Image Completion / Inpainting

- Goal: Remove object(s) from digital photographs, and then fill the hole with information extracted from the surrounding area, **preserving image structures**
- Issues:
  - Underconstrained problem
  - Requires inference about regions, boundaries, textures
  - Data-driven but needs scene semantics
  - Result should “look real” without seams or perceptual artifacts


Manual Editing using Photoshop

Photoshop CS5 “Content-Aware Fill”



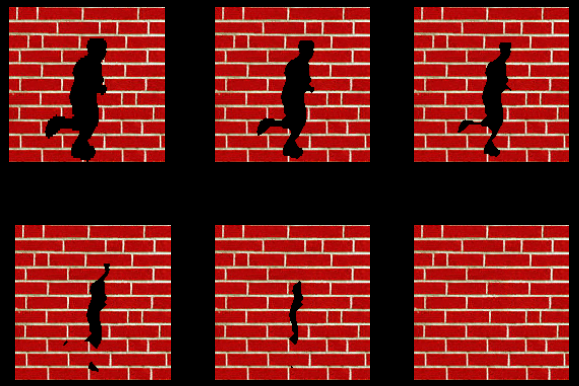

before

after

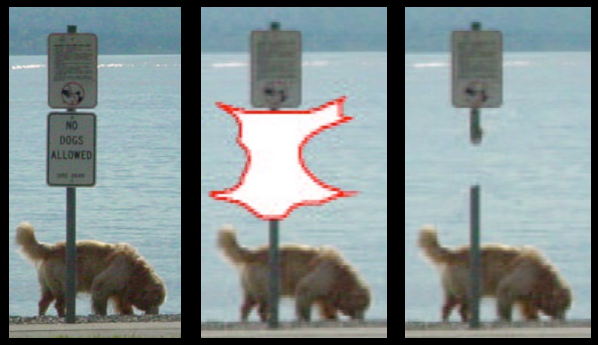
YouTube demos: http://www.youtube.com/watch?v=PTvxlfBW96k

Fundamental Issue: Texture vs. Structure

"Onion Skin" Filling Order



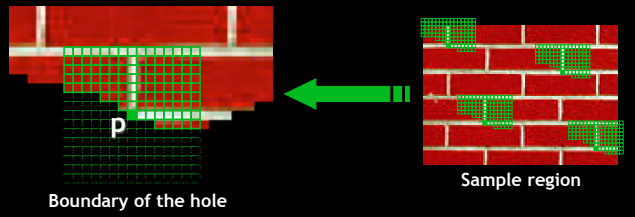
Standard Fill Order: "Onion Skin"



Original image | Image with hole | Filling in "onion skin" order

May lose linear structures and boundaries b/w texture regions

Hole Filling by Texture Synthesis



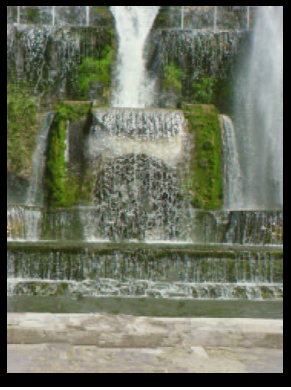
- Cheap and easy, but the filling order is crucial in preserving structure

Fill Order

- In what order should we fill the pixels?
  - choose pixels that have more neighbors already filled
  - choose pixels that are continuations of lines/curves/edges so that linear structures are propagated

Image Completion by Example-Based Inpainting

- A. Criminisi, P. Perez, and K. Toyama, CVPR 2003

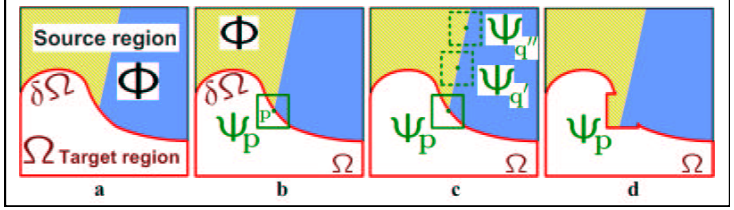


Criminisi's Approach

- Combine the strengths of texture synthesis and inpainting approaches
  - Use a texture synthesis algorithm to quickly fill patches with similar texture regions
  - ***Fill patches near linear structures first***, thereby finding structure/texture boundaries first, and then filling in the texture inside regions
- Result
  - Preserve linear structures
  - Avoid blurring

Best-First Filling Algorithm

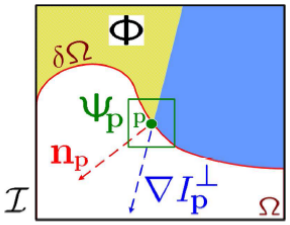
- Extract the initial fill-front, $\delta\Omega$, and compute priority values, $P(\mathbf{p})$, for all pixels $\mathbf{p}$ in $\delta\Omega$
- Repeat until no more unfilled pixels remain:
  1. Find target pixel $\mathbf{p}$ and patch $\Psi_{\mathbf{p}}$ with highest "priority," $P(\mathbf{p})$
  2. Find source patches, $\Psi_{\mathbf{q}'}$, $\Psi_{\mathbf{q}''}$, that most closely match the target patch
  3. Paste most similar source patch into target location
  4. Update fill front and priority values



Computing Fill Order

- Assign each pixel **p** centered on a patch Ψ_p a **priority value** *P*(**p**) that is a product of a confidence term, *C*(**p**), and data term, *D*(**p**):

$$P(\mathbf{p}) = C(\mathbf{p})D(\mathbf{p})$$



Confidence term:

$$C(\mathbf{p}) = \frac{\sum_{\mathbf{q}\in\Psi_\mathbf{p}\cap\bar{\Omega}} C(\mathbf{q})}{|\Psi_\mathbf{p}|}$$

