Vision Architectures
Since AlexNet
ImageNet Classification Challenge

- **2010**: Lin et al
- **2011**: Sanchez & Perronnin
- **2012**: Krizhevsky et al (AlexNet)
- **2013**: Zeiler & Fergus

<table>
<thead>
<tr>
<th>Year</th>
<th>Error Rate</th>
<th>Model Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010</td>
<td>28.2</td>
<td>Shallow, 8 layers</td>
</tr>
<tr>
<td>2011</td>
<td>25.8</td>
<td>Shallow, 8 layers</td>
</tr>
<tr>
<td>2012</td>
<td>16.4</td>
<td>19 layers</td>
</tr>
<tr>
<td>2013</td>
<td>11.7</td>
<td>22 layers</td>
</tr>
</tbody>
</table>

ImageNet Classification Challenge

Year | Error Rate | Network
--- | --- | ---
2010 | 28.2 | Lin et al
2011 | 25.8 | Sanchez & Perronnin
2012 | 16.4 | Krizhevsky et al (AlexNet)
2013 | 11.7 | Zeiler & Fergus
2014 | 7.3 | Simonyan & Zisserman (VGG)

Shallow:
- 8 layers (2010, 2011)
- 8 layers (2012)
- 19 layers (2013)

Deep:
VGG: Deeper Networks, Regular Design

VGG Design rules:
All conv are 3x3 stride 1 pad 1
All max pool are 2x2 stride 2
After pool, double #channels


Slide from Justin Johnson
VGG: Deeper Networks, Regular Design

VGG Design rules:
All conv are 3x3 stride 1 pad 1
All max pool are 2x2 stride 2
After pool, double #channels

Network has 5 convolutional stages:
Stage 1: conv-conv-pool
Stage 2: conv-conv-pool
Stage 3: conv-conv-pool
Stage 4: conv-conv-conv-[conv]-pool
Stage 5: conv-conv-conv-[conv]-pool

(VGG-19 has 4 conv in stages 4 and 5)


Slide from Justin Johnson
VGG: Deeper Networks, Regular Design

VGG Design rules:
All conv are 3x3 stride 1 pad 1
All max pool are 2x2 stride 2
After pool, double #channels

Option 1:
Conv(5x5, C -> C)

Params: 25C²
FLOPs: 25C²HW


Slide from Justin Johnson
VGG: Deeper Networks, Regular Design

VGG Design rules:
All conv are 3x3 stride 1 pad 1
All max pool are 2x2 stride 2
After pool, double #channels

Option 1:
Conv(5x5, C -> C)
Conv(3x3, C -> C)

Params: 25C^2
FLOPs: 25C^2HW

Option 2:
Conv(3x3, C -> C)
Conv(3x3, C -> C)

Params: 18C^2
FLOPs: 18C^2HW


Slide from Justin Johnson
VGG: Deeper Networks, Regular Design

VGG Design rules:
All conv are 3x3 stride 1 pad 1
All max pool are 2x2 stride 2
After pool, double #channels

Two 3x3 conv has same receptive field as a single 5x5 conv, but has fewer parameters and takes less computation!

Option 1:
Conv(5x5, C -> C)
Conv(3x3, C -> C)
Params: 25C^2
FLOPs: 25C^2HW

Option 2:
Conv(3x3, C -> C)
Conv(3x3, C -> C)
Params: 18C^2
FLOPs: 18C^2HW


Slide from Justin Johnson
VGG: Deeper Networks, Regular Design

VGG Design rules:
All conv are 3x3 stride 1 pad 1
All max pool are 2x2 stride 2
After pool, double #channels

Input: C x 2H x 2W
Layer: Conv(3x3, C\rightarrow C)

Memory: 4HWC
Params: 9C^2
FLOPs: 36HWC^2


Slide from Justin Johnson
VGG: Deeper Networks, Regular Design

VGG Design rules:
All conv are 3x3 stride 1 pad 1
All max pool are 2x2 stride 2
After max pool, double #channels

