

Training ResNet in practice:

- Batch Normalization after every CONV layer
- Xavier initialization from He et al.
- SGD + Momentum (0.9)
- Learning rate: 0.1, divided by 10 when validation error plateaus
- Mini-batch size 256
- Weight decay of 1e-5
- No dropout used
Case Study: ResNet

[He et al., 2015]

Experimental Results
- Able to train very deep networks without degrading (152 layers on ImageNet, 1202 on Cifar)
- Deeper networks now achieve lower training error as expected
- Swept 1st place in all ILSVRC and COCO 2015 competitions

**MSRA @ ILSVRC & COCO 2015 Competitions**

- **1st places in all five main tracks**
  - ImageNet Classification: “Ultra-deep” (quote Yann) 152-layer nets
  - ImageNet Detection: 16% better than 2nd
  - ImageNet Localization: 27% better than 2nd
  - COCO Detection: 11% better than 2nd
  - COCO Segmentation: 12% better than 2nd

ILSVRC 2015 classification winner (3.6% top 5 error) -- better than “human performance”! (Russakovsky 2014)
Batch Normalization
Batch Normalization

“you want unit gaussian activations? just make them so.”

consider a batch of activations at some layer. To make each dimension unit gaussian, apply:

\[
\hat{x}(k) = \frac{x^{(k)} - \mathbb{E}[x^{(k)}]}{\sqrt{\text{Var}[x^{(k)}]}}
\]

this is a vanilla differentiable function...
Batch Normalization

“I want unit Gaussian activations? just make them so.”

1. Compute the empirical mean and variance independently for each dimension.

2. Normalize

\[
\hat{x}(k) = \frac{x(k) - E[x(k)]}{\sqrt{\text{Var}[x(k)]}}
\]
Batch Normalization

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

\[
\hat{x}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{\text{Var}[x^{(k)}]}}
\]
Batch Normalization

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

Problem: do we necessarily want a unit gaussian input to a tanh layer?

\[
\hat{x}^{(k)} = \frac{x^{(k)} - E[x^{(k)}]}{\sqrt{\text{Var}[x^{(k)}]}}
\]
Batch Normalization

Normalize:

\[ \hat{x}(k) = \frac{x(k) - E[x(k)]}{\sqrt{\text{Var}[x(k)]}} \]

And then allow the network to squash the range if it wants to:

\[ y(k) = \gamma(k) \hat{x}(k) + \beta(k) \]

Note, the network can learn:

\[ \gamma(k) = \sqrt{\text{Var}[x(k)]} \]
\[ \beta(k) = E[x(k)] \]

to recover the identity mapping.
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, maybe

**Input:** Values of $x$ over a mini-batch: $\mathcal{B} = \{x_1...m\}$; Parameters to be learned: $\gamma, \beta$

**Output:** $\{y_i = \text{BN}_{\gamma, \beta}(x_i)\}$

\[
\begin{align*}
\mu_B & \leftarrow \frac{1}{m} \sum_{i=1}^{m} x_i \quad \text{\textit{\small // mini-batch mean}} \\
\sigma_B^2 & \leftarrow \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_B)^2 \quad \text{\textit{\small // mini-batch variance}} \\
\hat{x}_i & \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \quad \text{\textit{\small // normalize}} \\
y_i & \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) \quad \text{\textit{\small // scale and shift}}
\end{align*}
\]
Batch Normalization

- Makes deep networks much easier to train!
- Improves gradient flow
- Allows higher learning rates, faster convergence
- Networks become more robust to initialization
- Acts as regularization during training
- Zero overhead at test-time: can be fused with conv!
- Behaves differently during training and testing: this is a very common source of bugs!
Batch Normalization

Input: Values of \( x \) over a mini-batch: \( B = \{x_1, \ldots, x_m\} \); Parameters to be learned: \( \gamma, \beta \)

Output: \( \{y_i = \text{BN}_{\gamma, \beta}(x_i)\} \)

\[
\mu_B \leftarrow \frac{1}{m} \sum_{i=1}^{m} x_i \quad // \text{mini-batch mean}
\]

\[
\sigma^2_B \leftarrow \frac{1}{m} \sum_{i=1}^{m} (x_i - \mu_B)^2 \quad // \text{mini-batch variance}
\]

\[
\hat{x}_i \leftarrow \frac{x_i - \mu_B}{\sqrt{\sigma^2_B + \epsilon}} \quad // \text{normalize}
\]

\[
y_i \leftarrow \gamma \hat{x}_i + \beta \equiv \text{BN}_{\gamma, \beta}(x_i) \quad // \text{scale and shift}
\]

Note: at test time BatchNorm layer functions differently:

The mean/\( \text{std} \) are not computed based on the batch. Instead, a single fixed empirical mean of activations during training is used.

(e.g. can be estimated during training with running averages)