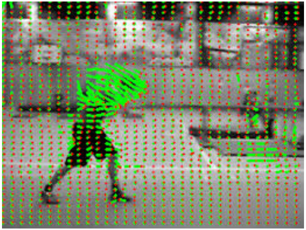
Deep Networks for Image-to-Image Prediction

COMPSCI 527 — Computer Vision

Outline

Image-to Image Prediction

- Recognition: 1 image $\rightarrow$ K label scores (funnel)
- Motion estimation: 2 images $\rightarrow$ 2 images
- Image segmentation: 1 image $\rightarrow$ K score images (K soft-max scores at every pixel)

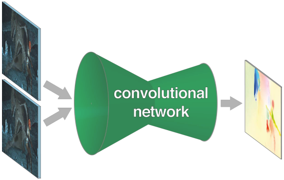


www.irisa.fr/texmex/people/jain

sthalles.github.io/deep_segmentation_network/

Architecture of Image-to Image Predictors

- The output is as large as the input
- *Retinotopic output*: values map to pixel locations
- The funnel-like architecture cannot be used
- An *hourglass* architecture is used instead



(image from Dosovitskiy *et al.*, FlowNet, 2015)

- A. k. a. *contraction-expansion*, *encoder-decoder*, . . .
- Let's see image motion estimation first,
  then image segmentation

Classical Approaches to Motion Estimation

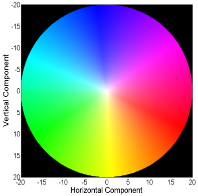
- For decades, global methods were cast as optimization problems to be solved at inference time
- Roughly: Find a flow field $\mathbf{u}(\mathbf{x})$ such that

$$\int [g(\mathbf{x} + \mathbf{u}(\mathbf{x})) - f(\mathbf{x})]^2 \, d\mathbf{x} + \lambda \int \left\| \frac{\partial \mathbf{u}}{\partial \mathbf{x}^T} \right\|^2 \, d\mathbf{x} \text{ is small}$$

- The resulting normal equation is discretized, and leads to a large, linear system in the unknowns $\mathbf{u}(\mathbf{x})$, one 2-vector per pixel
- The flow is not smooth at motion boundaries, various techniques have been proposed to improve results there
- However, these methods seem to work fairly well, see https://people.csail.mit.edu/celiu/OpticalFlow/

Why Use Neural Networks?

- A method based on neural networks needs many examples $(\mathbf{x}, y) = ((f, g), \mathbf{u})$



Why Use Neural Networks?

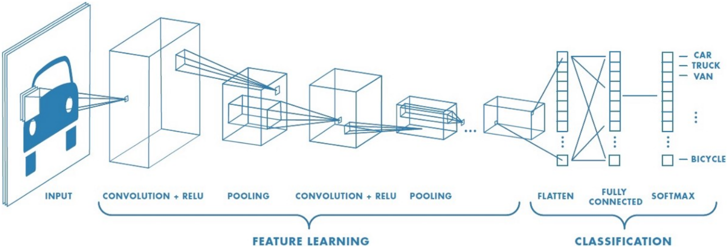
- Annotation is difficult: Hundreds of thousands or millions of flow vectors per example
- How do we know the flow at every pixel anyway?
- So why bother with deep learning?
- *Replace a complex optimization algorithm run at inference time with a deep network*
- At inference time, feed two images to a network and read the result at the output: *fast inference*
- Training is an even more complex optimization problem, but runs at training time
- Optimization assumes a very specific motion model. The neural network does not
- Therefore, *a neural network might do well even where the optimization algorithm doesn't*

Training Data and Loss

- Big question: How to annotate training data?
- Current best answer: computer graphics
- Sintel: http://sintel.is.tue.mpg.de
- Main limitation: Is graphics a good proxy for real video?
- Computer graphics is getting better and better
- Not hard to make good movies look worse!
- Loss: Discrepancy between true flow $\mathbf{v}(\mathbf{x})$ and computed flow $\mathbf{u}(\mathbf{x})$
- *End-Point Error (EPE)*: $\sqrt{\frac{1}{|\Omega|}\sum_{\mathbf{x}\in\Omega}\|\mathbf{u}(\mathbf{x}) - \mathbf{v}(\mathbf{x})\|^2}$

