CMP784
DEEP LEARNING

Lecture #05 – Convolutional Neural Networks

Aykut Erdem // Hacettepe University // Spring 2018
Previously on CMP784

• data preprocessing and normalization
• weight initializations
• ways to improve generalization
• babysitting the learning process
• hyperparameter selection
• optimization
Breaking news!

• Practical 1 is due Mar. 16, 23:59:59

• Project proposals is due Mar. 22!
  • about a half page
  • the research topic to be investigated,
  • what data you will use,
  • design overview,
  • a list of key readings.

• Practical 2 will be out on tomorrow!
  — Convolutional Neural Networks
  — Due Saturday, Apr. 6, 23:59:59

Image credit: wildml.com
Deadlines in the syllabus are closer than they appear.
Lecture Overview

• convolution layer
• pooling layer
• revolution of depth
• design guidelines
• residual connections
• semantic segmentation networks
• object detection networks
• backpropagation in CNNs

Disclaimer: Much of the material and slides for this lecture were borrowed from
— Andrea Vedaldi’s tutorial on Convolutional Networks for Computer Vision Applications
— Kaiming He’s ICML 2016 tutorial on Deep Residual Networks: Deep Learning Gets Way Deeper
— Ross Girshick’s talk on The Past, Present, and Future of Object Detection
— Fei-Fei Li, Andrej Karpathy and Justin Johnson’s CS231n class
Perceptron

[Rosenblatt 57]

• The goal is estimating the posterior probability of the binary label $y$ of a vector $x$:
Discovery of oriented cells in the visual cortex

[Hubel and Wiesel 59]
Convolution

• Convolution = Spatial filtering

\[(a * b)[i, j] = \sum_{i', j'} a[i', j']b[i - i', j - j']\]

• Different filters (weights) reveal a different characteristics of the input.
Convolution

• Convolution = Spatial filtering

\[(a \ast b)[i, j] = \sum_{i', j'} a[i', j'] b[i - i', j - j']\]

• Different filters (weights) reveal a different characteristics of the input.
Convolution

- Convolution = Spatial filtering

\[(a * b)[i, j] = \sum_{i', j'} a[i', j']b[i - i', j - j']\]

- Different filters (weights) reveal a different characteristics of the input.
Convolutional Neural Networks in a Nutshell

• A neural network model that consists of a sequence of local & translation invariant layers
  • Many identical copies of the same neuron: Weight/parameter sharing
  • Hierarchical feature learning

A bit of history

• Neurocognitron model by Fukushima (1980)
• The first convolutional neural network (CNN) model
• so-called “sandwich” architecture
  • simple cells act like filters
  • complex cells perform pooling
• Difficult to train
  • No backpropagation yet
A bit of history

• LeNet-5 model

A bit of history

- AlexNet model

Convolutional Neural Network

Convolutional layer

- Learn a filter bank (a set of filters) once
- Use them over the input data to extract features

\[ y = F \ast x + b \]
Data = 3D Tensors

• There is a vector of feature channels (e.g. RGB) at each spatial location (pixel).

\[ H \times W \times C = \text{channels} \]

\[ c = 1 \quad c = 2 \quad c = 3 \]

\[ \text{3D tensor} \]
Convolutions with 3D Filters

• Each filter acts on multiple input channels

  – **Local**
      Filters look locally

  – **Translation invariant**
      Filters act the same everywhere
Convolutional Layer

32x32x3 input

5x5x3 filter

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

1 number: the result of taking a dot product between the filter and a small 5x5x3 chunk of the input (i.e. 5*5*3 = 75-dimensional dot product + bias)
Convolutional Layer

32x32x3 input
5x5x3 filter

convolve (slide) over all spatial locations

activation map
Convolutional Layer

consider a second, green filter

32x32x3 input
5x5x3 filter

convolve (slide) over all spatial locations

activation maps
Convolutional Layer

- Multiple filters produce multiple output channels
- For example, if we had 6 5x5 filters, we’ll get 6 separate activation maps:

We stack these up to get an output of size 28x28x6.
Spatial Arrangement of Output Volume

- **Depth**: number of filters
- **Stride**: filter step size (when we “slide” it)
- **Padding**: zero-pad the input
Input Volume (+pad 1) (7x7x3)  
```
0 0 0 0 0 0 0
0 1 2 0 0 0 0
0 2 2 0 0 0 0
0 2 2 2 1 0 0
0 1 1 0 2 0 0
0 1 1 0 1 0 0
0 0 0 0 0 0 0
```

Filter W0 (3x3x3)  
```
-1 0 0
-1 0 0
-1 0 0
```

Filter W1 (3x3x3)  
```
-1 0 1
1 -1 1
0 0 -1
```

Output Volume (3x3x2)  
```
-3 -1 4
-2 -7 -4
1 -1 1
```

References: [95]
### Input Volume (+pad 1) (7x7x3)

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Filter W0 (3x3x3)

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Filter W1 (3x3x3)

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### Output Volume (3x3x2)

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### References

[95]
Convolutional layers

- Local receptive field
- Each column of hidden units looks at a different input patch
Convolutional layers

CONV, ReLU

e.g. 6
5x5x3 filters
Repeat linear / non-linear operators

3 CONV, ReLU e.g. 6 5x5x3 filters

6 28 24

5 CONV, ReLU e.g. 10 5x5x6 filters

10 24

….
Linear/Non-linear Chains

- The basic blueprint of most architectures
- Stack multiple layers of convolutions
Feature Learning

• Hierarchical layer structure allows to learn hierarchical filters (features).
Feature Learning

- Hierarchical layer structure allows to learn hierarchical filters (features).
Pooling layer

• makes the representations smaller and more manageable
• operates over each activation map independently:
• Max pooling, average pooling, etc.
Fully connected layer

• contains neurons that connect to the entire input volume, as in ordinary Neural Networks
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%
Convolutional Neural Network Demo

- ConvNetJS demo: training on CIFAR-10
Three Years of Progress
Revolution of Depth

AlexNet, 8 layers (ILSVRC 2012)

- 11x11 conv, 96, /4, pool/2
- 5x5 conv, 256, pool/2
- 3x3 conv, 384
- 3x3 conv, 384
- 3x3 conv, 256, pool/2
- fc, 4096
- fc, 4096
- fc, 1000

- 5 convolutional layers
- 3 fully connected layers
- ReLU
- End-to-end (no pre-training)
- Data augmentation
Revolution of Depth

AlexNet, 8 layers
(ILSVRC 2012)

VGG, 19 layers
(ILSVRC 2014)

- Very deep
- Simply deep

Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun.
INPUT: [224x224x3]  memory: 224*224*3 = 150K  params: 0

CONV3-64: [224x224x64] memory: 224*224*64 = 3.2M  params: (3*3*3)*64 = 1,728

CONV3-64: [224x224x64] memory: 224*224*64 = 3.2M  params: (3*3*64)*64 = 36,864

POOL2: [112x112x64] memory: 112*112*64 = 800K  params: 0

CONV3-128: [112x112x128] memory: 112*112*128 = 1.6M  params: (3*3*64)*128 = 73,728

CONV3-128: [112x112x128] memory: 112*112*128 = 1.6M  params: (3*3*128)*128 = 147,456

POOL2: [56x56x128] memory: 56*56*128 = 400K  params: 0

CONV3-256: [56x56x256] memory: 56*56*256 = 800K  params: (3*3*128)*256 = 294,912

CONV3-256: [56x56x256] memory: 56*56*256 = 800K  params: (3*3*256)*256 = 589,824

CONV3-256: [56x56x256] memory: 56*56*256 = 800K  params: (3*3*256)*256 = 589,824

POOL2: [28x28x256] memory: 28*28*256 = 200K  params: 0

CONV3-512: [28x28x512] memory: 28*28*512 = 400K  params: (3*3*256)*512 = 1,179,648