Input: C x 2H x 2W
Layer: Conv(3x3, C->C)
Memory: 4HWC
Params: 9C^2
FLOPs: 36HWC^2

Input: 2C x H x W
Layer: Conv(3x3, 2C -> 2C)
Memory: 2HWC
Params: 36C^2
FLOPs: 36HWC^2


Slide from Justin Johnson
### VGG: Deeper Networks, Regular Design

**VGG Design rules:**
- All conv are 3x3 stride 1 pad 1
- All max pool are 2x2 stride 2
- After pool, double #channels

<table>
<thead>
<tr>
<th>Input: C x 2H x 2W</th>
<th>Layer: Conv(3x3, C-&gt;C)</th>
<th>Memory: 4HWC</th>
<th>Params: 9C²</th>
<th>FLOPs: 36HWC²</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input: 2C x H x W</td>
<td>Conv(3x3, 2C -&gt; 2C)</td>
<td>Memory: 2HWC</td>
<td>Params: 36C²</td>
<td>FLOPs: 36HWC²</td>
</tr>
</tbody>
</table>

Conv layers at each spatial resolution take the same amount of computation!


Slide from Justin Johnson
AlexNet vs VGG-16: Much bigger network!

AlexNet vs VGG-16 (Memory, KB)

AlexNet total: 1.9 MB
VGG-16 total: 48.6 MB (25x)

AlexNet vs VGG-16 (Params, M)

AlexNet total: 61M
VGG-16 total: 138M (2.3x)

AlexNet vs VGG-16 (MFLOPs)

AlexNet total: 0.7 GFLOP
VGG-16 total: 13.6 GFLOP (19.4x)

Slide from Justin Johnson
ImageNet Classification Challenge

<table>
<thead>
<tr>
<th>Year</th>
<th>Error Rate</th>
<th>Authors/Models</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010</td>
<td>28.2</td>
<td>Lin et al</td>
</tr>
<tr>
<td>2011</td>
<td>25.8</td>
<td>Sanchez &amp; Perronnin</td>
</tr>
<tr>
<td>2012</td>
<td>16.4</td>
<td>Krizhevsky et al (AlexNet)</td>
</tr>
<tr>
<td>2013</td>
<td>11.7</td>
<td>Zeiler &amp; Fergus</td>
</tr>
<tr>
<td>2014</td>
<td>7.3</td>
<td>Simonyan &amp; Zisserman (VGG)</td>
</tr>
</tbody>
</table>

Shallow: 8 layers, 19 layers

Bar graph showing error rates from 2010 to 2014, with models listed for each year. The graph highlights the improvement in error rate over the years.
ImageNet Classification Challenge

Error Rate

28.2
25.8
16.4
11.7
7.3
6.7
3.6
3
2.3
5.1
0
5
10
15
20
25
30

2010
2011
2012
2013
2014
2014

Lin et al
Sanchez & Perronnin
Krizhevsky et al (AlexNet)
Zeiler & Fergus
Simonyan & Zisserman (VGG)
Szegedy et al (GoogLeNet)

Shallow
8 layers
8 layers
19 layers
22 layers
GoogLeNet: Focus on Efficiency

Many innovations for efficiency: reduce parameter count, memory usage, and computation

Szegedy et al, “Going deeper with convolutions”, CVPR 2015

Slide from Justin Johnson
GoogLeNet: Aggressive Stem

**Stem network** at the start aggressively downsamples input (Recall in VGG-16: Most of the compute was at the start)

Szegedy et al, “Going deeper with convolutions”, CVPR 2015

Slide from Justin Johnson
**GoogLeNet: Aggressive Stem**

_Sem network_ at the start aggressively downsamples input
(Recall in VGG-16: Most of the compute was at the start)

