Topics in AI (CPSC 532S):
Multimodal Learning with Vision, Language and Sound

Lecture 6: Convolutional Neural Networks (Part 3)
Logistics:

Assignment 2 is due on Monday
Computer Vision Problems

Categorization

- Multi-class:
  - Horse
  - Church
  - Toothbrush
  - Person

Detection

- Horse \((x, y, w, h)\)
- Horse \((x, y, w, h)\)
- Person \((x, y, w, h)\)
- Person \((x, y, w, h)\)

Segmentation

- Horse
- Person

Instance Segmentation

- Horse1
- Horse2
- Person1
- Person2

IMAGENET

- Multi-label:
  - Horse
  - Church
  - Toothbrush
  - Person
Computer Vision Problems (no language for now)

Categorization

Multi-class: Horse Church Toothbrush Person

IMAGENET
ILSVRC winner 2012

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
ILSVRC winner 2012

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
ILSVRC winner 2012

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
even deeper network with **computational efficiency**

- 22 layers
- Efficient “Inception” module
- No FC layers
- Only 5 million parameters! (12x less than AlexNet!)
- Better performance (@6.7 top 5 error)
GoogleLeNet: Inception Module

Idea: design good local topology ("network within network") and then stack these modules

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford

[ Szegedy et al., 2014 ]
**GoogleLeNet: Inception Module**

**Idea:** design good local topology (“network within network”) and then stack these modules.

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 at output depth-wise.

*slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford*
Idea: design good local topology ("network within network") and then stack these modules

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 at output depth-wise
**GoogleLeNet: Inception Module**

**Idea:** design good local topology (“network within network”) and then stack these modules

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 at output depth-wise

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, *cs231n Stanford*
**Idea:** design good local topology (“network within network”) and then stack these modules

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 at output depth-wise

![Diagram of Naive Inception module]

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, [cs231n Stanford](https://cs231n.stanford.edu)
**GoogleLeNet: Inception Module**

**Idea:** design good local topology (“network within network”) and then stack these modules

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 at output depth-wise

---

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
**Convolutional Layer: 1x1 convolutions**

56 x 56 x 64 image

32 filters of size, 1 x 1 x 64

56 x 56 x 32 image

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
**Idea:** design good local topology ("network within network") and then stack these modules

---

1x1 "bottleneck" layers

Naive Inception module

Inception module with dimension reduction

saves approximately 60% of computations

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
GoogleLeNet

even deeper network with **computational efficiency**

- 22 layers
- Efficient “Inception” module
- No FC layers
- Only 5 million parameters! (12x less than AlexNet!)
- Better performance (@6.7 top 5 error)

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford

[ Szegedy et al., 2014 ]
GoogleLeNet

even deeper network with **computational efficiency**

- 22 layers
- Efficient “Inception” module
- No FC layers
- Only 5 million parameters! (12x less than AlexNet!)
- Better performance (@6.7 top 5 error)
even deeper network with computational efficiency

- 22 layers
- Efficient “Inception” module
- No FC layers
- Only 5 million parameters! (12x less than AlexNet!)
- Better performance (@6.7 top 5 error)

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford

[ Szegedy et al., 2014 ]
GoogleLeNet

even deeper network with **computational efficiency**

- 22 layers
- Efficient “Inception” module
- No FC layers
- Only 5 million parameters! (12x less than AlexNet!)
- Better performance (@6.7 top 5 error)
Optimizing Deep Neural Networks

Consider multi-layer neural network with sigmoid activations and loss $C$

Source: http://neuralnetworksanddeeplearning.com/chap5.html
Optimizing **Deep** Neural Networks

\[
\frac{\partial C}{\partial b_1} = \sigma'(z_1) \times w_2 \times \sigma'(z_2) \times w_3 \times \sigma'(z_3) \times w_4 \times \sigma'(z_4) \times \frac{\partial C}{\partial a_4}
\]

Source: http://neuralnetworksanddeeplearning.com/chap5.html
Optimizing **Deep Neural Networks**

