Lecture 13: Modern Architectures
Last time: transformer encoder

Transformer encoder

Positional encoding

Multi-head self-attention

Layer norm

MLP

Layer norm

Positional encoding
Last time: transformer decoder

Transformer decoder

person wearing hat [END]

\[ \begin{align*}
    y_0 & \quad y_1 & \quad y_2 & \quad y_3 \\
\end{align*} \]

\[ \begin{align*}
    \text{c}_{0,0} & \quad \text{c}_{0,1} & \quad \text{c}_{0,2} \\
\end{align*} \]

\[ \begin{align*}
    \text{c}_{2,0} & \quad \text{c}_{2,1} & \quad \text{c}_{2,2} \\
\end{align*} \]

\[ \begin{align*}
    x_0 & \quad x_1 & \quad x_2 & \quad x_3 \\
\end{align*} \]

Vaswani et al, "Attention is all you need", NeurIPS 2017
<table>
<thead>
<tr>
<th>Model</th>
<th>Layers</th>
<th>Width</th>
<th>Heads</th>
<th>Params</th>
<th>Data</th>
<th>Training</th>
</tr>
</thead>
<tbody>
<tr>
<td>Transformer-Base</td>
<td>12</td>
<td>512</td>
<td>8</td>
<td>65M</td>
<td>?</td>
<td>8x P100 (12 hours)</td>
</tr>
<tr>
<td>Transformer-Large</td>
<td>12</td>
<td>1024</td>
<td>16</td>
<td>213M</td>
<td>?</td>
<td>8x P100 (3.5 days)</td>
</tr>
<tr>
<td>BERT-Base</td>
<td>12</td>
<td>768</td>
<td>12</td>
<td>119M</td>
<td>13GB</td>
<td>?</td>
</tr>
<tr>
<td>BERT-Large</td>
<td>24</td>
<td>1024</td>
<td>16</td>
<td>340M</td>
<td>13GB</td>
<td>?</td>
</tr>
<tr>
<td>XLNet-Large</td>
<td>24</td>
<td>1024</td>
<td>16</td>
<td>~340M</td>
<td>126GB</td>
<td>512x TPU-v3 (2.5 days)</td>
</tr>
<tr>
<td>RoBERTa</td>
<td>24</td>
<td>1024</td>
<td>16</td>
<td>355M</td>
<td>160GB</td>
<td>1024x V100 GPUs (1 day)</td>
</tr>
<tr>
<td>GPT-2</td>
<td>48</td>
<td>1600</td>
<td></td>
<td>1.5B</td>
<td>40GB</td>
<td>?</td>
</tr>
<tr>
<td>Megatron-LM</td>
<td>72</td>
<td>3072</td>
<td>32</td>
<td>8.3B</td>
<td>174GB</td>
<td>512x V100 GPU (9 days)</td>
</tr>
<tr>
<td>Turing-NLG</td>
<td>78</td>
<td>4256</td>
<td>28</td>
<td>17B</td>
<td>?</td>
<td>256x V100 GPUs</td>
</tr>
<tr>
<td>GPT-3</td>
<td>96</td>
<td>12288</td>
<td>96</td>
<td>175B</td>
<td>694GB</td>
<td>?</td>
</tr>
<tr>
<td>Gopher</td>
<td>80</td>
<td>16384</td>
<td>128</td>
<td>280B</td>
<td>10.55TB</td>
<td>4096x TPU-v3 (38 days)</td>
</tr>
</tbody>
</table>
Today: Modern Architectures
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]
Review: Convolution

32x32x3 image

3x3x3 filter $w$

Padding:
Preserve input spatial dimensions in output activations

Stride:
Downsample output activations
Review: Convolution

Each conv filter outputs a “slice” in the activation.
Review: Pooling

Single depth slice

max pool with 2x2 filters and stride 2
Today: Modern Architectures

Case Studies
- AlexNet
- VGG
- GoogLeNet
- ResNet
- ViT
- MLP Mixers

Also....
- SENet
- Wide ResNet
- ResNeXT
- DenseNet
- MobileNets
- NASNet
- EfficientNet
ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