<table>
<thead>
<tr>
<th>Layer</th>
<th>Input size</th>
<th>Layer</th>
<th>Output size</th>
<th>Memory (KB)</th>
<th>Params (K)</th>
<th>Flop (M)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C</td>
<td>H / W</td>
<td>filters</td>
<td>C</td>
<td>H / W</td>
<td></td>
</tr>
<tr>
<td>conv</td>
<td>3</td>
<td>224</td>
<td>64</td>
<td>64</td>
<td>112</td>
<td>3136</td>
</tr>
<tr>
<td>max-pool</td>
<td>64</td>
<td>112</td>
<td>3</td>
<td>64</td>
<td>56</td>
<td>784</td>
</tr>
<tr>
<td>conv</td>
<td>64</td>
<td>56</td>
<td>64</td>
<td>64</td>
<td>56</td>
<td>784</td>
</tr>
<tr>
<td>conv</td>
<td>64</td>
<td>56</td>
<td>192</td>
<td>192</td>
<td>56</td>
<td>2352</td>
</tr>
<tr>
<td>max-pool</td>
<td>192</td>
<td>56</td>
<td>3</td>
<td>192</td>
<td>28</td>
<td>588</td>
</tr>
</tbody>
</table>

Total from 224 to 28 spatial resolution:
Memory: 7.5 MB
Params: 124K
MFLOP: 418

Szegedy et al, “Going deeper with convolutions”, CVPR 2015

Slide from Justin Johnson
GoogleNet: Aggressive Stem

**Stem network** at the start aggressively downsamples input
(Recall in VGG-16: Most of the compute was at the start)

<table>
<thead>
<tr>
<th>Layer</th>
<th>Input size</th>
<th>Layer</th>
<th>Output size</th>
<th>C</th>
<th>H</th>
<th>W</th>
<th>filters</th>
<th>kernel</th>
<th>stride</th>
<th>pad</th>
<th>C</th>
<th>H</th>
<th>W</th>
<th>filters</th>
<th>memory (KB)</th>
<th>params (K)</th>
<th>flop (M)</th>
</tr>
</thead>
<tbody>
<tr>
<td>conv</td>
<td>3 224</td>
<td>64</td>
<td>7 2 3</td>
<td>64</td>
<td>112</td>
<td></td>
<td>3136</td>
<td>0</td>
<td>118</td>
<td></td>
<td>9</td>
<td>118</td>
<td></td>
<td>0</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>max-pool</td>
<td>64 112</td>
<td>64</td>
<td>3 2 1</td>
<td>64</td>
<td>56</td>
<td></td>
<td>784</td>
<td>0</td>
<td>2</td>
<td></td>
<td>0</td>
<td>2</td>
<td></td>
<td>0</td>
<td>2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>conv</td>
<td>64 56</td>
<td>64</td>
<td>1 1 0</td>
<td>64</td>
<td>56</td>
<td></td>
<td>784</td>
<td>4</td>
<td>13</td>
<td></td>
<td>4</td>
<td>13</td>
<td></td>
<td>0</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>conv</td>
<td>64 56 192</td>
<td>3 1 1</td>
<td>192 56</td>
<td>192</td>
<td>28</td>
<td></td>
<td>588</td>
<td>0</td>
<td>1</td>
<td></td>
<td>0</td>
<td>1</td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Total from 224 to 28 spatial resolution:
Memory: 7.5 MB
Params: 124K
MFLOP: 418

Compare VGG-16:
Memory: 42.9 MB (5.7x)
Params: 1.1M (8.9x)
MFLOP: 7485 (17.8x)

Szegedy et al, “Going deeper with convolutions”, CVPR 2015

Slide from Justin Johnson
GoogLeNet: Inception Module

Inception module
Local unit with parallel branches

Local structure repeated many times throughout the network

Szegedy et al, “Going deeper with convolutions”, CVPR 2015

Slide from Justin Johnson
GoogLeNet: Inception Module

**Inception module**
Local unit with parallel branches

Local structure repeated many times throughout the network

Uses 1x1 “Bottleneck” layers to reduce channel dimension before expensive conv (we will revisit this with ResNet!)

Szegedy et al, “Going deeper with convolutions”, CVPR 2015

Slide from Justin Johnson
GoogLeNet: Global Average Pooling

No large FC layers at the end! Instead uses **global average pooling** to collapse spatial dimensions, and one linear layer to produce class scores (Recall VGG-16: Most parameters were in the FC layers!)