Architectures: The Recognition Funnel

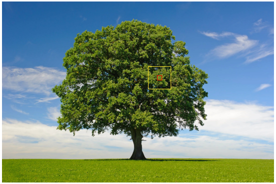
- A CNN used for classification looks like a funnel:



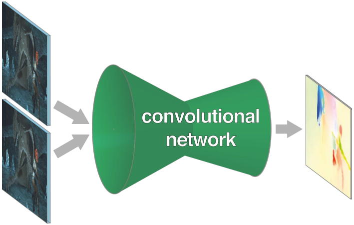
- Image in, category out
- Representation becomes more and more abstract
- For flow, the output is image-like, so the funnel won't work

Architectures: The Image-to-Image Hourglass

- However, abstraction is still useful

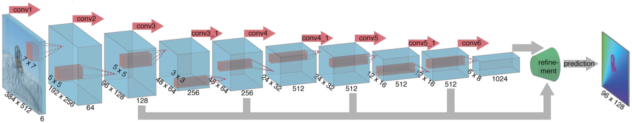


- Flow at low resolution may be coarse but less ambiguous
- First build an abstract view, then restore detail

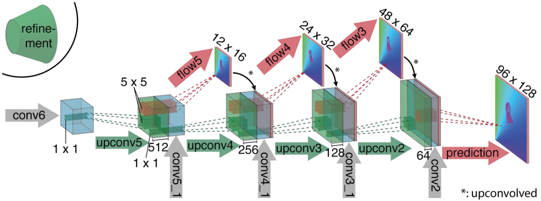


Architecture Detail: FlowNet, 2015

- *Encoder* (or contraction)



- *Decoder* (or expansion)



- Note the gray *skip connections* to restore detail

How to Decode: Up-Convolution

- We don't just want to upsample: Upsampling needs to be trainable
- *Up-convolution* is one way to upsample
- Best understood in the 1D case first
- Convolution with stride reduces resolution
- How to increase resolution instead?

Strided Convolution in Matrix Form

$g(y) = \sum_{x=0}^{p-1} k(x) f(sy - x)$

- Example: $\mathbf{f} \in \mathbb{R}^{12}$, stride $s = 2$, "same" format
  $\mathbf{k} = [a, b, c, d, e]$
- Then, $\mathbf{g} \in \mathbb{R}^{6}$ and $\mathbf{g} = K\mathbf{f}$ with $K \in \mathbb{R}^{6 \times 12}$

$$K = \begin{bmatrix} c & b & a & & & & & & & & & \\ e & d & c & b & a & & & & & & & \\ & & e & d & c & b & a & & & & & \\ & & & & e & d & c & b & a & & & \\ & & & & & & e & d & c & b & a & \\ & & & & & & & & e & d & c & b \end{bmatrix}$$

Up-Convolution

- The up-convolution corresponding to $\mathbf{g} = K\mathbf{f}$ is defined as $\varphi = K^T\mathbf{g}\,,\quad$ *not* the inverse of K

g_0	g_1	g_2	g_3	g_4	g_5
c	e				
b	d				
a	c	e			
	b	d			
	a	c	e		
		b	d		
		a	c	e	
			b	d	
			a	c	e
				b	d
				a	c
					b

Rewrite Up-Convolution as a Convolution

- *Dilute* **g** into γ with stride $s = 2$:

  $(g_0, g_1, g_2, g_3, g_4, g_5) \rightarrow (g_0, 0, g_1, 0, g_2, 0, g_3, 0, g_4, 0, g_5, 0)$

γ_0 g_0	γ_1 0	γ_2 g_1	γ_3 0	γ_4 g_2	γ_5 0	γ_6 g_3	γ_7 0	γ_8 g_4	γ_9 0	γ_{10} g_5	γ_{11} 0
c		*e*									
b		*d*									
a		*c*		*e*							
		b		*d*							
		a		*c*		*e*					
				b		*d*					
				a		*c*		*e*			
						b		*d*			
						a		*c*		*e*	
								b		*d*	
								a		*c*	
										b	