Expression for **gradient** of bias in **Layer 1**: \[
\frac{\partial C}{\partial b_1} = \sigma'(z_1)w_2\sigma'(z_2)w_3\sigma'(z_3)w_4\sigma'(z_4) \frac{\partial C}{\partial a_4}
\]

Expression for **gradient** of bias in **Layer 3**: \[
\frac{\partial C}{\partial b_3} = \sigma'(z_3)w_4\sigma'(z_4) \frac{\partial C}{\partial a_4}
\]

Optimizing Deep Neural Networks

Expression for gradient of bias in Layer 1: \[
\frac{\partial C}{\partial b_1} = \sigma'(z_1) w_2 \sigma'(z_2) w_3 \sigma'(z_3) w_4 \sigma'(z_4) \frac{\partial C}{\partial a_4}
\]

Expression for gradient of bias in Layer 3: \[
\frac{\partial C}{\partial b_3} = \sigma'(z_3) w_4 \sigma'(z_4) \frac{\partial C}{\partial a_4}
\]

Source: http://neuralnetworksanddeeplearning.com/chap5.html
Optimizing Deep Neural Networks

\[
\frac{\partial C}{\partial b_1} = \sigma'(z_1) \times w_2 \times \sigma'(z_2) \times w_3 \times \sigma'(z_3) \times w_4 \times \sigma'(z_4) \times \frac{\partial C}{\partial a_4}
\]

Observations:
- $|\text{weight}| < 1$ (due to initialization)
- max of derivative of sigmoid = $1/4$ @ 0

Source: http://neuralnetworksanddeeplearning.com/chap5.html
Optimizing Deep Neural Networks

Observations:
|weight| < 1 (due to initialization)
max of derivative of sigmoid = 1/4 @ 0

Source: http://neuralnetworksanddeeplearning.com/chap5.html
Optimizing Deep Neural Networks

This is called **vanishing gradient** problem

– makes deep networks hard to train
– later layers learn faster than earlier ones

\[
\frac{\partial C}{\partial b_1} = \sigma'(z_1) \times w_2 \times \sigma'(z_2) \times w_3 \times \sigma'(z_3) \times w_4 \times \sigma'(z_4) \times \frac{\partial C}{\partial a_4}
\]

\[
\frac{\partial C}{\partial b_1} = \sigma'(z_1) \underbrace{\times w_2 \sigma'(z_2)}_{< \frac{1}{4}} \underbrace{\times w_3 \sigma'(z_3)}_{< \frac{1}{4}} \times w_4 \sigma'(z_4) \frac{\partial C}{\partial a_4}
\]

\[
\frac{\partial C}{\partial b_3} = \sigma'(z_3) w_4 \sigma'(z_4) \frac{\partial C}{\partial a_4}
\]

Source: http://neuralnetworksanddeeplearning.com/chap5.html
Exploding gradient problem

- makes weights large (e.g., 100)
- make bias such that pre-activation = 0

Source: http://neuralnetworksanddeeplearning.com/chap5.html
even deeper network with **computational efficiency**

- 22 layers
- Efficient “Inception” module
- No FC layers
- Only 5 million parameters! (12x less than AlexNet!)
- Better performance (@6.7 top 5 error)
ILSVRC winner 2012

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
ResNet

even deeper — **152 layers**!

using residual connections

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, *cs231n* Stanford
ResNet: Motivation

What happens when we continue to stacking deeper layers on a “plain” CNN

[ He et al., 2015 ]
ResNet: Motivation

What happens when we continue to stacking deeper layers on a “plain” CNN

What's the problem?

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
ResNet: Motivation

What happens when we continue to stacking deeper layers on a “plain” CNN

What's the problem?

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
ResNet: Motivation

**Hypothesis:** deeper models are harder to optimize (optimization problem)
ResNet: Motivation

Hypothesis: deeper models are harder to optimize (optimization problem)

Observation: the deeper model should (conceptually) perform just as well (e.g., take shallower model and use identity for all remaining layers)
**ResNet: Motivation**

**Hypothesis:** deeper models are harder to optimize (optimization problem)

**Observation:** the deeper model should (conceptually) perform just as well (e.g., take shallower model and use identity for all remaining layers)

How do we implement this idea in practice
**Solution:** use network to fit residual mapping instead of directly trying to fit a desired underlying mapping

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

Use layers to fit residual \[ F(x) = H(x) - X \] instead of \( H(x) \) directly.

