Texture

What is a texture?
Texture

What is a texture?
What is a texture?

Texture
Texture

• Texture: **stochastic** pattern that is **stationary**
  ("looks the same" at all locations)
• May be structured or random
Texture

Stochastic
Stationary
Texture

Stochastic     Stationary
Goal

- Computational representation of texture
  - Textures generated by same stationary stochastic process have same representation
  - Perceptually similar textures have similar representations
Applications

- Segmentation
- 3D Reconstruction
- Classification
- Synthesis

http://animals.nationalgeographic.com/
Applications

- **Segmentation**
- **3D Reconstruction**
- **Classification**
- **Synthesis**
Applications

• Segmentation
• 3D Reconstruction
• Classification
• Synthesis
Applications

- Segmentation
- 3D Reconstruction
- Classification
- Synthesis
Applications

- Segmentation
- 3D Reconstruction
- Classification
- Synthesis
Texture Representation?

- What makes a good texture representation?
  - Textures generated by same stationary stochastic process have same representation
  - Perceptually similar textures have similar representations
Statistics of filter banks
Filter-Based Texture Representation

• Research suggests that the human visual system performs local spatial frequency analysis (Gabor filters)

J. J. Kulikowski, S. Marcelja, and P. Bishop.
Theory of spatial position and spatial frequency relations in the receptive fields of simple cells in the visual cortex.
Texture Representation

• Analyze textures based on the responses of linear filters
  – Use filters that look like patterns (spots, edges, bars, …)
  – Compute magnitudes of filter responses

• Represent textures with statistics of filter responses within local windows
  – Histogram of feature responses for all pixels in window
Texture Representation Example

original image

derivative filter responses, squared

statistics to summarize patterns in small windows

<table>
<thead>
<tr>
<th>Win. #1</th>
<th>mean $\frac{d}{dx}$ value</th>
<th>mean $\frac{d}{dy}$ value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>4</td>
<td>10</td>
</tr>
</tbody>
</table>

Grauman
Texture Representation Example

original image

derivative filter responses, squared

<table>
<thead>
<tr>
<th></th>
<th>mean $\frac{d}{dx}$ value</th>
<th>mean $\frac{d}{dy}$ value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Win. #1</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>Win. #2</td>
<td>18</td>
<td>7</td>
</tr>
</tbody>
</table>

statistics to summarize patterns in small windows

Grauman
Texture Representation Example

original image

derivative filter responses, squared

<table>
<thead>
<tr>
<th></th>
<th>mean $\frac{d}{dx}$ value</th>
<th>mean $\frac{d}{dy}$ value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Win. #1</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>Win. #2</td>
<td>18</td>
<td>7</td>
</tr>
<tr>
<td>Win. #9</td>
<td>20</td>
<td>20</td>
</tr>
</tbody>
</table>

statistics to summarize patterns in small windows
Texture Representation Example

- Statistics to summarize patterns in small windows

<table>
<thead>
<tr>
<th>Dimension 1 (mean d/dx value)</th>
<th>Dimension 2 (mean d/dy value)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Win. #1</td>
<td>4</td>
</tr>
<tr>
<td>Win. #2</td>
<td>18</td>
</tr>
<tr>
<td>Win. #9</td>
<td>20</td>
</tr>
</tbody>
</table>
Texture Representation Example

Statistics to summarize patterns in small windows

<table>
<thead>
<tr>
<th>Dimension 1 (mean d/dx value)</th>
<th>Dimension 2 (mean d/dy value)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Win. #1</td>
<td>4</td>
</tr>
<tr>
<td>Win. #2</td>
<td>18</td>
</tr>
<tr>
<td>Win. #9</td>
<td>20</td>
</tr>
</tbody>
</table>

Far: dissimilar textures
Close: similar textures
Filter Banks

- Previous example used two filters, resulting in 2-dimensional feature vector
  - x and y derivatives revealed local structure
- Filter bank: many filters
  - Higher-dimensional feature space
  - Distance still related to similarity of local structure
Filter banks

• What filters to put in the bank?
  – Combination of different scales, orientations, patterns
You Try: Can you match the texture to the response?

