Lecture 9: CNN Architectures
Administrative

A2 due Wed May 1.

** If late days past Friday are used, assignment will not be graded before midterm.
Administrative

**Midterm**: In-class Tue May 7. Covers material through Lecture 10 (Thu May 2).

Midterm room assignments will be posted on Piazza.

**Midterm review session**: Fri May 3 discussion section
Sample midterm will be released Wed on Piazza.
A3 will be released next Wed May 8, due Wed May 22
Last 2 lectures: Training neural networks

1. **One time setup**
   - activation functions, preprocessing, weight initialization,
   - regularization, gradient checking

2. **Training dynamics**
   - babysitting the learning process,
   - parameter updates, hyperparameter optimization

3. **Evaluation**
   - model ensembles, test-time augmentation
One more thing: Transfer Learning

“You need a lot of data if you want to train/use CNNs”
One more thing: Transfer Learning

“You need a lot of data if you want to train/use CNNs”
Transfer Learning with CNNs

1. Train on Imagenet

Razavian et al., “CNN Features Off-the-Shelf: An Astounding Baseline for Recognition”, CVPR Workshops 2014
Transfer Learning with CNNs

1. Train on Imagenet

2. Small Dataset (C classes)

Razavian et al., “CNN Features Off-the-Shelf: An Astounding Baseline for Recognition”, CVPR Workshops 2014
Transfer Learning with CNNs

1. Train on Imagenet

2. Small Dataset (C classes)

   - Freeze these
   - Reinitialize this and train

3. Bigger dataset

   - Freeze these
   - Train these
   - With bigger dataset, train more layers
   - Lower learning rate when finetuning; 1/10 of original LR is good starting point

---

Razavian et al., “CNN Features Off-the-Shelf: An Astounding Baseline for Recognition”, CVPR Workshops 2014
<table>
<thead>
<tr>
<th></th>
<th>very similar dataset</th>
<th>very different dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>very little data</strong></td>
<td>?</td>
<td>?</td>
</tr>
<tr>
<td><strong>quite a lot of data</strong></td>
<td>?</td>
<td>?</td>
</tr>
</tbody>
</table>

