Various topics

Datasets
CNN Architectures
CapsNets
Sequence models

Last time..

CNNs

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)

5x5x3 filter

32x32x3 image

activation map

activation maps

$$W_{out} = \frac{W_{in} - F + 2P}{S} + 1$$

$$H_{out} = \frac{H_{in} - F + 2P}{S} + 1$$

$$D_{out} = K$$

5x5 x 8 weights

N input channels

M output channels

1	1	2	4
5	**6**	7	**8**
3	2	1	0
1	2	3	**4**

2x2 max pooling

6	8
3	4

Agenda

- **Datasets**, challenges and metrics for Image classifications
  - ImageNet, ILSVRC
- **CNN Architectures / Base models** (AlexNet, VGG, Inception, ResNet, etc.)
- CapsuleNets
- RNN / LSTM (sequence models)

Datasets, Challenges & Metrics

Datasets

MNIST:
- Training: 60k
- Test: 10k
- Classes: 10
- Size: 28x28

Fashion-MNIST:
- Training: 60k
- Test: 10k
- Classes: 10
- Size: 28x28

CIFAR-10:
- Training: 50k
- Test: 10k
- Classes: 10
- Size: 32x32 (x3, color)

CIFAR-100:
- Training: 50k
- Test: 10k
- Classes: 100
- Size: 32x32 (x3, color)

Datasets: ImageNet

- 14 millions+ labeled images, 22,000 categories
- Organized according to the **WordNet** hierarchy:
  - meaningful concepts (possibly multiple words or word phrases) = "synonym set" or «synset"
  - 100,000 synsets, majority of them are nouns (80,000+)
- Goal: on average 1000 images to illustrate each synset

Challenge (competition): ILSVRC

ILSVRC: ImageNet Large Scale Visual Recognition Challenge (subset of ImageNet)

- ILSVRC uses a subset of ImageNet
- 1000 categories / clases
- Roughly 1.2 million training images, 50,000 validation images and 100,000 testing images.
- Around 1000 images in each of 1000 categories.

ImageNet Challenge

IMAGENET

- 1,000 object classes (categories).
- Images:
  - 1.2 M train
  - 100k test.

Metric: Top-1 / Top-5 accuracy?

ILSVRC: Image Classification: Metrics

- **Prediction**: distribution, sum=1
- **Top-1** score / accuracy:
  - **Correct**: target label = top predicted label (the one having the highest probability)
- **Top-5** score / accuracy:
  - **Correct**: target label = among top 5 predictions (the 5 ones with the highest probabilities)
- **Score** # times the target label matched the predicted label, divided by the number of data-points evaluated

ILSVRC: Image Classification: Metrics (2)

- **Example 1**
  - Picture of «**dog**»
  - Prediction:
    - «cat»: 0.1
    - «dog»: 0.2
    - «house»: 0.0
    - «mouse»: 0.7
  - Top-1: ? **Wrong**
  - Top-3: ? **Correct**

- **Example 2**
  - Picture of **cat**
  - Prediction:
    - Tiger: 0.4
    - Dog: 0.3
    - Cat: 0.1
    - Lynx: 0.09
    - Lion: 0.08
    - Bird: 0.02
    - Bear: 0.01
  - Top-1: ? **Wrong**
  - Top-5: ? **Correct**

CNN Architectures

Stanford: CS231n: CNN Archtectures

Today: CNN Architectures

Case Studies

- AlexNet
- VGG
- GoogLeNet
- ResNet

Also....

- NiN (Network in Network)
- Wide ResNet
- ResNeXT
- Stochastic Depth
- Squeeze-and-Excitation Network
- DenseNet
- FractalNet
- SqueezeNet
- NASNet

- Today: discuss some of the **CNN architectures** used
- Go into **depth** of some (**commonly** used, **winners** of image net competition)
- **Briefly** mention some of the other models (interesting either from a **historical** perspective or as recent areas of **research**)

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]

- Quick review: one of the first ConvNets successfully used in practice
- Character recognition

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

- AlexNet: first CNN-based winner of ILSVRC, cut the error rate by a large margin, widely used as a base architecture for many years, still used for transfer learning for example.

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.

- AlexNet: first large scale CNN, won the ImageNet classification challenge in 2012, outperformed all previous non-DL-based models by large margin, in many ways AlexNet started widespread research and usage of CNNs
- Basic Architecture: C-M-N-1_C-M-N-2_C-C-C-M_FC-FC-FC, similar to LeNet but more layers in total

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

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

- Padding used?, HxWxD?

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**

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

- Parameteres vs. Weights and Biases

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

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

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!

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.

- Good exercise to go through this and figure out the output sizes and number of parameters involved at every layer

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.

- If you do that, you get something like this
- Why 1000 neurons in last FC layer?

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.

- Some more detail..

Case Study: AlexNet

[Krizhevsky et al. 2012]

[55x55x48] x 2

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)

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.

- Split into two rows, only 3GB, couldn't fit entire network, two GPUs, half the future maps on each, 48 maps on each GPU for given layer

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.