Filters

A

B

C

1

2

3

Mean abs responses
Application: Retrieval

• Retrieve similar images based on texture

Textons

- Elements (“textons”) either identical or come from some statistical distribution
- Can analyze in natural images
Clustering Textons

- Output of bank of $n$ filters can be thought of as vector in $n$-dimensional space
- Can *cluster* these vectors using $k$-means [Malik et al.]
- Result: dictionary of most common textures
K-means clustering
Revisiting k-means

Most well-known and popular clustering algorithm:

Start with some initial cluster centers

Iterate:
- Assign/cluster each example to closest center
- Recalculate centers as the mean of the points in a cluster

Credit: David Kauchak
K-means: an example
K-means: Initialize centers randomly
K-means: assign points to nearest center
K-means: readjust centers
K-means: assign points to nearest center

Credit: David Kauchak
K-means: readjust centers

Credit: David Kauchak
K-means: assign points to nearest center

Credit: David Kauchak
K-means: readjust centers

Credit: David Kauchak
K-means: assign points to nearest center

No changes: Done

Credit: David Kauchak
Iterate:
- **Assign/cluster each example to closest center**
- Recalculate centers as the mean of the points in a cluster

How do we do this?

Credit: David Kauchak
K-means

Iterate:

- Assign/cluster each example to closest center
- Recalculate centers as the mean of the points in a cluster

Where are the cluster centers?
K-means

Iterate:

- Assign/cluster each example to closest center
- Recalculate centers as the mean of the points in a cluster

How do we calculate these?
K-means

Iterate:

- Assign/cluster each example to closest center
- Recalculate centers as the mean of the points in a cluster

Mean of the points in the cluster:

\[ \mu(C) = \frac{1}{|C|} \sum_{x \in C} x \]
K-means loss function

K-means tries to minimize what is called the “k-means” loss function:

\[
\text{loss} = \sum_{i=1}^{n} d(x_i, \mu_k)^2 \quad \text{where } \mu_k \text{ is cluster center for } x_i
\]

that is, the sum of the squared distances from each point to the associated cluster center.
Minimizing k-means loss

Iterate:

1. Assign/cluster each example to closest center
2. Recalculate centers as the mean of the points in a cluster

\[
\text{loss} = \sum_{i=1}^{n} d(x_i, \mu_k)^2 \quad \text{where } \mu_k \text{ is cluster center for } x_i
\]

Does each step of k-means move towards reducing this loss function (or at least not increasing)?

Credit: David Kauchak
Minimizing k-means loss

Iterate:

1. Assign/cluster each example to closest center
2. Recalculate centers as the mean of the points in a cluster

\[ \text{loss} = \sum_{i=1}^{n} d(x_i, \mu_k)^2 \text{ where } \mu_k \text{ is cluster center for } x_i \]

This isn’t quite a complete proof/argument, but:

1. Any other assignment would end up in a larger loss
1. The mean of a set of values minimizes the squared error
Minimizing k-means loss

Iterate:
1. Assign/cluster each example to closest center
2. Recalculate centers as the mean of the points in a cluster

\[ \text{loss} = \sum_{i=1}^{n} d(x_i, \mu_k)^2 \text{ where } \mu_k \text{ is cluster center for } x_i \]

Does this mean that k-means will always find the minimum loss/clustering?
Minimizing k-means loss

Iterate:
1. Assign/cluster each example to closest center
2. Recalculate centers as the mean of the points in a cluster

\[
\text{loss} = \sum_{i=1}^{n} d(x_i, \mu_k)^2 \quad \text{where } \mu_k \text{ is cluster center for } x_i
\]

NO! It will find a minimum.