$\bar{\Omega} = I - \Omega$

Measures fraction of *P*'s neighbors already filled

Data term:

$$D(\mathbf{p}) = \frac{|\nabla I_\mathbf{p}^\perp \cdot \mathbf{n}_\mathbf{p}|}{\alpha}$$

Measures strength and orientation of edge at *P*

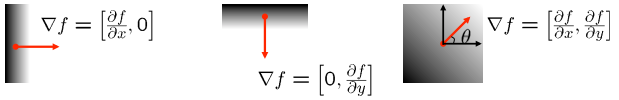
C measures fraction of patch already filled
D measures the degree to which an edge ends at **p**
α is a constant

Image Gradient

The gradient of an image:

$$\nabla f = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]$$

The gradient points in the direction of most rapid change in intensity

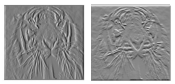


The gradient direction (*orientation* of edge normal) is given by:

$$\theta = \tan^{-1}\left(\frac{\partial f}{\partial y} / \frac{\partial f}{\partial x}\right)$$

The edge *strength* is given by the gradient magnitude

$$\|\nabla f\| = \sqrt{\left(\frac{\partial f}{\partial x}\right)^2 + \left(\frac{\partial f}{\partial y}\right)^2}$$



Slide credit Steve Seitz

Derivatives with Convolution

For 2D function, *f*(*x*, *y*), the partial derivative is:

$$\frac{\partial f(x,y)}{\partial x} = \lim_{\varepsilon\to 0} \frac{f(x+\varepsilon, y) - f(x,y)}{\varepsilon}$$

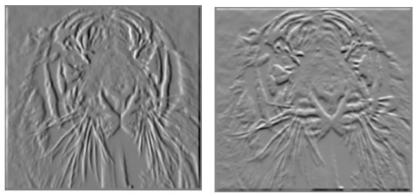
For discrete data, we can approximate using ***finite differences***:

$$\frac{\partial f(x,y)}{\partial x} \approx \frac{f(x+1,y) - f(x,y)}{1}$$

To implement using convolution, what would be the associated filter?

Partial Derivatives of an Image

$\frac{\partial f(x,y)}{\partial x}$



$\frac{\partial f(x,y)}{\partial y}$

-1	1

-1
1

or

1
-1

Assorted Finite Difference Filters

Prewitt: $M_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}$; $M_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix}$

Sobel: $M_x = \begin{bmatrix} -1 & 0 & 1 \\ -2 & 0 & 2 \\ -1 & 0 & 1 \end{bmatrix}$; $M_y = \begin{bmatrix} 1 & 2 & 1 \\ 0 & 0 & 0 \\ -1 & -2 & -1 \end{bmatrix}$

Roberts: $M_x = \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix}$; $M_y = \begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix}$

```
>> My = fspecial('sobel');
>> outim = imfilter(double(im), My);
>> imagesc(outim);
>> colormap gray;
```

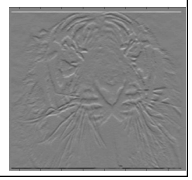


Image Gradient

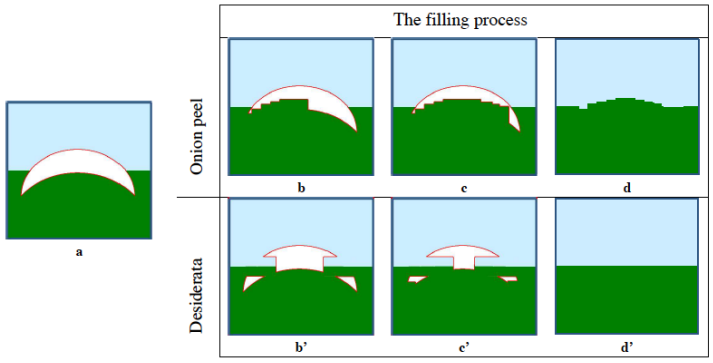
In Matlab:

[Gmag, Gdir] = imgradient(f)

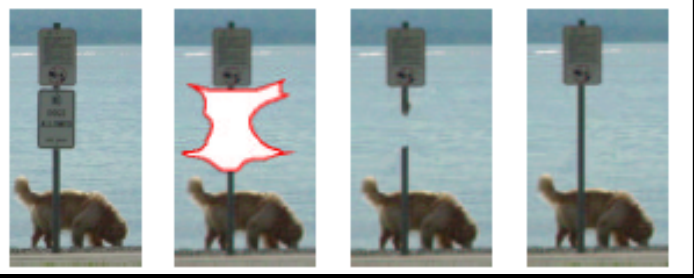
- uses Sobel operator by default

[Gmag, Gdir] = imgradient(f, 'prewitt')
uses Prewitt operator
Etc.