- Sanchez & Perronnin (2011)
- Krizhevsky et al (AlexNet, 2012)
- Zeiler & Fergus (2013)
- Simonyan & Zisserman (VGG, 2014)
- Szegedy et al (GoogLeNet, 2014)
- He et al (ResNet, 2015)
- Shao et al (2016)
- Hu et al (SENet, 2017)
- Russakovsky et al (Human, 2018)

- Layer Count: Shallow (8 layers), 19 layers, 22 layers, 152 layers, 152 layers, 152 layers.
ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

First CNN-based winner

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

shallow
8 layers
8 layers
19 layers
22 layers
152 layers
152 layers
152 layers
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]**

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

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

[Krizhevsky et al. 2012]

Input: 227x227x3 images

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

=>

Output volume [55x55x96]

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

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

[Krizhevsky et al. 2012]

Input: 227x227x3 images

First layer (CONV1): 96 11x11 filters applied at stride 4
=>
Output volume [55x55x96]
Parameters: (11*11*3 + 1)*96 = 35K

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

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?

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.
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:

[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.
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)

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

First CNN-based winner:

- Krizhevsky et al (AlexNet) - 19 layers
- Simonyan & Zisserman (VGG) - 22 layers
- He et al (ResNet) - 152 layers
- Hu et al (SENet) - 152 layers
- Russakovsky et al - 152 layers

Shallow:

- Lin et al
- Sanchez & Perronnin
- Krizhevsky et al (AlexNet) - 8 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)
- Shao et al
- Hu et al (SENet)
- Russakovsky et al

ZFNet: Improved hyperparameters over AlexNet

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

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

Ali Farhadi, Aditya Kusupati

Lecture 13 - 28

Nov 16, 2023
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%
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) 7.3
- **2014**: Szegedy et al (GoogLeNet) 6.7
- **2015**: He et al (ResNet) 3.6
- **2016**: Shao et al 3.0
- **2017**: Hu et al (SENet) 2.3
- **Human**: 5.1

**Deeper Networks**
- **2011**: 8 layers
- **2012**: 8 layers
- **2013**: 19 layers
- **2014**: 22 layers
- **2015**: 152 layers
- **2016**: 152 layers
- **2017**: 152 layers

*Note: Layer counts for Deeper Networks are approximate.*
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

[Simonyan and Zisserman, 2014]

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

[Simonyan and Zisserman, 2014]

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

[Simonyan and Zisserman, 2014]

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

![Diagram of VGGNet architecture](image-url)
Case Study: VGGNet
[Simonyan and Zisserman, 2014]

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

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

Q: 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
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 = 800K 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
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=800K params: 0

CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*128)*256 = 589,924

CONV3-256: [56x56x256] memory: 56*56*256=800K params: (3*3*256)*256 = 589,924

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

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

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 (for a forward pass)

TOTAL params: 138M parameters
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
INPUT: [224x224x3]  memory: 224*224*3=150K  params: 0  

CONV3-64: [224x224x64]  memory: 224*224*64=3.2M  params: (3*3*64)*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=800K  params: 0  
CONV3-256: [56x56x256]  memory: 56*56*256=800K  params: (3*3*128)*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  
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  
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

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
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

Shallow: 8 layers
Deeper Networks: 19 layers, 22 layers, 152 layers, 152 layers, 152 layers

Human: 5.1
Case Study: GoogLeNet

[ Szegedy et al., 2014 ]

Deeper networks, with computational efficiency

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

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 channel-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 channel-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 are the output sizes of all different filter operations?

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

Naive Inception module

Module input:
28x28x256

Input

Filter concatenation

1x1 conv, 128
3x3 conv, 192
5x5 conv, 96
3x3 pool
Case Study: GoogLeNet
[Szegedy et al., 2014]

Example: Q1: What are the output sizes of all different filter operations?

Naive Inception module

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

Naive Inception module

Example:

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

Q2: What is output size after filter concatenation?

Module input: 28x28x256

Input

Filter concatenation

28x28x128
28x28x192
28x28x96
28x28x256

1x1 conv, 128
3x3 conv, 192
5x5 conv, 96
3x3 pool

Example:

Input: 28x28x256

1x1 conv, 128

3x3 pool
Case Study: GoogLeNet
[Szegedy et al., 2014]

Naive Inception module

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

Example:

Q2: What is output size after filter concatenation?