**More specific**

**More generic**
<table>
<thead>
<tr>
<th>More specific</th>
<th>More generic</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Image</strong></td>
<td></td>
</tr>
<tr>
<td>Conv-64</td>
<td>Conv-64</td>
</tr>
<tr>
<td>Conv-64</td>
<td>Conv-64</td>
</tr>
<tr>
<td>MaxPool</td>
<td>MaxPool</td>
</tr>
<tr>
<td>Conv-512</td>
<td>Conv-512</td>
</tr>
<tr>
<td>Conv-512</td>
<td>Conv-512</td>
</tr>
<tr>
<td>MaxPool</td>
<td>MaxPool</td>
</tr>
<tr>
<td>Conv-256</td>
<td>Conv-256</td>
</tr>
<tr>
<td>Conv-256</td>
<td>Conv-256</td>
</tr>
<tr>
<td>MaxPool</td>
<td>MaxPool</td>
</tr>
<tr>
<td>Conv-128</td>
<td>Conv-128</td>
</tr>
<tr>
<td>Conv-128</td>
<td>Conv-128</td>
</tr>
<tr>
<td>MaxPool</td>
<td>MaxPool</td>
</tr>
<tr>
<td>Conv-64</td>
<td>Conv-64</td>
</tr>
<tr>
<td>Conv-64</td>
<td>Conv-64</td>
</tr>
<tr>
<td>FC-4096</td>
<td>FC-4096</td>
</tr>
<tr>
<td>Conv-1000</td>
<td>Conv-1000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>very similar dataset</th>
<th>very different dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>very little data</strong></td>
<td>Use Linear Classifier on top layer</td>
</tr>
<tr>
<td><strong>quite a lot of data</strong></td>
<td>Finetune a few layers</td>
</tr>
<tr>
<td>More generic</td>
<td>More specific</td>
</tr>
<tr>
<td>--------------</td>
<td>--------------</td>
</tr>
<tr>
<td><strong>Image</strong></td>
<td><strong>Conv-64</strong></td>
</tr>
<tr>
<td><strong>Conv-64</strong></td>
<td><strong>Conv-64</strong></td>
</tr>
<tr>
<td><strong>MaxPool</strong></td>
<td><strong>MaxPool</strong></td>
</tr>
<tr>
<td><strong>Conv-128</strong></td>
<td><strong>Conv-128</strong></td>
</tr>
<tr>
<td><strong>MaxPool</strong></td>
<td><strong>MaxPool</strong></td>
</tr>
<tr>
<td><strong>Conv-256</strong></td>
<td><strong>Conv-256</strong></td>
</tr>
<tr>
<td><strong>Conv-512</strong></td>
<td><strong>Conv-512</strong></td>
</tr>
<tr>
<td><strong>MaxPool</strong></td>
<td><strong>MaxPool</strong></td>
</tr>
<tr>
<td><strong>Conv-512</strong></td>
<td><strong>Conv-512</strong></td>
</tr>
<tr>
<td><strong>MaxPool</strong></td>
<td><strong>MaxPool</strong></td>
</tr>
<tr>
<td><strong>Conv-512</strong></td>
<td><strong>Conv-512</strong></td>
</tr>
<tr>
<td><strong>MaxPool</strong></td>
<td><strong>MaxPool</strong></td>
</tr>
<tr>
<td><strong>Conv-256</strong></td>
<td><strong>Conv-256</strong></td>
</tr>
<tr>
<td><strong>Conv-128</strong></td>
<td><strong>Conv-128</strong></td>
</tr>
<tr>
<td><strong>MaxPool</strong></td>
<td><strong>MaxPool</strong></td>
</tr>
<tr>
<td><strong>Conv-64</strong></td>
<td><strong>Conv-64</strong></td>
</tr>
<tr>
<td><strong>Conv-64</strong></td>
<td><strong>Conv-64</strong></td>
</tr>
<tr>
<td><strong>FC-1000</strong></td>
<td><strong>FC-4096</strong></td>
</tr>
<tr>
<td><strong>FC-4096</strong></td>
<td><strong>FC-4096</strong></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>very similar dataset</th>
<th>very different dataset</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>very little data</strong></td>
<td>Use Linear Classifier on top layer</td>
</tr>
<tr>
<td><strong>quite a lot of data</strong></td>
<td>Finetune a few layers</td>
</tr>
<tr>
<td></td>
<td>You’re in trouble… Try linear classifier from different stages</td>
</tr>
<tr>
<td></td>
<td>Finetune a larger number of layers</td>
</tr>
</tbody>
</table>
Transfer learning with CNNs is pervasive… (it’s the norm, not an exception)

Object Detection
(Fast R-CNN)

Image Captioning: CNN + RNN

Figure copyright Ross Girshick, 2015. Reproduced with permission.

Figure copyright IEEE, 2015. Reproduced for educational purposes.
Transfer learning with CNNs is pervasive…
(it’s the norm, not an exception)

Object Detection
(Fast R-CNN)

- CNN pretrained on ImageNet

Image Captioning: CNN + RNN

Figure copyright Ross Girshick, 2015. Reproduced with permission.

Figure copyright IEEE, 2015. Reproduced for educational purposes.
Transfer learning with CNNs is pervasive…
(it’s the norm, not an exception)

Object Detection
(Fast R-CNN)

CNN pretrained on ImageNet

Image Captioning: CNN + RNN

Word vectors pretrained with word2vec

Figure copyright IEEE, 2015. Reproduced for educational purposes.

Figure copyright Ross Girshick, 2015. Reproduced with permission.
Transfer learning with CNNs is pervasive…
But recent results show it might not always be necessary!

He et al, “Rethinking ImageNet Pre-training”, arXiv 2018
Takeaway for your projects and beyond:
Have some dataset of interest but it has < ~1M images?

1. Find a very large dataset that has similar data, train a big ConvNet there
2. Transfer learn to your dataset

Deep learning frameworks provide a “Model Zoo” of pretrained models so you don’t need to train your own

TensorFlow: https://github.com/tensorflow/models
PyTorch: https://github.com/pytorch/vision
Today: CNN Architectures

Case Studies
- AlexNet
- VGG
- GoogLeNet
- ResNet

Also....
- SENet
- NiN (Network in Network)
- Wide ResNet
- ResNeXT
- DenseNet
- FractalNet
- MobileNets
- NASNet
Review: LeNet-5

[LeCun et al., 1998]

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]
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