Filling Order: Onion-Skin vs Edge-driven



Results



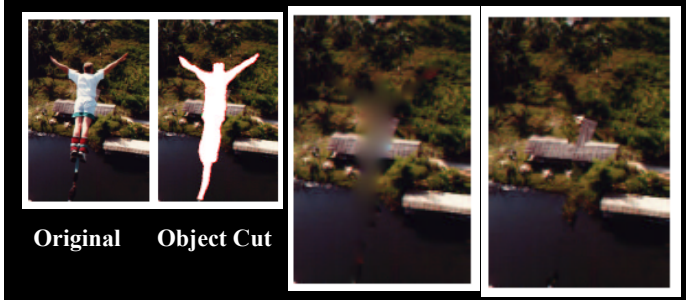
Original | Object Cut | Texture Synth | Criminisi

Visualization of Fill Order

http://research.microsoft.com/en-us/projects/i3l/patchworks.aspx

Results: Filling Large Regions

12% of image area filled



Original **Object Cut** **Texture Synthesis** **Criminisi**

Results: Filling Large Regions

Results: Filling Large Regions

10% of image area filled

Results

Inpainting Examples

http://research.microsoft.com/en-us/projects/i3l/patchworks.aspx

Inpainting Examples

Image Completion with Structure Propagation

J. Sun, L. Yuan, J. Jia, and H. Shum
SIGGRAPH 2005

Image Completion with Structure Propagation

- The method of Criminisi *et al*. does **not** guarantee continuity of salient structures such as curves or junctions
- Missing structure is hard to recover automatically, but can be easily specified **manually**

Algorithm

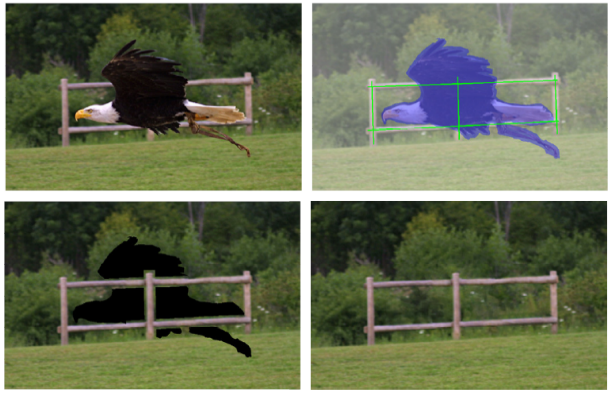
1. User ***interactively*** draws a few curves on the image that extend from given structures into hole, specifying most salient missing structures and partitioning image into regions
2. Synthesize patches along the curves, defining structure boundaries
3. Fill in remaining holes using patch-based texture synthesis with samples from same segmented regions

Example Result

Comparison with Criminisi *et al*.

Sun *et al*. **Criminisi *et al*.**

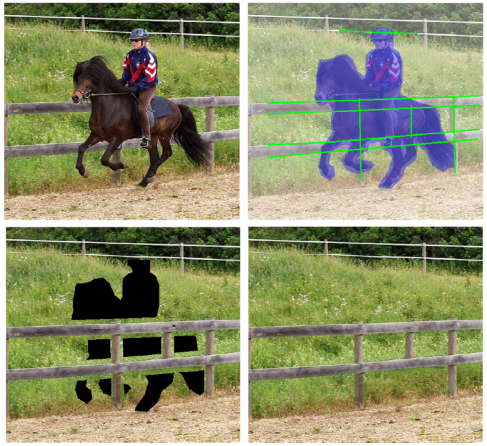
Example Result



Comparison with Criminisi *et al*.

Sun *et al.* **Criminisi *et al.***

Example Result



Comparison with Criminisi *et al*.

Sun *et al.* **Criminisi *et al.***

Scene Completion Using Millions of Photographs

James Hays and Alexei A. Efros
SIGGRAPH 2007

Slides by J. Hays and A. Efros

Efros and Leung (pixel-based texture synthesis) result

Criminisi *et al*. (structure-based fill order) result

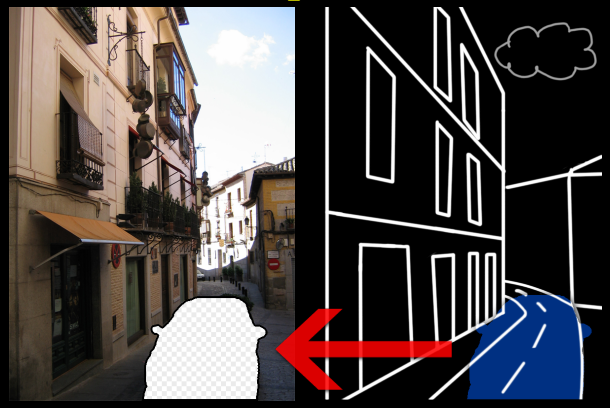
Contour extension gone wrong

Criminisi *et al*. result

Microsoft Digital Image Pro Smart “erase” result

Need more info on **scene semantics**

Scene Semantics for Image Completion



Manually added

Sun *et al*. result

Sun *et al*. result

Data

2.3 Million unique images from Flickr groups and keyword searches

landscapes, travel, city, outdoors, vacation, etc.

Scene Completion Result

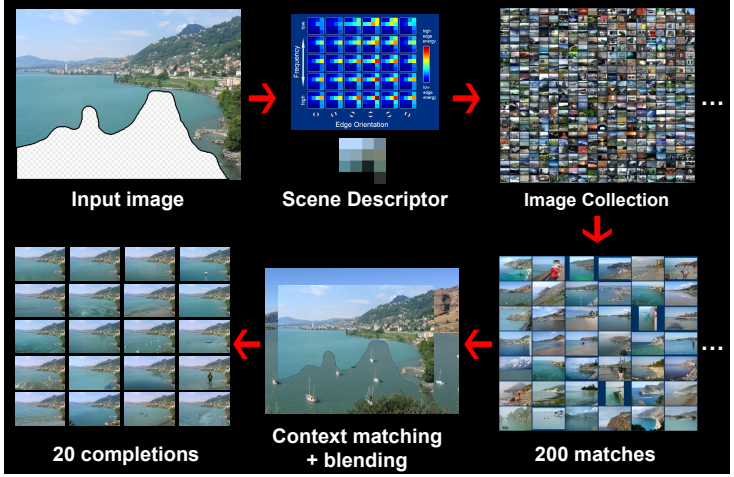
Challenges

- Computational
  - How to efficiently search millions or billions of source images?
- Finding semantically appropriate image fragments
  - Generically (e.g., car) and specifically (red BMW SUV)
- Differences in appearance
  - Lighting, color, shadows, etc.

