We now have a complete framework for performing CBIR and evaluating the results:

- Feature extraction. E.g., mean and standard deviation features.
- Distance measures: L1, L2.
- Evaluation: PR curves and average precision.

Now, we will investigate:

- Additional features and feature normalization
- " distance measures.

Histogram features

Instead of using just the mean and standard deviation to summarize the distribution of pixel values, how about a richer representation?

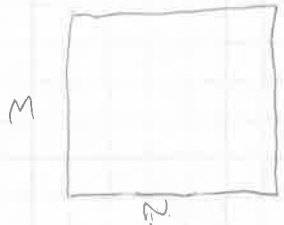
→ Estimate the probability density function (PDF) accurately, the probability mass function (PMF)) using a histogram.

Given a grayscale image of size $M \times N$ with pixel values $0, 1, \ldots, L-1$;

1) Select # of bins: d

2) Divide pixel range evenly into d bins.

3) Let h_i $i = 1, \ldots d$ be the number of pixels in image with values in ith bin.



bins 1: $0 \leq p < 8$
2: $8 \leq p < 16$
⋮
32: $246 \leq p < 256$

$L = 256$

$d = 32$

h_i = # pixels with values in i^{th} bin.

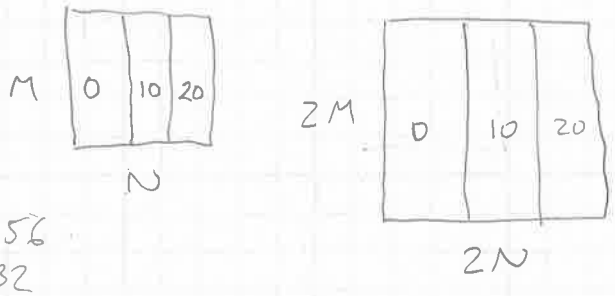
Now have a d-dimensional representation of image

$$h = (h_1, \ldots, h_d)$$

Given two images with histogram features $h1$ and $h2$, can compute image similarity using, for example, L2 distance:

$$d_{L2}(h1, h2) = \sqrt{\frac{1}{d} \sum_{i=1}^{d} (h1_i - h2_i)^2}$$

Q: Are these histogram features, as we've computed them, invariant to image size?



$L = 256$
$d = 32$

$h1 = \left(\frac{MN}{3}, \frac{MN}{3}, \frac{MN}{3}, 0 \ldots, 0\right)$ $\quad h2 = \left(\frac{4MN}{3}, \frac{4MN}{3}, \frac{4MN}{3}, 0 \ldots, 0\right)$.

$$d_{L2}(h1, h2) \neq 0.$$

A: No, not invariant to image size.

Q: How to make invariant to image size?

A: Compute normalized histogram.

$$\hat{h}_i = \frac{\#\text{ pixels in } i^{th} \text{ bin}}{\text{total \# pixels in image } (M \cdot N)}$$

This is same as making vector h have unit L1 length.

$$\sum_{i=1}^{d} \hat{h}_i = 1 \qquad \|\hat{h}\|_1 = 1$$

4/23/18

Can also perform other feature normalizations.

E.g., Normalize feature so that L2 norm = 1.

$$\hat{h}_i = \frac{h_i}{\|h\|_2} = \frac{h_i}{\sqrt{\sum_{i=1}^{d} h_i^2}}$$

Now, $\|\hat{h}\|_2 = 1$.

Grayscale histogram features

Adv.:
- Simple to compute
- Richer representation than just mean and stdev
- Low-dimensional
- Rotation invariant
- Can be made invariant to image size

Disadv.:
- Still do not capture spatial arrangement of pixel values.
- Not invariant to intensity shifts.

Other distance measures, particularly for comparing histograms:

Intersection

$$d_{intersection} = \sum_{i=1}^{d} \min(h1_i, h2_i)$$

Inner-product

$$d_{inner\text{-}product} = \sum_{i=1}^{d} h1_i \cdot h2_i$$

Chi-square

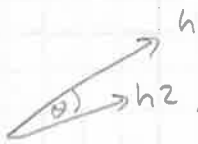
$$d_{chi\text{-}square} = \sum_{i=1}^{d} \frac{(h1_i - h2_i)^2}{h1_i + h2_i}$$

Bhattacharyya

$$d_B = \sqrt{1 - \sum_{i=1}^{d} \frac{\sqrt{h1_i h2_i}}{\sqrt{\sum_{j=1}^{d} h1_j \sum_{k=1}^{d} h2_k}}}$$

Cosine

Consider features as vectors and compute cosine of angle between them



$$\cos\theta = \frac{h1 \cdot h2}{\|h1\|_2 \|h2\|_2} = \frac{\sum_{i=1}^{d} h1_i h2_i}{\sqrt{\sum_{j=1}^{d} h1_j \sum_{k=1}^{d} f2_k}}$$

Note: This is a similarity measure that has a maximum of 1.