CONV3-512: [28x28x512] memory: 28*28*512 = 400K  params: (3*3*512)*512 = 2,359,296

CONV3-512: [28x28x512] memory: 28*28*512 = 400K  params: (3*3*512)*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)*512 = 2,359,296

CONV3-512: [14x14x512] memory: 14*14*512 = 100K  params: (3*3*512)*512 = 2,359,296

CONV3-512: [14x14x512] memory: 14*14*512 = 100K  params: (3*3*512)*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 ~ = 93MB / image  
(only forward! ~*2 for bwd)

TOTAL params: 138M parameters
INPUT: [224x224x3] memory: 224*224*3=150K params: 0 (not counting biases)
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*3)*64 = 1,728
CONV3-64: [224x224x64] memory: 224*224*64=3.2M params: (3*3*64)*64 = 36,864
POOL2: [112x112x64] memory: 112*112*64=800K params: 0
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*64)*128 = 73,728
CONV3-128: [112x112x128] memory: 112*112*128=1.6M params: (3*3*128)*128 = 147,456
POOL2: [56x56x128] memory: 56*56*128=400K params: 0
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 294,912
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,824
POOL2: [28x28x256] memory: 28*28*256=200K params: 0
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*256)*512 = 1,179,648
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*512 = 2,359,296
CONV3-512: [28x28x512] memory: 28*28*512=400K params: (3*3*512)*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)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*512 = 2,359,296
CONV3-512: [14x14x512] memory: 14*14*512=100K params: (3*3*512)*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 ~= 93MB / image
(Total forward! ~*2 for bwd)
TOTAL params: 138M parameters

VGG-16 Net
Revolution of Depth

AlexNet, 8 layers (ILSVRC 2012)
- 11x11 conv, 96, /4, pool/2
- 5x5 conv, 256, pool/2
- 3x3 conv, 384
- 3x3 conv, 384
- 3x3 conv, 384
- 3x3 conv, 256, pool/2
- fc, 4096
- fc, 4096
- fc, 1000

VGG, 19 layers (ILSVRC 2014)
- 3x3 conv, 64
- 3x3 conv, 64, pool/2
- 3x3 conv, 128
- 3x3 conv, 128, pool/2
- 3x3 conv, 256
- 3x3 conv, 256
- 3x3 conv, 256
- 3x3 conv, 256, pool/2
- 3x3 conv, 512
- 3x3 conv, 512
- 3x3 conv, 512
- 3x3 conv, 512, pool/2
- 3x3 conv, 512
- 3x3 conv, 512
- 3x3 conv, 512
- 3x3 conv, 512, pool/2
- fc, 4096
- fc, 4096
- fc, 1000

GoogLeNet, 22 layers (ILSVRC 2014)
- Branching
- Bootleneck
- Skip connection

Kaiming He, Xiangyu Zhang, Shaoqing Ren, & Jian Sun.

Simply deep
GoogLeNet

Inception module

[Szegedy et al., 2014]
GoogLeNet

Fun features:
- Only 5 million params! (Removes FC layers completely)

Compared to AlexNet:
- 12X less params
- 2x more compute
- 6.67% (vs. 16.4%)
Revolution of Depth

AlexNet, 8 layers (ILSVRC 2012)

VGG, 19 layers (ILSVRC 2014)

ResNet, 152 layers (ILSVRC 2015)
Revolution of Depth

AlexNet, 8 layers
(ILSVRC 2012)

VGG, 19 layers
(ILSVRC 2014)

ResNet, 152 layers
(ILSVRC 2015)
Residual Net (ResNet)

The ResNet architecture introduces the concept of residual learning, which allows the network to learn identity mappings more effectively. This is achieved by adding a shortcut connection that skips one or more layers, allowing the network to learn the difference between the input and the output directly. This helps in addressing the vanishing gradient problem that can occur in deep networks.

