Instance Segmentation (Continued)
Network Visualization
• **Assignment 2**
  • 🚨 We are into the grace period!🚨
  • No exception other than for emergencies.

• **Project Proposal Feedback is Out**
  • Talk to the TA (over OHs) who graded your proposal for more detailed feedback.

• **Assignment 3 out soon**
Computer Vision Tasks

Classification

Semantic Segmentation

Object Detection

Instance Segmentation

CAT
GRASS, CAT, TREE, SKY
DOG, DOG, CAT
DOG, DOG, CAT

No spatial extent
No objects, just pixels
Multiple Object

This image is CC0 public domain
Semantic Segmentation Idea: Fully Convolutional

Design a network with only convolutional layers without downsampling operators to make predictions for pixels all at once!

Input: 3 x H x W

Convolutions: D x H x W

Scores: C x H x W

Predictions: H x W

Conv
Conv
Conv
Conv
argmax

Loss: Pixel-wise cross entropy!
Learnable Upsampling: Transposed Convolution

Q: Why is it called transpose convolution?

Input: $2 \times 2$
Output: $4 \times 4$

$3 \times 3$ transpose convolution, stride 2 pad 1

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

Sum where output overlaps

Learnable Upsampling: Transposed 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**: $D_3 \times H/4 \times W/4$
- **High-res**: $D_1 \times H/2 \times W/2$
- **Predictions**: $H \times W$

“Slow” R-CNN

Classify regions with SVMs

Forward each region through ConvNet

Warped image regions (224x224 pixels)

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

Input image

Problem: Very slow! Need to do ~2k independent forward passes for each image!

Idea: Pass the image through convnet before cropping! Crop the conv feature instead!

Fast R-CNN

Regions of Interest (RoIs) from a proposal method

“Backbone” network: AlexNet, VGG, ResNet, etc

Run whole image through ConvNet

Crop + Resize features

“conv5” features

Per-Region Network

Box offset

Linear + softmax

Object category

Linear

CNN

ConvNet

Input image

“Slow” R-CNN

SVMs

Conv Net

SVMs

Conv Net

SVMs

Conv Net

Input image

Girshick, “Fast R-CNN”, ICCV 2015. Figure copyright Ross Girshick, 2015; source. Reproduced with permission.
Cropping Features: RoI Pool

Project proposal onto features

“Snap” to grid cells

Divide into 2x2 grid of (roughly) equal subregions

Max-pool within each subregion

Image features: C x H x W
(e.g. 512 x 20 x 15)

Region features (here 512 x 2 x 2;
In practice e.g 512 x 7 x 7)

Region features always the same size even if input regions have different sizes!

Problem: Region features slightly misaligned

Cropping Features: RoI Align

Project proposal onto features

No “snapping”!

Sample at regular points in each subregion using bilinear interpolation

Max-pool within each subregion

Region features (here 512 x 2 x 2; In practice e.g 512 x 7 x 7)

Input Image
(e.g. 3 x 640 x 480)

CNN

Image features: C x H x W
(e.g. 512 x 20 x 15)

He et al, “Mask R-CNN”, ICCV 2017
**Faster R-CNN:**
Make CNN do proposals!

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

Otherwise same as Fast R-CNN:
Crop features for each proposal, classify each one

Figure copyright 2015, Ross Girshick; reproduced with permission
Faster R-CNN:
Make CNN do proposals!
Single-Stage Object Detectors: YOLO / SSD / RetinaNet

- 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)
- Looks a lot like RPN, but category-specific!

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

Lin et al, “Focal Loss for Dense Object Detection”, ICCV 2017
Instance Segmentation

Classification

Semantic Segmentation

Object Detection

Instance Segmentation

CAT

GRASS, CAT, TREE, SKY

DOG, DOG, CAT

DOG, DOG, CAT

No spatial extent

No objects, just pixels

Multiple Object
Object Detection: Faster R-CNN
Instance Segmentation: Mask R-CNN

Add a small mask network that operates on each RoI and predicts a 28x28 binary mask.

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

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

CNN + RPN

RoI Align

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

Predict a mask for each of C classes

C x 28 x 28
Mask R-CNN: Example Mask Training Targets
Mask R-CNN: Example Mask Training Targets
Mask R-CNN: Example Mask Training Targets
Mask R-CNN: Example Mask Training Targets
Mask R-CNN: Very Good Results!

He et al, "Mask R-CNN", ICCV 2017
Mask R-CNN
Also does pose

He et al, “Mask R-CNN”, ICCV 2017
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, ...