Note: Some of these measures are equivalent especially for certain feature normalizations.

E.g.: If h is L_2 normalized ($\|h\|_2 = 1$) then

$$d_{cosine} = d_{L2} = d_{inner\text{-}product}$$

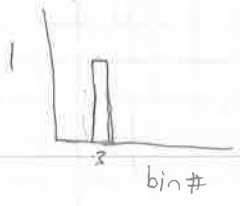
If L_1 normalized $d_{L1} = d_{intersection}$

Can be faster to compute one of these equivalent distance measures.

Earth Mover's Distance (EMD)

Suppose have three images with the following L1 normalized histograms

I1



I2

I3

Most of the measures above will result in image I2 being more similar to I3 than to I1.

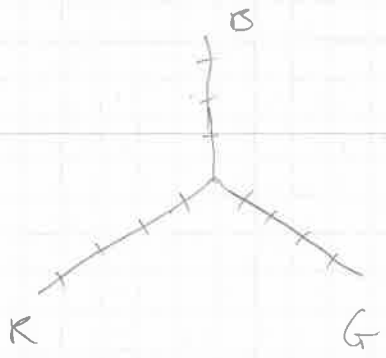
The EMD takes into account how far apart the bins are.

The analogy is that of moving dirt. What is the least amount of effort required to transform one distribution to another.

See internet for details.

Color histogram features

Pixels in color images typically have three values (e.g., RGB), so now compute color histogram by binning 3D space.



1) Pick the number of bins along each color dimension.

$$d_R, d_G, d_B$$

Now, the bins are cubes.

2) Count # pixels in each bin.

3) Form 1D color histogram feature vector by ordering bins (any order is OK as long as remain consistent between images).

4) Feature vector now has dimension

$$d = d_R d_G d_B$$

5) Can normalize:

L1: by dividing by # pixels in image

L2: by dividing by $\|h\|_2$

Note: Most of distance measures above can be applied to color histogram features.

EMD now needs to be able to know distance between bins in 3D color space.

Often use colorspace other than RGB:

- HSV
- Luv

Color histogram vs. grayscale histogram features:

Ad: Captures color information

Disad: Higher dimensional feature vector that can be sparse.

ex. $d_R = d_G = d_B = 32 \rightarrow d = 32768$

usually $d_R = d_G = d_B = 8$ or so $\Rightarrow d = 512$

Texture Features

How can we characterize/summarize the spatial distribution of pixel values?

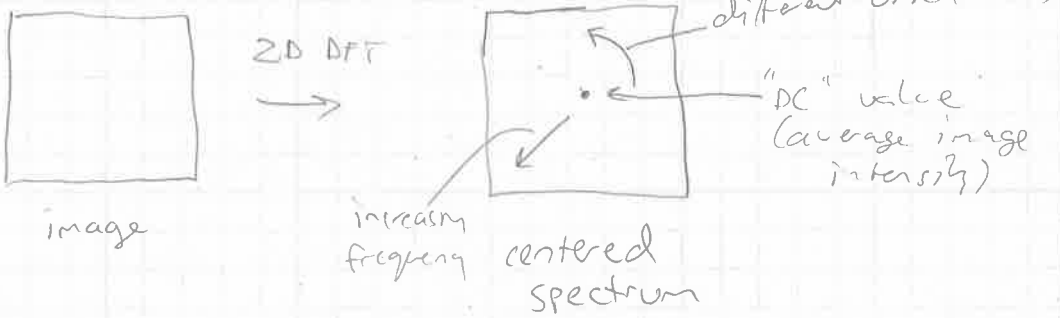
Visual perceptual phenomenon of texture.
<show slide of Brodatz textures>
Lots of texture descriptors.

We'll first consider frequency based ones called Gabor texture features.