Mathematically, this can be represented as:

\[ H(x) = F(x) + x \]

where \( H(x) \) is the output of the network, \( F(x) \) is the feature map generated by the network, and \( x \) is the input. The addition of \( x \) helps in learning the residual mapping, which is particularly useful in deeper networks.
How deep is enough?

- GoogLeNet (2012)
- VGG-M (2013)
- VGG-VD-16 (2014)
- AlexNet (2012)

16 convolutional layers
50 convolutional layers
152 convolutional layers


How deep is enough?

• 3 × more accurate in 3 years
Speed

• 5 × slower

Remark: 101 ResNet layers same size/speed as 16 VGG-VD layers
Reason: Far fewer feature channels (quadratic speed/space gain)
Moral: Optimize your architecture
Model Size

- Num. of parameters is about the same

- **Remark:** 101 ResNet layers same size/speed as 16 VGG-VD layers
- **Reason:** Far fewer feature channels (quadratic speed/space gain)
- **Moral:** Optimize your architecture
ResNeXt: Both Wider and Deeper

• shortcut, bottleneck, and multi-branch
• Better accuracy (when having the same FLOPs/#params as ResNet)

Inception: heterogeneous multi-branch
ResNeXt: uniform multi-branch

Saining Xie et al. "Aggregated Residual Transformations for Deep Neural Networks". CVPR 2017
Dense Connections


Slide credit: Gao Huang
Dense Connections

: Element-wise addition

Dense Connections

+ : Element-wise addition

- Identity mappings promote gradient propagation.

Dense Connections

: Channel-wise concatenation

DenseNet

- 201 layers, 20M parameters
- Densely connected blocks
- Alleviates vanishing gradient
- Strengthens feature propagation
- Encourages feature reuse

Recent Advances: Design Guidelines
Design Guidelines

**Guideline 1: Avoid tight bottlenecks**

- **From bottom to top**
  - The spatial resolution $H \times W$ decreases
  - The number of channels $C$ increases

- **Guideline**
  - Avoid tight information bottleneck
  - Decrease the data volume $H \times W \times C$ slowly

---


Receptive Field

Must be large enough

• Receptive field of a neuron
  – The image region influencing a neuron
  – Anything happening outside is invisible to the neuron

• Importance
  – Large image structures cannot be detected by neurons with small receptive fields

• Enlarging the receptive field
  – Large filters
  – Chains of small filters
Guideline 2: Prefer small filter chains

**Remark:** 101 ResNet layers same size/speed as 16 VGG-VD layers

**Reason:** Far fewer feature channels (quadratic speed/space gain)

**Moral:** Optimize your architecture
Design Guidelines

Guideline 3: Keep the number of channels at bay

\[ H \times W \times C \]

- \( C = \text{num. input channels} \)
- \( K = \text{num. output channels} \)

Num. of operations

\[
\frac{H \times H_f}{\text{stride}} \times \frac{W \times W_f}{\text{stride}} \times C \times K
\]

Num. of parameters

\[ H_f \times W_f \times C \times K \]

complexity \( \propto C \times K \)
Guideline 4: Less computations with filter groups

$M$ filters

Consider instead $G$ groups of $M/G$ filters

split channels

filter groups

put back

complexity $\propto (C \times K) / G$
AlexNet

Design Guidelines

Guideline 4:
Less computations with filter groups

\[
\begin{align*}
\text{Full filters:} & & \begin{bmatrix} C \times K \end{bmatrix} \times \begin{bmatrix} x \end{bmatrix} \\
\text{complexity:} & & C \times K \\
\end{align*}
\]

\[
\begin{align*}
\text{Group-sparse filters:} & & \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \times \begin{bmatrix} x \end{bmatrix} \\
\text{complexity:} & & C \times K / G \\
\end{align*}
\]