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

Module input: 28x28x256

Input

Filter concatenation

1x1 conv, 128

3x3 conv, 192

5x5 conv, 96

3x3 pool

28x28x128

28x28x192

28x28x96

28x28x256
Case Study: GoogLeNet
[Szegedy et al., 2014]

Example:

Q2: What is output size after filter concatenation?

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

Conv Ops:

- [1x1 conv, 128] 28x28x128x1x1x256
- [3x3 conv, 192] 28x28x192x3x3x256
- [5x5 conv, 96] 28x28x96x5x5x256

Total: 854M ops

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

Example:

Module input: 28x28x256

1x1 conv, 128
3x3 conv, 192
5x5 conv, 96
3x3 pool

Filter concatenation

Naive Inception module
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!

Q2: What is output size after filter concatenation?

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

28x28x128  28x28x192  28x28x96  28x28x256

Module input: 28x28x256

Input

Filter concatenation

1x1 conv, 128  3x3 conv, 192  5x5 conv, 96  3x3 pool
Case Study: GoogLeNet
[Szegedy et al., 2014]

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

Example:

Q2: What is output size after filter concatenation?

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

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

Naive Inception module
Review: 1x1 convolutions

1x1 CONV with 32 filters

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

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

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

Ali Farhadi, Aditya Kusupati  Lecture 13 - 60  Nov 16, 2023
Review: 1x1 convolutions

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

Projects depth to lower dimension (combination of feature maps)

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

1x1 conv “bottleneck”
layers
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

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!
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
- 27x less params than VGG-16
- ILSVRC’14 classification winner (6.7% top 5 error)
ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

- Sanchez & Perronnin (2011)
- Zeiler & Fergus (2013)
- Simonyan & Zisserman (VGG) (2014)
- Szegedy et al (GoogLeNet) (2014)
- Shao et al (2016)

“Revolution of Depth”

- 2010: Lin et al, shallow
- 2011: Sanchez & Perronnin, 8 layers
- 2012: Krizhevsky et al (AlexNet), 8 layers
- 2013: Zeiler & Fergus, 8 layers
- 2014: Simonyan & Zisserman (VGG), 19 layers
- 2014: Szegedy et al (GoogLeNet), 22 layers
- 2015: He et al (ResNet), 152 layers
- 2016: Shao et al, 152 layers
- 2017: Hu et al (SENet), 152 layers
- 2018: Russakovsky et al (Human), 192 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?

![Graphs showing training and test error over iterations for 20-layer and 56-layer ResNet models.](image)
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 test and training error

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

[He et al., 2015]

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

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

[He et al., 2015]

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

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

What should the deeper model learn to be at least as good 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

```
H(x)
```

```
<table>
<thead>
<tr>
<th>conv</th>
</tr>
</thead>
<tbody>
<tr>
<td>relu</td>
</tr>
<tr>
<td>conv</td>
</tr>
</tbody>
</table>
```

“Plain” layers
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.

### Diagram

- **“Plain” layers**
  - $X$ → \( \text{conv} \) → \( \text{relu} \) → \( \text{conv} \)
  - $H(x)$

- **Residual block**
  - $X$ → \( \text{conv} \) → \( \text{relu} \) → \( \text{conv} \)
  - $H(x) = F(x) + x$
  - Identity mapping: $H(x) = x$ if $F(x) = 0$
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.

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

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

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

Instead of:
\[
H(x)
\]
Case Study: ResNet

[He et al., 2015]

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

\[ F(x) + x \]

**Residual block**

\[ \text{relu} \]

\[ 3x3 \text{ conv} \]

\[ \text{identity} \]

\[ \text{relu} \]

\[ 3x3 \text{ conv} \]

\[ \text{Pool} \]
Case Study: ResNet

[He et al., 2015]

- 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)

Reduce the activation volume by half.
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 (stem)
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 (stem)
- No FC layers at the end (only FC 1000 to output classes)
- (In theory, you can train a ResNet with input image of variable sizes)
Case Study: ResNet

[He et al., 2015]

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

[He et al., 2015]

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

[He et al., 2015]

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

[He et al., 2015]

Training ResNet in practice:

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

[He et al., 2015]

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

MSRA @ ILSVRC & COCO 2015 Competitions

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

[He et al., 2015]

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

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


Comparing complexity...