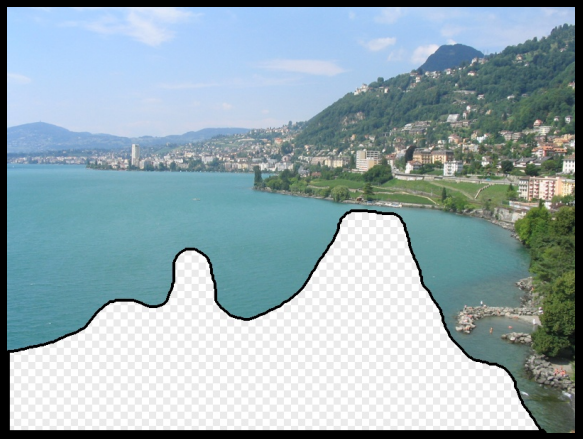
Finding a Good Fill Patch

1. Find a set of candidate images that are "semantically similar scenes" to the input image
   – Don't assume image tags or structured datasets (e.g., ImageNet)
2. Find most similar patch from each candidate image to fill hole

The Algorithm



Scene Matching



How to Find Semantically-Similar Scenes? The "Gist" of a Scene

If this is a street, this must be a pedestrian

***Gist*: abstract meaning of scene; low-dim., global image rep**

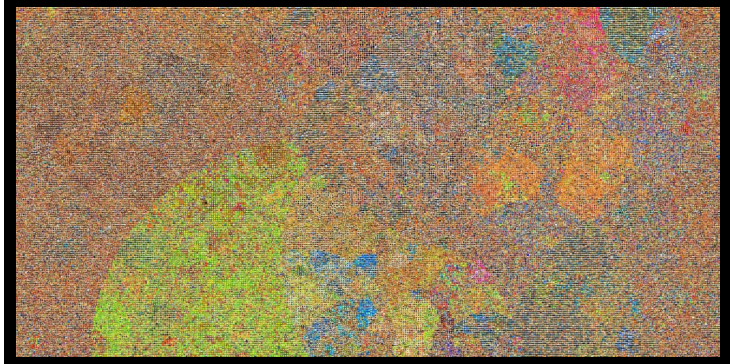
- Obtained within 150 ms *(Biederman, 1981, Thorpe S. et.al 1996)*
- Obtained without attention *(Oliva & Schyns, 1997, Wolfe,J.M. 1998)*
- Possibly derived via statistics of low-level structures *(e.g. Swain & Ballard, 1991)*

Image Representations

- The space of all possible images is huge!
  - $32 \times 32 \times 8$ bit image $\Rightarrow 10^{7400}$ possible images
  - In 100 years a person only sees about 10^{11} images
- But the number of "natural" / "real" images is much, much smaller and there are semantic clusters
- How many images are needed to be able to find a similar image to match any given natural input image?

80 Million Tiny Images

A. Torralba, R. Fergus, and W. T. Freeman



Dataset available at: groups.csail.mit.edu/vision/TinyImages

What is the Gist?
A Scene-based Image Descriptor

What features are sufficient to represent a scene?

- **Statistics of local, low-level features**
- **Color histograms**
- **Oriented band-pass filters**

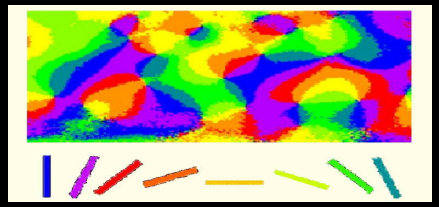
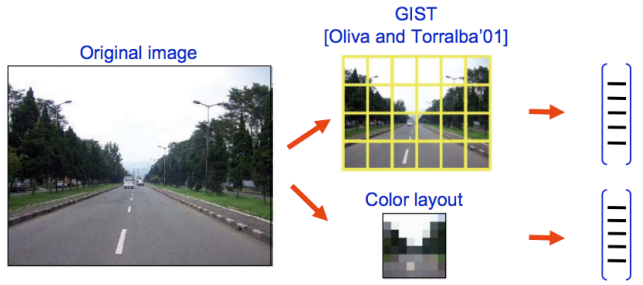


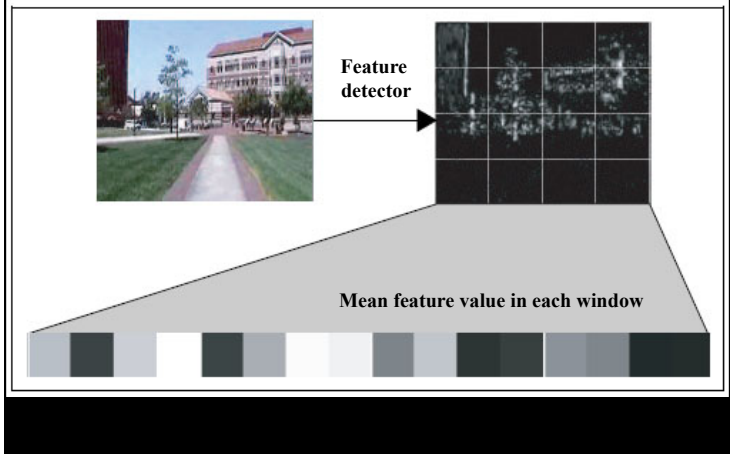
Image representation



Distance between images:

measured by Euclidean distance between image descriptors (GIST and color have equal weight) [Hays&Efros'07]

