CSE 559A: Computer Vision

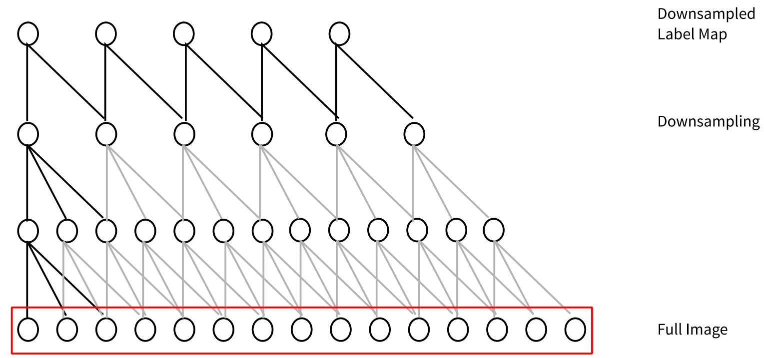
Fall 2018: T-R: 11:30-1pm @ Lopata 101

Instructor: Ayan Chakrabarti (ayan@wustl.edu).
Course Staff: Zhihao Xia, Charlie Wu, Han Liu

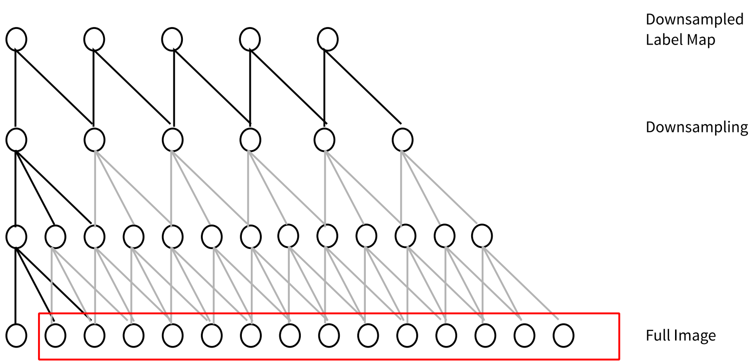
http://www.cse.wustl.edu/~ayan/courses/cse559a/