Detectron2 (PyTorch)
https://github.com/facebookresearch/detectron2
Mask R-CNN, RetinaNet, Faster R-CNN, RPN, Fast R-CNN, R-FCN, ...

Finetune on your own dataset with pre-trained models
Beyond 2D Object Detection...
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.
Objects + Relationships = Scene Graphs

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

Scene Graph Prediction

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

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: Simple 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: Monocular Camera

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

3D Shape Prediction: Mesh R-CNN

Gkioxari et al., Mesh RCNN, ICCV 2019
Recap: Lots of computer vision tasks!

- **Classification**
  - CAT
  - No spatial extent

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

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

- **Instance Segmentation**
  - DOG, DOG, CAT
  - This image is CC0 public domain
Visualizing Neural Networks
Interpreting a Linear Classifier: Visual Viewpoint

airplane
automobile
bird
cat
deer
dog
frog
horse
ship
truck

Input image

W

0.2 -0.5
0.1 2.0
1.5 1.3
2.1 0.0
0 0.25
0.2 -0.3

b

1.1
3.2
-1.2

Score

-96.8
437.9
61.95

plane car bird cat deer dog frog horse ship truck
First Layer: Visualize Filters

AlexNet:
64 x 3 x 11 x 11

Huang et al, "Densely Connected Convolutional Networks", CVPR 2017
First Layer: Visualize Filters

AlexNet:
64 x 3 x 11 x 11

ResNet-18:
64 x 3 x 7 x 7

ResNet-101:
64 x 3 x 7 x 7

DenseNet-121:
64 x 3 x 7 x 7

Huang et al, "Densely Connected Convolutional Networks", CVPR 2017
Visualize the filters/kernels (raw weights)

We can visualize filters at higher layers, but not that interesting

(these are taken from ConvNetJS CIFAR-10 demo)
4096-dimensional feature vector for an image (layer immediately before the classifier)

Run the network on many images, collect the feature vectors
Last Layer: Nearest Neighbors

Recall: Nearest neighbors in pixel space

Test image L2 Nearest neighbors in feature space

Last Layer: Dimensionality Reduction

Visualize the “space” of FC7 feature vectors by reducing dimensionality of vectors from 4096 to 2 dimensions.

Simple algorithm: Principal Component Analysis (PCA)

More complex: \( \text{t-SNE} \)

Visualize MNIST:
Last Layer: Dimensionality Reduction

Van der Maaten and Hinton, "Visualizing Data using t-SNE", JMLR 2008
Krizhevsky et al, "ImageNet Classification with Deep Convolutional Neural Networks", NIPS 2012. Figure reproduced with permission.

See high-resolution versions at http://cs.stanford.edu/people/karpathy/cnnemebd/
conv5 feature map is 128x13x13; visualize as 128 13x13 grayscale images
Visualizing Activations

conv5 feature map is 128x13x13; visualize as 128 13x13 grayscale images

Neural nets learn distributed representations over many layers. Difficult to visualize everything!
Which pixels matter: Saliency via Occlusion

Mask part of the image before feeding to CNN, check how much predicted probabilities change

P(elephant) = 0.95

P(elephant) = 0.75

Zeiler and Fergus, “Visualizing and Understanding Convolutional Networks”, ECCV 2014
Which pixels matter: Saliency via Occlusion

Mask part of the image before feeding to CNN, check how much predicted probabilities change
Which pixels matter: Saliency via Backprop

Forward pass: Compute probabilities

Figures copyright Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman, 2014; reproduced with permission.
Which pixels matter: Saliency via Backprop

Forward pass: Compute probabilities

Compute gradient of (unnormalized) class score with respect to image pixels, take absolute value and max/sum over RGB channels

Figures copyright Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman, 2014; reproduced with permission.
Saliency Maps

Figures copyright Karen Simonyan, Andrea Vedaldi, and Andrew Zisserman, 2014; reproduced with permission.
Gradient-based Saliency Visualization

Given a trained model, we can perform forward pass given an input to get scores, softmax probabilities, loss and then backwards pass to get gradients.

- Note: We are keeping parameters/weights frozen
  - Do not use gradients w.r.t. weights to perform updates
Gradient-based Saliency Visualization

**Idea:** We can backprop to the image

- Sensitivity of loss to individual pixel changes
- Large sensitivity implies important pixels
- Called **Saliency Maps**

**In practice:**

- Instead of loss, find gradient of classifier **scores** (pre-softmax)
- Take absolute value of gradient
- Sum across all channels

Gradient-based Saliency Visualization

Applying traditional (non-learned) computer vision segmentation algorithms on gradients gets us **object segmentation for free**!