Gist Feature Extraction



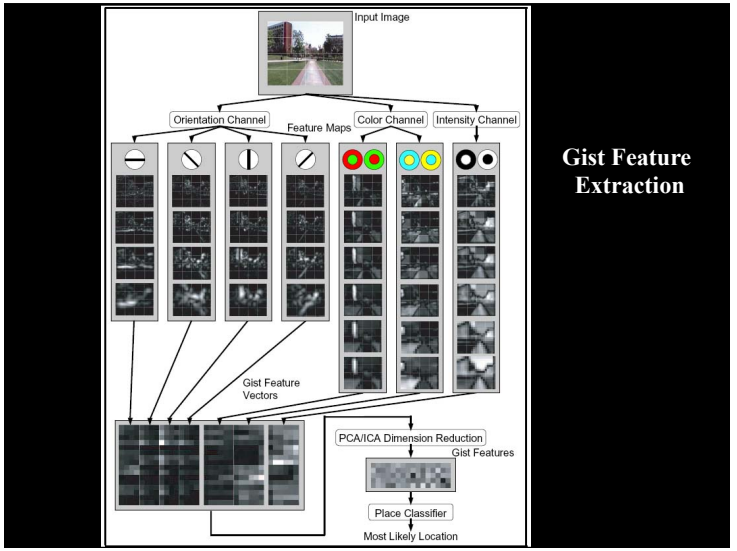
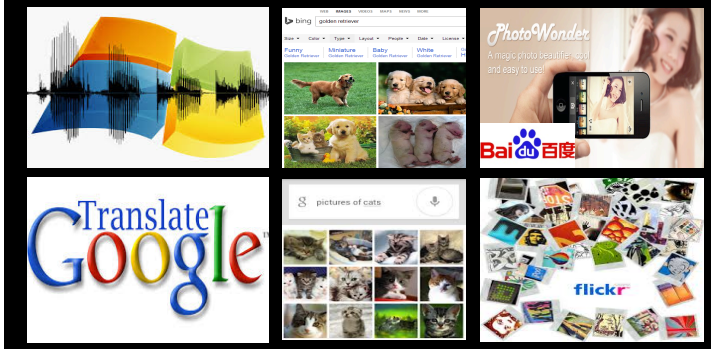


Image Features by Deep Learning

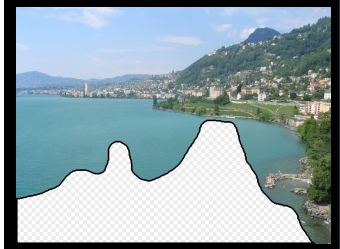


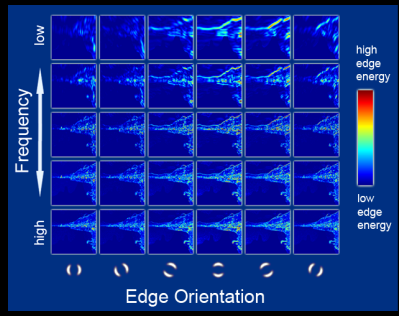
Deep Learning - breakthrough in visual and speech recognition

Learning a Hierarchy of Feature Extractors

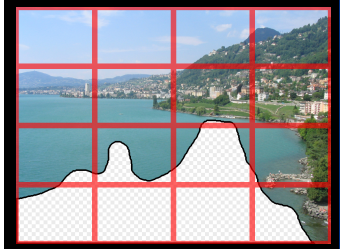
- Each layer of hierarchy extracts features from output of previous layer
- All the way from pixels → classifier / features
- Layers have (nearly) the same structure

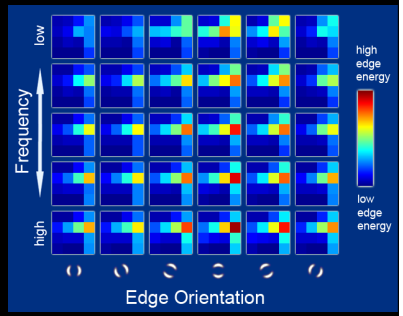
Scene Descriptor





Scene Descriptor

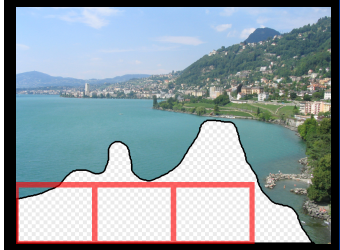


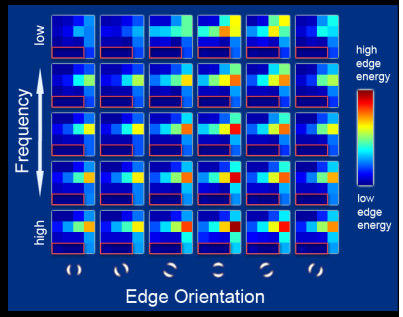


6 orientations, 5 scales, 4 x 4 spatial resolution

Gist scene descriptor (Oliva and Torralba 2001)

