Texture

- SVD Matlab demo
- Texture
- Bag-of-words
- (Spatial) pyramid matching
Let us represent a linear transformation as follows:

\[ y = Ax, \quad A \in \mathbb{R}^{n \times m} \quad (1) \]

where \( A \) is a matrix with \( n \) columns and \( m \) rows. This document uses the singular value decomposition (SVD) to decompose \( A \) into a series of geometric transformations, focusing on intuition rather than a precise formulation.

For simplicity, let \( n = 2 \) and \( m = 3 \), such that \( A \) transforms points in 2D to 3D.

Figure 1: Visualizing a matrix \( A \in \mathbb{R}^{2 \times 3} \) as a transformation of points from \( \mathbb{R}^2 \) to \( \mathbb{R}^3 \).

Orthonormal basis:

First, let us recall that the projection of a vector \( x \in \mathbb{R}^n \) along a unit vector \( v \) (e.g., \( v^T v = 1 \)) can be written as \( v^T x \). Let us construct a set of \( n \) unit vectors and write them as a matrix \( V = [v_1, v_2, \ldots, v_n] \). We can then compute the projection or coordinates of vector \( x \) along the unit vectors with a matrix multiplication \( p = V^T x \).

If all the unit vectors are orthogonal to each other (\( v_i^T v_j = 0 \) for \( i \neq j \)), then \( V^T V = I \). This implies that \( V \) can be thought of as a rotation matrix (whose inverse is \( V^T \)), making it easy to undo the projection. The set of vectors in \( V \) form an orthonormal basis for \( \mathbb{R}^n \). Let us construct an orthonormal basis for the output space \( U = [u_1, u_2, \ldots] \).

Finally, we have:

\[ y = Ax \]

\[ y = U \Sigma V^T x \]
Recall: SVD

Let us represent a linear transformation as follows:

\[ y = Ax, \quad A \in \mathbb{R}^{n \times m} \]  

(1)

where \( A \) is a matrix with \( n \) columns and \( m \) rows. This document uses the singular value decomposition (SVD) to decompose \( A \) into a series of geometric transformations, focusing on intuition rather than a precise formulation. For simplicity, let \( n = 2 \) and \( m = 3 \), such that \( A \) transforms points in 2D to 3D.

Figure 1: Visualizing a matrix \( A \in \mathbb{R}^{2 \times 3} \) as a transformation of points from \( \mathbb{R}^2 \) to \( \mathbb{R}^3 \).

Orthonormal basis:

First, let us recall that the projection of a vector \( x \in \mathbb{R}^n \) along a unit vector \( v \) (e.g., \( v^Tv = 1 \)) can be written as \( v^Tx \). Let us construct a set of \( n \) unit vectors and write them as a matrix

\[
V = \begin{bmatrix} v_1 & v_2 & \ldots & v_n \end{bmatrix}
\]

We can then compute the projection or coordinates of vector \( x \) along the unit vectors with a matrix multiplication

\[
p = V^T x.
\]

If all the unit vectors are orthogonal to each other (\( v_i^Tv_j = 0 \) for \( i \neq j \)), then \( V^TV = I \). This implies that \( V \) can be thought of as a rotation matrix (whose inverse is \( V^T \)), making it easy to undo the projection. The set of vectors in \( V \) form an orthonormal basis for \( \mathbb{R}^n \). Let us similarly construct an orthonormal basis for the output space

\[
U = \begin{bmatrix} u_1 & u_2 & \ldots & u_m \end{bmatrix}, U^TU = I
\]

\[
V = \begin{bmatrix} v_1 & v_2 & \ldots & v_n \end{bmatrix}, V^TV = I
\]  

\[
\Sigma = \begin{bmatrix} \sigma_1 & 0 & 0 & \ldots \\ 0 & \sigma_2 & 0 & \ldots \\ 0 & 0 & \sigma_3 & \ldots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}
\]
Recall: SVD

\[ y = U \Sigma V^T x \]

Notation: \( u_i \) = left singular vector, \( \sigma_i \) = singular values, \( v_i \) = right singular vectors

Any linear operator can be thought of as mapping from \( \mathbb{R}^n \) to \( \mathbb{R}^m \)

1. projection (with right singular vectors)
2. scaling (with singular values),
3. reconstruction (with left singular vectors)
Recall: SVD