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
ResNet

Full details

— Stacked **residual blocks**
— Every residual block consists of **two 3x3 filters**
— Periodically double # of filters and downsample spatially using stride of 2
— Additional convolutional layer in the beginning
— **No FC layers** at the end (only FC to output 1000 classes)
ILSVRC winner 2012

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
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 Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
Regularization: Stochastic Depth

Effectively “dropout” but for layers

Stochastically with some probability turn off some layer (for each batch)

Effectively trains a collection of neural networks

* adopted from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
ResNet: A little theory

One can view a sequence of outputs from residual layers as a **Dynamical System**

[Cheng et al., ICLR 2018]
ResNet: A little theory

One can view a sequence of outputs from residual layers as a **Dynamical System**

$$Y_{j+1} = Y_j + G(Y_j, \theta_j)$$

[ Cheng et al., ICLR 2018 ]
ResNet: A little theory

One can view a sequence of outputs from residual layers as a Dynamical System.

What happens if you take more layers and take smaller steps?

[Chen et al., NIPS 2018 best paper]
ResNet: A little theory

One can view a sequence of outputs from residual layers as a **Dynamical System**

What happens if you take more layers and take smaller steps?

You can actually treat a neural network as an **ODE**: 

$$ \frac{dh(t)}{dt} = f(h(t), t, \theta) $$

[ Chen et al., NIPS 2018 best paper ]
Comparing Complexity


* adopted from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
Computer Vision Problems

**Categorization**

- **Multi-class:**
  - Horse
  - Church
  - Toothbrush
  - Person

- **Multi-label:**
  - Horse
  - Church
  - Toothbrush
  - Person

**Detection**

- **Horse (x, y, w, h)**
- **Horse (x, y, w, h)**
- **Person (x, y, w, h)**
- **Person (x, y, w, h)**

**Segmentation**

- **Horse**
- **Person**

**Instance Segmentation**

- **Horse1**
- **Horse2**
- **Person1**
- **Person2**
Computer Vision Problems (no language for now)

Segmentation

Horse
Person
Semantic **Segmentation**

Label **every pixel** with a category label (without differentiating instances)

*slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford*
Semantic **Segmentation**: Sliding Window

Extract patches

Classify center pixel with CNN

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford

[ Farabet et al, TPAMI 2013 ]
[ Pinheiro et al, ICML 2014 ]
Semantic **Segmentation**: Sliding Window

**Problem:** VERY inefficient, no reuse of computations for overlapping patches

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
Semantic **Segmentation**: Fully Convolutional CNNs

Design a network as a number of convolutional layers to make predictions for all pixels at once!

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
Semantic **Segmentation**: Fully Convolutional CNNs

Design a network as a number of convolutional layers to make predictions for all pixels at once!

![Diagram](image)

**Input Image**  
$3 \times H \times W$

**Convolutions**  
$D \times H \times W$

**Class Scores**  
$C \times H \times W$

**Predicted Labels**  
$H \times W$

**Problem:** Convolutions at the original image scale will be very expensive

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
Semantic **Segmentation:** Fully Convolutional CNNs

Design a network as a number of convolutional layers with **downsampling** and **upsampling** inside the network!

Input **Image**

3 x H x W

High-res:

D₁ x H/2 x W/2

Med-res:

D₂ x H/4 x W/4

Med-res:

D₂ x H/4 x W/4

Low-res:

D₃ x H/4 x W/4

High-res:

D₁ x H/2 x W/2

Predicted **Labels**

H x W

[ Long et al, CVPR 2015 ]
[ Noh et al, ICCV 2015 ]

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**
Semantic **Segmentation**: Fully Convolutional CNNs

Design a network as a number of convolutional layers with **downsampling** and **upsampling** inside the network!

