Recap

Filters/Kernels and Stride

Learning filters

CNNs are partially connected networks

Tensorflow conv2d function

Padding

Application to MNIST/CIFAR
Today’s goal – continue to learn about CNNs

(1) Convolutional Neural Network (CNN) architecture

(2) First successful CNN - AlexNet
   - Pooling and translational invariance

(3) Deeper CNNs!
   - Residual Blocks
   - Batch normalization
Bias Term in Convolution Layers

Just like a fully connected layer, we can have a learnable additive bias for convolution.
Adding a Bias in Tensorflow

If you use tf.nn.conv2d, bias can be added with:

```
tf.nn.bias_add(value, bias)
```

Conv2D output Bias variable to add
e.g. `tf.Variable(tf.random.normal([16]))` for a conv2d output with 16 channels

Full documentation here: [https://www.tensorflow.org/api_docs/python/tf/nn/bias_add](https://www.tensorflow.org/api_docs/python/tf/nn/bias_add)
Adding a Bias in Tensorflow

If you are using keras layers, bias is included by default:

```
import tensorflow as tf

tf.keras.layers.Conv2D(filters, kernel_sz, strides, padding, use_bias = True)
```

Full documentation here: https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/keras/layers/Conv2D
Our neural network so far
Convolutional Neural Network Architecture
CNN Architecture
CNN Architecture

Image description: CNN architecture diagram with layers and operations including ReLU, pooling, linear layer, and softmax.
CNN Architecture
CNN Architecture

This part learns to extract features from the image.
CNN Architecture

A single convolutional layer
CNN Architecture

Activation after filter passes over image
CNN Architecture

This part learns to perform a specific task (e.g. classification) using those features
CNN Architecture

Flattened data (beginning of the fully connected portion)

linear layer softmax

output
CNN Architecture

Fully connected layers to classify input

<table>
<thead>
<tr>
<th>Linear layer</th>
<th>Softmax</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>∑</td>
<td>σ</td>
<td></td>
</tr>
</tbody>
</table>
CNN Architecture

Why multiple convolutional layers?

Input

Label=“Llama”
Feature Extraction using multiple convolution layers

Hierarchy of features
Sequence of layers detect broader and broader features
## Example: Network Dissection

<table>
<thead>
<tr>
<th>Layer 3 active regions</th>
<th>Layer 4 active regions</th>
<th>Layer 5 active regions</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1" alt="Layer 3" /></td>
<td><img src="image2" alt="Layer 4" /></td>
<td><img src="image3" alt="Layer 5" /></td>
</tr>
<tr>
<td>&quot;Eye Detector&quot;</td>
<td>&quot;Eyes and Nose Detector&quot;</td>
<td>&quot;Dog Face Detector&quot;</td>
</tr>
</tbody>
</table>

http://netdissect.csail.mit.edu/

Any questions?
ILSVRC 2012
(ImageNet Large Scale Visual Recognition Challenge)

The classification task on ImageNet:

For each image, assign 5 labels in order of decreasing confidence. Success if one of these labels matches the ground truth.

Predictions:

1. Carpet
2. Zebra
3. Llama
4. Flower
5. Horse

https://commons.wikimedia.org/wiki/File:Common_zebra_1.jpg
ILSVRC 2012

Percentage that model fails to classify is known as **Top 5 Error Rate**

![Puffer Fish](https://commons.wikimedia.org/wiki/File:Puffer_Fish_DSC01257.JPG)

**Predictions:**

1. Sponge 
2. Person 
3. Llama 
4. Flower 
5. Boat
AlexNet: Why CNNs Are a Big Deal

Major performance boost on ImageNet at ILSRCV 2012
Top 5 error rate of 15.3% compared to 26.2% achieved by 2nd place

Note: SuperVision is the name of Alex's team

http://image-net.org/challenges/LSVRC/2012/analysis/
AlexNet

- 60 million parameters
- 5 Convolutional Layers
- 3 Fully Connected Layers

[Alex Krizhevsky et al. 2012]

Pooling
Max Pooling

Max pooling with stride 2 and 2x2 filters

Max of pixels in window
Max Pooling

Max pooling with stride 2 and 2x2 filters

Max of pixels in window

Max of pixels in window
Max Pooling

Max pooling with stride 2 and 2x2 filters

Max of pixels in window

<table>
<thead>
<tr>
<th>6</th>
<th>3</th>
<th>1</th>
<th>-3</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Max Pooling

Max pooling with stride 2 and 2x2 filters

Max of pixels in window
Max Pooling

Max pooling with stride 2 and 2x2 filters

<table>
<thead>
<tr>
<th>6</th>
<th>3</th>
<th>1</th>
<th>-3</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>1</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td></td>
<td>3</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Max of pixels in window

<table>
<thead>
<tr>
<th>6</th>
<th>2</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>3</td>
</tr>
</tbody>
</table>
Max Pooling

Max pooling with stride 2 and 2x2 filters

Why use Max Pooling?
Pooling: Motivation

Max Pooling

- Keeps track of regions with highest activations, indicating object presence
- Controllable way to lower (coarser) resolution (down sample the convolution output)
Other Pooling Techniques

Average pooling with stride 2 and 2x2 filters

Average pixel values in each window
Learning a Pooling Function

- The network can learn its own pooling function
- Implement via a strided convolution layer

Learned filter weights
So...did we achieve our goal of translational invariance?
What was Translational Invariance again?

• To make a neural net $f$ robust in this same way, it should ideally satisfy \textit{translational invariance}: $f(T(x)) = f(x)$, where
  • $x$ is the input image
  • $T$ is a translation (i.e. a horizontal and/or vertical shift)

$$f\left(\begin{array}{c} \text{input image} \\ \end{array}\right) \Rightarrow f(\begin{array}{c} \text{shifted image} \\ \end{array})$$
Are CNNs Translation Invariant?