Scene Descriptor



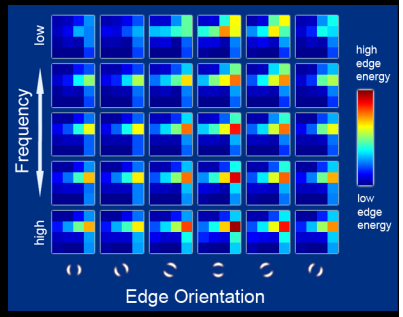


Exclude missing regions

Gist scene descriptor (Oliva and Torralba 2001)

Scene Descriptor

+

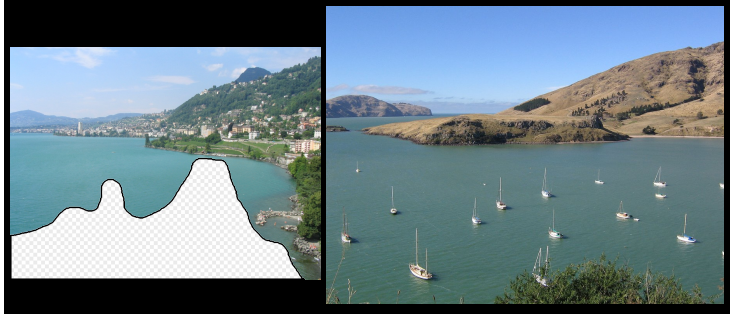


Color descriptor: color of the query image downsampled to 4 x 4

Distances calculated by SSD between query image descriptors & images in database

Gist scene descriptor (Oliva and Torralba 2001)

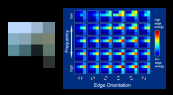
Step 2: Context Matching to Find Best Patch



- Query image template = band within 80 pixels of hole boundary
- SSD match with all translations and 3 scales of database image

Result Ranking

Assign each of the 200 results a score that is the sum of:

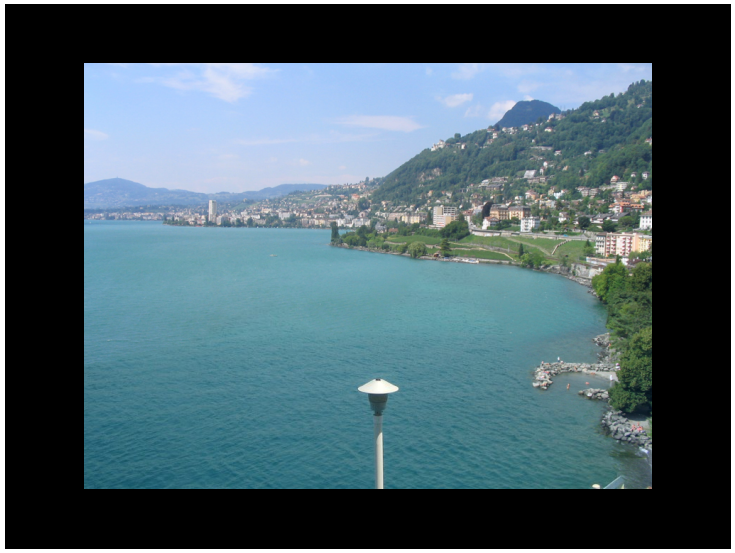


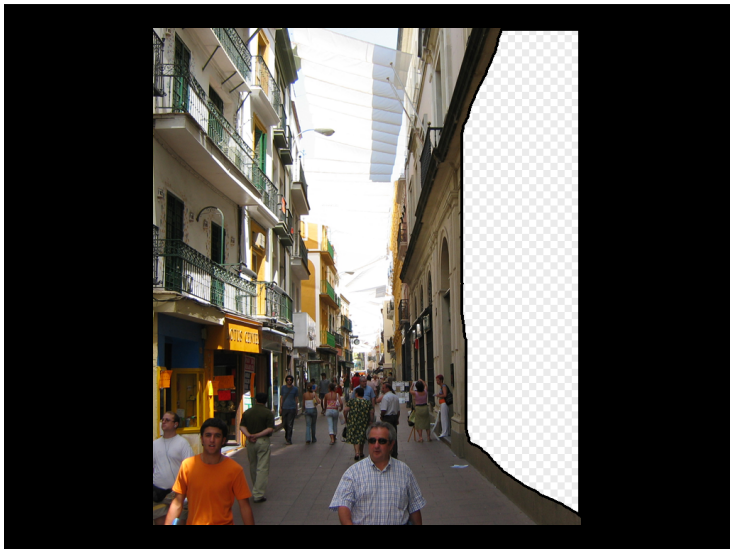
Scene matching distance

Context matching distance
(color + texture)