November 15, 2018

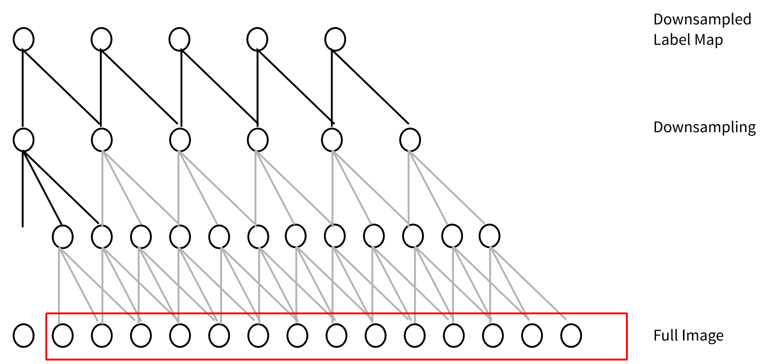
MORE ABOUT ARCHITECTURES



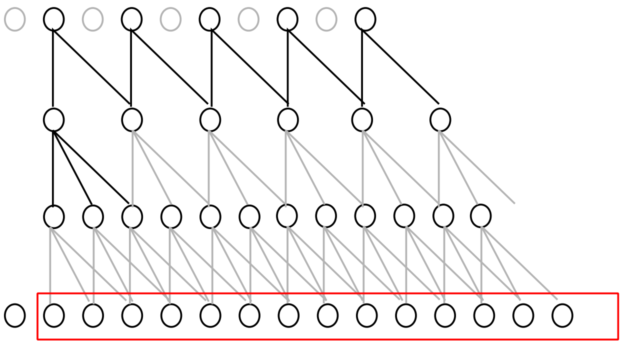
MORE ABOUT ARCHITECTURES



MORE ABOUT ARCHITECTURES



MORE ABOUT ARCHITECTURES



Downsampled
Label Map

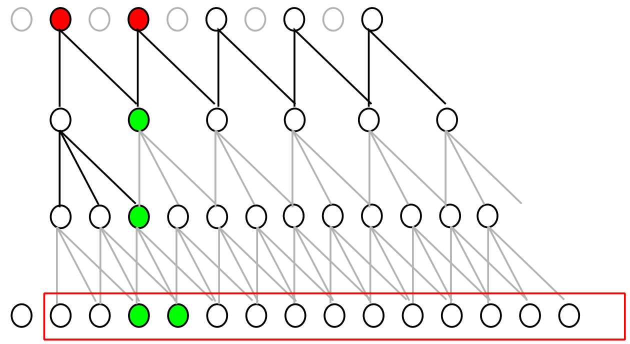
Downsampling

Full Image

MORE ABOUT ARCHITECTURES

Do this for all shifts.

Still Sharing Computation !



Downsampled
Label Map

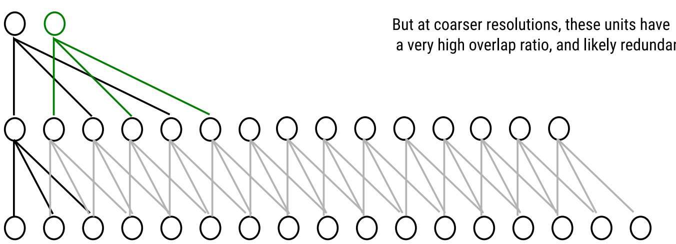
Downsampling

Full Image

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.

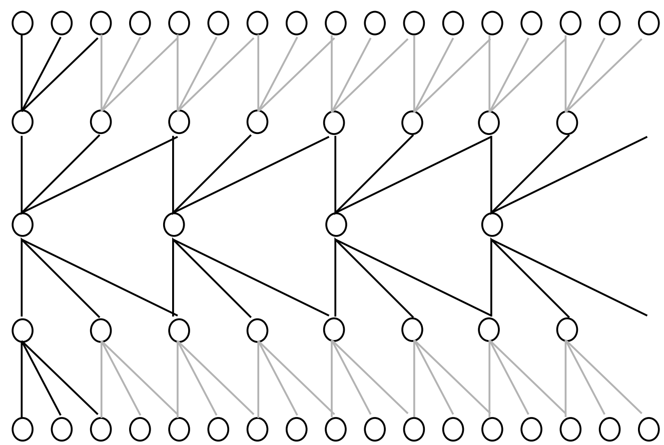


But at coarser resolutions, these units have a very high overlap ratio, and likely redundant information.

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.

MORE ABOUT ARCHITECTURES

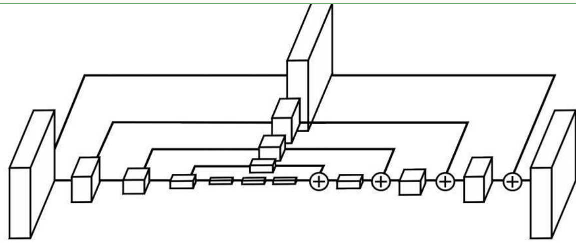


Convolution with
Up-sampling

Fractional-stride

Convolution Transpose

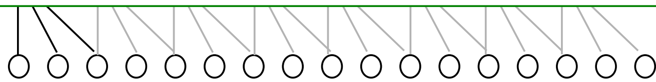
MORE ABOUT ARCHITECTURES



Newell et al., Hour-glass Networks.

Another popular architecture: U-Net

Notice skip connection at the same resolution.



TRANSPOSE CONVOLUTION

- 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 $A_s A_k$?

$$(A_s A_k)^T = A_k^T A_s^T$$

- What does A_s^T represent ?
- Upsampling by filling in zeros. A_k^T 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.

