We saw before:

A series of matrix multiplications:

- \( x \mapsto W_1^T x \mapsto h_1 = f(W_1^T x) \mapsto W_2^T h_1 \mapsto h_2 = f(W_2^T h_1) \mapsto W_3^T h_2 \mapsto h_3 = f(W_3^T h_3) \mapsto W_4^T h_3 = \hat{y} \)
Convolutional Networks

- Neural Networks that use convolution in place of general matrix multiplication in at least one layer

Next:
- What is convolution?
- What is pooling?
- What is the motivation for such architectures (remember LeNet?)
LeNet-5 (LeCun, 1998)

The original Convolutional Neural Network model goes back to 1989 (LeCun)
AlexNet (Krizhevsky, Sutskever, Hinton 2012)

- ImageNet 2012 15.4% error rate
Convolutional Neural Networks

Figure: Andrej Karpathy
Now let’s deconstruct them...
Convolution

Convolve image with kernel having weights $w$ (learned by backpropagation)
Convolution

\( w^T x \)
Convolution
Convolution

$w^T x$
Convolution
Convolution

$w^T x$
Convolution

\[ w^T x \]
Convolution
Convolution

\[ w^T x \]
Convolution
Convolution

$w^T x$
Convolution
Convolution

\[w^T x\]
Convolution
Convolution

\[ w^T x \]
Convolution
Convolution

\[ W^T x \]
Convolution
Convolution

$w^T x$
Convolution
Convolution

\[ w^T x \]
Convolution

$w^T x$
Convolution
Convolution
Convolution
Convolution
Convolution
Convolution
Convolution
Convolution

What is the number of parameters?
Output Size

- We used stride of 1, kernel with receptive field of size 3 by 3
- Output size:
  \[ \frac{N - K}{S} + 1 \]

- In previous example: \( N = 6, K = 3, S = 1 \), output size = 4
- For \( N = 8, K = 3, S = 1 \), output size is 6
Zero Padding

- Often, we want the output of a convolution to have the same size as the input. Solution: Zero padding.
- In our previous example:

```
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0
```

- Common to see convolution layers with stride of 1, filters of size $K$, and zero padding with $\frac{K-1}{2}$ to preserve size.
Learn Multiple Filters
Learn Multiple Filters

- If we use 100 filters, we get 100 feature maps

Figure: I. Kokkinos
In General

- We have only considered a 2-D image as a running example.
- But we could operate on volumes (e.g. RGB Images would be depth 3 input, filter would have same depth).

Image from Wikipedia
For convolutional layer:

- Suppose input is of size $W_1 \times H_1 \times D_1$
- Filter size is $K$ and stride $S$
- We obtain another volume of dimensions $W_2 \times H_2 \times D_2$
- As before:

$$W_2 = \frac{W_1 - K}{S} + 1 \quad \text{and} \quad H_2 = \frac{H_1 - K}{S} + 1$$

- Depths will be equal
Example volume: $28 \times 28 \times 3$ (RGB Image)
100 $3 \times 3$ filters, stride 1
What is the zero padding needed to preserve size?
Number of parameters in this layer?
For every filter: $3 \times 3 \times 3 + 1 = 28$ parameters
Total parameters: $100 \times 28 = 2800$
Figure: Andrej Karpathy
After obtaining feature map, apply an elementwise non-linearity to obtain a transformed feature map (same size)
Figure: Andrej Karpathy
Pooling
Pooling

\[
\max \{ a_i \}
\]
Pooling

\[ \text{max}\{a_i\} \]
Pooling

\[ \text{max}\{a_i\} \]
Pooling

- Other options: Average pooling, L2-norm pooling, random pooling
Pooling

- We have multiple feature maps, and get an equal number of subsampled maps
- This changes if cross channel pooling is done
So what’s left: Fully Connected Layers

Figure: Andrej Karpathy
Filters are of size $5 \times 5$, stride 1
Pooling is $2 \times 2$, with stride 2
How many parameters?
**Input image:** $227 \times 227 \times 3$

**First convolutional layer:** 96 filters with $K = 11$ applied with stride $= 4$

**Width and height of output:** \[
\frac{227 - 11}{4} + 1 = 55
\]
Number of parameters in first layer?

11 X 11 X 3 X 96 = 34848
AlexNet

- Next layer: Pooling with 3 X 3 filters, stride of 2
- Size of output volume: 27
- Number of parameters?
AlexNet

- Popularized the use of ReLUs
- Used heavy data augmentation (flipped images, random crops of size 227 by 227)
- Parameters: Dropout rate 0.5, Batch size = 128, Weight decay term: 0.0005, Momentum term $\alpha = 0.9$, learning rate $\eta = 0.01$, manually reduced by factor of ten on monitoring validation loss.
Short Digression: How do the features look like?
Layer 1 filters

This and the next few illustrations are from Rob Fergus
Layer 2 Patches

Layer 2: Top 9 Patches

- Patches from validation images that give maximal activation of a given feature map
Layer 2 Patches