Inception-v4: Resnet + Inception!


Comparing complexity...


Comparing complexity...

GoogLeNet: most efficient


Comparing complexity...

AlexNet:
Smaller compute, still memory heavy, lower accuracy


Comparing complexity...


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)
- Human

Network ensembling

- 8 layers (2010 - Lin et al)
- 8 layers (2011 - Sanchez & Perronnin)
- 19 layers (2012 - Krizhevsky et al)
- 22 layers (2013 - Zeiler & Fergus)
- 152 layers (2014 - Simonyan & Zisserman)
- 152 layers (2014 - Szegedy et al)
- 152 layers (2015 - He et al)
- 152 layers (2016 - Shao et al)
- 152 layers (2017 - Hu et al)
- 152 layers (Human - Russakovsky et al)
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)
- Shao et al
- Hu et al (SENet)
- Russakovsky et al

Adaptive feature map reweighting

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

- 8 layers
- 19 layers
- 22 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

Completion of the challenge: Annual ImageNet competition no longer held after 2017 -> now moved to Kaggle.
How have transformers affected architectures?

Transformer Encoder Block:

**Inputs**: Set of vectors $\mathbf{x}$

**Outputs**: Set of vectors $\mathbf{y}$

Self-attention is the only interaction between vectors.

Layer norm and MLP operate independently per vector.

Highly scalable, highly parallelizable, but high memory usage.

Vaswani et al, "Attention is all you need", NeurIPS 2017
Notice the residual connections!!

Residual connections inherited from ResNet’s design.

Allows for better gradients to flow through all the transformers blocks.

Vaswani et al, “Attention is all you need”, NeurIPS 2017
How to incorporate transformers to vision?

Idea #1: pass the image pixels into the transformer encoder.

So, each $z_{0,0}$ is a pixel.

What is the problem with this idea?
How to incorporate transformers to vision?

Idea #1: pass the image pixels into the transformer encoder.

So, each $z_{0,0}$ is a pixel.

Q. What is the problem with this idea?
A. Memory issue: Assume images are 224x224 pixels. This means that self attention will produce $224^4 = 10^9$ values!
How to incorporate transformers to vision?

Idea #2: Divide image into patches and pass those patches into the transformer.

So, each $z_{0,0}$ is a 16x16x3 patch.

Q. What operation do you know already that operates over patches?

Dosovitskiy et al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021
Idea #2: Divide image into patches and pass those patches into the transformer.

So, each \( z_{0,0} \) is a 16x16x3 patch.

Q. What operation do you know already that operates over patches?
Yes it’s a convolution.

Q. What is the kernel size and stride and padding?
Idea #2: Divide image into patches and pass those patches into the transformer. So, each $z_{0,0}$ is a 16x16x3 patch.

Q. Does this solve the memory problem?
A. $16 \times 16 \times 3 = 768$.
So self-attention will produce $768 \times 768 < 10^5$.
Since transformers are permutation invariant, we want to add position encoding to each patch.

- Patches are 768D.
- Position encoding is some learned 768D.

Pick any consistent ordering of patches (e.g. top left patch is always first).

Simply Add position encoding and patch representation.
How to turn the output to a class prediction?

1. Add special [CLS] token.
   Similar to <start> and <end> tokens in NLP.
   Output CLS representation makes the final prediction using a linear layer.
Common ViT architectures

<table>
<thead>
<tr>
<th>Model</th>
<th>Layers</th>
<th>Hidden size $D$</th>
<th>MLP size</th>
<th>Heads</th>
<th>Params</th>
</tr>
</thead>
<tbody>
<tr>
<td>ViT-Base</td>
<td>12</td>
<td>768</td>
<td>3072</td>
<td>12</td>
<td>86M</td>
</tr>
<tr>
<td>ViT-Large</td>
<td>24</td>
<td>1024</td>
<td>4096</td>
<td>16</td>
<td>307M</td>
</tr>
<tr>
<td>ViT-Huge</td>
<td>32</td>
<td>1280</td>
<td>5120</td>
<td>16</td>
<td>632M</td>
</tr>
</tbody>
</table>

Common patch sizes: 32, 16, 14…
Smaller patches results in larger more powerful models.