- Convolution is *translation equivariant*
  - A translated input results in an output translated by the same amount
  
  \[ f(T(I)) = T(f(I)) \]

- \((T(I) \otimes K)(x, y) = T(I \otimes K)(x, y)\)

\[
\begin{align*}
\begin{array}{c}
\text{f (\includegraphics[width=2cm]{zebra.png})} \\
\downarrow T
\end{array}
&= \\
\begin{array}{c}
\text{f (\includegraphics[width=2cm]{zebra.png})} \\
\downarrow T
\end{array}
\end{align*}
\]

*Here, \((I \otimes K)(x, y) = \sum_{m} \sum_{n} I(x + m, y + n)K(m, n)\)*
Are CNNs Translation Invariant?

• Max pooling is intended to give invariance to small translations
  • The highest activation pixel can shift around within the pooling window, and the output does not change

\[
\begin{align*}
f(\begin{pmatrix} 6 & 3 \\ 4 & 1 \end{pmatrix}) &= 6 \\
f(\begin{pmatrix} 1 & 5 \\ 6 & 3 \end{pmatrix}) &= 6 \\
f(\begin{pmatrix} 2 & 6 \\ 2 & 4 \end{pmatrix}) &= 6
\end{align*}
\]
So how does it all come together?

Convolution is **translation equivariant**

Max pooling gives invariance to small translations
Are CNNs Translation Invariant?

- Answer: CNNs are *sort of* translation invariant
  - Shifting the content of the image around tends not to drastically effect the output classification probabilities...

Are CNNs Translation Invariant?

- Answer: CNNs are “sort of” translation invariant
  - Shifting the content of the image around tends not to drastically effect the output classification probabilities...
Are CNNs Translation Invariant?

- Answer: CNNs are “sort of” translation invariant
  - Shifting the content of the image around tends not to drastically effect the output classification probabilities...
  - ...but they are not, strictly speaking, translation invariant

Are CNNs Translation Invariant?

- Is it possible to build a truly translation invariant CNN?
  - Yes!
  - Have to properly “pre-filter” images before pooling them
  - Comes from signal processing theory (The Sampling Theorem)
  - Take CS 1230 (Computer Graphics) if you want to learn about this!

Other Invariances

Rotation/Viewpoint Invariance
Other Invariances

Rotation/Viewpoint Invariance

Size Invariance
Other Invariances

• All of these are desirable
• How do CNNs fare?
  • Max pooling gives some small amount of size invariance...
  • ...but in general, CNNs don’t do well with big changes in size, pose, or lighting
• Consequence of not having these invariances?
  • Need **lots** of training data
  • Have to show the network examples of everything under different poses, lighting, etc.
  • Data Augmentation
More Complicated Networks

AlexNet:

VGG:

224x224x3 → 112x112x128 → 56x56x256 → 28x28x512 → 14x14x512 → 7x7x512 → 1x4096 → 1x1000 → output
Can you guess what was the biggest bottleneck to adding more layers?
Vanishing gradients!

Revolution of Depth

Vanishing Gradient

Deep Layers

Somewhere in the middle

Initial Layers

ILSVRC'14 VGG

ILSVRC'13

ILSVRC'12 AlexNet

ILSVRC'11

ILSVRC'10

eNet Classification top-5 error (%)

19 layers 7.3
8 layers 11.7
8 layers 16.4
shallow 25.8

28.2

More Complicated Networks

ResNet:
Lots of layers, tons of learnable parameters
Avoids Vanishing Gradient problem
but how?

More Complicated Networks

ResNet:

Lots of layers, tons of learnable parameters
Avoids Vanishing Gradient problem

Residual Blocks

- In very deep nets, each layer often needs to learn just a small transformation of the preceding layer (identity + change)
- Idea: explicitly design the network such that the output of each layer is the identity + some deviation from it
  - Deviation is known as a residual
Residual Blocks

- In very deep nets, each layer often needs to learn just a small transformation of the preceding layer (identity + change)
- Idea: explicitly design the network such that the output of each layer is the identity + some deviation from it
  - Deviation is known as a residual
- Allows gradient to flow through two pathways
- **Significantly stabilizes training of very deep networks**

Batch Normalization (stabilizing training)

Idea: normalize the activations for each feature at each layer

Why might we want to do this?
Batch Normalization: Motivation

More stable inputs = faster training

MNIST test accuracy vs number of training steps

Batch Normalization: Implementation

For each feature $x$, Start by calculating the batch mean and standard deviation for each feature:

$$\mu_{\text{batch}} = \frac{\sum_{i=0}^{\text{batch\_size}} x_i}{\text{batch\_size}}$$

$$\sigma_{\text{batch}} = \sqrt{\frac{\sum_{i=0}^{\text{batch\_size}} (x_i - \mu_{\text{batch}})^2}{\text{batch\_size}}}$$
Batch Normalization: Implementation

Normalize by subtracting feature x’s batch mean, then divide by batch standard deviation.

\[ x' = \frac{x - \mu_{\text{batch}}}{\sigma_{\text{batch}}} \]

Feature x now has mean 0 and variance 1 along the batch
Batch Normalization in Tensorflow

```
tf.keras.layers.BatchNormalization(input)
```

Documentation:  
https://www.tensorflow.org/versions/r2.0/api_docs/python/tf/keras/layers/BatchNormalization
Recap

CNNs

Architecture

AlexNet + Pooling

CNNs are “sort of” translationally invariant

Many layers = vanishing gradient

Deeper CNNs

ResNet + Residual blocks

Batch normalization

Revolution of Depth