=>

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]**

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

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

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
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.
Case Study: AlexNet

[Krizhevsky et al. 2012]

Full (simplified) AlexNet architecture:

- **INPUT**
  - [227x227x3]

- **CONV1**: 96 11x11 filters at stride 4, pad 0
  - [27x27x96]

- **MAX POOL1**: 3x3 filters at stride 2
  - [27x27x96]

- **NORM1**: Normalization layer

- **CONV2**: 256 5x5 filters at stride 1, pad 2
  - [13x13x256]

- **MAX POOL2**: 3x3 filters at stride 2
  - [13x13x256]

- **NORM2**: Normalization layer

- **CONV3**: 384 3x3 filters at stride 1, pad 1
  - [13x13x384]

- **CONV4**: 384 3x3 filters at stride 1, pad 1
  - [13x13x384]

- **CONV5**: 256 3x3 filters at stride 1, pad 1
  - [13x13x256]

- **MAX POOL3**: 3x3 filters at stride 2
  - [6x6x256]

- **FC6**: 4096 neurons
  - [4096]

- **FC7**: 4096 neurons
  - [4096]

- **FC8**: 1000 neurons (class scores)
  - [1000]

Historical note: Trained on GTX 580 GPU with only 3 GB of memory. Network spread across 2 GPUs, half the neurons (feature maps) on each GPU.

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)

CONV1, CONV2, CONV4, CONV5: Connections only with feature maps on same GPU

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)

CONV3, FC6, FC7, FC8: Connections with all feature maps in preceding layer, communication across GPUs

Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.
ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

- Lin et al
- Sanchez & Perronnin
- Krizhevsky et al (AlexNet)
- Zeiler & Fergus
- Simonyan & Zisserman (VGG)
- Szegedy et al (GoogLeNet)
- He et al (ResNet)
- Shao et al
- Hu et al (SENet)
- Russakovsky et al

Layer depths:
- shallow
- 8 layers
- 19 layers
- 22 layers
- 152 layers (3 times)
ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

- Lin et al
- Sanchez & Perronnin
- Krizhevsky et al (AlexNet)
- Zeiler & Fergus
- Simonyan & Zisserman (VGG)
- Szegedy et al (GoogLeNet)
- He et al (ResNet)
- Shao et al
- Hu et al (SENet)
- Russakovsky et al

First CNN-based winner: Krizhevsky et al (AlexNet)

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

2010: 28.2
2011: 25.8
2012: 16.4
2013: 11.7
2014: 7.3
2014: 6.7
2015: 3.6
2016: 3
2017: 2.3
Human: 5.1
ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

- Lin et al
- Sanchez & Perronnin
- Krizhevsky et al (AlexNet)
- Zeiler & Fergus
- Simonyan & Zisserman (VGG)
- Szegedy et al (GoogLeNet)
- He et al (ResNet)
- Shao et al
- Hu et al (SENet)
- Russakovsky et al

- ZFNet: Improved hyperparameters over AlexNet
- Shallow: 8 layers
- 2010: 28.2
- 2011: 25.8
- 2012: 16.4
- 2013: 11.7
- 2014: 7.3
- 2014: 6.7
- 2015: 3.6
- 2016: 3
- 2017: 2.3
- Human: 5.1
- 152 layers
- 152 layers
- 152 layers
- 19 layers
- 22 layers
- 8 layers
### ZFNet

[Zeiler and Fergus, 2013]

**Input Image**
- Image size: 224
- Filter size: 7
- Stride 2

**Layers**
- **Layer 1**: 110
  - Input: 224 x 224
  - Convolution: 7x7 with stride 2
  - Pooling: 3x3 max pool with stride 2
  - 55 x 55
- **Layer 2**: 26
  - Convolution: 3x3 with stride 2
  - Pooling: 3x3 max pool with stride 2
  - 26 x 26
- **Layer 3**: 13
  - Convolution: 1x1
  - Pooling: 3x3 max pool with stride 2
  - 13 x 13
- **Layer 4**: 13
  - Convolution: 1x1
  - Pooling: 3x3 max pool with stride 2
  - 13 x 13
- **Layer 5**: 13
  - Convolution: 1x1
  - Pooling: 3x3 max pool with stride 2
  - 13 x 13
- **Layer 6** and **Layer 7**: 4096 units each
- **Output**: Class softmax

