ANNOUNCEMENTS

- Last recitation, for PSET 5, this Friday!
- Make sure you are all working on final projects. Leave yourself enough time to write the report.

OBJECT DETECTION

Rich feature hierarchies for accurate object detection and semantic segmentation
Tech report (v5)

Ross Girshick, Jeff Donahue, Trevor Darrell, Jitendra Malik
UC Berkeley

R-CNN: Regions with CNN features

1. Input image  
2. Extract region proposals (~2k)  
3. Compute CNN features  
4. Classify regions
Newer methods also use a neural network to generate region proposals.

Efficient Implementations: bulk of the computation happens once on the entire image, and you crop a feature map for each region.

Even Faster Methods: Discretize image locations into grid, and directly output upto a fixed number of bounding boxes for each grid block.

Say you want to train a network to solve a problem.

- The task is complex, so you need a large network.
- But you don’t have enough training data to train such a network.
- Pick a related task for which you do have a lot of training data
  - ImageNet is a great database for this for a variety of semantic tasks
- Train a network (like VGG-16) to solve that task.
- Then, choose the output of some intermediate layer of that network
- Use it as a feature vector, and learn a smaller network for your problem which goes from those features to the desired output.
TRANSFER LEARNING

- VGG-16 does well on ImageNet classification and gives you a feature representation that is surprisingly useful for a broad range of tasks.

Remember computing encoding $\tilde{x}$ from $x$. VGG-16’s pool5, fc1, fc2, features can be the $\tilde{x}$ for many tasks.

One can also “initialize” a network with the VGG-16 architecture to one trained with ImageNet, and then “finetune” by replacing the final layer as classification for another task.

In general, empirical question to determine when training on Task A will provide good features for Task B.

OTHER TASKS
Treat *segmentation* as "Classification" of a patch treated as an image

Treat *segmentation* as "Classification" of the center pixel of a patch, with the patch given as input and treated as an image
OTHER TASKS

Treat *segmentation* as "Classification" of the center pixel of a patch, with the patch given as input and treated as an image. Same approach: Pre-train on imagenet, use activations or finetune.

FULLY-CONVOLUTIONAL NETWORKS

Let's think in 1D. And assume we had two conv layers, and then a fully connected layer.

The problem: Lots of patches (one for each pixel)

Let's think in 1D. And assume we had two conv layers, and then a fully connected layer.
FULLY-CONVOLUTIONAL NETWORKS

Let's think in 1D. And assume we had two conv layers, and then a fully connected layer.

FULLY-CONVOLUTIONAL NETWORKS

Let's think in 1D. And assume we had two conv layers, and then a fully connected layer.

FULLY-CONVOLUTIONAL NETWORKS

Let's think in 1D. And assume we had two conv layers, and then a fully connected layer.

FULLY-CONVOLUTIONAL NETWORKS

Let's think in 1D. And assume we had two conv layers, and then a fully connected layer.
FULLY-CONVOLUTIONAL NETWORKS

Let's think in 1D. And assume we had two conv layers, and then a fully connected layer.

FULLY-CONVOLUTIONAL NETWORKS

Let's think in 1D. And assume we had two conv layers, and then a fully connected layer.

FULLY-CONVOLUTIONAL NETWORKS

Let's think in 1D. And assume we had two conv layers, and then a fully connected layer.

FULLY-CONVOLUTIONAL NETWORKS

Let's think in 1D. And assume we had two conv layers, and then a fully connected layer.

No need to repeat computations.
FULLY-CONVOLUTIONAL NETWORKS

Let's think in 1D. And assume we had two conv layers, and then a fully connected layer.

No need to repeat computations.

FULLY-CONVOLUTIONAL NETWORKS

FULLY-CONVOLUTIONAL NETWORKS

FULLY-CONVOLUTIONAL NETWORKS

FULLY-CONVOLUTIONAL NETWORKS
But what about downsampling?

- Option 0: Just don’t use downsampling
  Bad, because down-sampling is a way to quickly increase the “receptive field” of your network.

- Option 1: Just produce a label map at lower-resolution.
  - Option 2: If you downsample by $N$ (typically $N = 2^K$)
    Feed every $(N - 1) \times (N - 1)$ “shifted” version of your input through this FCN.

  Bad because if you down-sample multiple times, you’re still re-computing activations prior to the last-downsampling.