Immediate consequences (by appealing to geometric intuition)

\[ ||A||_F = A(:)^T A(:) \]

Low rank: \[ \min_{A': \text{rank}(A') \leq k} ||A - A'||_F = U(:, 1 : k) \Sigma (1 : k, 1 : k) V(:, 1 : k)^T. \]

Homogenous least-squares: \[ \min_{h: h^T h = 1} ||Ah||^2 = V(:, \text{end}) \]

Pseudoinverse: \[ A^+ = \arg \min_{A^+} ||A^+ A - I||_F = V \begin{bmatrix} \frac{1}{\sigma_1} & 0 & 0 & \cdots \\ 0 & \frac{1}{\sigma_2} & 0 & \cdots \\ 0 & 0 & \frac{1}{\sigma_3} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}^T U^T \]
Extension: positive-semidefinite (PSD) matrices

Let’s construct a square matrix $B = A^T A$

Any PSD matrix can be thought of as mapping from $\mathbb{R}^n$ to $\mathbb{R}^n$

1. projection (with eigenvectors)
2. scaling (with eigenvalues),
3. reconstruction (with same eigenvectors)

For $V = \text{identity}$, $B = \text{scaling}$
For general $V$, $B = \text{scaling in rotated coordinate system}$
Matlab demo

Steerable & Separable Filter Banks
(how can we efficiently exploit for convolution?)

Eigenfaces
Texture

- SVD Matlab demo
- Texture
- Bag-of-words
- (Spatial) pyramid matching
How do we define texture?
Continuum of regularity

stochastic

regular
Continuum of regularity

Each pixel is drawn iid from some probability distribution over colors $P(I)$

Each pixel is color is determined completely by its coordinates w.r.t. the rest of the texture
Continuum of regularity

Each pixel is drawn from some probability distribution over colors conditioned on the value of its neighbors $P(I|N)$.
Pre-attentive texture discrimination

(Julesz, 1981)

“textons”

160 ms, outside foveal gaze
Instantaneous, or effortless texture discrimination
Representing textures

Let’s encode the texture as a distribution over localized visual elements, or “textons”

1. How do we represent a texton?

2. How do we represent a distribution over them?
Let’s build a discrete probability distribution (or histogram) over small KxK patches, where each pixel can take on one of \( N = 256 \) values.

What would be the size of this histogram?

Hopeless! \( N^{KK} \)
Let’s build a discrete probability distribution (or histogram) over small KxK patches, where each pixel can take on one of N = 256 values.

Project KxK patch into M filters

Let’s represent marginal distribution instead of joint

\[ P(f_1, f_2, \ldots) = N^M \text{ table entries} \]

\[ P(f_1)P(f_2)P(f_3) = N^M \text{ table entries} \]
Use collection of histograms for a set of filters to represent texture
Is there an efficient way to capture *joint* statistics of filter responses?
Capture joint statistics via histograms of vector quantized features
K-means

visual intuition
Cost function

$$\min_{Z,D} C(Z, D, X) \quad \text{where} \quad C(Z, D, X) = \sum_{i} \|x_i - d_{z_i}\|^2$$

- $x_i$: $i^{th}$ input vector (to be clustered)
- $z_i \in \{1 \ldots K\}$: $i^{th}$ label
- $d_k$: $k^{th}$ dictionary element (or mean)
Coordinate descent optimization

\[
\min_{Z,D} C(Z, D, X) \quad \text{where} \quad C(Z, D, X) = \sum_i \|x_i - d_{zi}\|^2
\]

1. \( \min_Z C(Z, D, X) \)

2. \( \min_D C(Z, D, X) \)
Training vs testing

Training: \[ \min_{Z,D} C(Z, D, X_{\text{train}}) \]

Testing: \[ \min_{Z} C(Z, D, X_{\text{test}}) \]
Question: how can we visualize mean (or texton)

Let $B$ be a matrix of vectorized filters: $B = [b_1, \ldots b_t]$

Given an vectorized image patch $p_i$, compute filter responses with a linear projection: $x_i = B^T p_i$

Given a mean $d_i$, compute corresponding image with: $\text{pseudoinv}(B^T)d_i$
What Are Textons?