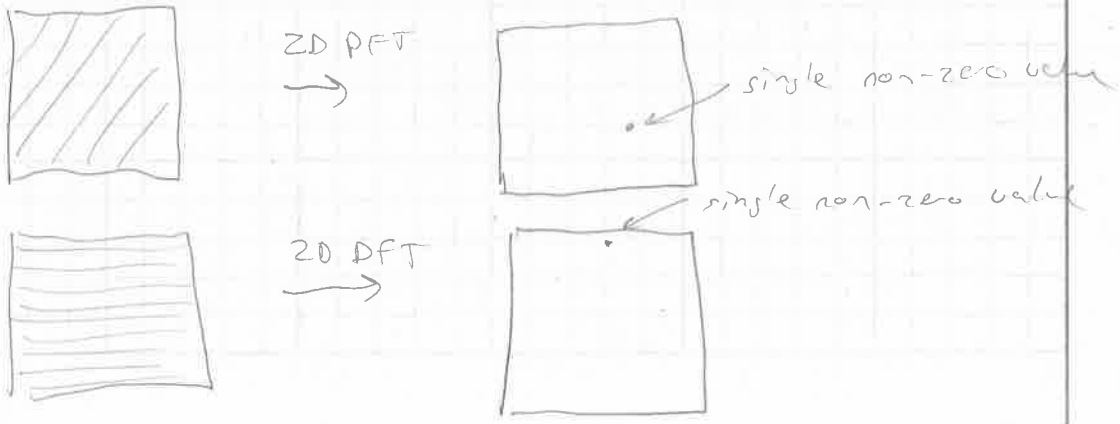
In general:
- Texture is extracted from grayscale images only (so, convert color to grayscale image before extracting texture features).
- Texture features need to be general – not specific to detecting any single pattern.

One way to think about texture patterns is that they have a particular orientation and scale.

This should bring to mind 2D Fourier representations of images.



Example



Idea: Form a texture feature descriptor based on 2D DFT of image.

Option 1: Feature vector is entire 2D DFT spectrum.

Can then compare images by comparing their spectrums "pixel-wise."

This is too "rigid" though. Patterns which might be visually similar might have slightly different frequency components - a freq-to-freq comparison wouldn't capture this.

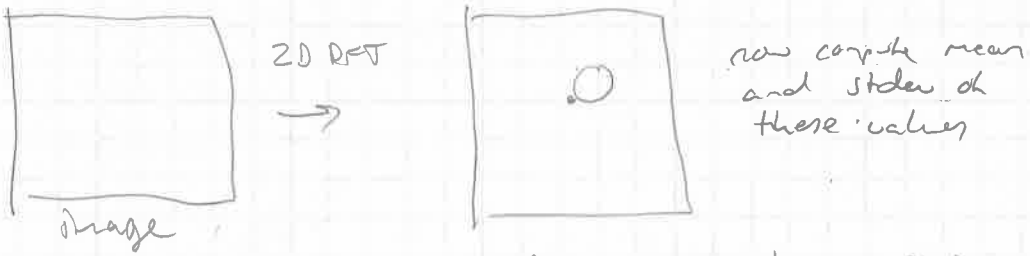
↓ 4 factor

Option 2: Feature vector summarizes portions of 2D DFT spectrum.

Partition spectrum into regions and summarize values using, say, mean and standard deviation.

This partitioning can be accomplished by multiplying by "windows". However, we know that multiplying by sharp windows causes ringing in the spatial domain.

So, instead multiply by Gaussian windows

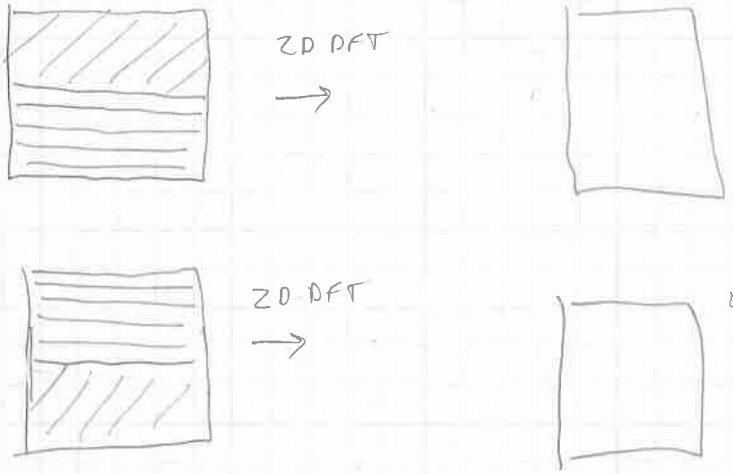


If have n Gaussian windows, will have feature vector of length $2n$: $[\mu_1, \sigma_1, \mu_2, \sigma_2, \ldots, \mu_n, \sigma_n]$

