Lecture 11: Detection and Segmentation
Administrative

1. Midterms being graded
   - Please don’t discuss midterms until next week - some students not yet taken

2. A2 being graded

3. Project milestones due Wed 5/16
Last Time: Recurrent Networks
Last Time: Recurrent Networks

PANDURAS:
Alas, I think he shall be come approached and the day
When little brain would be attain'd into being never fed,
And who is but a chain and subjects of his death,
I should not sleep.

Second Senator:
They are away this miseries, produced upon my soul,
Breaking and strongly should be buried, when I perish
The earth and thoughts of many states.

DUKE VINCENZIO:
Well, your wit is in the care of side and that.

Second Lord:
They would be ruled after this chamber, and
My fair nues began out of the fact, to be conveyed,
Whose noble souls I'll have the heart of the wars.

Clown:
Come, sir, I will make did behold your worship.

VIOLA:
I'll drink it.
Last Time: Recurrent Networks

A cat sitting on a suitcase on the floor

A cat is sitting on a tree branch

A woman is holding a cat in her hand

Two people walking on the beach with surfboards

A tennis player in action on the court

A person holding a computer mouse on a desk

So far: Image Classification

Class Scores
Cat: 0.9
Dog: 0.05
Car: 0.01
...

Fully-Connected: 4096 to 1000

Vector: 4096

This image is CC0 public domain

Figure copyright Alex Krizhevsky, Ilya Sutskever, and Geoffrey Hinton, 2012. Reproduced with permission.
Today: Detection, Segmentation
Other Computer Vision Tasks

- **Semantic Segmentation**
  - **GRASS, CAT, TREE, SKY**
  - No objects, just pixels

- **Classification + Localization**
  - **CAT**
  - Single Object

- **Object Detection**
  - **DOG, DOG, CAT**
  - Multiple Object

- **Instance Segmentation**
  - **DOG, DOG, CAT**
  - Multiple Object
Other Computer Vision Tasks

- **Semantic Segmentation**
  - GRASS, CAT, TREE, SKY
  - No objects, just pixels

- **2D Object Detection**
  - DOG, DOG, CAT
  - Object categories + 2D bounding boxes

- **3D Object Detection**
  - Car
  - Object categories + 3D bounding boxes
Semantic Segmentation

Semantic Segmentation

GRASS, CAT, TREE, SKY
No objects, just pixels

2D Object Detection

DOG, DOG, CAT
Object categories + 2D bounding boxes

3D Object Detection

Car
Object categories + 3D bounding boxes

This image is CC0 public domain
Semantic Segmentation

Label each pixel in the image with a category label

Don’t differentiate instances, only care about pixels
Semantic Segmentation Idea: Sliding Window

- Full image
- Extract patch
- Classify center pixel with CNN

Cow
Cow
Grass

Farabet et al, "Learning Hierarchical Features for Scene Labeling," TPAMI 2013
Pinheiro and Collobert, "Recurrent Convolutional Neural Networks for Scene Labeling", ICML 2014
Semantic Segmentation Idea: Sliding Window

Full image

Extract patch

Classify center pixel with CNN

Cow

Cow

Grass

Problem: Very inefficient! Not reusing shared features between overlapping patches

Pinheiro and Collobert, “Recurrent Convolutional Neural Networks for Scene Labeling”, ICML 2014
Semantic Segmentation Idea: Fully Convolutional

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

Input: $3 \times H \times W$

Convolutions: $D \times H \times W$

Scores: $C \times H \times W$

Predictions: $H \times W$
Semantic Segmentation Idea: Fully Convolutional

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

Input: $3 \times H \times W$

Convolutions: $D \times H \times W$

Scores: $C \times H \times W$

Predictions: $H \times W$

Problem: convolutions at original image resolution will be very expensive ...
Semantic Segmentation Idea: Fully Convolutional

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

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

**High-res:**
$$D_1 \times H/2 \times W/2$$

**Med-res:**
$$D_2 \times H/4 \times W/4$$

**Low-res:**
$$D_3 \times H/4 \times W/4$$

**High-res:**
$$D_1 \times H/2 \times W/2$$

**Predictions:**
$$H \times W$$

---

Semantic Segmentation Idea: Fully Convolutional

**Downsampling:**
Pooling, strided convolution

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

- **Input:** $3 \times H \times W$
- **High-res:** $D_1 \times H/2 \times W/2$
- **Low-res:** $D_3 \times H/4 \times W/4$
- **Med-res:** $D_2 \times H/4 \times W/4$
- **Med-res:** $D_2 \times H/4 \times W/4$
- **High-res:** $D_1 \times H/2 \times W/2$
- **Predictions:** $H \times W$


In-Network upsampling: “Unpooling”

**Nearest Neighbor**

Input: 2 x 2  

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

Output: 4 x 4  

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

**“Bed of Nails”**

Input: 2 x 2  

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

Output: 4 x 4  

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>4</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
In-Network upsampling: “Max Unpooling”

Max Pooling
Remember which element was max!