Song-Chun Zhu\textsuperscript{1}, Cheng-en Guo\textsuperscript{1}, Yingnian Wu\textsuperscript{2}, and Yizhou Wang\textsuperscript{1}

1 Introduction

Texton refers to fundamental micro-structures in generic natural images and the basic elements in early (pre-attentive) visual perception\cite{8}. In practice, the study of textons has important implications in a series of problems. Firstly, decomposing an image into its constituent components reduces information redundancy and, thus, leads to better image coding algorithms. Secondly, the decomposed image representation often has much reduced dimensions and less dependence between variables (coefficients), therefore it facilitates image modeling which is necessary for image segmentation and recognition. Thirdly, in biological vision the micro-structures in natural images provide an ecological clue for understanding the functions of neurons in the early stage of biological vision systems\cite{1,13}. However, in the literature of computer vision and visual perception, the word “texton” remains a vague concept and a precise mathematical definition has yet to be found.


Capture joint statistics via histograms of vector quantized features
Overall pipeline

Filter responses

Filter

Filter

Filter

K-Means

Histogram
Texture recognition

polka-dot

brick

mesh

match
Extension: matrix formulation

\[
\min_{D, Z} \| X - DZ \|_F^2
\]

\[
X = [x_1, \ldots x_n]
\]

\[
D = [d_1, \ldots, d_K]
\]

\[
Z = [z_1, \ldots z_n]
\]

K-means: \( z_i = [\ldots, 0, 1, 0, \ldots] \)
Sparse reconstructions

\[
\min_{D,Z} \| X - DZ \|_F^2 \quad \text{subject to sparse constraints on } Z
\]

\[X = [x_1, \ldots, x_n]\]
\[D = [d_1, \ldots, d_K]\]
\[Z = [z_1, \ldots, z_n]\]

K-means: \[z_i = [\ldots, 0, 1, 0, \ldots]\]

L0 sparse-coding: \[\|z_i\|_0 \leq M\] (greedy algorithms known as “matching pursuit”)

L1 sparse-coding: \[\|z_i\|_1 \leq M\] (convex program)
L1 sparse dictionary learning

- Emergence of Simple-Cell Receptive Field Properties by Learning a Sparse Code for Natural Images.
Sparse reconstructions

\[
\min_{D,Z} \|X - DZ\|_F^2 \quad \text{subject to sparse constraints on } Z
\]

\[
X = [x_1, \ldots x_n]
\]

\[
D = [d_1, \ldots, d_K]
\]

\[
Z = [z_1, \ldots z_n]
\]

K-means: \( z_i = [\ldots, 0, 1, 0, \ldots] \)

L0 sparse-coding: \( \|z_i\|_0 \leq M \)

L1 sparse-coding: \( \|z_i\|_1 \leq M \)

Can be written equivalently as: \( \min_{D,Z} \|X - DZ\|_F^2 + R(Z) \)
Sparse reconstructions

\[
\min_{D,Z} \|B^T I - DZ\|_F^2 \quad \text{subject to sparse constraints on } Z
\]

\[
I = [i_1, \ldots, x_n]
\]

\[
D = [d_1, \ldots, d_K]
\]

\[
Z = [z_1, \ldots, z_n]
\]

\[
B = [b_1, \ldots, b_t]
\]

\(i_n\) is the \(n^{th}\) image patch

\(b_i\) is the \(t^{th}\) filter in the filter bank
Learning Feature Representations with K-means

Adam Coates and Andrew Y. Ng

Stanford University, Stanford CA 94306, USA
{acoates,ang}@cs.stanford.edu

2.1 Pre-processing

Before running a learning algorithm on our input data points \( x(i) \), it is useful to normalize the brightness and contrast of the patches. That is, for each \( x(i) \) we subtract out the mean of the intensities and divide by the standard deviation. A small value is added to the variance before division to avoid divide by zero and also suppress noise. For pixel intensities in the range \([0, 255]\), adding 10 to the variance is often a good starting point:

\[
\tilde{x}(i) = \frac{x(i) - \text{mean}(\tilde{x}(i))}{\text{var}(\tilde{x}(i)) + 10}
\]

where \( \tilde{x}(i) \) are unnormalized patches and "mean" and "var" are the mean and variance of the elements of \( \tilde{x}(i) \).