Surprising because **not part of supervision**

Intermediate Features via (guided) backprop

Pick a single intermediate neuron, e.g. one value in 128 x 13 x 13 conv5 feature map

Compute gradient of activation value with respect to image pixels

Zeiler and Fergus, “Visualizing and Understanding Convolutional Networks”, ECCV 2014
Intermediate Features via (guided) backprop

Pick a single intermediate neuron, e.g. one value in 128 x 13 x 13 conv5 feature map

Compute gradient of activation value with respect to image pixels

**Guided backprop**: suppress pathways that have negative gradients --- only backprop positive gradients through each ReLU
Guided Backprop Results

From: Springenberg et al., “Striving For Simplicity: The All Convolutional Net”
Note: These images were created by a slightly different method called deconvolution, which ends up being similar to guided backprop.

From: “Visualizing and Understanding Convolutional Networks, Zeiler & Fergus, 2014.”
VGG Layer-by-Layer Visualization

From: “Visualizing and Understanding Convolutional Networks, Zeiler & Fergus, 2014.”
VGG Layer-by-Layer Visualization

From: “Visualizing and Understanding Convolutional Networks, Zeiler & Fergus, 2014.”
VGG Layer-by-Layer Visualization

From: "Visualizing and Understanding Convolutional Networks, Zeiler & Fergus, 2014."
Guided Grad-CAM

Grad-CAM

Conv Feature Maps

Guided Backpropagation

Neuron Importance

Guided Grad-CAM

Grad-CAM

Guided Backpropagation

Rectified Conv Feature Maps

Backprop till conv

Gradients

Activations

What animal is in this picture? Dog

What animal is in this picture? Cat

Summary

- Gradients are important not just for optimization, but also for analyzing what neural networks have learned.
- Standard backprop not always the most informative for visualization purposes.
- Several ways to modify the gradient flow to improve visualization results.
Optimizing the Input Images
Idea: Since we have the gradient of scores w.r.t. inputs, can we optimize the image itself to maximize the score?

Why?

- Generate images from scratch!
- Adversarial examples

We can perform **gradient ascent** on image for image generation

- Start from random/zero image
- Use scores to avoid minimizing other class scores instead

Often need **regularization term** to induce statistics of natural imagery

- E.g. small pixel values, spatial smoothness

\[
\text{argmax } S_c(I) - \lambda \|I\|_2^2
\]

\[
I = I + \alpha \frac{\partial S_c}{\partial I}
\]

Note: You might have to squint!

Can improve results with various tricks:

- Clipping or normalization of small values & gradients
- Gaussian blurring

From: Yosinski et al., “Understanding Neural Networks Through Deep Visualization”, 2015
Improved Results

From: Yosinski et al., “Understanding Neural Networks Through Deep Visualization”, 2015
Summary

We can optimize the input image to generate examples to increase class scores or activations.

This can show us a great deal about what examples (not in the training set) activate the network.
We can perform gradient ascent on image.

Rather than start from zero image, why not real image?

And why not optimize the score of an arbitrary (incorrect!) class?

Surprising result: You need very small amount of pixel changes to make the network confidently wrong!

\[ \text{argmax } S_c(I) - \lambda \|I\|^2, \]

where \( c = \text{cat} \)

Example of Adversarial Noise

Note this problem is not specific to deep learning!

- Other methods also suffer from it
- Can show how **linearity** (even at the end) can bring this about
  - Can add many small values that add up in right direction

From: Goodfellow et al., “Explaining and Harnessing Adversarial Examples”, 2015
Variations of Attacks

Single-Pixel Attacks!


White vs. Black-Box Attacks of Increasing Complexity

Chakraborty et al., Adversarial Attacks and Defences: A Survey, 2018
Summary of adversarial Attacks/Defenses

Similar to other security-related areas, it’s an active cat-and-mouse game. Several defenses such as:

- Training with adversarial examples
- Perturbations, noise, or re-encoding of inputs

There are not universal methods that are robust to all types of attacks.
Style Transfer: Separating Style from Content

So far, we’ve seen how to generate images for certain classes / activations through backpropagation / gradient-based optimization.

Can we use similar ideas to generate images by combining the style and the content from different images?
Each layer of CNN gives $C \times H \times W$ tensor of features; $H \times W$ grid of $C$-dimensional vectors
Neural Texture Synthesis: Gram Matrix

Each layer of CNN gives $C \times H \times W$ tensor of features;
$H \times W$ grid of $C$-dimensional vectors

Outer product of two $C$-dimensional vectors gives $C \times C$ matrix measuring co-occurrence
Neural Texture Synthesis: Gram Matrix

Each layer of CNN gives $C \times H \times W$ tensor of features; $H \times W$ grid of $C$-dimensional vectors

Outer product of two $C$-dimensional vectors gives $C \times C$ matrix measuring co-occurrence

Average over all HW pairs of vectors, giving Gram matrix $G$ of shape $C \times C$

Gram matrix captures the statistics of the texture rather than the content of the image
Neural Texture Synthesis: Gram Matrix

Each layer of CNN gives $C \times H \times W$ tensor of features; $H \times W$ grid of $C$-dimensional vectors.

