Convolutional Neural Networks
Architectures
Application to other problems
Reading

• http://cs231n.github.io/convolutional-networks/

• Papers cited from the slides
Announcements

• Artifact voting for P3 is open
  – Closes Wednesday 11:59pm
• Material starting today is not on the final exam, but might be of interest anyway.
• Today:
  – GoogLeNet, ResNet, etc.
  – CNNs applied to other problems
• Wednesday:
  – Review; Deep dream, style transfer (if time)
• Friday:
  – Review; bilateral space (if time)
Goals

• Gain intuition for some of the trends in CNN architectures since 2012:
  – Deeper networks
  – Smaller filters
  – Parallel filters, bottleneck layers
  – Residual connections

• Understand how to solve other problems with CNNs:
  – Classification on non-ImageNet datasets
  – Semantic Segmentation
  – Object detection
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]
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.
ImageNet Large Scale Visual Recognition Challenge (ILSVRRC) winners

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

**Deeper Networks**
- 152 layers (2014, 2015, 2016)
- 19 layers (2014)
- 22 layers (2014)

**Note**: The diagram illustrates the evolution of ImageNet challenge winners, highlighting the trend towards deeper networks improving accuracy over time.
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
WE NEED TO GO DEEPER
Case Study: GoogleLeNet

[Szegedy et al., 2014]

Deeper networks, with computational efficiency

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

Inception module
Case Study: GoogLeNet

[ Szegedy et al., 2014 ]

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 depth-wise
Case Study: GoogLeNet
[Szegedy et al., 2014]

Example:

Q3: 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] \(28 \times 28 \times 128 \times 1 \times 1 \times 256\)
- [3x3 conv, 192] \(28 \times 28 \times 192 \times 3 \times 3 \times 256\)
- [5x5 conv, 96] \(28 \times 28 \times 96 \times 5 \times 5 \times 256\)

Total: 854M ops

Very expensive compute

Pooling layer also preserves feature depth, which means total depth after concatenation can only grow at every layer!
Case Study: GoogLeNet

[_szegedy et al., 2014_]

Example:

Q3: What is the output size after filter concatenation?

\[ 28 \times 28 \times (128 + 192 + 96 + 256) = 529k \]

Solution: “bottleneck” layers that use 1x1 convolutions to reduce feature depth.
Reminder: 1x1 convolutions

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

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

Projects depth to lower dimension (combination of feature maps)
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
Case Study: GoogLeNet

[Szegedy et al., 2014]

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
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

Classifier output (removed 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-Sofmax)
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
- No FC layers
- 12x less params than AlexNet
- ILSVRC’14 classification winner (6.7% top 5 error)
WE NEED TO GO DEEPER
ImageNet Large Scale Visual Recognition Challenge (ILSVRC) winners

```
<table>
<thead>
<tr>
<th>Year</th>
<th>Model</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>2010</td>
<td>Lin et al</td>
<td>28.2</td>
</tr>
<tr>
<td>2011</td>
<td>Sanchez &amp; Perronnin</td>
<td>25.8</td>
</tr>
<tr>
<td>2012</td>
<td>Krizhevsky et al (AlexNet)</td>
<td>16.4</td>
</tr>
<tr>
<td>2013</td>
<td>Zeiler &amp; Fergus</td>
<td>11.7</td>
</tr>
<tr>
<td>2014</td>
<td>Simonyan &amp; Zisserman (VGG)</td>
<td>7.3</td>
</tr>
<tr>
<td>2014</td>
<td>Szegedy et al (GoogLeNet)</td>
<td>6.7</td>
</tr>
<tr>
<td>2015</td>
<td>He et al (ResNet)</td>
<td>3.6</td>
</tr>
<tr>
<td>2016</td>
<td>Shao et al</td>
<td>3.0</td>
</tr>
<tr>
<td>2017</td>
<td>Hu et al (SENet)</td>
<td>2.3</td>
</tr>
<tr>
<td></td>
<td>Russakovsky et al</td>
<td>5.1</td>
</tr>
</tbody>
</table>
```

“Revolution of Depth”
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?