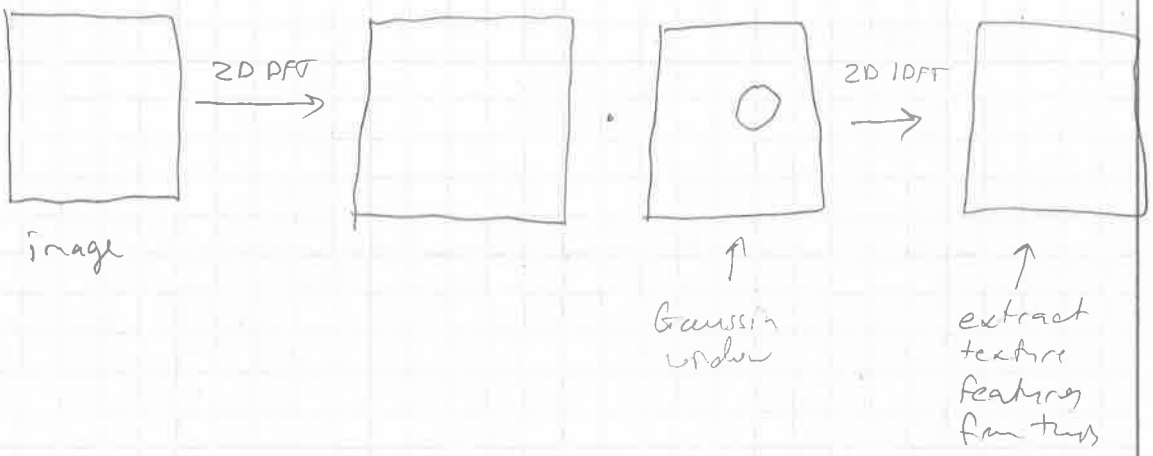
Design choice: Number, sizes, and locations of Gaussian windows.

Deriving the texture features in the frequency domain doesn't tell us <u>where</u> in the image the pattern occurs.

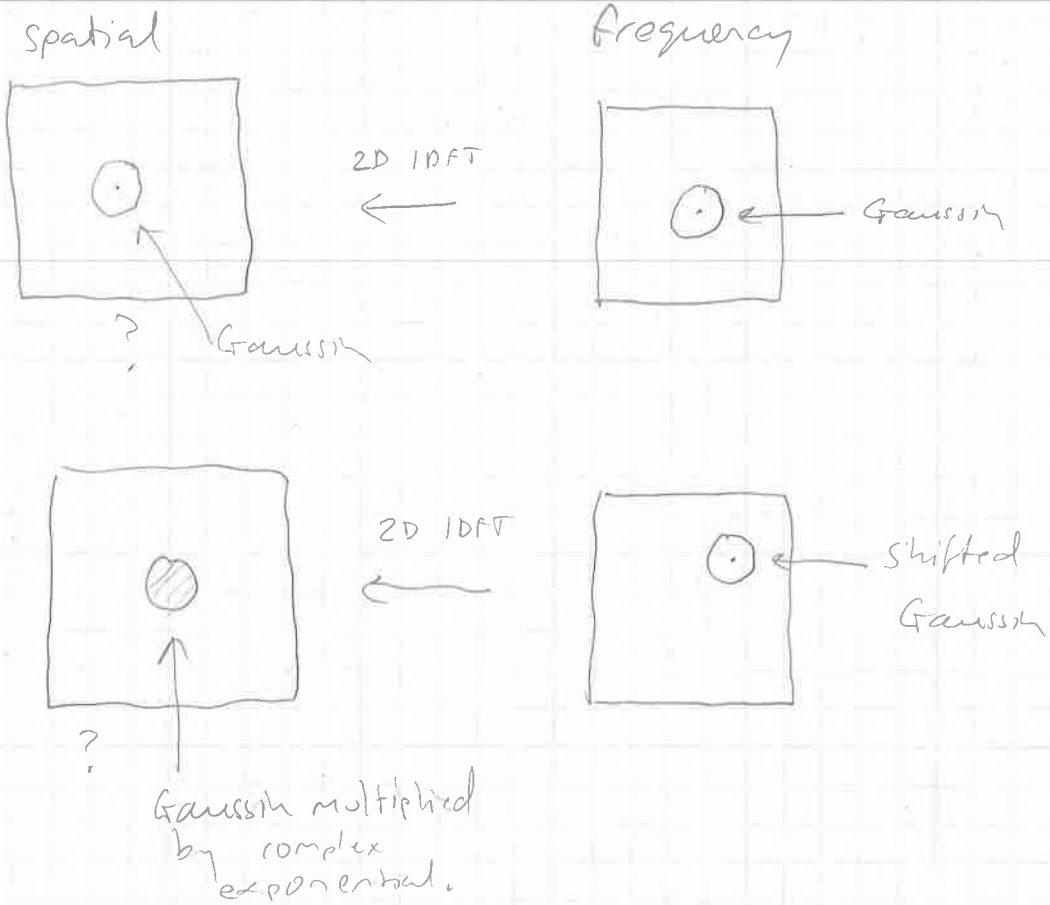
Consider two images:



Option 3: Filter image in frequency domain by multiplying by a Gaussian window and transform result back to spatial domain.



Multiplying by a Gaussian window in the frequency domain = convolving by ? in the spatial domain.

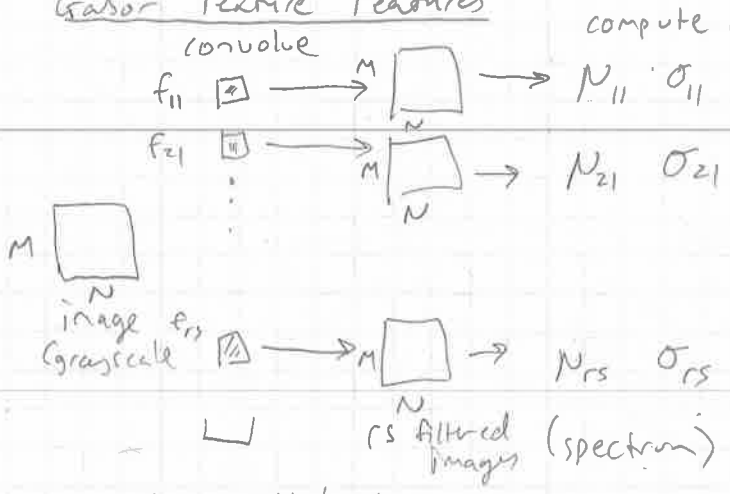


"sinusoidally modulated Gaussian"

Talk about modulation of signals in general.

<show slides of Gabor filters>

Gabor Texture Features



Gabor filterbank
- r orientations (typically ~6)
- s scales (typically 4-5)

rs filters in total

f_{ij} : filter at orientation i and scale j

$i = 1 \ldots r$
$j = 1 \ldots s$

Feature vector is mean and standard deviation (of magnitude) of filtered images.

Gabor texture feature =

$$[\mu_{11}, \sigma_{11}, \mu_{21}, \sigma_{21}, \ldots, \sigma_{r1}, \mu_{r1}, \mu_{12}, \sigma_{12}, \ldots, \mu_{r2}, \sigma_{r2}, \ldots, \mu_{rs}, \sigma_{rs}]$$

Mathematical Form of Gabor Filters

We want to derive a bank of Gabor filters at different scales and orientations.

The mathematical specification of the filters could be in either the spatial or frequency domain.

<show plots of Gabor filters in spatial and frequency domains>

Spatial domain: modulated Gaussians

frequency domain: shifted Gaussians

A 2D Gabor function and its Fourier transform can be written as:

$$g(x,y) = \left(\frac{1}{2\pi\sigma_x\sigma_y}\right) \exp\left[\underbrace{-\frac{1}{2}\left(\frac{x^2}{\sigma_x^2}+\frac{y^2}{\sigma_y^2}\right)}_{\text{Gaussian}} + \underbrace{2\pi j W x}_{\text{modulation}}\right]$$

$$G(u,v) = \exp\left\{-\frac{1}{2}\left[\frac{(u-W)^2}{\sigma_u^2}+\frac{v^2}{\sigma_v^2}\right]\right\}$$

where $\sigma_u = \frac{1}{2\pi\sigma_x}$ and $\sigma_v = \frac{1}{2\pi\sigma_y}$

W is the modulation in the spatial domain or shift in the frequency domain.