<table>
<thead>
<tr>
<th>Layer</th>
<th>Input size</th>
<th>Layer</th>
<th>Output size</th>
<th>C</th>
<th>H/W</th>
<th>filters</th>
<th>kernel</th>
<th>stride</th>
<th>pad</th>
<th>C</th>
<th>H/W</th>
<th>memory (KB)</th>
<th>params (k)</th>
<th>flop (M)</th>
</tr>
</thead>
<tbody>
<tr>
<td>avg-pool</td>
<td>1024 7</td>
<td></td>
<td></td>
<td>1024</td>
<td>1</td>
<td>7</td>
<td>1</td>
<td>0</td>
<td></td>
<td>1024</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>fc</td>
<td>1024 1000</td>
<td></td>
<td></td>
<td>1000</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000</td>
<td></td>
<td>0</td>
<td>1025</td>
<td>1</td>
</tr>
</tbody>
</table>

Compare with VGG-16:
GoogLeNet: Global Average Pooling

No large FC layers at the end! Instead uses global average pooling to collapse spatial dimensions, and one linear layer to produce class scores (Recall VGG-16: Most parameters were in the FC layers!)

<table>
<thead>
<tr>
<th>Layer</th>
<th>Input size</th>
<th>Layer</th>
<th>Output size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>C</td>
<td>H/W</td>
<td>filters</td>
</tr>
<tr>
<td>avg-pool</td>
<td>1024</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>fc</td>
<td>1024</td>
<td>1000</td>
<td></td>
</tr>
</tbody>
</table>

Compare with VGG-16:

<table>
<thead>
<tr>
<th>Layer</th>
<th>C</th>
<th>H/W</th>
<th>filters</th>
<th>kernel</th>
<th>stride</th>
<th>pad</th>
<th>C</th>
<th>H/W</th>
<th>memory (KB)</th>
<th>params (K)</th>
<th>flop (M)</th>
</tr>
</thead>
<tbody>
<tr>
<td>flatten</td>
<td>512</td>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>25088</td>
<td></td>
<td>98</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fc6</td>
<td>25088</td>
<td>4096</td>
<td>4096</td>
<td>4096</td>
<td>4</td>
<td></td>
<td>16</td>
<td>102760</td>
<td>103</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fc7</td>
<td>4096</td>
<td>4096</td>
<td>4096</td>
<td>4096</td>
<td>4</td>
<td></td>
<td>16</td>
<td>16777</td>
<td>17</td>
<td></td>
<td></td>
</tr>
<tr>
<td>fc8</td>
<td>4096</td>
<td>1000</td>
<td>1000</td>
<td>1000</td>
<td>4</td>
<td></td>
<td>4</td>
<td>4096</td>
<td>4</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Slide from Justin Johnson
GoogLeNet: Auxiliary Classifiers

Training using loss at the end of the network didn’t work well: Network is too deep, gradients don’t propagate cleanly

As a hack, attach “auxiliary classifiers” at several intermediate points in the network that also try to classify the image and receive loss

GoogLeNet was before batch normalization! With BatchNorm no longer need to use this trick
ImageNet Classification Challenge

Error Rate

Shallow

2010: Lin et al

2011: Sanchez & Perronnin

2012: Krizhevsky et al (AlexNet)

2013: Zeiler & Fergus

2014: Simonyan & Zisserman (VGG)

2014: Szegedy et al (GoogLeNet)

19 layers

22 layers

8 layers

8 layers

16.4

11.7

7.3

6.7

5.1

3.6

3

2.3

5

10

15

20

25
ImageNet Classification Challenge

- 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)

Error Rates:
- 2010: 28.2%
- 2011: 25.8%
- 2012: 16.4%
- 2013: 11.7% (19 layers)
- 2014: 7.3% (22 layers)
- 2014: 6.7%
- 2015: 3.6% (152 layers)

Shallow: 8 layers

Deep: 152 layers
Residual Networks

Once we have Batch Normalization, we can train networks with 10+ layers. What happens as we go deeper?
Residual Networks

Once we have Batch Normalization, we can train networks with 10+ layers. What happens as we go deeper?