- Input 227 (vs. 224) needed to get the numbers right

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

- 2013 ImageNet challenge was won by ZFNet (from the authors)

ZFNet

[Zeiler and Fergus, 2013]

AlexNet but:
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%

- For the most part: better accuracy by improving hyper parameteres
- Same general structure, same # layers, but made a few changes to the conv. layers.

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

- 2014, more different, e.g. depth, jump in performance, close race, GoogleNet was the winner, VGGNet close behind (but won the localization challenge)

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

- Idea: deeper networks with smaller filters
- From 8 to 16/19 layers, 3x3 conv. with periodic pooling
- Simple, elegant network architecture, able to get 7.3% top five error

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: 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

(not counting biases)

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

- Same calculations as for AlexNet

(not counting biases)

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

TOTAL memory: 24M * 4 bytes ~= 96MB / image (for a forward pass)
TOTAL params: 138M parameters

- Forward pass: 100MB/image (5GB of total memory, store 50)
- 138 milloin parameters (60 million for AlexNet), Depth: # layers (vs. # channels)

(not counting biases)

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

Note:

Most memory is in early CONV

Most params are in late FC

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

- Most memory in early Conv
- Most parameters in late FC

(not counting biases)

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

- CONV3-64: 3by3 convs with 64 filters
- conv1-1, conv1-2: Groups of filters

VGG16

Common names

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

- Some more details

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

- Winning net from 2014 was GoogLeNet (also known as Inception-v1)

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)

Inception module

- Deeper too, but also looked at the problem of computational efficiency.
- Using an **inception module**, stacking a lot of these modules on top of each other
- No FC layers, in total only 5 mill parameters (12x less than AlexNet), and got 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

Inception module

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

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]

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

Example:

Naive Inception module

Case Study: GoogLeNet

[Szegedy et al., 2014]

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

Example:

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

Naive Inception module

Case Study: GoogLeNet

[Szegedy et al., 2014]

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

Example:

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

Naive Inception module

Case Study: GoogLeNet

[Szegedy et al., 2014]

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

Example:

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

Naive Inception module

Case Study: GoogLeNet

[Szegedy et al., 2014]

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

Example:

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

Naive Inception module

- Zero padding needed

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?

Naive Inception module

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?

28x28x(128+192+96+256) = 28x28x672

Naive Inception module

- Kept the same spatial dimensions and blew up the depth

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?

28x28x(128+192+96+256) = 28x28x672

Naive Inception module

Conv Ops:

[1x1 conv, 128] 28x28x128x1x1x256

[3x3 conv, 192] 28x28x192x3x3x256

[5x5 conv, 96] 28x28x96x5x5x256

Total: 854M ops

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?

28x28x(128+192+96+256) = 28x28x672

Naive Inception module

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!

- Input dept: 256, output depth: 672, depth will just keep increasing

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?

28x28x(128+192+96+256) = **529k**

Naive Inception module

Solution: “bottleneck” layers that use 1x1 convolutions to reduce feature depth

Reminder: 1x1 convolutions

Reminder: 1x1 convolutions

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

1x1 conv “bottleneck” layers

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

Inception module

Case Study: GoogLeNet

[Szegedy et al., 2014]

Full GoogLeNet architecture

Stem Network: Conv-Pool-2x Conv-Pool

Case Study: GoogLeNet

[Szegedy et al., 2014]

Full GoogLeNet architecture

Stacked Inception Modules

Case Study: GoogLeNet

[Szegedy et al., 2014]

Full GoogLeNet architecture

Classifier output

Case Study: GoogLeNet

[Szegedy et al., 2014]

Full GoogLeNet architecture

Classifier output
(removed expensive FC layers!)

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
- No FC layers
- 12x less params than AlexNet
- ILSVRC’14 classification winner (6.7% top 5 error)

Inception module

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

ResNet

- **Problems** of Plain Network (no skip / shortcut connection)
  - vanishing/exploding gradients
    - **backpropagation**: multiplying n of these small (= gradient zero/vanished) / large (= gradient large/exploded) numbers to compute gradients of the “front” layers in an n-layer network
  - expect **deeper** network will have more **accurate** prediction, experiments show something else (more a optimization than overfitting problem)

ResNet (2)

- **Solution**: add skip / shortcut connections
  - Output: H(x) = F(x) + x
  - Weight layers learn a kind of residual mapping: F(x) = H(x) - x
  - Even if there is vanishing gradient for the weight layers, we always still have the **identity x** to transfer back to earlier layers.

ResNet (3)

- Architecture (figure):
  - 34-layer residual network (ResNet, top)
  - 34-layer plain network (middle)
  - The VGG-19 (bottom)
- 3 types of skip / shortcut connections when the input dimensions are smaller than the output dimensions.