Q: What’s strange about these training and test curves?  
[Hint: look at the order of the curves]

56-layer model performs worse on both training and test error -> The deeper model performs worse, but it’s not caused by overfitting!
Case Study: ResNet

[He et al., 2015]

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

The deeper model should be able to perform at least as well 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
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

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

Use layers to fit residual \( F(x) = H(x) - x \) instead of \( H(x) \) directly
Case Study: ResNet

[He et al., 2015]

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

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
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
- No FC layers at the end (only FC 1000 to output classes)
Case Study: ResNet

[He et al., 2015]

Total depths of 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]

Experimental Results
- Able to train very deep networks without degrading (152 layers on ImageNet, 1202 on Cifar)
- Deeper networks now achieve lowing 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
Comparing complexity...

Inception-v4: Resnet + Inception!


Comparing complexity...


Comparing complexity...


And so on and so forth...

- Dizzying number of papers since then proposing more architecture tricks and hacks.
- A couple notable examples:
  - FractalNet
  - DenseNet
Do we though?
(open question)
Beyond ResNets...

FractalNet: Ultra-Deep Neural Networks without Residuals

[Larsson et al. 2017]

- Argues that key is transitioning effectively from shallow to deep and residual representations are not necessary
- Fractal architecture with both shallow and deep paths to output
- Trained with dropping out sub-paths
- Full network at test time
Beyond ResNets...

Densely Connected Convolutional Networks

[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
And so on and so forth...

- So we’ve beat the crap out of ImageNet... what now?
And so on and so forth...

- So we’ve beat the crap out of ImageNet... what now?
  - Can we do image classification on other datasets?
  - Can we do things other than image classification?
And so on and so forth...

• So we’ve beat the crap out of ImageNet... what now?
  – Can we do image classification on other datasets?
  – Can we do things other than image classification?
Transfer Learning

“You need a lot of data if you want to train/use CNNs”
Transfer Learning

“You need a lot of data if you want to train/use CNNs”

well... sort of

BUSTED
Transfer Learning with CNNs

1. Train on Imagenet

- FC-1000
- FC-4096
- FC-4096
- MaxPool
- Conv-512
- Conv-512
- MaxPool
- Conv-512
- Conv-512
- MaxPool
- Conv-256
- Conv-256
- MaxPool
- Conv-128
- Conv-128
- MaxPool
- Conv-64
- Conv-64

- Image
Transfer Learning with CNNs

1. Train on Imagenet

FC-1000
FC-4096
FC-4096
MaxPool
Conv-512
Conv-512
MaxPool
Conv-512
Conv-512
MaxPool
Conv-256
Conv-256
MaxPool
Conv-128
Conv-128
MaxPool
Conv-64
Conv-64
Image

2. Small Dataset (C classes)

FC-C
FC-4096
FC-4096
MaxPool
Conv-512
Conv-512
MaxPool
Conv-512
Conv-512
MaxPool
Conv-256
Conv-256
MaxPool
Conv-128
Conv-128
MaxPool
Conv-64
Conv-64
Image

Reinitialize this and train

Freeze these
Transfer Learning with CNNs

1. Train on Imagenet
   - FC-1000
   - FC-4096
   - FC-4096
   - MaxPool
   - Conv-512
   - Conv-512
   - MaxPool
   - Conv-512
   - Conv-512
   - MaxPool
   - Conv-512
   - Conv-512
   - MaxPool
   - Conv-256
   - Conv-256
   - MaxPool
   - Conv-128
   - Conv-128
   - MaxPool
   - Conv-64
   - Conv-64

2. Small Dataset (C classes)
   - FC-C
   - FC-4096
   - MaxPool
   - Conv-512
   - Conv-512
   - MaxPool
   - Conv-512
   - Conv-512
   - MaxPool
   - Conv-256
   - Conv-256
   - MaxPool
   - Conv-128
   - Conv-128
   - MaxPool
   - Conv-64
   - Conv-64

3. Bigger dataset
   - FC-C
   - FC-4096
   - MaxPool
   - Conv-512
   - Conv-512
   - MaxPool
   - Conv-512
   - Conv-512
   - MaxPool
   - Conv-256
   - Conv-256
   - MaxPool
   - Conv-128
   - Conv-128
   - MaxPool
   - Conv-64
   - Conv-64

Reinitialize this and train
Freeze these
Train these
With bigger dataset, train more layers
Freeze these
Lower learning rate when finetuning; 1/10 of original LR is good starting point
Figure: Razavian et al.: CNN Features off-the-shelf: an Astounding Baseline for Recognition
Transfer learning with CNNs is pervasive…
(it’s the norm, not an exception)

Object Detection
(Fast R-CNN)

Image Captioning: CNN + RNN

Figure copyright Ross Girshick, 2015. Reproduced with permission.