Deeper model does worse than shallow model!

Initial guess: Deep model is overfitting since it is much bigger than the other model.


Slide from Justin Johnson
Residual Networks

Once we have Batch Normalization, we can train networks with 10+ layers. What happens as we go deeper?

In fact the deep model seems to be underfitting since it also performs worse than the shallow model on the training set! It is actually underfitting.
Residual Networks

A deeper model can emulate a shallower model: copy layers from shallower model, set extra layers to identity

Thus deeper models should do at least as good as shallow models

**Hypothesis:** This is an optimization problem. Deeper models are harder to optimize, and in particular don’t learn identity functions to emulate shallow models.
Residual Networks

A deeper model can emulate a shallower model: copy layers from shallower model, set extra layers to identity

Thus deeper models should do at least as good as shallow models

**Hypothesis:** This is an optimization problem. Deeper models are harder to optimize, and in particular don’t learn identity functions to emulate shallow models

**Solution:** Change the network so learning identity functions with extra layers is easy!
Residual Networks

**Solution:** Change the network so learning identity functions with extra layers is easy!

```
H(x)

conv
 relu
conv
 X

"Plain" block
```

```
F(x) + x

relu

F(x)

relu

X

Residual Block
```

Residual Networks

**Solution:** Change the network so learning identity functions with extra layers is easy!

If you set these to 0, the whole block will compute the identity function!


Slide from Justin Johnson
Residual Networks

A residual network is a stack of many residual blocks.

Regular design, like VGG: each residual block has two 3x3 conv.

Network is divided into stages: the first block of each stage halves the resolution (with stride-2 conv) and doubles the number of channels.

Residual Networks

Uses the same aggressive stem as GoogleNet to downsample the input 4x before applying residual blocks:

<table>
<thead>
<tr>
<th>Layer</th>
<th>Input size</th>
<th>Layer</th>
<th>Output size</th>
<th>params (k)</th>
<th>flop (M)</th>
</tr>
</thead>
<tbody>
<tr>
<td>C</td>
<td>H/W</td>
<td>filters</td>
<td>kernel</td>
<td>stride</td>
<td>pad</td>
</tr>
<tr>
<td>conv</td>
<td>3 224</td>
<td>64</td>
<td>7</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>
| max-pool | 64 112 | 3       | 2       | 1      | 64  | 56   |       | 784        | 0 2  


Slide from Justin Johnson
Residual Networks

Like GoogLeNet, no big fully-connected-layers: instead use **global average pooling** and a single linear layer at the end


Slide from Justin Johnson
Residual Networks

**ResNet-18:**

Stem: 1 conv layer
Stage 1 (C=64): 2 res. block = 4 conv
Stage 2 (C=128): 2 res. block = 4 conv
Stage 3 (C=256): 2 res. block = 4 conv
Stage 4 (C=512): 2 res. block = 4 conv
Linear

ImageNet top-5 error: 10.92
GFLOP: 1.8

Error rates are 224x224 single-crop testing, reported by torchvision

Slide from Justin Johnson
Residual Networks

**ResNet-18:**
- Stem: 1 conv layer
- Stage 1 (C=64): 2 res. block = 4 conv
- Stage 2 (C=128): 2 res. block = 4 conv
- Stage 3 (C=256): 2 res. block = 4 conv
- Stage 4 (C=512): 2 res. block = 4 conv
- Linear

ImageNet top-5 error: 10.92
GFLOP: 1.8

**ResNet-34:**
- Stem: 1 conv layer
- Stage 1: 3 res. block = 6 conv
- Stage 2: 4 res. block = 8 conv
- Stage 3: 6 res. block = 12 conv
- Stage 4: 3 res. block = 6 conv
- Linear