Unfortunately, the k-means loss function is generally not convex and for most problems has many, many minima

We’re only guaranteed to find one of them
K-means: Initialize centers randomly

What would happen here?

Seed selection ideas?

Credit: David Kauchak
K-means: Initialize furthest from centers

Pick a random point for the first center
K-means: Initialize furthest from centers

What point will be chosen next?

Credit: David Kauchak
K-means: Initialize furthest from centers

Furthest point from center

What point will be chosen next?

Credit: David Kauchak
K-means: Initialize furthest from centers

Furthest point from center

What point will be chosen next?

Credit: David Kauchak
K-means: Initialize furthest from centers

Any issues/concerns with this approach?

Furthest point from center

Credit: David Kauchak
Furthest points concerns

If $k = 4$, which points will get chosen?
Furthest points concerns

If we do a number of trials, will we get different centers?

Credit: David Kauchak
Furthest points concerns

Doesn’t deal well with outliers

Credit: David Kauchak
K-means

- But usually k-means works pretty well
  - Especially with large number of points and large number of centers $k$
- Variations: kmeans++, etc
- Alternatives: spectral clustering, hierarchical (bottom-up, agglomerative or top-down, divisive)
Coming back to textons
Clustering Textons

• Output of bank of $n$ filters can be thought of as vector in $n$-dimensional space
• Can *cluster* these vectors using $k$-means [Malik et al.]
• Result: dictionary of most common textures
Clustering Textons

Image

Clustered Textons

Texton to Pixel Mapping
Using Texture in Segmentation

• Compute histogram of how many times each of the $k$ clusters occurs in a neighborhood

• Define similarity of histograms $h_i$ and $h_j$ using $\chi^2$

$$\chi^2 = \frac{1}{2} \sum_k \frac{(h_i(k) - h_j(k))^2}{h_i(k) + h_j(k)}$$

• Different histograms $\rightarrow$ separate regions
Application: Segmentation
Texture synthesis
Markov Random Fields

• Different way of thinking about textures
• Premise: probability distribution of a pixel depends on values of neighbors
• Probability the same throughout image
  – Extension of Markov chains
Motivation from Language

- Shannon (1948) proposed a way to synthesize new text using N-grams
  - Use a large text to compute probability distributions of each letter given N–1 previous letters
  - Starting from a seed repeatedly sample the conditional probabilities to generate new letters
  - Can do this with image patches!
Texture Synthesis Based on MRF

• For each pixel in destination:
  – Take already-synthesized neighbors
  – Find closest match in original texture
  – Copy pixel to destination

• Efros & Leung 1999
  – Speedup by Wei & Levoy 2000
  – Extension to copying whole blocks by Efros & Freeman 2001
Efros & Leung Algorithm

- Compute output pixels in scanline order (top-to-bottom, left-to-right)
Efros & Leung Algorithm

- Find candidate pixels based on similarities of pixel features in neighborhoods.
Efros & Leung Algorithm

• Similarities of pixel neighborhoods can be computed with squared differences (SSD) of pixel colors and/or filter bank responses

\[ \| (\begin{array}{c} \text{Pixel Neighborhood 1} \\ \text{Pixel Neighborhood 2} \end{array} - \begin{array}{c} \text{Reference Neighborhood 1} \\ \text{Reference Neighborhood 2} \end{array} ) \|^2 \]
Efros & Leung Algorithm

• For each pixel p:
  – Find the best matching K windows from the input image
  – Pick one matching window at random
  – Assign p to be the center pixel of that window
Synthesis Results
Synthesis Results

white bread  
brick wall
Hole Filling

• Fill pixels in “onion skin” order
  – Within each “layer”, pixels with most neighbors are synthesized first
  – Normalize error by the number of known pixels
  – If no close match can be found, the pixel is not synthesized until the end
Hole Filling
Extrapolation
Special video

https://ghc.anitab.org/ghc-17-livestream/

(Wednesday keynote, 16:20 min - 44:00 min)