After normalization, we can try to run K-means on the new input patches. The centroids that are obtained (i.e., the columns of the dictionary \( D \)) are visualized as patches in Figure 1a. It can be seen that K-means tends to learn low-frequency edge-like centroids. This result has been reproduced many times in the past [16,37,2]. Unfortunately, it turns out that these centroids tend to work poorly in recognition tasks [11]. One explanation for this result is that the correlations between nearby pixels (i.e., low-frequency variations in the images) tend to be very strong even after brightness and contrast normalization. In the presence of these correlations, K-means tends to generate many highly correlated centroids rather than spreading the centroids out to span the data more evenly. A cartoon depicting this problem is shown on the left of Figure 1b. To remedy this situation, one should use whitening (also called "sphering") to rescale the input data to remove these correlations [22]. This tends to cause K-means to...
2.1 Pre-processing

Before running a learning algorithm on our input data points \( x_i \), it is useful to normalize the brightness and contrast of the patches. That is, for each \( x_i \) we subtract out the mean of the intensities and divide by the standard deviation. A small value is added to the variance before division to avoid divide by zero and also suppress noise. For pixel intensities in the range \([0, 255]\), adding 10 to the variance is often a good starting point:

\[
x_i = \tilde{x}_i - \text{mean}(\tilde{x}_i) \quad \text{p} \quad \text{var}(\tilde{x}_i) + 10
\]

where \( \tilde{x}_i \) are unnormalized patches and "mean" and "var" are the mean and variance of the elements of \( \tilde{x}_i \).

After normalization, we can try to run K-means on the new input patches. The centroids that are obtained (i.e., the columns of the dictionary \( D \)) are visualized as patches in Figure 1a. It can be seen that K-means tends to learn low-frequency edge-like centroids. This result has been reproduced many times in the past [16, 37, 2]. Unfortunately, it turns out that these centroids tend to work poorly in recognition tasks [11]. One explanation for this result is that the correlations between nearby pixels (i.e., low-frequency variations in the images) tend to be very strong even after brightness and contrast normalization. In the presence of these correlations, K-means tends to generate many highly correlated centroids rather than spreading the centroids out to span the data more evenly. A cartoon depicting this problem is shown on the left of Figure 1b. To remedy this situation, one should use whitening (also called "sphering") to rescale the input data to remove these correlations [22]. This tends to cause K-means to do better in recognition tasks [11].

**Whitening:** choose linear transformation \( B \) such that \( \mathbb{E}[(B^TX)(X^TB)] = \text{Identity} \)

![Image of whitening process](image-url)
Dictionary learning

$$\min_{D,Z} ||X - DZ||^2_F$$

Some folks claim that k-means should always be replaced by sparse coding (never hurts, sometimes better)

… k-means is far simpler, right?
“In between” k-means and sparse coding

\[
\min_{D, Z} \|X - DZ\|_F^2 \quad \text{subject to sparse constraints on } Z
\]

K-means: \( z_i = [\ldots, 0, 1, 0, \ldots] \)

L0 sparse-coding: \( \|z_i\|_0 \leq M \)  
(For \( M = 1 \), we can solve for \( Z \) in closed form)

Generalizes k-means with cosine distance to allow for negative coefficients
Example Natural Materials

- Terrycloth
- Rough Plastic
- Plaster
- Sponge
- Rug
- Painted Spheres

Columbia-Utrecht Database (http://www.cs.columbia.edu/CAVE)

polka-dot
brick
mesh
Texture

- SVD Matlab demo
- Texture
- Bag-of-words
- (Spatial) pyramid matching
Analogy with “bag-of-words” for document processing

Political observers say that the government of Zorgia does not control the political situation. The government will not hold elections …

The ZH-20 unit is a 200Gigahertz processor with 2Gigabyte memory. Its strength is its bus and high-speed memory……
Bag-of-visual-words

Training images → Filter responses → Clustering
Object → Bag of ‘words’
Recognition with bag-of-words

- Summarize entire image based on its distribution (histogram) of word occurrences.
- Compare to stored library of images (or class-specific models)

Image credit: Fei-Fei Li
Texture

- SVD Matlab demo
- Texture
- Bag-of-words
- (Spatial) pyramid matching
Digression: alternative to quantization

Approximate matching with histogram similarity

\[ \mathbf{X} = \{ \mathbf{x}_1, \ldots, \mathbf{x}_m \}; \quad \mathbf{x}_i \in \mathbb{R}^d \]

\[ \mathbf{Y} = \{ \mathbf{y}_1, \ldots, \mathbf{y}_n \}; \quad \mathbf{y}_i \in \mathbb{R}^d \]
Aside: what’s the “right” way to compare histograms?

\[ D_r(x, y) = \left( \sum_i (x_i - y_i)^r \right)^{\frac{1}{r}} \]

euclidean (r=2) or manhattan (r=1)
Aside: what’s the “right” way to compare histograms?