Note tradeoff between $\sigma_x \longleftrightarrow \sigma_u$

$\sigma_y \longleftrightarrow \sigma_v$

Note $g(x,y)$ is complex

Q: How to generate Gabor functions at different scales and orientations?

A: A self-similar filter bank can be obtained by appropriate dilations and rotations of $g(x,y)$ through the generator function:

$$g_{mn}(x,y) = a^{-m} g(x',y'), \quad a>1 \quad m,n \in \text{integer}$$

$$x' = a^{-m}(x\cos\theta + y\sin\theta)$$

$$y' = a^{-m}(-x\sin\theta + y\cos\theta)$$

where $\theta = \frac{n\pi}{K}$ $\quad n: 0,\ldots,K-1$ is the # of orientations

$m = 0,\ldots,S-1$ is the # of scales

4/30/19 ↓

The scale factor a^{-m} is meant to ensure the energy of the function is independent of m.

We need to specify $\begin{cases} U_l & \text{lower frequency of interest} \\ U_h & \text{upper frequency " "} \end{cases}$

Then σ_u and σ_v (and thus σ_x and σ_y) and a can be determined.

Filters can be applied in spatial domain through convolution or frequency domain through multiplication.

Summary of computing Gabor texture features:

1) Pick S = # of scales (usually 4-5)
K = # of orientations (usually 6-8)

U_l : lower frequency of interest (ex: $U_l = 0.05$)
U_h : upper " " " (ex: $U_h = 0.4$)

2) Compute <u>discretized</u> versions of KS filters

$$g_{mn}(x,y) \qquad \begin{matrix} m=0,\ldots,K-1 \\ n=0,\ldots,S-1 \end{matrix}$$

These are the spatial filters.

3) Given an image I, compute KS filtered images by convolving I with $g_{mn}(x,y)$

$$I_{mn} \qquad \begin{matrix} m=0,\ldots,K-1 \\ n=0,\ldots,S-1 \end{matrix}$$

Compute mean and standard deviation of magnitude of each I_{mn}

$$\mu_{mn} = \text{mean}(|I_{mn}|)$$

$$\sigma_{mn} = \text{stdev}(|I_{mn}|)$$

4) Form $2KS$ feature vector

$$f_{Gabor} = [\mu_{11}, \sigma_{11}, \mu_{21}, \sigma_{21}, \ldots, \mu_{K1}, \sigma_{K1}, \mu_{K2}, \sigma_{K2}, \ldots, \mu_{KS}, \sigma_{KS}]$$

This feature vector can then be compared using L2 distance, for example.

Feature normalization revisited

Suppose the dynamic range of some feature components/dimensions are much larger than others:

$$f = [f_1, f_2, \ldots, f_d]$$

f_1: ranges between $0 - 1 \times 10^6$

f_2: ranges between $0 - 1$

f_d: ranges between $-0.001 - 0.001$

Visualize 2D features

Using standard distance measures such as L1 and L2 will give greater weight to those components/dimensions with greater dynamic ranges.

If this is not what is intended and would instead prefer that all components/dimensions are weighted equally, then perform feature normalization so that all components/dimensions have "zero-mean and unit variance":

$$f' = [f_1', f_2', \ldots, f_d']$$

where $$f_i' = \frac{f_i - \mu_i}{\sigma_i}$$

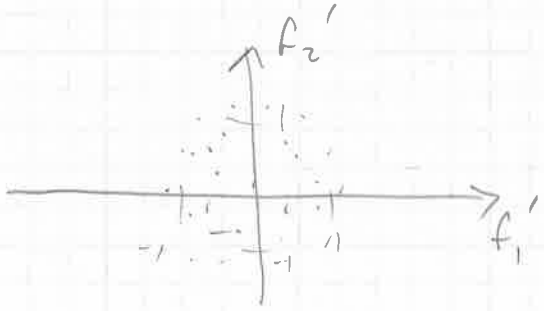
μ_i is the mean of component/dimension i over all features in your dataset

$$\mu_i = \frac{1}{M} \sum_{j=1}^{M} f_i \quad \text{where } M = \#\text{ features (images) in dataset.}$$

σ_i is the standard deviation of i^{th} component/dimension over whole dataset

$$\sigma_i = \sqrt{\frac{1}{M} \sum_{j=1}^{M} (f_i - \mu_i)^2}$$

f_i' will now have zero mean and unit variance (standard deviation)



For CBIR:

After extracting features from images in your target set, normalize each feature to have zero mean and unit variance.