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>6</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>5</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>4</td>
<td>8</td>
</tr>
</tbody>
</table>

Input: 4 x 4

Output: 2 x 2

Max Unpooling
Use positions from pooling layer

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

Input: 2 x 2

Output: 4 x 4

Rest of the network

Corresponding pairs of downsampling and upsampling layers
Learnable Upsampling: Transpose Convolution

Recall: Typical 3 x 3 convolution, stride 1 pad 1

Input: 4 x 4

Output: 4 x 4
Learnable Upsampling: 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
**Learnable Upsampling: Transpose Convolution**

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

Input: 4 x 4

Output: 4 x 4

Dot product between filter and input
Learnable Upsampling: Transpose Convolution

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

Input: 4 x 4

Output: 2 x 2
Learnable Upsampling: Transpose Convolution

Recall: Normal 3 x 3 convolution, stride 2 pad 1
Learnable Upsampling: Transpose Convolution

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

Input: 4 x 4

Dot product between filter and input

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

Stride gives ratio between movement in input and output

Output: 2 x 2
Learnable Upsampling: Transpose Convolution

3 x 3 transpose convolution, stride 2 pad 1

Input: 2 x 2  
Output: 4 x 4
Learnable Upsampling: Transpose Convolution

3 x 3 \texttt{transpose} convolution, stride 2 pad 1

Input: 2 x 2

Output: 4 x 4
Learnable Upsampling: Transpose Convolution

3 x 3 transpose convolution, stride 2 pad 1

Input gives weight for filter

Sum where output overlaps

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

Stride gives ratio between movement in output and input

Input: 2 x 2

Output: 4 x 4
Learnable Upsampling: Transpose Convolution

3 x 3 transpose convolution, stride 2 pad 1

- Input gives weight for filter
- Sum where output overlaps
- Filter moves 2 pixels in the output for every one pixel in the input
- Stride gives ratio between movement in output and input

Input: 2 x 2
Output: 4 x 4
Learnable Upsampling: Transpose Convolution

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

- Input gives weight for filter
- Sum where output overlaps
- Filter moves 2 pixels in the output for every one pixel in the input
- Stride gives ratio between movement in output and input

Other names:
- Deconvolution (bad)
- Upconvolution
- Fractionally strided convolution
- Backward strided convolution

**3 x 3 transpose convolution, stride 2 pad 1**
Learnable Upsampling: 1D Example

Output contains copies of the filter weighted by the input, summing at where at overlaps in the output.

Need to crop one pixel from output to make output exactly 2x input.
Convolution as Matrix Multiplication (1D Example)

We can express convolution in terms of a matrix multiplication

\[
\vec{x} \ast \vec{a} = X \vec{a}
\]

\[
\begin{bmatrix}
  x & y & x & 0 & 0 & 0 \\
  0 & x & y & x & 0 & 0 \\
  0 & 0 & x & y & x & 0 \\
  0 & 0 & 0 & x & y & x \\
  0 & 0 & 0 & 0 & x & y \\
\end{bmatrix}
\begin{bmatrix}
  0 \\
  a \\
  b \\
  c \\
  d \\
  0
\end{bmatrix}
= \begin{bmatrix}
  ay + bz \\
  ax + by + cz \\
  bx + cy + dz \\
  cx + dy
\end{bmatrix}
\]

Example: 1D conv, kernel size=3, stride=1, padding=1
Convolution as Matrix Multiplication (1D Example)

We can express convolution in terms of a matrix multiplication

\[ \tilde{x} \ast \tilde{a} = X \tilde{a} \]

Example: 1D conv, kernel size=3, stride=1, padding=1

\[
\begin{bmatrix}
    x & y & x & 0 & 0 & 0 \\
    0 & x & y & x & 0 & 0 \\
    0 & 0 & x & y & x & 0 \\
    0 & 0 & 0 & x & y & x \\
    0 & 0 & 0 & 0 & x & y \\
    0 & 0 & 0 & 0 & 0 & x
\end{bmatrix}
\begin{bmatrix}
    0 \\
    a \\
    b \\
    c \\
    d \\
    0
\end{bmatrix}
= 
\begin{bmatrix}
    ay + bz \\
    ax + by + cz \\
    bx + cy + dz \\
    cx + dy
\end{bmatrix}
\]