ResNet (4)

- Bottleneck Design
  - to reduce the complexity
  - 1x1 conv layers are added to the start and end of network (NiN, GoogLeNet)
  - 1×1 conv can reduce the number of connections (parameters) while not degrading the performance of the network so much.
  - With the bottleneck design, 34-layer ResNet become 50-layer ResNet (there are deeper network with the bottleneck design: ResNet-101 and ResNet-152)

The Basic Block (Left) and The Proposed Bottleneck Design (Right)

ResNet (5)

- When plain network is used, 18-layer is better than 34-layer, due to the vanishing gradient problem
- When ResNet is used, 34-layer is better than 18-layer, vanishing gradient problem has been solved by skip connections.
- If we compare 18-layer plain network and 18-layer ResNet, there is no much difference. This is because vanishing gradient problem does not appear for shallow network.

Validation Error:
18-Layer and 34-Layer Plain Network (Left), 18-Layer and 34-Layer ResNet (right)

	plain	ResNet
18 layers	27.94	27.88
34 layers	28.54	**25.03**

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

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

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

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)

No FC layers besides FC 1000 to output classes

Global average pooling layer after last conv layer

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)

Case Study: ResNet

[He et al., 2015]

Training ResNet in practice:

- Batch Normalization after every CONV layer
- Xavier 2/ 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 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

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)

Additional readings: **Inception** versions

- **Review: Inception v1 (GoogLeNet)**
- Review: Inception-v2 / BN-Inception (Batch Normalization)
- Review: Inception-v3 (Factorization)
- Review: Inception-v4

Not in bold = orientation

Additional readings: **ResNet** versions

- **Review: ResNet**
- Review: Pre-Activation ResNet with Identity Mapping

Not in bold = orientation

Comparing complexity...

An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Figures copyright Alfredo Canziani, Adam Paszke, Eugenio Culurciello, 2017. Reproduced with permission.

Comparing complexity...

Inception-v4: Resnet + Inception!

An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Figures copyright Alfredo Canziani, Adam Paszke, Eugenio Culurciello, 2017. Reproduced with permission.

Comparing complexity...

VGG: Highest memory, most operations

An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Figures copyright Alfredo Canziani, Adam Paszke, Eugenio Culurciello, 2017. Reproduced with permission.

Comparing complexity...

An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Figures copyright Alfredo Canziani, Adam Paszke, Eugenio Culurciello, 2017. Reproduced with permission.

Comparing complexity...

AlexNet:
Smaller compute, still memory heavy, lower accuracy

An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Figures copyright Alfredo Canziani, Adam Paszke, Eugenio Culurciello, 2017. Reproduced with permission.

Comparing complexity...

ResNet:
Moderate efficiency depending on model, highest accuracy

An Analysis of Deep Neural Network Models for Practical Applications, 2017.

Figures copyright Alfredo Canziani, Adam Paszke, Eugenio Culurciello, 2017. Reproduced with permission.

Additional Architectures

More for orientation (not part of syllabus)

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 (FC, i.e. 1x1 conv layers)
- Precursor to GoogLeNet and ResNet "bottleneck" layers
- Philosophical inspiration for GoogLeNet

(a) Linear convolution layer

(b) Mlpconv layer

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)

Basic residual block

Wide residual block

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

Improving ResNets...

Deep Networks with Stochastic Depth

[Huang et al. 2016]

- Motivation: reduce vanishing gradients and training time through short networks during training
- Randomly drop a subset of layers during each training pass
- Bypass with identity function
- Use full deep network at test time

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

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

	Inception-v3	Inception-v4	Inception-Resnet-v2	Resnet-200	Wrn-68-3	Fusion（Val.）	Fusion（Test）
Err. (%)	4.20	4.01	3.52	4.26	4.65	2.92 (-0.6)	2.99

ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

Improving ResNets...

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)

Beyond ResNets...

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

Figures copyright Larsson et al., 2017. Reproduced with permission.

Beyond ResNets...

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

Dense Block

Efficient networks...

SqueezeNet: AlexNet-level Accuracy With 50x Fewer Parameters and <0.5Mb Model Size

[Iandola et al. 2017]

- Fire modules consisting of a ‘squeeze’ layer with 1x1 filters feeding an ‘expand’ layer with 1x1 and 3x3 filters
- AlexNet level accuracy on ImageNet with 50x fewer parameters
- Can compress to 510x smaller than AlexNet (0.5Mb)

Figure copyright Iandola, Han, Moskewicz, Ashraf, Dally, Keutzer, 2017. Reproduced with permission.

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

Normal Cell

Reduction Cell

Summary: CNN Architectures

Case Studies

- AlexNet
- VGG
- GoogLeNet
- ResNet

Also....

- NiN (Network in Network)
- Wide ResNet
- ResNeXT
- Stochastic Depth
- Squeeze-and-Excitation Network
- DenseNet
- FractalNet
- SqueezeNet
- NASNet

Summary: CNN Architectures

- VGG, GoogLeNet, ResNet all in wide use, available in model zoos
- ResNet current best default, also consider SENet when available
- Trend towards extremely deep networks
- Significant research centers around design of layer / skip connections and improving gradient flow
- Efforts to investigate necessity of depth vs. width and residual connections
- Even more recent trend towards meta-learning

Next time: CapsNets, RNNs