**Details**
- **CONV1**: change from (11x11 stride 4) to (7x7 stride 2)
- **CONV3,4,5**: instead of 384, 384, 256 filters use 512, 1024, 512

**ImageNet top 5 error**: 16.4% -> 11.7%

---

Fei-Fei Li & Justin Johnson & Serena Yeung

Lecture 9 - 37 April 30, 2019
ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

- **Lin et al** (2010) - 28.2
- **Sanchez & Perronnin** (2011) - 25.8
- **Krizhevsky et al** (AlexNet) (2012) - 16.4
- **Zeiler & Fergus** (2013) - 11.7
- **Simonyan & Zisserman (VGG)** (2014) - 7.3
- **Szegedy et al (GoogLeNet)** (2014) - 6.7
- **He et al (ResNet)** (2015) - 3.6
- **Shao et al** (2016) - 3
- **Hu et al (SENet)** (2017) - 2.3
- **Russakovsky et al** (2018) - 5.1

### Networks

- **Shallow Networks**: 8 layers each
  - 2010
  - 2011
  - 2012
- **Deeper Networks**: 19 layers, 22 layers, 152 layers each
  - 2013
  - 2014
  - 2015

**Legend**:
- Shallow: 8 layers
- Deeper: 19, 22, 152 layers
Case Study: VGGNet

[Simonyan and Zisserman, 2014]

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

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: $3 \times (3^2C^2)$ vs. $7^2C^2$ for $C$ channels per layer
**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

---

VGG16
<table>
<thead>
<tr>
<th>Layer Type</th>
<th>Input Shape</th>
<th>Memory (Bytes)</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>INPUT</td>
<td>[224x224x3]</td>
<td>224<em>224</em>3 = 150K</td>
<td>0</td>
</tr>
<tr>
<td>CONV3-64:</td>
<td>[224x224x64]</td>
<td>224<em>224</em>64 = 3.2M</td>
<td>(3<em>3</em>3)*64 = 1,728</td>
</tr>
<tr>
<td>CONV3-64:</td>
<td>[224x224x64]</td>
<td>224<em>224</em>64 = 3.2M</td>
<td>(3<em>3</em>64)*64 = 36,864</td>
</tr>
<tr>
<td>POOL2:</td>
<td>[112x112x64]</td>
<td>112<em>112</em>64 = 800K</td>
<td>0</td>
</tr>
<tr>
<td>CONV3-128:</td>
<td>[112x112x128]</td>
<td>112<em>112</em>128 = 1.6M</td>
<td>(3<em>3</em>64)*128 = 73,728</td>
</tr>
<tr>
<td>CONV3-128:</td>
<td>[112x112x128]</td>
<td>112<em>112</em>128 = 1.6M</td>
<td>(3<em>3</em>128)*128 = 147,456</td>
</tr>
<tr>
<td>POOL2:</td>
<td>[56x56x128]</td>
<td>56<em>56</em>128 = 400K</td>
<td>0</td>
</tr>
<tr>
<td>CONV3-256:</td>
<td>[56x56x256]</td>
<td>56<em>56</em>256 = 800K</td>
<td>(3<em>3</em>128)*256 = 294,912</td>
</tr>
<tr>
<td>CONV3-256:</td>
<td>[56x56x256]</td>
<td>56<em>56</em>256 = 800K</td>
<td>(3<em>3</em>256)*256 = 589,824</td>
</tr>
<tr>
<td>POOL2:</td>
<td>[28x28x256]</td>
<td>28<em>28</em>256 = 200K</td>
<td>0</td>
</tr>
<tr>
<td>CONV3-512:</td>
<td>[28x28x512]</td>
<td>28<em>28</em>512 = 400K</td>
<td>(3<em>3</em>256)*512 = 1,179,648</td>
</tr>
<tr>
<td>CONV3-512:</td>
<td>[28x28x512]</td>
<td>28<em>28</em>512 = 400K</td>
<td>(3<em>3</em>512)*512 = 2,359,296</td>
</tr>
<tr>
<td>POOL2:</td>
<td>[14x14x512]</td>
<td>14<em>14</em>512 = 100K</td>
<td>0</td>
</tr>
<tr>
<td>CONV3-512:</td>
<td>[14x14x512]</td>
<td>14<em>14</em>512 = 100K</td>
<td>(3<em>3</em>512)*512 = 2,359,296</td>
</tr>
<tr>
<td>POOL2:</td>
<td>[7x7x512]</td>
<td>7<em>7</em>512 = 25K</td>
<td>0</td>
</tr>
<tr>
<td>FC:</td>
<td>[1x1x4096]</td>
<td>4096</td>
<td>7<em>7</em>512*4096 = 102,760,448</td>
</tr>
<tr>
<td>FC:</td>
<td>[1x1x4096]</td>
<td>4096</td>
<td>4096*4096 = 16,777,216</td>
</tr>
<tr>
<td>FC:</td>
<td>[1x1x1000]</td>
<td>1000</td>
<td>4096*1000 = 4,096,000</td>
</tr>
</tbody>
</table>