**Downsampling** = Pooling  
**Upsampling** = ???

Input **Image**  
$3 \times H \times W$  

**Predicted Labels**  
$H \times W$

- High-res: $D_1 \times H/2 \times W/2$
- Med-res: $D_2 \times H/4 \times W/4$
- Low-res: $D_3 \times H/4 \times W/4$

[ Long et al, CVPR 2015 ]  
[ Noh et al, ICCV 2015 ]
In-network **Up Sampling** (a.k.a “Unpooling”)

Nearest Neighbor

\[
\begin{array}{ll|ll}
1 & 2 & 1 & 1 \\
3 & 4 & 3 & 3 \\
\end{array}
\begin{array}{ll|ll}
& & 2 & 2 \\
& & 4 & 4 \\
\end{array}
\]

**Input:** 2 x 2  \hspace{1cm} **Output:** 4 x 4

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
In-network **Up Sampling** (a.k.a “Unpooling”)

Nearest Neighbor

Input: 2 x 2

<table>
<thead>
<tr>
<th>1 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 4</td>
</tr>
</tbody>
</table>

Output: 4 x 4

<table>
<thead>
<tr>
<th>1 1 2 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 3 4 4</td>
</tr>
</tbody>
</table>

“Bed of Nails”

Input: 2 x 2

<table>
<thead>
<tr>
<th>1 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>3 4</td>
</tr>
</tbody>
</table>

Output: 4 x 4

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

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
In-network **Up Sampling:** Max Unpooling

Max Pooling
Remember which element was max!

Max Unpooling
Use positions from pooling layer

```
Input: 4 x 4
1 2 3 6
3 5 2 1
7 3 4 8

Output: 2 x 2
5 6
7 8

Rest of the network

Input: 2 x 2
0 0
0 1
0 0
0 0
0 0

Output: 4 x 4
1 2
0 1
3 0
0 4
```

Corresponding pairs of downsampling and upsampling layers

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
In-network **Up Sampling**: Transpose Convolution

**Recall:** Normal 3 x 3 convolution, stride 1 pad 1

Input: 4 x 4

Dot product between filter and input

Output: 4 x 4

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
In-network **Up Sampling**: Transpose Convolution

**Recall**: Normal 3 x 3 convolution, stride 1 pad 1

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
In-network **Up Sampling**: Transpose Convolution

**Recall**: Normal 3 x 3 convolution, stride 2 pad 1

![Diagram showing Up Sampling process](image)

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
**In-network Up Sampling:** Transpose Convolution

**Recall:** Normal 3 x 3 convolution, stride 2 pad 1

- **Input:** 4 x 4
- **Output:** 2 x 2
- Dot product between filter and input
- **Filter moves 2 pixels in the input for every one pixel in the output**
- **Stride gives ratio in movement in input vs output**

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
In-network **Up Sampling:** Transpose Convolution

3 x 3 transpose convolution, stride 2 pad 1

**Input:** 2 x 2  
**Output:** 4 x 4

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, *cs231n Stanford*
In-network **Up Sampling**: Transpose Convolution

3 x 3 transpose convolution, stride 2 pad 1

**Input**: 2 x 2  
**Output**: 4 x 4

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
In-network **Up Sampling:** Transpose Convolution

3 x 3 **transpose** convolution, stride 2 pad 1

Input: 2 x 2

Output: 4 x 4

Filter moves 2 pixels in the output for every one pixel in the input

Stride gives ratio in movement in output vs input

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, **cs231n Stanford**
Output contains copies of the filter weighted multiplied by the input, summing at overlaps in the output.

* slide from Fei-Dei Li, Justin Johnson, Serena Yeung, cs231n Stanford
U-Net Architecture

ResNet-like Fully convolutional CNN

[ Ronneberger et al, CVPR 2015 ]
Computer **Vision Problems** (no language for now)

**Categorization**
- **Multi-class:** Horse, Church, Toothbrush, Person

**Detection**
- Horse \((x, y, w, h)\)
- Horse \((x, y, w, h)\)
- Person \((x, y, w, h)\)
- Person \((x, y, w, h)\)

**Segmentation**
- Horse
- Person

**Instance Segmentation**
- Horse1
- Horse2
- Person1
- Person2

**IMAGENET**

**Multi-label:** Horse, Church, Toothbrush, Person