Figure copyright IEEE, 2015. Reproduced for educational purposes.
Transfer learning with CNNs is pervasive…
(it’s the norm, not an exception)

Object Detection (Fast R-CNN)

CNN pretrained on ImageNet

Image Captioning: CNN + RNN

Girshick, "Fast R-CNN", ICCV 2015
Figure copyright Ross Girshick, 2015. Reproduced with permission.

Figure copyright IEEE, 2015. Reproduced for educational purposes.

Slide: Fei-Fei Li, Justin Johnson, & Serena Yeung
And so on and so forth...

- So we’ve beat the crap out of ImageNet... what now?
  - Can we do image classification on other datasets?
  - Can we do things other than image classification?
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

Classification + Localization

Object Detection

Instance Segmentation

GRASS, CAT, TREE, SKY

No objects, just pixels

CAT

Single Object

DOG, DOG, CAT

Multiple Object

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

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

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: 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:** $H \times W$

Slide: Fei-Fei Li, Justin Johnson, & Serena Yeung
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:

Med-res:

Low-res:

High-res:

Predictions:

$D_1 \times H/2 \times W/2$

$D_2 \times H/4 \times W/4$

$D_3 \times H/4 \times W/4$

$D_1 \times H/2 \times W/2$

$H \times W$


Slide: Fei-Fei Li, Justin Johnson, & Serena Yeung
Semantic Segmentation Idea: Fully Convolutional

Downsampling: Pooling, strided convolution

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

Upsampling: ???

Input: $3 \times H \times W$

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

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

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: $H \times W$


In-Network upsampling: “Unpooling”

Nearest Neighbor

Input: 2 x 2

Output: 4 x 4

“Bed of Nails”

Input: 2 x 2

Output: 4 x 4
In-Network upsampling: “Max Unpooling”

Max Pooling
Remember which element was max!

Max Unpooling
Use positions from pooling layer

Input: 4 x 4
Output: 2 x 2
Rest of the network

Input: 2 x 2
Output: 4 x 4

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

Dot product between filter and input

Output: 4 x 4
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 $3 \times 3$ convolution, **stride 2** pad 1

Input: $4 \times 4$  

Dot product between filter and input  

Output: $2 \times 2$
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 transpose convolution, stride 2 pad 1

Input gives weight for filter

Input: 2 x 2

Output: 4 x 4
Learnable Upsampling: Transpose Convolution

3 x 3 transpose convolution, stride 2 pad 1

Input: 2 x 2

Output: 4 x 4

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
Learnable Upsampling: Transpose Convolution

3 x 3 transpose convolution, stride 2 pad 1

Input: 2 x 2
Output: 4 x 4

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 gives weight for filter
Learnable Upsampling: Transpose Convolution

Other names:
- Deconvolution (bad)
- Upconvolution
- Fractionally strided convolution
- Backward strided 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 between movement in output and input

Sum where output overlaps

Input gives weight for filter
Learnable Upsampling: 1D Example

Input

Output

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

Need to crop one pixel from output to make output exactly 2x input.
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
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.
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
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)

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

- **Softmax classifier**
- **Linear + softmax**
- **Linear**
- **Bounding-box regressors**
- **Fully-connected layers**
- **“RoI Pooling” layer**
- **“conv5” feature map of image**
- **Regions of Interest (RoIs) from a proposal method**
- **ConvNet**
- **Forward whole image through ConvNet**

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

CNN

Roi Align

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

Conv

Conv

Predict a mask for each of C classes

C x 14 x 14

He et al, "Mask R-CNN", arXiv 2017
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.
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: Impact of Deep Learning

Figure copyright Ross Girshick, 2015. Reproduced with permission.
Other Problems

• Fine-grained recognition (e.g., dog/bird species)
• Instance segmentation
• Face detection and recognition
• Motion estimation
• Feature detection and description
• Depth estimation
• Novel view synthesis
• ...and many others