**TOTAL memory:** 24M * 4 bytes \(\approx\) 96MB / image (for a forward pass)

**TOTAL params:** 138M parameters

---

Fei-Fei Li & Justin Johnson & Serena Yeung  
Lecture 9 - 45  
April 30, 2019
INPUT: [224x224x3]  memory: 224*224*3=150K  params: 0

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 = 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 = 2,359,296

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
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 =~ 96MB / image (only forward! ~*2 for bwd)

TOTAL params: 138M parameters
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

- **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**

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

Deep networks have shown significant improvements in performance on the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) over the years. The use of deeper networks, such as ResNet and SENet, has led to a decrease in error rates, with the latest year showing a human-level performance.
Case Study: GoogLeNet

[Szegedy et al., 2014]

Deeper networks, with computational efficiency

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

[Szegedy et al., 2014]

“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 depth-wise
Case Study: GoogLeNet

[Szegedy et al., 2014]

Naive Inception module

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 depth-wise

Q: What is the problem with this? [Hint: Computational complexity]
Case Study: GoogLeNet

[Szegedy et al., 2014]

Example:

Q: What is the problem with this?
[Hint: Computational complexity]
Case Study: GoogLeNet

[Szegedy et al., 2014]

Example:

Q1: What is the output size of the 1x1 conv, with 128 filters?

Q: What is the problem with this? [Hint: Computational complexity]

Naive Inception module
Case Study: GoogLeNet
[Szegedy et al., 2014]

Example:

Q1: What is the output size of the 1x1 conv, with 128 filters?

Naive Inception module

Q: What is the problem with this? [Hint: Computational complexity]
Case Study: GoogLeNet

[Szegedy et al., 2014]

Example:

Q: What is the problem with this?
[Hint: Computational complexity]

Q2: What are the output sizes of all different filter operations?

Naive Inception module
Case Study: GoogLeNet
[Szegedy et al., 2014]

Example:

Q: What is the problem with this?
[Hint: Computational complexity]

28x28x256

1x1 conv, 128

3x3 conv, 192

5x5 conv, 96

3x3 pool

Naive Inception module

Q2: What are the output sizes of all different filter operations?

28x28x128

28x28x192

28x28x96

28x28x256

Module input: 28x28x256

Filter concatenation
Case Study: GoogLeNet

[Szegedy et al., 2014]

Example:

Q3: What is output size after filter concatenation?

28x28x128

1x1 conv, 128

28x28x192

3x3 conv, 192

28x28x96

5x5 conv, 96

28x28x256

3x3 pool

Q: What is the problem with this?
[Hint: Computational complexity]
Case Study: GoogLeNet

[Szegedy et al., 2014]

Example:

Q3: What is output size after filter concatenation?

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

Module input:

\(28 \times 28 \times 256\)

1x1 conv, 128

28x28x128

28x28x192

3x3 conv, 192

28x28x96

5x5 conv, 96

28x28x256

3x3 pool

Filter concatenation

Input

Naive Inception module

Q: What is the problem with this?