If query image is from target set, already have normalized feature.

If query image is not from target set, compute feature and normalize it using μ_i and σ_i from target set.

Make sure you don't compare original and normalized feature vectors.

Note: This kind of normalization can also be accomplished by the distance measure: Mahalanobis distance

Even though the design of the Gabor texture features tries to keep dynamic range of feature components/dimensions equal, it is still a good idea to normalize them to have zero mean and unit variance before comparing them.

50: [Regularity Dom Orient1 Dom Orient2 / Dom Scale1 Dom Scale2]

< CBGIR demo >

5/21/19 ↓

Connections to MPEG-7 - Perceptual Browsing Component (PBC)

Connections between Gabor filters and visual system.

Connections between " " and CNNs.

Other texture features

Texture features that explicitly model spatial arrangement of pixel values

"what values tend to occur next to what values"

Gray-Level Co-Occurrence Matrices (GLCM)

Spatial co-occurrence of pixel values

Let $f(x,y)$ be an $M \times N$ image with pixel values $i = 1, \ldots, L$ e.g., $i = 0, \ldots, 255$

A GLCM is an $L \times L$ matrix that counts the number of times two pixel values occur in some spatial arrangement (e.g., next to each other)

$GLCM_{adj}(i,j)$ = # times pixel with value i occurs next to a pixel with value j.

Next to can be 4-connected or 8-connected

The spatial arrangement can also directional.

e.g., above, below, diagonal, etc.

Can interpret GLCM as probabilities.

Intuitively: The diagonal or near-diagonal entries of a GLCM will be larger for images composed of patches with the same or similar values (with respect to the spatial arrangement).

GLCM entries are usually normalized so sum to one; why?

The GLCM is large though: e.g., 255×255

So, instead summarize GLCM using scalar quantities that can form a texture feature descriptor/vector.

Robert Haralick in 1973 proposed 14 scalar quantities that can be derived from GLCMs.

Five most common:

1. Angular second moment = ASM = $\sum_{i=1}^{L}\sum_{j=1}^{L}(GLCM(i,j))^2$

ASM ↓ image more random

ASM ↑ " " structured

2. Contrast = CON = $\sum_{n=0}^{L-1} n^2 \left\{ \sum_{|i-j|=n} GLCM(i,j) \right\}$

(for GLCM with larger off-diagonal values)

CON ↑ image has quickly varying intensities

3. Inverse different moment = IDM = $\sum_{i=1}^{L}\sum_{j=1}^{L} \frac{GLCM(i,j)}{1+(i-j)^2}$

or near diagonal

IDM ↑ GLCM with large diagonal values
image with constant or near constant values

4. Entropy = ENT = $-\sum_{i=1}^{L}\sum_{j=1}^{L} GLCM(i,j) \log GLCM(i,j)$

ENT ↑ is GLCM is evenly distributed
image that is noisy or random

5. Correlation = COR = $\sum_{i=1}^{L}\sum_{j=1}^{L} \frac{(ij)\,GLCM(i,j) - \mu_i' \mu_j'}{\sigma_i' \sigma_j'}$

where

$\mu_i' = \sum_{i=1}^{L}\sum_{j=1}^{L} i\,GLCM(i,j)$ and similar for μ_j'

$\sigma_j'^2 = \sum_{i=1}^{L}\sum_{j=1}^{L} GLCM(i,j)\,(i-\mu_i')^2$ and similar for σ_j'

Typically compute a set of GLCMs and derived scalar quantities for a set of spatial arrangements (offsets)

E.g.: Four adjacencies:

Given five values for each adjacency, form 20D feature vector.

$$f_{GLCM} = [ASM_1, CON_1, IDM_1, ENT, CORR_1, \ldots, CORR_4]$$

Can then compare using L1, L2 distance etc.