Layer 2: Top-9 Patches
Layer 3 Patches

Layer 3: Top-9 Patches
Layer 3 Patches

Layer 3: Top-9 Patches
Layer 4 Patches

Layer 4: Top-9 Patches
Evolution of Filters
Evolution of Filters

Caveat?
Back to Architectures
ImageNet 2013

- Was won by a network similar to AlexNet (Matthew Zeiler and Rob Fergus)
- Changed the first convolutional layer from $11 \times 11$ with stride of 4, to $7 \times 7$ with stride of 2
- AlexNet used 384, 384 and 256 layers in the next three convolutional layers, ZF used 512, 1024, 512
- ImageNet 2013: 14.8 % (reduced from 15.4 %) (top 5 errors)
VGGNet (Simonyan and Zisserman, 2014)

Best model: Column D.

Error: 7.3% (top five error)
VGGNet (Simonyan and Zisserman, 2014)

- Total number of parameters: 138 Million (calculate!)
- Memory (Karpathy): 24 Million X 4 bytes ≈ 93 MB per image
- For backward pass the memory usage is doubled per image
- Observations:
  - Early convolutional layers take most memory
  - Most parameters are in the fully connected layers
Going Deeper

Classification: ImageNet Challenge top-5 error

Figure: Kaiming He, MSR
Network in Network

(a) Linear convolution layer

(b) Mlpconv layer

M. Lin, Q. Chen, S. Yan, Network in Network, ICLR 2014
Google LeNet

*Szegedy et al, Going Deeper With Convolutions, CVPR 2015*

- **Error:** 6.7% (top five error)
The Inception Module

- Parallel paths with different receptive field sizes - capture sparse patterns of correlation in stack of feature maps
- Also include auxiliary classifiers for ease of training
- Also note 1 by 1 convolutions
### Google LeNet

<table>
<thead>
<tr>
<th>type</th>
<th>patch size/strides</th>
<th>output size</th>
<th>depth</th>
<th>#1×1 reduce</th>
<th>#3×3 reduce</th>
<th>#5×5 reduce</th>
<th>pool proj</th>
<th>params</th>
<th>ops</th>
</tr>
</thead>
<tbody>
<tr>
<td>convolution</td>
<td>7×7/2</td>
<td>112×112×64</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2.7K</td>
<td>34M</td>
</tr>
<tr>
<td>max pool</td>
<td>3×3/2</td>
<td>56×56×64</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>112K</td>
<td>360M</td>
</tr>
<tr>
<td>convolution</td>
<td>3×3/1</td>
<td>56×56×192</td>
<td>2</td>
<td></td>
<td>64</td>
<td>192</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>max pool</td>
<td>3×3/2</td>
<td>28×28×192</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>inception (3a)</td>
<td>28×28×256</td>
<td>2</td>
<td>64</td>
<td>96</td>
<td>128</td>
<td>32</td>
<td>32</td>
<td>159K</td>
<td>128M</td>
</tr>
<tr>
<td>inception (3b)</td>
<td>28×28×480</td>
<td>2</td>
<td>128</td>
<td>128</td>
<td>192</td>
<td>32</td>
<td>96</td>
<td>380K</td>
<td>304M</td>
</tr>
<tr>
<td>max pool</td>
<td>3×3/2</td>
<td>14×14×480</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>inception (4a)</td>
<td>14×14×512</td>
<td>2</td>
<td>192</td>
<td>96</td>
<td>208</td>
<td>16</td>
<td>48</td>
<td>364K</td>
<td>73M</td>
</tr>
<tr>
<td>inception (4b)</td>
<td>14×14×512</td>
<td>2</td>
<td>160</td>
<td>112</td>
<td>224</td>
<td>24</td>
<td>64</td>
<td>437K</td>
<td>88M</td>
</tr>
<tr>
<td>inception (4c)</td>
<td>14×14×512</td>
<td>2</td>
<td>128</td>
<td>128</td>
<td>256</td>
<td>24</td>
<td>64</td>
<td>463K</td>
<td>100M</td>
</tr>
<tr>
<td>inception (4d)</td>
<td>14×14×528</td>
<td>2</td>
<td>112</td>
<td>144</td>
<td>288</td>
<td>32</td>
<td>64</td>
<td>580K</td>
<td>119M</td>
</tr>
<tr>
<td>inception (4e)</td>
<td>14×14×832</td>
<td>2</td>
<td>256</td>
<td>160</td>
<td>320</td>
<td>32</td>
<td>128</td>
<td>840K</td>
<td>170M</td>
</tr>
<tr>
<td>max pool</td>
<td>3×3/2</td>
<td>7×7×832</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>inception (5a)</td>
<td>7×7×832</td>
<td>2</td>
<td>256</td>
<td>160</td>
<td>320</td>
<td>32</td>
<td>128</td>
<td>1072K</td>
<td>54M</td>
</tr>
<tr>
<td>inception (5b)</td>
<td>7×7×1024</td>
<td>2</td>
<td>384</td>
<td>192</td>
<td>384</td>
<td>48</td>
<td>128</td>
<td>1388K</td>
<td>71M</td>
</tr>
<tr>
<td>avg pool</td>
<td>7×7/1</td>
<td>1×1×1024</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>dropout (40%)</td>
<td>1×1×1024</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>linear</td>
<td>1×1×1000</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1000K</td>
<td>1M</td>
</tr>
<tr>
<td>softmax</td>
<td>1×1×1000</td>
<td>0</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