[Hint: Computational complexity]
Case Study: GoogLeNet
[Szegedy et al., 2014]

Q: What is the problem with this?
[Hint: Computational complexity]

Example:
Q3: 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] \(28 \times 28 \times 128 \times 1 \times 1 \times 256\)
- [3x3 conv, 192] \(28 \times 28 \times 192 \times 3 \times 3 \times 256\)
- [5x5 conv, 96] \(28 \times 28 \times 96 \times 5 \times 5 \times 256\)

Total: 854M ops
Case Study: GoogLeNet
[Szegedy et al., 2014]

Example: Naive Inception module

Q: What is the problem with this? [Hint: Computational complexity]

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:

Q: 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 depth
Reminder: 1x1 convolutions

1x1 CONV with 32 filters

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

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]

Naive Inception module

Inception module with dimension reduction
Case Study: GoogLeNet

[Szegedy et al., 2014]

Inception module with dimension reduction

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
Case Study: GoogLeNet

[Szegedy et al., 2014]

Full GoogLeNet architecture

Stem Network:
Conv-Pool-
2x Conv-Pool
Case Study: GoogLeNet

[Szegedy et al., 2014]
Case Study: GoogLeNet

[Szegedy et al., 2014]

Full GoogLeNet architecture

Classifier output
Case Study: GoogLeNet

[Szegedy et al., 2014]

Full GoogLeNet architecture

Note: after the last convolutional layer, a global average pooling layer is used that spatially averages across each feature map, before final FC layer. No longer multiple expensive FC layers!

Classifier output
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]

Full GoogLeNet architecture

22 total layers with weights
(parallel layers count as 1 layer => 2 layers per Inception module. Don’t count auxiliary output layers)

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
- ILSVRC’14 classification winner (6.7% top 5 error)
ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

- Lin et al
- Sanchez & Perronnin
- Krizhevsky et al (AlexNet)
- Zeiler & Fergus
- Simonyan & Zisserman (VGG)
- Szegedy et al (GoogLeNet)
- He et al (ResNet)
- Shao et al
- Hu et al (SENet)
- Russakovsky et al

“Revolution of Depth”

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

Shallow models (8 layers):
- 2010: Lin et al
- 2011: Sanchez & Perronnin
- 2012: Krizhevsky et al
- 2013: Zeiler & Fergus

Deep models (19 layers, 22 layers, 152 layers):
- 2014: Simonyan & Zisserman
- 2014: Szegedy et al
- 2015: He et al
- 2016: Shao et al
- 2017: Hu et al
- Human: Russakovsky et al

Legend:
- Shallow models (8 layers)
- Deep models (19 layers, 22 layers, 152 layers)
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?

Q: What’s strange about these training and test curves? [Hint: look at the order of the curves]
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]

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

[He et al., 2015]

Hypothesis: the problem is an *optimization* problem, deeper models are harder to optimize

The deeper model should be able to perform at least as well as the shallower model.

A solution by construction is copying the learned layers from the shallower model and setting additional layers to identity mapping.
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

“Plain” layers

Residual block

\[ H(x) \]

\[ F(x) \]

\[ F(x) + x \]
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

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

Use layers to fit residual \( F(x) = H(x) - x \) instead of \( H(x) \) directly
Case Study: ResNet

[He et al., 2015]

Full ResNet architecture:
- Stack residual blocks
- Every residual block has two 3x3 conv layers

Full ResNet architecture:
- Stack residual blocks
- Every residual block has two 3x3 conv layers

\[
F(x) + x
\]

\[
F(x)
\]

\[
3x3 \text{ conv}
\]

\[
X
\]

Residual block

 relu

 relu

 softmax

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

 pooling

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 (/2 in each dimension)
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 (/2 in each dimension)
- Additional conv layer at the beginning
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 (/2 in each dimension)
- Additional conv layer at the beginning
- No FC layers at the end (only FC 1000 to output classes)
Case Study: ResNet
[He et al., 2015]

Total depths of 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]

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

1x1 conv, 64 filters to project to 28x28x64

3x3 conv operates over only 64 feature maps

1x1 conv, 256 filters projects back to 256 feature maps (28x28x256)

28x28x256 output
Case Study: ResNet
[He et al., 2015]

Training ResNet in practice:

- Batch Normalization after every CONV layer
- Xavier $2/3$ 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 $1 \times 10^{-5}$
- 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 lowing 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
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 lowing training error as expected
- Swept 1st place in all ILSVRC and COCO 2015 competitions

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


Comparing complexity...


Comparing complexity...


Comparing complexity...


Comparing complexity...

AlexNet:
Smaller compute, still memory heavy, lower accuracy


Comparing complexity...