Groups = filters, seen as a matrix, have a “block” structure
Design Guidelines

Guideline 5:
Low-rank decompositions

Filter bank $3 \times 3 \times C \times K$

Vertical $1 \times 3 \times C \times K$

Horizontal $3 \times 1 \times K \times K$

Groups $3 \times 3 \times C/G \times K/G$

"Network in network" $1 \times 1 \times K \times K$

Make sure to mix the information
Recent Advances:
Batch Normalization
Batch Normalization

- **Condition features**
  - Standardize the response of each feature channel within the batch
    - Average over spatial locations
    - Also, average over multiple images in the batch (e.g. 16-256)

- **Standardize** the response of each feature channel within the batch

Batch Normalization

• Training vs. testing modes

Training: batch-specific moment averages are collected

Testing: moment averages are used instead of batch-specific moments

• Moments (mean & variance)
  – Training: compute anew for each batch
  – Testing: fixed to their average values
Batch Normalization

• Utilization

  - Batch normalization is used after filtering, before ReLU
  - It is always followed by channel-specific scaling factor $s$ and bias $b$
  - Noisy bias/variance estimation replaces dropout regularization
Recent Advances:
Residual Learning
Residual Learning

Fixed identity // learned residual

\[ x_{n+5} = x_n + (\phi_{\text{ReLU}} \circ \phi_* \circ \phi_{\text{ReLU}} \circ \phi_*)(x_n) \]

Residual Learning

• “Overly deep” plain nets have higher training error
• A general phenomenon, observed in many datasets
• This is optimization issue, deeper models are harder to optimize
Residual Learning

• Richer solution space

• A deeper model should not have higher training error

• A solution by construction:
  – original layers: copied from a
  – learned shallower model
  – extra layers: set as identity
  – at least the same training error
Residual Learning

- The loss surface of a 56-layer net using the CIFAR-10 dataset, both without (left) and with (right) residual connections.

Hao Li et al., "Visualizing the Loss Landscape of Neural Nets". ICLR 2018
Summary

• **Impact of deep learning** in computer vision
  • 2012: Amazing results by AlexNet in the ImageNet challenge
  • 2013-15: Massive 3 improvement
  • 2016-19: Further massive improvements not unlikely

• What have we learned
  • Several *incremental refinements*
  • AlexNet was just a first proof of concept after all

• Things that work
  • Deeper architectures
  • Smarter architectures (groups, low rank decompositions, ...)
  • Batch normalization
  • Residual connections
Transfer Learning with Convolutional Neural Networks
Beyond CNNs

• Do features extracted from the CNN generalize other tasks and datasets?
  • Donahue et al. (2013), Chatfield et al. (2014), Razavian et al. (2014), Yosinski et al. (2014), etc.

• CNN activations as deep features
• Finetuning CNNs
CNN activations as deep features

• CNNs discover effective representations. Why not to use them?
CNN activations as deep features

- CNNs discover effective representations. Why not to use them?

Layer 1 Filters (Gabor and color blobs)
CNN activations as deep features

- CNNs discover effective representations. Why not to use them?

Layer 1 Filters (Gabor and color blobs)
Layer 2
Layer 5

Zeiler et al., 2014

Slide credit: Jason Yosinski
CNN activations as deep features

- CNNs discover effective representations. Why not?
CNNs as deep features

- CNNs discover effective representations. Why not to use them?

Transfer Learning with CNNs

- A CNN trained on a (large enough) dataset generalizes to other visual tasks
Transfer Learning with CNNs

• Keep layers 1-7 of our ImageNet-trained model fixed
• Train a new softmax classifier on top using the training images of the new dataset.

1. Train on Imagenet
2. Small dataset: feature extractor
   - Freeze these
   - Train this
3. Medium dataset: finetuning
   - more data = retrain more of the network (or all of it)
   - Freeze these
   - tip: use only ~1/10th of the original learning rate in finetuning top layer, and ~1/100th on intermediate layers
How transferable are features in CNN networks?

- Divide ImageNet into man-made objects A (449 classes) and natural objects B (551 classes)
- The transferability of features decreases as the distance between the base task and target task increases