ImageNet top-5 error: 8.58
GFLOP: 3.6

Error rates are 224x224 single-crop testing, reported by torchvision

Slide from Justin Johnson
Residual Networks

**ResNet-18:**
Stem: 1 conv layer
Stage 1 (C=64): 2 res. block = 4 conv
Stage 2 (C=128): 2 res. block = 4 conv
Stage 3 (C=256): 2 res. block = 4 conv
Stage 4 (C=512): 2 res. block = 4 conv
Linear

ImageNet top-5 error: 10.92
GFLOP: 1.8

**ResNet-34:**
Stem: 1 conv layer
Stage 1: 3 res. block = 6 conv
Stage 2: 4 res. block = 8 conv
Stage 3: 6 res. block = 12 conv
Stage 4: 3 res. block = 6 conv
Linear

ImageNet top-5 error: 8.58
GFLOP: 3.6

**VGG-16:**
ImageNet top-5 error: 9.62
GFLOP: 13.6

Error rates are 224x224 single-crop testing, reported by [torchvision](https://pytorch.org/)
Residual Networks: Basic Block


Slide from Justin Johnson
Residual Networks: Basic Block


Slide from Justin Johnson
Residual Networks: Bottleneck Block


Slide from Justin Johnson
Residual Networks: Bottleneck Block

More layers, less computational cost!

"Basic" Residual block

- Conv(3x3, C→C)
- Conv(3x3, C→C)
- FLOPs: 9HWC^2

Total FLOPs: 18HWC^2

"Bottleneck" Residual block

- Conv(3x1, C→4C)
- Conv(3x3, C→C)
- Conv(1x1, 4C→C)
- FLOPs: 4HWC^2
- FLOPs: 9HWC^2
- FLOPs: 4HWC^2

Total FLOPs: 17HWC^2


Slide from Justin Johnson
Residual Networks

Error rates are 224x224 single-crop testing, reported by torchvision

<table>
<thead>
<tr>
<th>Block type</th>
<th>Stem layers</th>
<th>Stage 1</th>
<th>Stage 2</th>
<th>Stage 3</th>
<th>Stage 4</th>
<th>FC layers</th>
<th>GFLOP</th>
<th>ImageNet top-5 error</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-18</td>
<td>Basic</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>ResNet-34</td>
<td>Basic</td>
<td>1</td>
<td>3</td>
<td>6</td>
<td>4</td>
<td>8</td>
<td>6</td>
<td>12</td>
</tr>
</tbody>
</table>
Residual Networks

ResNet-50 is the same as ResNet-34, but replaces Basic blocks with Bottleneck Blocks. This is a great baseline architecture for many tasks even today!

<table>
<thead>
<tr>
<th>Block type</th>
<th>Stem layers</th>
<th>Stage 1</th>
<th>Stage 2</th>
<th>Stage 3</th>
<th>Stage 4</th>
<th>FC layers</th>
<th>GFLOP</th>
<th>ImageNet top-5 error</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-18</td>
<td>Basic</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>1</td>
<td>1.8</td>
</tr>
<tr>
<td>ResNet-34</td>
<td>Basic</td>
<td>1</td>
<td>3</td>
<td>6</td>
<td>4</td>
<td>8</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>ResNet-50</td>
<td>Bottle</td>
<td>1</td>
<td>3</td>
<td>9</td>
<td>4</td>
<td>12</td>
<td>6</td>
<td>1</td>
</tr>
</tbody>
</table>

Error rates are 224x224 single-crop testing, reported by torchvision
Residual Networks

Deeper ResNet-101 and ResNet-152 models are more accurate, but also more computationally heavy.