Convolution transpose multiplies by the transpose of the same matrix:

\[ \tilde{x} \ast^T \tilde{a} = X^T \tilde{a} \]

\[
\begin{bmatrix}
    x & 0 & 0 & 0 \\
    y & x & 0 & 0 \\
    z & y & x & 0 \\
    0 & z & y & x \\
    0 & 0 & z & y \\
    0 & 0 & 0 & z
\end{bmatrix}
\begin{bmatrix}
    a \\
    b \\
    c \\
    d
\end{bmatrix}
= 
\begin{bmatrix}
    ax \\
    ay + bx \\
    az + by + cx \\
    bz + cy + dx \\
    cx + dy \\
    dz
\end{bmatrix}
\]

Example: 1D conv, kernel size=3, stride=1, padding=1

When stride=1, convolution transpose is just a regular convolution (with different padding rules)
Convolution as Matrix Multiplication (1D Example)

We can express convolution in terms of a matrix multiplication

\[ \vec{x} \ast \vec{a} = X \vec{a} \]

\[
\begin{bmatrix}
  x & y & x & 0 & 0 & 0 \\
  0 & 0 & x & y & x & 0 \\
\end{bmatrix}
\begin{bmatrix}
  0 \\
  a \\
  b \\
  c \\
  d \\
  0
\end{bmatrix}
= \begin{bmatrix}
  ay + bz \\
  bx + cy + dz
\end{bmatrix}
\]

Example: 1D conv, kernel size=3, stride=2, padding=1
Convolution as Matrix Multiplication (1D Example)

We can express convolution in terms of a matrix multiplication

\[ \tilde{x} \ast \tilde{a} = X \tilde{a} \]

\[
\begin{bmatrix}
  x & y & z & 0 & 0 & 0 \\
  0 & 0 & x & y & z & 0 \\
\end{bmatrix}
\begin{bmatrix}
  0 \\
  a \\
  b \\
  c \\
  d \\
  0 \\
\end{bmatrix}
= \begin{bmatrix}
  ay + bz \\
  bx + cy + dz \\
\end{bmatrix}
\]

Example: 1D conv, kernel size=3, stride=2, padding=1

Convolution transpose multiplies by the transpose of the same matrix:

\[ \tilde{x} \ast^T \tilde{a} = X^T \tilde{a} \]

\[
\begin{bmatrix}
  x & 0 \\
  y & 0 \\
  z & x \\
  0 & y \\
  0 & z \\
  0 & 0 \\
\end{bmatrix}
\begin{bmatrix}
  a \\
  b \\
\end{bmatrix}
= \begin{bmatrix}
  ax \\
  ay \\
  az + bx \\
  by \\
  bz \\
  0 \\
\end{bmatrix}
\]

When stride>1, convolution transpose is no longer a normal convolution!
Semantic Segmentation Idea: Fully Convolutional

**Downsampling:**
Pooling, strided convolution

**Upsampling:**
Unpooling or strided transpose convolution

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

- **Input:** $3 \times H \times W$
- **High-res:** $D_1 \times H/2 \times W/2$
- **Med-res:** $D_2 \times H/4 \times W/4$
- **Low-res:** $D_3 \times H/4 \times W/4$
- **Predictions:** $H \times W$

---

Aside: Multi-view 3D Reconstruction

<table>
<thead>
<tr>
<th>Image 1</th>
<th>Image 2</th>
<th>Image 3</th>
<th>Reconstruction</th>
</tr>
</thead>
</table>

Aside: Multi-view 3D Reconstruction

Aside: Multi-view 3D Reconstruction

2D image sequence → 3D Convolutional LSTM → 3D Voxelized Reconstruction

2D Object Detection

Semantic Segmentation

GRASS, CAT, TREE, SKY
No objects, just pixels

2D Object Detection

DOG, DOG, CAT
Object categories + 2D bounding boxes

3D Object Detection

Car
Object categories + 3D bounding boxes
Classification + Localization

Class Scores
Cat: 0.9
Dog: 0.05
Car: 0.01
...

Vector: 4096

Fully Connected: 4096 to 1000

Box Coordinates
(x, y, w, h)

Fully Connected: 4096 to 4

Treat localization as a regression problem!