- Square matrix
- Can fill new columns with anything we like

Up-Convolution as a Convolution

γ_0	γ_1	γ_2	γ_3	γ_4	γ_5	γ_6	γ_7	γ_8	γ_9	γ_{10}	γ_{11}
g_0	0	g_1	0	g_2	0	g_3	0	g_4	0	g_5	0
c	*d*	*e*									
b	*c*	*d*	*e*								
a	*b*	*c*	*d*	*e*							
	a	*b*	*c*	*d*	*e*						
		a	*b*	*c*	*d*	*e*					
			a	*b*	*c*	*d*	*e*				
				a	*b*	*c*	*d*	*e*			
					a	*b*	*c*	*d*	*e*		
						a	*b*	*c*	*d*	*e*	
							a	*b*	*c*	*d*	*e*
								a	*b*	*c*	*d*
									a	*b*	*c*

- Up-convolution is the convolution of a diluted input with the reverse of the original kernel k, that is, with

$$\kappa(y) \stackrel{\text{def}}{=} k(p-1-y)$$

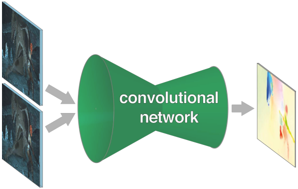
- Up-convolution can be written as follows:

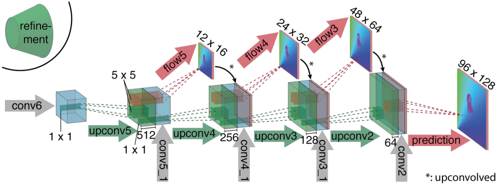
$$\phi(x) = \sum_{y=0}^{p-1} \kappa(y)\gamma(x-y)$$

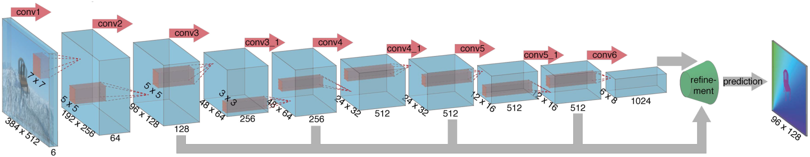
Up-Convolution Summary

- To reduce resolution, convolve and then sample
- Efficiently, do convolution with stride:
  $g(y) = \sum_{x=0}^{p-1} k(x) f(sy - x)$
- To increase resolution, dilute and then convolve
- Efficiently, do diluted convolution
  $\phi(x) = \sum_{y=0}^{p-1} \kappa(y) \gamma(x - y)$
  where $\gamma(y) = \begin{cases} g\left(\frac{y}{s}\right) & \text{if } y \overset{s}{=} 0 \\ 0 & \text{otherwise} \end{cases} \quad \text{for} \quad 0 \leq y \leq sn$
- More efficiently: $\phi(x) = \sum_{y \overset{s}{=} x,\ y=0}^{p-1} \kappa(y)\, g\left(\frac{x-y}{s}\right)$

FlowNet, 2015





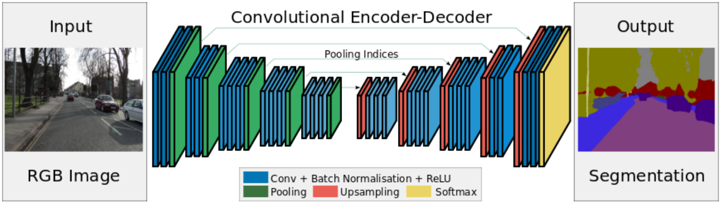


Demos at https://www.youtube.com/watch?v=JSzUdVBmQP4

Unsupervised Training?

- Loss based on End-Point Error: $\|\mathbf{u}(\mathbf{x}) - \mathbf{v}(\mathbf{x})\|^2$
- Requires supervision $\mathbf{v}$
- Loss based on Photometric Error + Regularization Term: $[g(\mathbf{x} + \mathbf{u}(\mathbf{x})) - f(\mathbf{x})]^2 + \lambda \left\| \frac{\partial \mathbf{u}}{\partial \mathbf{x}^T} \right\|^2$
- Only f, g are needed
- Issue: Correct flow implies small loss, but the converse is not necessarily true, mainly because of the aperture problem
- Works, but not as well
- However, we can bring massive amounts of data to bear

Architectures for Image Segmentation



https://mi.eng.cam.ac.uk/projects/segnet/ (2015)

- Still encoder-decoder
- *K* soft-max scores per pixel
- Pooling in the encoder
- Upsamples by “unpooling”, copies pooling indices from the encoder