ResNet:
Moderate efficiency depending on model, highest accuracy


Forward pass time and power consumption


<table>
<thead>
<tr>
<th>Year</th>
<th>Winner</th>
<th>Network</th>
<th>Layers</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010</td>
<td>Lin et al</td>
<td>Shallow</td>
<td>8 layers</td>
</tr>
<tr>
<td>2011</td>
<td>Sanchez &amp; Perronnin</td>
<td>8 layers</td>
<td>8 layers</td>
</tr>
<tr>
<td>2012</td>
<td>Krizhevsky et al (AlexNet)</td>
<td>19 layers</td>
<td>19 layers</td>
</tr>
<tr>
<td>2013</td>
<td>Zeiler &amp; Fergus</td>
<td>152 layers</td>
<td>152 layers</td>
</tr>
<tr>
<td>2014</td>
<td>Simonyan &amp; Zisserman (VGG)</td>
<td>152 layers</td>
<td>152 layers</td>
</tr>
<tr>
<td>2014</td>
<td>Szegedy et al (GoogLeNet)</td>
<td>152 layers</td>
<td>152 layers</td>
</tr>
<tr>
<td>2015</td>
<td>He et al (ResNet)</td>
<td>152 layers</td>
<td>152 layers</td>
</tr>
<tr>
<td>2016</td>
<td>Shao et al</td>
<td>Network ensembling</td>
<td>152 layers</td>
</tr>
<tr>
<td>2017</td>
<td>Hu et al (SENet)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2018</td>
<td>Russakovsky et al</td>
<td>Human</td>
<td>5.1</td>
</tr>
</tbody>
</table>
Improving ResNets...

“Good Practices for Deep Feature Fusion”

[Shao et al. 2016]

- Multi-scale ensembling of Inception, Inception-Resnet, Resnet, Wide Resnet models
- ILSVRC’16 classification winner

<table>
<thead>
<tr>
<th></th>
<th>Inception-v3</th>
<th>Inception-v4</th>
<th>Inception-Resnet-v2</th>
<th>Resnet-200</th>
<th>Wrn-68-3</th>
<th>Fusion (Val.)</th>
<th>Fusion (Test)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Err. (%)</td>
<td>4.20</td>
<td>4.01</td>
<td>3.52</td>
<td>4.26</td>
<td>4.65</td>
<td>2.92 (-0.6)</td>
<td>2.99</td>
</tr>
</tbody>
</table>
ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

- Lin et al
- Sanchez & Perronnin
- Krizhevsky et al (AlexNet)
- Zeiler & Fergus
- Simonyan & Zisserman (VGG)
- Szegedy et al (GoogLeNet)
- He et al (ResNet)
- Russakovsky et al
- Shao et al
- Hu et al (SENet)

Adaptive feature map reweighting

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

- 2010: Lin et al, shallow
- 2011: Sanchez & Perronnin, 8 layers
- 2012: Krizhevsky et al (AlexNet), 8 layers
- 2013: Zeiler & Fergus, 19 layers
- 2014: Simonyan & Zisserman (VGG), 22 layers
- 2014: Szegedy et al (GoogLeNet), 19 layers
- 2015: He et al (ResNet), 22 layers
- 2016: Shao et al, 152 layers
- 2017: Hu et al (SENet), 152 layers
- Human, 152 layers
Squeeze-and-Excitation Networks (SENet) [Hu et al. 2017]

- Add a “feature recalibration” module that learns to adaptively reweight feature maps
- Global information (global avg. pooling layer) + 2 FC layers used to determine feature map weights
- ILSVRC’17 classification winner (using ResNeXt-152 as a base architecture)
ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

- Lin et al
- Sanchez & Perronnin
- Krizhevsky et al (AlexNet)
- Zeiler & Fergus
- Simonyan & Zisserman (VGG)
- Szegedy et al (GoogLeNet)
- He et al (ResNet)
- Hu et al (SENet)
- Russakovsky et al

Bar chart showing the years and corresponding accuracy:
- 2010: 28.2%
- 2011: 25.8%
- 2012: 16.4%
- 2013: 11.7%
- 2014: 7.3%
- 2014: 6.7%
- 2015: 3.6%
- 2016: 3%
- 2017: 2.3%
- Human: 5.1%

Layers:
- Shallow: 8 layers
- 8 layers
- 8 layers
- 19 layers
- 22 layers
- 152 layers
- 152 layers
- 152 layers
ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