This image is CC0 public domain
Classification + Localization

Class Scores
- Cat: 0.9
- Dog: 0.05
- Car: 0.01

Correct label: Cat

Softmax Loss

Vector: 4096

Fully Connected: 4096 to 1000

Box Coordinates (x, y, w, h)

L2 Loss

Fully Connected: 4096 to 4

Correct box: (x’, y’, w’, h’)

Treat localization as a regression problem!
Classification + Localization

Class Scores
Cat: 0.9
Dog: 0.05
Car: 0.01
...

Multitask Loss

Correct label: Cat

Softmax Loss

Correct box: (x’, y’, w’, h’)

L2 Loss

Box Coordinates (x, y, w, h)

Fully Connected: 4096 to 4

Vector: 4096

Treat localization as a regression problem!

This image is CC0 public domain
Classification + Localization

Often pretrained on ImageNet (Transfer learning)

Treat localization as a regression problem!

Class Scores
Cat: 0.9
Dog: 0.05
Car: 0.01
...

Softmax Loss

Correct box: (x', y', w', h')

L2 Loss

Vector: 4096

Fully Connected: 4096 to 4

Box Coordinates (x, y, w, h)

Fully Connected: 4096 to 1000

Correct label: Cat

This image is CC0 public domain
Object Detection as Regression?

CAT: (x, y, w, h)

DOG: (x, y, w, h)

CAT: (x, y, w, h)

DUCK: (x, y, w, h)

DUCK: (x, y, w, h)

....
Object Detection as Regression?

Each image needs a different number of outputs!

CAT: \((x, y, w, h)\) \hspace{1cm} 4 numbers

DOG: \((x, y, w, h)\)
DOG: \((x, y, w, h)\)
CAT: \((x, y, w, h)\)

DUCK: \((x, y, w, h)\) \hspace{1cm} Many numbers!
DUCK: \((x, y, w, h)\)
Object Detection as Classification: Sliding Window

Apply a CNN to many different crops of the image, CNN classifies each crop as object or background

Dog? NO
Cat? NO
Background? YES
Object Detection as Classification: Sliding Window

Apply a CNN to many different crops of the image, CNN classifies each crop as object or background.

Dog? YES
Cat? NO
Background? NO
Object Detection as Classification: Sliding Window

Apply a CNN to many different crops of the image, CNN classifies each crop as object or background

Dog? YES
Cat? NO
Background? NO
Object Detection as Classification: Sliding Window

Apply a CNN to many different crops of the image, CNN classifies each crop as object or background.

Dog? NO
Cat? YES
Background? NO
Object Detection as Classification: Sliding Window

Apply a CNN to many different crops of the image, CNN classifies each crop as object or background

Problem: Need to apply CNN to huge number of locations, scales, and aspect ratios, very computationally expensive!
Region Proposals / Selective Search

- Find “blobby” image regions that are likely to contain objects
- Relatively fast to run; e.g. Selective Search gives 2000 region proposals in a few seconds on CPU

Alexe et al., “Measuring the objectness of image windows”, TPAMI 2012
Uijlings et al., “Selective Search for Object Recognition”, IJCV 2013
Cheng et al., “BING: Binarized normed gradients for objectness estimation at 300fps”, CVPR 2014
Zitnick and Dollar, “Edge boxes: Locating object proposals from edges”, ECCV 2014
R-CNN

Figure copyright Ross Girshick, 2015; source, Reproduced with permission.
R-CNN

Figure copyright Ross Girshick, 2015; [source](#). Reproduced with permission.


Regions of Interest (RoI) from a proposal method (~2k)
R-CNN


Figure copyright Ross Girshick, 2015; [source](#). Reproduced with permission.
R-CNN

Figure copyright Ross Girshick, 2015; source. Reproduced with permission.
R-CNN

Classify regions with SVMs
Forward each region through ConvNet
Warped image regions
Regions of Interest (RoI) from a proposal method (~2k)

Figure copyright Ross Girshick, 2015; source. Reproduced with permission.
R-CNN

Linear Regression for bounding box offsets

Classify regions with SVMs

Forward each region through ConvNet

Warped image regions

Regions of Interest (RoI) from a proposal method (~2k)

R-CNN: Problems

- Ad hoc training objectives
  - Fine-tune network with softmax classifier (log loss)
  - Train post-hoc linear SVMs (hinge loss)
  - Train post-hoc bounding-box regressions (least squares)
- Training is slow (84h), takes a lot of disk space
- Inference (detection) is slow
  - 47s / image with VGG16 [Simonyan & Zisserman. ICLR15]
  - Fixed by SPP-net [He et al. ECCV14]
Fast R-CNN