- Option 3: Dilated Convolutions

$$N = 2^K$$
Dilated Convolution

In your original network, everytime you have downsampling, you should remove that stride (set it to 1), and multiply it with the dilation factor of all subsequent spatial layers (conv and pool).
**DEEP ARCHITECTURES**

- Deeper networks are able to express more complex functions. Larger Hypothesis Space.
- Empirically, deeper networks tend to overfit less than wider networks (i.e., more layers better than more channels)

- But, you need to back-propagate gradients of the loss computed at the final output back through a larger number of layers.
- With multiple non-linearities, gradients have a higher probability of vanishing.
- So, deeper networks are harder to train.
DEEP ARCHITECTURES

- Deeper networks are able to express more complex functions. Larger hypothesis space.
- Empirically, deeper networks tend to overfit less than wider networks (i.e., more layers better than more channels).
- But, you need to back-propagate gradients of the loss computed at the final output back through a larger number of layers.
- With multiple non-linearities, gradients have a higher probability of vanishing.
- So, deeper networks are harder to train.
- Solution: provide an easier path for gradients back to initial layers.

DEEP ARCHITECTURES

Deeply-Supervised Nets

Deep Residual Learning for Image Recognition

Kaiming He Xiangyu Zhang Shaoqing Ren Jian Sun
Microsoft Research
{kahe, v-xiangz, v-shren, jiansun}@microsoft.com

Figure 2. Residual learning: a building block.
Deep Residual Learning for Image Res

Kaiming He  Xiangyu Zhang  Shaoqing Ren
Microsoft Research
{khe, x-zhang, s-rene}@microsoft.com

Figure 2. Residual learning: a building block.

Abstract

Recent work has shown that convolutional networks can be substantially deeper, more accurate, and efficient to train if they contain shorter connections between layers close to the input and those close to the output. In this paper, we embrace this observation and introduce the Dense Convolutional Network (DenseNet), which connects every layer to every other layer. This architecture encourages feature propagation, which we show empirically has several compelling advantages: they alleviate the vanishing-gradient problem, improve accuracy, encourage feature reuse, and substantially reduce the number of parameters.

Densely Connected Convolutional Networks

Guo Huang* Cornell University
ghuang@cornell.edu
Zhang Liu* Tsinghua University
liuzhang38@mails.tsinghua.edu.cn
Kilian Q. Weinberger Cornell University
kw@cs.cornell.edu
Laurens van der Maaten Facebook AI Research
laurens@fb.com

Figure 1: A 5-layer dense block, with a growth rate of δ = 4. Each layer takes all preceding feature maps as input.
MORE ABOUT ARCHITECTURES

Do this for all shifts.

Still Sharing Computation!
- We need downsampling because that quickly increases receptive field.

- But, we need to get an output for every pixel. So we apply the network on all overlapping receptive fields: efficiently using dilated convolutions.

But at coarser resolutions, these units have a very high overlap ratio, and likely redundant information.

But we still need to compute them with our current architecture: because the values won't exactly be the same.
MORE ABOUT ARCHITECTURES

- We need downsampling because that quickly increases receptive field.
- But, we need to get an output for every pixel. So we apply the network on all overlapping receptive fields: efficiently using dilated convolutions.

![Diagram]

But at coarser resolutions, these units have a very high overlap ratio, and likely redundant information.

But we still need to compute them with our current architecture: because the values won’t exactly be the same.

Solution: Change the architecture. Build one that goes from image to image, using downsampling to increase receptive field and then upsampling to get back to the original resolution, within the network itself.
As it suggests, it is the transpose of the operation of Convolution with Stride.

In fact, this represents the operation for back-propagating gradients through a convolution-with-stride layer.

Lets go back to our matrix vector notation, represent convolution with $A_k$ and downsampling with $A_s$.

$$y = A_s A_k x$$

- What is the transpose of this operation? Of $A_s A_k$?
  $$ (A_s A_k)^T = A^T_s A^T_k$$

- What does $A^T_k$ represent?
  - Upsampling by filling in zeros. $A^T_k$ is still convolution (with a flipped kernel, but doesn't matter).
  - So a convolution-transpose layer effectively does up-sampling with zeros, and then a regular convolution.
  - But up-sampling with zeros often leads to artifacts. Newer architectures don't use convolution transpose.
    Instead, they do bilinear or nearest-neighbor interpolation on the feature maps to increase resolution, and then do a regular convolution.