Graph-cut cost

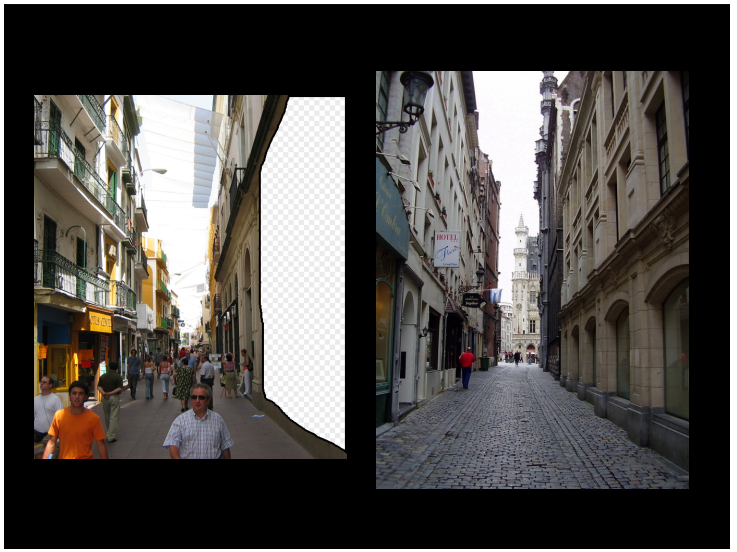
Top 20 Results

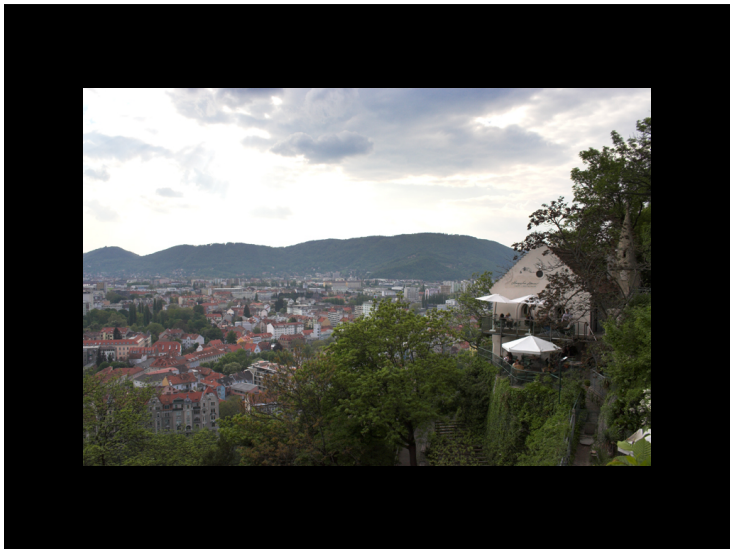


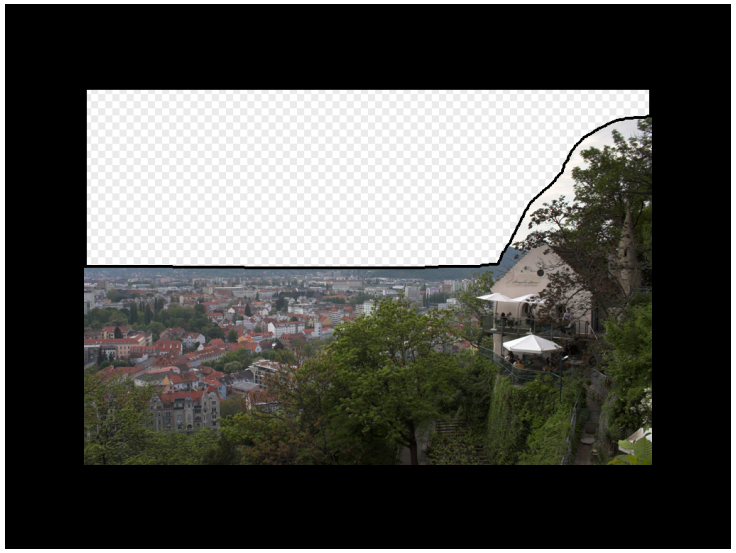


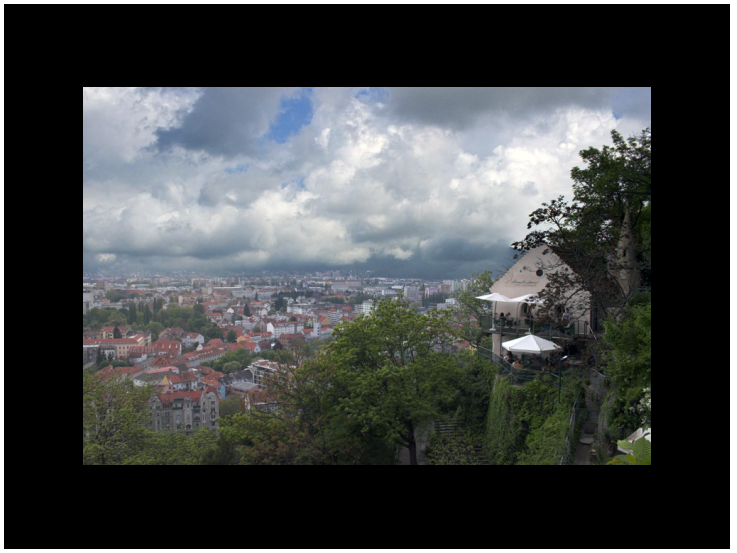

… 200 scene matches

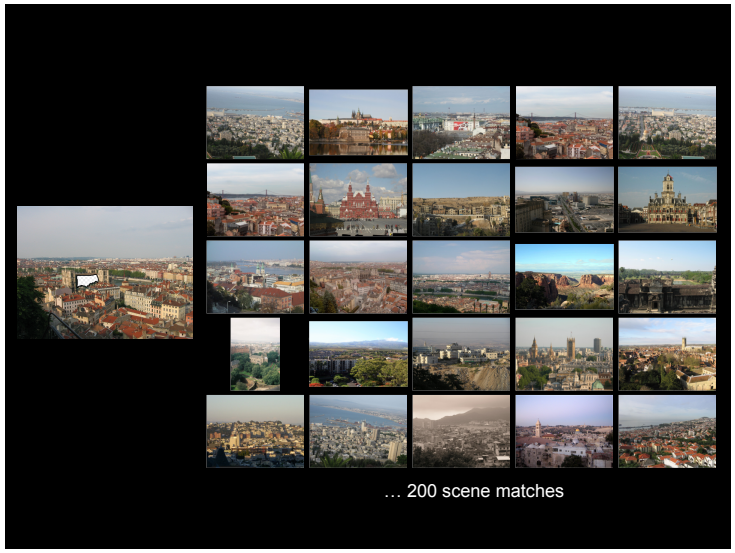

… 200 scene matches










… 200 scene matches

Failures

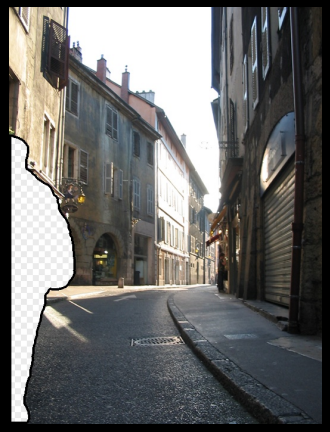
Failures

Failures

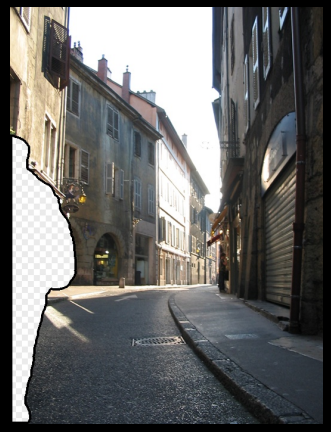
Failures

Failures

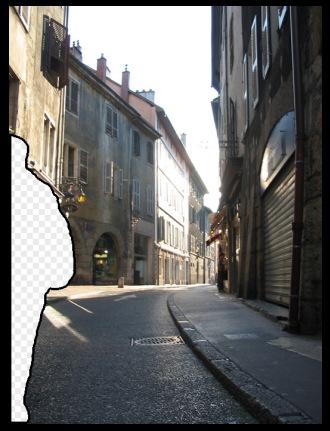
Failures



Failures

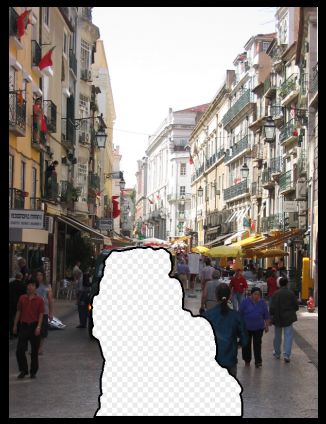


Failures

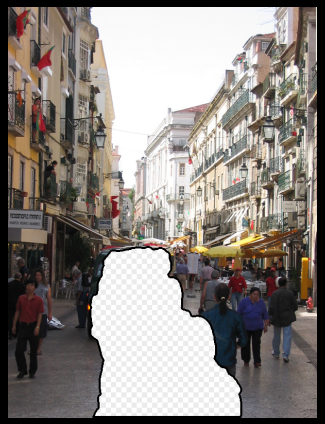


Failures

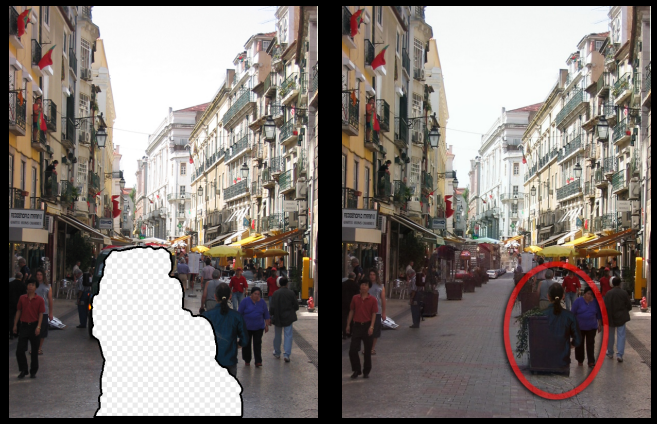