Figure copyright Ross Girshick, 2015; source. Reproduced with permission.
Fast R-CNN

Regions of Interest (RoIs) from a proposal method

“conv5” feature map of image

Forward whole image through ConvNet

ConvNet

Input image

Girshick, “Fast R-CNN”, ICCV 2015. Figure copyright Ross Girshick, 2015; source. Reproduced with permission.
Fast R-CNN

Figure copyright Ross Girshick, 2015; source. Reproduced with permission.
Fast R-CNN: RoI Pooling

Hi-res input image: 3 x 640 x 480 with region proposal

Hi-res conv features: 512 x 20 x 15;
Projected region proposal is e.g. 512 x 18 x 8 (varies per proposal)

Divide projected proposal into 7x7 grid, max-pool within each cell
RoI conv features: 512 x 7 x 7 for region proposal

Fully-connected layers expect low-res conv features: 512 x 7 x 7

Fast R-CNN

Figure copyright Ross Girshick, 2015; source. Reproduced with permission.
Fast R-CNN

Regions of Interest (RoIs) from a proposal method

ConvNet

Input image

Softmax classifier

Lineral + softmax

Linear

Bounding-box regressors

Fully-connected layers

“RoI Pooling” layer

“conv5” feature map of image

Forward whole image through ConvNet

Figure copyright Ross Girshick, 2015; source. Reproduced with permission.
Fast R-CNN
(Training)

Log loss + Smooth L1 loss

Multi-task loss

softmax
Linear

FCs

ConvNet

Input image

Figure copyright Ross Girshick, 2015; source. Reproduced with permission.
Fast R-CNN
(Training)

Log loss + Smooth L1 loss
Multi-task loss

softmax
Linear
FCs
ConvNet
Input image

Figure copyright Ross Girshick, 2015; source. Reproduced with permission.
R-CNN vs SPP vs Fast R-CNN

Girshick, "Fast R-CNN", ICCV 2015
R-CNN vs SPP vs Fast R-CNN

Problem:
Runtime dominated by region proposals!

Faster R-CNN:
Make CNN do proposals!

Insert Region Proposal Network (RPN) to predict proposals from features

Jointly train with 4 losses:
1. RPN classify object / not object
2. RPN regress box coordinates
3. Final classification score (object classes)
4. Final box coordinates

Figure copyright 2015, Ross Girshick; reproduced with permission
Faster R-CNN: Make CNN do proposals!
Mask R-CNN

He et al, "Mask R-CNN", arXiv 2017

CNN

Classification Scores: C
Box coordinates (per class): 4 * C

Rol Align

Conv

Conv

Predict a mask for each of C classes

C x 14 x 14

256 x 14 x 14

256 x 14 x 14
Mask R-CNN: Very Good Results!

He et al, "Mask R-CNN", arXiv 2017
Figures copyright Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick, 2017.
Reproduced with permission.
Mask R-CNN
Also does pose

He et al, "Mask R-CNN", arXiv 2017
Figures copyright Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick, 2017.
Reproduced with permission.
Detection without Proposals: YOLO / SSD

Input image
3 x H x W

Divide image into grid
7 x 7

Image a set of base boxes centered at each grid cell
Here B = 3

Within each grid cell:
- Regress from each of the B base boxes to a final box with 5 numbers: (dx, dy, dh, dw, confidence)
- Predict scores for each of C classes (including background as a class)

Output:
7 x 7 x (5 * B + C)

Detection without Proposals: YOLO / SSD

Go from input image to tensor of scores with one big convolutional network!

Input image
3 x H x W

Divide image into grid
7 x 7

Image a set of **base boxes**
centered at each grid cell
Here B = 3

Within each grid cell:
- Regress from each of the B base boxes to a final box with 5 numbers:
  (dx, dy, dh, dw, confidence)
- Predict scores for each of C classes (including background as a class)

Output:
7 x 7 x (5 * B + C)

Object Detection: Lots of variables ...

**Base Network**
- VGG16
- ResNet-101
- Inception V2
- Inception V3
- Inception
- ResNet
- MobileNet

**Object Detection architecture**
- Faster R-CNN
- R-FCN
- SSD

**Takeaways**
- Faster R-CNN is slower but more accurate
- SSD is much faster but not as accurate

Huang et al, “Speed/accuracy trade-offs for modern convolutional object detectors”, CVPR 2017

Object Detection: Impact of Deep Learning

Figure copyright Ross Girshick, 2015. Reproduced with permission.
Open Source Frameworks

Lots of good implementations on GitHub!