- **Lin et al**
- **Sanchez & Perronnin**
- **Krizhevsky et al (AlexNet)**
- **Zeiler & Fergus**
- **Simonyan & Zisserman (VGG)**
- **Szegedy et al (GoogLeNet)**
- **He et al (ResNet)**
- **Russakovsky et al**
- **Hu et al (SENet)**
- **Shao et al**
- **Human**

**Completion of the challenge:**
Annual ImageNet competition no longer held after 2017 -> now moved to Kaggle.
But research into CNN architectures is still flourishing
Network in Network (NiN)

[Lin et al. 2014]

- Mlpconv layer with “micronetwork” within each conv layer to compute more abstract features for local patches
- Micronetwork uses multilayer perceptron
- Precursor to GoogLeNet and ResNet “bottleneck” layers
- Philosophical inspiration for GoogLeNet

Figures copyright Lin et al., 2014. Reproduced with permission.
Improving ResNets...

Identity Mappings in Deep Residual Networks

[He et al. 2016]

- Improved ResNet block design from creators of ResNet
- Creates a more direct path for propagating information throughout network (moves activation to residual mapping pathway)
- Gives better performance
Improving ResNets...

Wide Residual Networks

[Zagoruyko et al. 2016]

- Argues that residuals are the important factor, not depth
- User wider residual blocks (F x k filters instead of F filters in each layer)
- 50-layer wide ResNet outperforms 152-layer original ResNet
- Increasing width instead of depth more computationally efficient (parallelizable)
Improving ResNets...

Aggregated Residual Transformations for Deep Neural Networks (ResNeXt)

[Xie et al. 2016]

- Also from creators of ResNet
- Increases width of residual block through multiple parallel pathways ("cardinality")
- Parallel pathways similar in spirit to Inception module
Other ideas...

FractalNet: Ultra-Deep Neural Networks without Residuals

[Larsson et al. 2017]

- Argues that key is transitioning effectively from shallow to deep and residual representations are not necessary
- Fractal architecture with both shallow and deep paths to output
- Trained with dropping out sub-paths
- Full network at test time
Other ideas...

Densely Connected Convolutional Networks

[Huang et al. 2017]

- Dense blocks where each layer is connected to every other layer in feedforward fashion
- Alleviates vanishing gradient, strengthens feature propagation, encourages feature reuse
Efficient networks...

MobileNets: Efficient Convolutional Neural Networks for Mobile Applications

[Howard et al. 2017]

- Depthwise separable convolutions replace standard convolutions by factorizing them into a depthwise convolution and a 1x1 convolution that is much more efficient
- Much more efficient, with little loss in accuracy
- Follow-up MobileNetV2 work in 2018 (Sandler et al.)
- Other works in this space e.g. ShuffleNet (Zhang et al. 2017)
Meta-learning: Learning to learn network architectures...

Neural Architecture Search with Reinforcement Learning (NAS)

[Zoph et al. 2016]

- “Controller” network that learns to design a good network architecture (output a string corresponding to network design)
- Iterate:
  1) Sample an architecture from search space
  2) Train the architecture to get a “reward” R corresponding to accuracy
  3) Compute gradient of sample probability, and scale by R to perform controller parameter update (i.e. increase likelihood of good architecture being sampled, decrease likelihood of bad architecture)
Meta-learning: Learning to learn network architectures...

Learning Transferable Architectures for Scalable Image Recognition

[Zoph et al. 2017]

- Applying neural architecture search (NAS) to a large dataset like ImageNet is expensive
- Design a search space of building blocks (“cells”) that can be flexibly stacked
- NASNet: Use NAS to find best cell structure on smaller CIFAR-10 dataset, then transfer architecture to ImageNet
- Many follow-up works in this space e.g. AmoebaNet (Real et al. 2019) and ENAS (Pham, Guan et al. 2018)
Summary: CNN Architectures

Case Studies
- AlexNet
- VGG
- GoogLeNet
- ResNet

Also....
- SENet
- NiN (Network in Network)
- Wide ResNet
- ResNeXT
- DenseNet
- FractalNet
- MobileNets
- NASNet
Summary: CNN Architectures

- Many popular architectures available in model zoos
- ResNet and SENet currently good defaults to use
- Networks have gotten increasingly deep over time
- Many other aspects of network architectures are also continuously being investigated and improved
- Even more recent trend towards meta-learning

- Next time: Recurrent neural networks