TRAINING IN PRACTICE

- Remember: Gradient Descent is Fragile

The Effect of Parameterization

$$f(x;\theta) = \theta\, x$$
$$f'(x;\theta') = 2\theta'\, x$$

- Two representations of the same hypothesis space.
- Let's initialize $\theta' = \theta/2$. Say $\theta = 10, \theta' = 5$.
- Compute loss with respect to the same example.
  - Say $\nabla_f L = \nabla_{f'} = 1, x = 1$
  - What are ∇_θ and $\nabla_{\theta'}$?
- $\nabla_\theta = 1, \nabla_{\theta'} = 2$.
- Update with learning rate = 1
- Updated $\theta = 9, \theta = 3$
- $f(x) = 9x, f'(x) = 6x$

TRAINING IN PRACTICE

Initialization

- Because we're using a first order method, it is important to make sure that the activations of all layers are the same magnitude.
- Because we have RELUs $y = \max(0, x)$, it is important to make sure roughly half the expectations are positive.
- Normalize your inputs to be 0-mean and unit variance.
  - Compute dataset mean and standard deviation, subtract and divide from all inputs.
  - For images, you usually compute the mean over all pixels---so single normalization for all pixels in the input.
    - But different for different color channels.
  - Sometimes, you will want 'per-location' normalization.
  - Other times, you will normalize each input by its own pixel mean and standard deviation.

TRAINING IN PRACTICE

Initialization

- Initialize all biases to 0. Why ? Don't want to shift the mean.
- Now initialize weights randomly so that variance of outputs = variance of inputs = 1.
    - Use approximation $var(wx) = var(w)var(x)$ for scalar w and x.
- Say you have a fully connection layer: $y = W^T x$
    - x is N-dimensional. We assume its 0-mean unit-variance coming in.
    - W is $N \times M$ dimensional.
    - We will initialize W with 0-mean and variance σ^2.
    - What should be the value of σ^2 ? Take 5 mins.
- $\sigma^2 = 1/N.$
- Now what about a convolution layer with kernel size $K \times K \times C_1 \times C_2$.
    - Initialize kernel with 0-mean and variance σ^2. What should σ^2 be ?
- $\sigma^2 = 1/(K^2 C_1)$

TRAINING IN PRACTICE

Initialization

- Actually, using normal distributions is sometimes unstable.
- Probability for values very far from the mean is low, but not 0.
- When you sample millions of weights, you might end up with such a high value!
- Solution: Use truncated distributions that are forced to have 0 probability outside a range.
    - Uniform distribution, "truncated-normal"
    - Figure out what parameters of this distribution should be to have equivalent variance.

TRAINING IN PRACTICE

Initialization

- But this only ensures zero-mean unit-variance at initialization.
- As your weights update, they can begin to give you biased weights.
- Another option, add normalization in the network itself !

Sergey Ioffe and Christian Szegedy, "Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift".

BATCH-NORMALIZATION

- **Batch-Norm Layer**

$$y = BN(x)$$

$$y = \frac{x - \text{Mean}(x)}{\sqrt{\text{Var}(x) + \epsilon}}$$

Here, mean and variance are interpreted per-channel

So for each channel, you compute mean of the values of that channel activation at all spatial locations

across all examples in the training set.

But this would be too hard to do in each iteration. So the BN layer just does this normalization over a batch.

And back-propagates through it.

BATCH-NORMALIZATION

- **Batch-Norm Layer**

$$x = (x)_{bijc}$$

$$\mu_c = \frac{1}{BHW} \sum_{bij} x_{bijc}$$

$$\sigma_c^2 = \frac{1}{BHW} \sum_{bij} (x_{bijc} - \mu_c)^2$$

$$y_{bijc} = \frac{x_{bijc} - \mu_c}{\sqrt{\sigma_c^2 + \epsilon}}$$

- The BN layer has no parameters.

What about during back-propagation ?

- When you back-propagate through it to x, you go back-propagate through the computation of mean and variance.
- At test time, replace μ_c and σ_c^2 as mean and variance over the entire training set.

BATCH-NORMALIZATION

- Typically apply BN before a RELU.
- Typical use: RELU(BN(Conv(x,k))+b)
  - Don't add bias before BN as its pointless
  - Learn bias post-BN
  - Can also learn a scale: RELU(a BN(Conv(x,k))+b)
- Leads to significantly faster training.
- But you need to make sure you are normalizing over a diverse enough set of samples.