<table>
<thead>
<tr>
<th>Block type</th>
<th>Stem layers</th>
<th>Stage 1</th>
<th>Stage 2</th>
<th>Stage 3</th>
<th>Stage 4</th>
<th>FC layers</th>
<th>GFLOP</th>
<th>ImageNet top-5 error</th>
</tr>
</thead>
<tbody>
<tr>
<td>ResNet-18</td>
<td>Basic</td>
<td>1</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>ResNet-34</td>
<td>Basic</td>
<td>1</td>
<td>3</td>
<td>6</td>
<td>4</td>
<td>8</td>
<td>6</td>
<td>12</td>
</tr>
<tr>
<td>ResNet-50</td>
<td>Bottle</td>
<td>1</td>
<td>3</td>
<td>9</td>
<td>4</td>
<td>12</td>
<td>6</td>
<td>18</td>
</tr>
<tr>
<td>ResNet-101</td>
<td>Bottle</td>
<td>1</td>
<td>3</td>
<td>9</td>
<td>4</td>
<td>12</td>
<td>23</td>
<td>69</td>
</tr>
<tr>
<td>ResNet-152</td>
<td>Bottle</td>
<td>1</td>
<td>3</td>
<td>9</td>
<td>8</td>
<td>24</td>
<td>36</td>
<td>108</td>
</tr>
</tbody>
</table>

Error rates are 224x224 single-crop testing, reported by torchvision
Residual Networks

- Able to train very deep networks
- Deeper networks do better than shallow networks (as expected)
- Swept 1st place in all ILSVRC and COCO 2015 competitions
- Still widely used today!

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


Slide from Justin Johnson
Improving Residual Networks: Block Design

Original ResNet block

“Pre-Activation” ResNet Block

Note ReLU **after** residual:
Cannot actually learn identity function since outputs are nonnegative!

Note ReLU **inside** residual:
Can learn true identity function by setting Conv weights to zero!

He et al., “Identity mappings in deep residual networks”, ECCV 2016

Slide from Justin Johnson
Improving Residual Networks: Block Design

Original ResNet block

- ReLU
- Batch Norm
- Conv
- ReLU
- Batch Norm
- Conv

“Pre-Activation” ResNet Block

- Conv
- ReLU
- Batch Norm
- Conv
- ReLU
- Batch Norm

Slight improvement in accuracy (ImageNet top-1 error)

ResNet-152: 21.3 vs 21.1
ResNet-200: 21.8 vs 20.7

Not actually used that much in practice


Slide from Justin Johnson
Recap

Multiple smaller convs v/s a single large conv

Global average pooling vs fully connected heads

Auxiliary Supervision
Today’s plan

• Finish talking about other architectures since AlexNet
• Designing neural networks for vision problems
  • Segmentation
  • ...

Recap

- **Conv(3x3, C->C)**
- **Conv(5x5, C->C)**
- **Conv(3x3, C->C)**

- Multiple smaller convs v/s a single large conv

- Global average pooling vs fully connected heads

- **Residual Connections**

- **Softmax**
  - **FC 1000**
  - **Pool**
  - **3x3 conv, 512**

- **Auxiliary Supervision**

- **Test error**
  - 56-layer
  - 20-layer

- **Iterations**
  - Shallow (8 layers)
  - 2010 Lin et al.
  - 2011 Sanchez & Parmee
  - 2012 Krizhevsky et al.
  - 2013 Zeiler & Fergus
  - 2013 Simonyan & Zisserman
  - 2014 Sengoddy et al.
  - 2014 He et al.
  - 2015 Huang et al.

- **Error Rate**
  - 2010: 28.2%
  - 2011: 25.8%
  - 2012: 16.4%
  - 2013: 11.7%
  - 2014: 7.3%
  - 2015: 6.7%
  - 2016: 3.6%

- **Fully Connected Heads**
  - **relu**
  - **F(x)**
  - **conv**
  - **relu**
  - **F(x) + x**
  - **conv**
  - **relu**
  - **X**
Comparing Complexity


Slide from Justin Johnson
Comparing Complexity

Inception-v4: Resnet + Inception!


Slide from Justin Johnson
Comparing Complexity

VGG: Highest memory, most operations


Slide from Justin Johnson
Comparing Complexity

GoogLeNet:
Very efficient!