Nomenclature: ViT-B/32 means that its a ViT model that uses Base values for layers, hidden size, mlp vize, and head. /32 means the input image patches are 32x32.

Dosovitskiy et al, “An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale”, ICLR 2021
Comparing ResNets with ViTs

Models are initially trained on a large dataset called JFT-300M

And then the last linear layer is finetuned on ImageNet-1.5M

ViT performs worse when only 10M images are used from JFT. But ViT outperforms ResNets with larger training data (300M images from JFT).

Dosovitskiy et al, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale", ICLR 2021
Self-attention is expensive… can we design something simpler?

Every self-attention is expensive. We want each input to “interact” with other tokens but can we simplify the operation a bit?

Vaswani et al, “Attention is all you need”, NeurIPS 2017
MLP-Mixer: an all-MLP architecture

Tolstikhin et al., “MLP-Mixer: An all-MLP architecture for vision”, NeurIPS2021
MLP-Mixer: an all-MLP architecture

Input: $N \times C$
- $N$ patches with $C$ channels each

MLP 1: $C \rightarrow C$
- apply to each of the $N$ patches

MLP 2: $N \rightarrow N$
- apply to each of the $C$ patches

Tolstikhin et al, “MLP-Mixer: An all-MLP architecture for vision”, NeurIPS2021
MLP-Mixer: The MLPs are sort of like **convs**

Cool idea; but hasn’t taken off yet.

---

**Input:** \(N \times C\)
- \(N\) patches with \(C\) channels each

**MLP 1:** \(C \rightarrow C\),
- apply to each of the \(N\) patches

**MLP 2:** \(N \rightarrow N\),
- apply to each of the \(C\) patches

---

Tolstikhin et al, “MLP-Mixer: An all-MLP architecture for vision”, NeurIPS2021
MLP-Mixer: Many concurrent and followups


But research has continued since ImageNet
(Will go over following slides if time,
Otherwise skip to the summary slides in the end.)
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
- 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...

Densely Connected Convolutional Networks (DenseNet)

[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
- Showed that shallow 50-layer network can outperform deeper 152 layer ResNet
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
- Much more efficient, with little loss in accuracy
- Follow-up MobileNetV2 work in 2018 (Sandler et al.)
- ShuffleNet: Zhang et al, CVPR 2018
Learning to search for 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)
Learning to search for 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)
EfficientNet: Smart Compound Scaling

[Tan and Le. 2019]

- Increase network capacity by scaling width, depth, and resolution, while balancing accuracy and efficiency.
- Search for optimal set of compound scaling factors given a compute budget (target memory & flops).
- Scale up using smart heuristic rules

depth: \( d = \alpha^\phi \)
width: \( w = \beta^\phi \)
resolution: \( r = \gamma^\phi \)

s.t. \( \alpha \cdot \beta^2 \cdot \gamma^2 \approx 2 \)
\( \alpha \geq 1, \beta \geq 1, \gamma \geq 1 \)
Efficient networks...

https://openai.com/blog/ai-and-efficiency/
Summary: Modern Architectures

Case Studies
- AlexNet
- VGG
- GoogLeNet
- ResNet
- Vit
- MLP-Mixer

Also....
- SENet
- Wide ResNet
- ResNeXT
- DenseNet
- MobileNets
- NASNet
Main takeaways

**AlexNet** showed that you can use CNNs to train Computer Vision models.

**ZFNet, VGG** shows that bigger networks work better.

**GoogLeNet** is one of the first to focus on efficiency using 1x1 bottleneck convolutions and global avg pool instead of FC layers.

**ResNet** showed us how to train extremely deep networks:
- Limited only by GPU & memory!
- Showed diminishing returns as networks got bigger.

After ResNet: CNNs were better than the human metric and focus shifted to efficient networks:
- Lots of tiny networks aimed at mobile devices: **MobileNet, ShuffleNet**

**Neural Architecture Search** can now automate architecture design.

**ViT** is the current favorite architecture but requires a lot of compute and data.

**MLP-Mixers** have presented an alternative to transformers but they haven’t taken off.
Summary: Modern Architectures

- Many popular architectures available in model zoos
- ResNet-50 and ViT currently good defaults to use
- Next time: Structure prediction
Next time: Structured Prediction