TensorFlow Detection API:
https://github.com/tensorflow/models/tree/master/research/object_detection
Faster RCNN, SSD, RFCN, Mask R-CNN

Caffe2 Detectron:
https://github.com/facebookresearch/Detectron
Mask R-CNN, RetinaNet, Faster R-CNN, RPN, Fast R-CNN, R-FCN

Finetune on your own dataset with pre-trained models
Aside: Object Detection + Captioning = Dense Captioning

Figure copyright IEEE, 2016. Reproduced for educational purposes.
Aside: Object Detection + Captioning
= Dense Captioning

Johnson, Karpathy, and Fei-Fei, "DenseCap: Fully Convolutional Localization Networks for Dense Captioning", CVPR 2016
Figure copyright IEEE, 2016. Reproduced for educational purposes.
Aside: Visual Genome

108,077 Images
5.4 Million Region Descriptions
1.7 Million Visual Question Answers
3.8 Million Object Instances
2.8 Million Attributes
2.3 Million Relationships
Everything Mapped to Wordnet Synsets

Aside: Scene Graph Generation

Xu, Zhu, Choy, and Fei-Fei, “Scene Graph Generation by Iterative Message Passing”, CVPR 2017
Figure copyright IEEE, 2018. Reproduced for educational purposes.
3D Object Detection

Semantic Segmentation

GRASS, CAT, TREE, SKY
No objects, just pixels

2D Object Detection

DOG, DOG, CAT
Object categories + 2D bounding boxes

3D Object Detection

Car
Object categories + 3D bounding boxes
Simplified Camera Model

Image source: https://www.pcmag.com/encyclopedia_images/_FRUSTUM.GIF
Simplified Camera Model

A point on the image plane corresponds to a ray in the 3D space.
Simplified Camera Model

A point on the image plane corresponds to a *ray* in the 3D space.

A 2D bounding box on an image is a *frustrum* in the 3D space.

Image source: https://www.pcmag.com/encyclopedia_images/_FRUSTUM.GIF
Scale & Distance Ambiguity

Objects of different scales & distances may look exactly the same in the image!
Simplified Camera Model

A point on the image plane corresponds to a ray in the 3D space.

A 2D bounding box on an image is a frustrum in the 3D space.

Localize an object in 3D: The object can be anywhere in the camera viewing frustrum!
3D Object Detection

2D Object Detection:
2D bounding box
(x, y, w, h)

3D Object Detection:
3D oriented bounding box
(x, y, z, w, h, l, r, p, y)

Simplified bbox: no roll & pitch

Much harder problem than 2D object detection!
3D Object Detection
3D Object Detection: Monocular Camera

- Same idea as Faster RCNN, but proposals are in 3D
- 3D bounding box proposal, regress 3D box parameters + class score

3D Object Detection: Monocular View

3D Object Detection: Camera + LiDAR

Velodyne (HDL-64e)

3D Point Cloud
3D Object Detection: Camera + LiDAR

- Combine 3D proposals from multiple views & sensors
- Regress 3D box parameters + class score

Chen, Xiaozhi, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. "Multi-view 3d object detection network for autonomous driving." CVPR 2017
3D Object Detection: Camera + LiDAR

Chen, Xiaozhi, Huimin Ma, Ji Wan, Bo Li, and Tian Xia. "Multi-view 3D object detection network for autonomous driving." CVPR 2017
RGB-Depth Camera

Kinect (Xbox One)

This image is CC0 public domain
RGB-Depth Camera

Registered RGB + depth point cloud
Point Cloud Voxelization

1. Capture RGB-D point cloud of a scene.
2. Partition the 3D space into a regular 3D grid.
3. For each grid cell that has a point fall into it, fill the cell with the RGB value of that point.
   A bit like “3D image”
3D Object Detection: RGB-Depth Camera

Voxelized RGB-D point cloud

3D region proposals

Object categories + 3D bounding boxes

“Faster RCNN in 3D”

Recap

Semantic Segmentation

GRASS, CAT, TREE, SKY
No objects, just pixels

2D Object Detection

DOG, DOG, CAT
Object categories + 2D bounding boxes

3D Object Detection

Car
Object categories + 3D bounding boxes
Next time:
Visualizing CNN features
DeepDream + Style Transfer