Slide from Justin Johnson
Comparing Complexity

AlexNet: Low compute, lots of parameters

Comparing Complexity

ResNet: Simple design, moderate efficiency, high accuracy


Slide from Justin Johnson
ImageNet Classification Challenge

- 2010: Lin et al
- 2011: Sanchez & Perronnin
- 2012: Krizhevsky et al (AlexNet)
- 2013: Szegedy et al (GoogLeNet)
- 2014: He et al (ResNet)

Shallow:
- 8 layers

Deep:
- 2011: 16.4 layers
- 2012: 11.7 layers
- 2013: 19 layers
- 2014: 22 layers
- 2015: 3.6 layers

Error Rate
A. Dosovitskiy et al., *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale.*
$y_i = \sum_j w_{ij} x_{ij}$

$w_{ij} = \text{softmax}_j(x_i^T x_j / \sqrt{d_k})$

$w_{ij} = \frac{e^{x_i^T x_j}}{\sum_j e^{x_i^T x_j}}$
Attention (with key, query and value)

\[ y_i = \sum_j w_{ij} W_v x_{ij} \]

\[ w_{ij} = \text{softmax}_j \left( (W_q x_i)^T W_k x_j / \sqrt{d_k} \right) \]

Source: [http://peterbloem.nl/blog/transformers](http://peterbloem.nl/blog/transformers)  See also: Attention is all you need
Attention (Vision Transformers)

<table>
<thead>
<tr>
<th></th>
<th>Ours-JFT (ViT-H/14)</th>
<th>Ours-JFT (ViT-L/16)</th>
<th>Ours-I21K (ViT-L/16)</th>
<th>BiT-L (ResNet152x4)</th>
<th>Noisy Student (EfficientNet-L2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ImageNet</td>
<td>88.55 ± 0.04</td>
<td>87.76 ± 0.03</td>
<td>85.30 ± 0.02</td>
<td>87.54 ± 0.02</td>
<td>88.4/88.5*</td>
</tr>
<tr>
<td>ImageNet ReAL</td>
<td>90.72 ± 0.05</td>
<td>90.54 ± 0.03</td>
<td>88.62 ± 0.05</td>
<td>90.54</td>
<td>90.55</td>
</tr>
<tr>
<td>CIFAR-10</td>
<td>99.50 ± 0.06</td>
<td>99.42 ± 0.03</td>
<td>99.15 ± 0.03</td>
<td>99.37 ± 0.06</td>
<td></td>
</tr>
<tr>
<td>CIFAR-100</td>
<td>94.55 ± 0.04</td>
<td>93.90 ± 0.05</td>
<td>93.25 ± 0.05</td>
<td>93.51 ± 0.08</td>
<td></td>
</tr>
<tr>
<td>Oxford-IIIT Pets</td>
<td>97.56 ± 0.03</td>
<td>97.32 ± 0.11</td>
<td>94.67 ± 0.15</td>
<td>96.62 ± 0.23</td>
<td></td>
</tr>
<tr>
<td>Oxford Flowers-102</td>
<td>99.68 ± 0.02</td>
<td>99.74 ± 0.00</td>
<td>99.61 ± 0.02</td>
<td>99.63 ± 0.03</td>
<td></td>
</tr>
<tr>
<td>VTAB (19 tasks)</td>
<td>77.63 ± 0.23</td>
<td>76.28 ± 0.46</td>
<td>72.72 ± 0.21</td>
<td>76.29 ± 1.70</td>
<td></td>
</tr>
<tr>
<td>TPUv3-core-days</td>
<td>2.5k</td>
<td>0.68k</td>
<td>0.23k</td>
<td>9.9k</td>
<td>12.3k</td>
</tr>
</tbody>
</table>

Table 2: Comparison with state of the art on popular image classification benchmarks. We report mean and standard deviation of the accuracies, averaged over three fine-tuning runs. Vision Transformer models pre-trained on the JFT-300M dataset outperform ResNet-based baselines on all datasets, while taking substantially less computational resources to pre-train. ViT pre-trained on the smaller public ImageNet-21k dataset performs well too. *Slightly improved 88.5% result reported in Touvron et al. (2020).

A. Dosovitskiy et al., *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale.*
Scale Better with More Data