*C. Szegedy et al, Going Deeper With Convolutions, CVPR 2015*
Google LeNet

- Has 5 Million or 12X fewer parameters than AlexNet
- Gets rid of fully connected layers
Inception v2, v3

- Use Batch Normalization during training to reduce dependence on auxiliary classifiers
- More aggressive factorization of filters

*C. Szegedy et al, Rethinking the Inception Architecture for Computer Vision, CVPR 2016*
Why do CNNs make sense? (Brain Stuff next time)
Convolutions: Motivation

Convolution leverages four ideas that can help ML systems:

- Sparse interactions
- Parameter sharing
- Equivariant representations
- Ability to work with inputs of variable size

Sparse Interactions

- Plain Vanilla NN ($y \in \mathbb{R}^n, x \in \mathbb{R}^m$): Need matrix multiplication $y = Wx$ to compute activations for each layer (every output interacts with every input)
- Convolutional networks have sparse interactions by making kernel smaller than input
  - $\implies$ need to store fewer parameters, computing output needs fewer operations ($O(m \times n)$ versus $O(k \times n)$)
Motivation: Sparse Connectivity

- Fully connected network: $h_3$ is computed by full matrix multiplication with no sparse connectivity
Motivation: Sparse Connectivity

Kernel of size 3, moved with stride of 1

$h_3$ only depends on $x_2, x_3, x_4$
Connections in CNNs are sparse, but units in deeper layers are connected to all of the input (larger receptive field sizes)
Motivation: Parameter Sharing

- Plain vanilla NN: Each element of $W$ is used exactly once to compute output of a layer.
- In convolutional networks, parameters are tied: weight applied to one input is tied to value of a weight applied elsewhere.
- Same kernel is used throughout the image, so instead learning a parameter for each location, only a set of parameters is learnt.
- Forward propagation remains unchanged $O(k \times n)$.
- Storage improves dramatically as $k \ll m, n$. 

Lecture 7 Convolutional Neural Networks
Motivation: Equivariance

- Let’s first formally define convolution:

\[ s(t) = (x \ast w)(t) = \int x(a)w(t - a)da \]

- In Convolutional Network terminology \( x \) is referred to as the **input**, \( w \) as the **kernel** and \( s \) as the **feature map**

- Discrete Convolution:

\[ S(i, j) = (I \ast K)(i, j) = \sum_{m} \sum_{n} I(m, n)K(i - m, j - n) \]

- Convolution is commutative, thus:

\[ S(i, j) = (I \ast K)(i, j) = \sum_{m} \sum_{n} I(i - m, j - n)K(m, n) \]
Aside

- The latter is usually more straightforward to implement in ML libraries (less variation in range of valid values of $m$ and $n$).
- Neither are usually used in practice in Neural Networks.
- Libraries implement *Cross Correlation*, same as convolution, but without flipping the kernel.

$$S(i, j) = (I \ast K)(i, j) = \sum_m \sum_n I(i + m, j + n)K(m, n)$$
Motivation: Equivariance

- **Equivariance:** $f$ is equivariant to $g$ if $f(g(x)) = g(f(x))$
- The form of parameter sharing used by CNNs causes each layer to be equivariant to translation
- That is, if $g$ is any function that translates the input, the convolution function is equivariant to $g$
Motivation: Equivariance

- Implication: While processing time series data, convolution produces a timeline that shows when different features appeared (if an event is shifted in time in the input, the same representation will appear in the output).
- Images: If we move an object in the image, its representation will move the same amount in the output.
- This property is useful when we know some local function is useful everywhere (e.g., edge detectors).
- Convolution is not equivariant to other operations such as change in scale or rotation.
Pooling: Motivation

- Pooling helps the representation become slightly *invariant* to small translations of the input.
- Reminder: Invariance: $f(g(x)) = f(x)$
- If input is translated by small amount: values of most pooled outputs don’t change.
Pooling: Invariance

Figure: Goodfellow et al.
Pooling

- Invariance to local translation can be useful if we care more about whether a certain feature is present rather than exactly where it is.
- Pooling over spatial regions produces invariance to translation, what if we pool over separately parameterized convolutions?
- Features can learn which transformations to become invariant to (Example: Maxout Networks, Goodfellow et al. 2013)
- One more advantage: Since pooling is used for downsampling, it can be used to handle inputs of varying sizes.
Next time

- More Architectures
- Variants on the CNN idea
- More motivation
- Group Equivariance
- Equivariance to Rotation
Quiz!