Outer product of two $C$-dimensional vectors gives $C \times C$ matrix measuring co-occurrence.

Average over all $H \times W$ pairs of vectors, giving **Gram matrix $G$** of shape $C \times C$.

Efficient to compute; reshape features from $C \times H \times W$ to $=C \times HW$ then compute $G$ from all pairs of feature vectors.

Gram matrix captures the statistics of the **texture** rather than the content of the image.
Neural Texture Synthesis

1. Pretrain a CNN on ImageNet (VGG-19)
2. Run input texture forward through CNN, record activations on every layer; layer \( i \) gives feature map of shape \( C_i \times H_i \times W_i \)
3. At each layer compute the Gram matrix giving outer product of features:

\[
G_{ij}^l = \sum_k F_{ik}^l F_{jk}^l \quad \text{shape } C_i \times C_i
\]

\[
\mathcal{L}(\tilde{x}, \hat{x}) = \sum_{l=0}^{L} w
\]
Neural Texture Synthesis

1. Pretrain a CNN on ImageNet (VGG-19)
2. Run input texture forward through CNN, record activations on every layer; layer $i$ gives feature map of shape $C_i \times H_i \times W_i$
3. At each layer compute the Gram matrix giving outer product of features:
   \[ G_{ij}^l = \sum_k F_{ik}^l F_{jk}^l \text{(shape } C_i \times C_i) \]
4. Initialize generated image from random noise
5. Pass generated image through CNN, compute Gram matrix on each layer
6. Compute loss: weighted sum of L2 distance between Gram matrices
7. Backprop to get gradient on image
8. Make gradient step on image
9. GOTO 5

Gatys, Ecker, and Bethge, “Texture Synthesis Using Convolutional Neural Networks”, NIPS 2015
Figure copyright Leon Gatys, Alexander S. Ecker, and Matthias Bethge, 2015. Reproduced with permission.
Neural Texture Synthesis

Reconstructing texture from higher layers recovers larger features from the input texture
Neural Style Transfer

Content Image + Style Image

This image is licensed under CC-BY 3.0

Starry Night by van Gogh is in the public domain

Gatys, Ecker, and Bethge, "Texture Synthesis Using Convolutional Neural Networks", NIPS 2015
Neural Style Transfer

Content Image + Style Image = Style Transfer!

Starry Night by Van Gogh is in the public domain

This image is licensed under CC-BY 3.0

This image copyright Justin Johnson, 2015. Reproduced with permission.

Feature Inversion

Given a CNN feature vector for an image, find a new image that:
- Matches the given feature vector
- “looks natural” (image prior regularization)

\[ x^* = \arg\min_{x \in \mathbb{R}^{H \times W \times C}} \ell(\Phi(x), \Phi_0) + \lambda \mathcal{R}(x) \]

\[ \ell(\Phi(x), \Phi_0) = \|\Phi(x) - \Phi_0\|^2 \]

\[ \mathcal{R}_{V^\beta}(x) = \sum_{i,j} \left( (x_{i,j+1} - x_{i,j})^2 + (x_{i+1,j} - x_{i,j})^2 \right)^{\frac{\beta}{2}} \]

Feature Inversion

Reconstructing from different layers of VGG-16

Figure from Johnson, Alahi, and Fei-Fei, “Perceptual Losses for Real-Time Style Transfer and Super-Resolution”, ECCV 2016. Copyright Springer, 2016. Reproduced for educational purposes.
Figure adapted from Johnson, Alahi, and Fei-Fei, “Perceptual Losses for Real-Time Style Transfer and Super-Resolution”, ECCV 2016. Copyright Springer, 2016. Reproduced for educational purposes.

Figure adapted from Johnson, Alahi, and Fei-Fei, “Perceptual Losses for Real-Time Style Transfer and Super-Resolution”, ECCV 2016. Copyright Springer, 2016. Reproduced for educational purposes.
Neural Style Transfer

Figure copyright Justin Johnson, 2015.
Summary

- Generating images through optimization is a powerful concept!
- Besides fun and art, methods such as stylization also useful for understanding what the network has learned
- Also useful for other things such as data augmentation