Failures



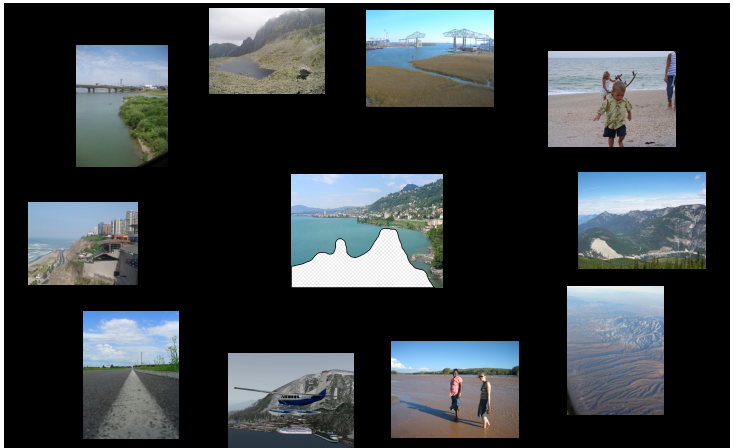
Failures



Failures



Why does it Work?



10 nearest neighbors from a collection of 20,000 images

10 nearest neighbors from a collection of 2 million images

Image Completion: Summary

- The key challenge is propagating the image "structure"
- Approaches:
  - For simple enough problems, the heuristic of extending existing edges (Criminisi *et al*.) is sufficient
  - For more complex problems, the user can provide the high-level structure information (Sun *et al*.)
  - Alternatively, high-level information can be obtained from a large database of images (Hays and Efros)