- Euclidean distance ($r=2$) or Manhattan distance ($r=1$)
- Chi-squared distance ([derived from chi-squared text in statistics](https://en.wikipedia.org/wiki/Chi-squared_distance))
- K-L divergence ([log probability of seeing x under model y](https://en.wikipedia.org/wiki/Kullback%E2%80%93Leibler_divergence))

$$D_r(x, y) = \left( \sum_i (x_i - y_i)^r \right)^{\frac{1}{r}}$$

$$Chi(x, y) = \sum_i \frac{(x_i - y_i)^2}{x_i + y_i}$$

$$D_{KL}(x, y) = \sum_i x_i \log \frac{x_i}{y_i}$$
Earth mover’s distance

Cast as “transportation problem”
Earth mover’s distance

Bipartite network flow

\[
\min \sum_{i,j} c_{ij} f_{ij} \quad s.t.
\]

\[
f_{ij} \geq 0
\]

\[
\sum_{i} f_{ij} = y_j
\]

\[
\sum_{j} f_{ij} = x_i
\]

\[
EMD(x, y) = \sum_{i,j} c_{ij} f_{ij}
\]

Similarity Kernels

[sometimes more intuitive to define than distance functions]

\[ D(x, y) = K(x, x) + K(y, y) - 2K(x, y) \]

\[ K_{lin}(x, y) = \sum_{i} x_i y_i \quad \text{[what’s corresponding distance function?]} \]
Similarity Kernels

[sometimes more intuitive to define than distance functions]

\[ D(x, y) = K(x, x) + K(y, y) - 2K(x, y) \]

\[ K_{lin}(x, y) = \sum_i x_i y_i \]

\[ K_{int}(x, y) = \sum_i \min(x_i, y_i) \]

What happens if \( x, y \) are binary vectors?
Similarity Kernels

[sometimes more intuitive to define than distance functions]

\[
D(x, y) = K(x, x) + K(y, y) - 2K(x, y)
\]

\[
K_{lin}(x, y) = \sum_{i} x_i y_i
\]

\[
K_{int}(x, y) = \sum_{i} \text{min}(x_i, y_i)
\]

\[
K_{bat}(x, y) = \sum_{i} \sqrt{x_i y_i}
\]

It turns out, we can compute transformations \( f(x) \) and \( f(y) \) such that L2 distance in transformed space corresponds to these kernal terms (allows use of linear predictors)

http://www.robots.ox.ac.uk/~vgg/software/homkermap/
Histogram intersection kernel

\[ \mathcal{I}(H(X), X(Y)) = \sum_k \min(H(X_k), X(Y_k)) \]
But what about bin effects (partial credit for near matches)?
Back to correspondence matching

Count matches obtained from larger bins
Counting new matches

Histogram intersection

\[ I(H(X), H(Y)) = \sum_{j=1}^{r} \min(H(X)_j, H(Y)_j) \]

matches at this level

\[ N_i = I(H_i(X), H_i(Y)) - I(H_{i-1}(X), H_{i-1}(Y)) \]

matches at previous level

Difference in histogram intersections across levels counts *number of new pairs* matched
Giving partial credit for new matches

Weight new matches inversely proportional to bin size

\[ \frac{1}{2^i} \]
Pyramid match kernel

$K_\Delta \left( \Psi(X), \Psi(Y) \right) = \sum_{i=0}^{L} \frac{1}{2^i} \left( \mathcal{I}(H_i(X), H_i(Y)) - \mathcal{I}(H_{i-1}(X), H_{i-1}(Y)) \right)$

- Weights inversely proportional to bin size
- Normalize kernel values to avoid favoring large sets
**Spatial Pyramid Matching**

Quantize features into words, but build pyramid in space

Nifty way to encode constraints like “eye” words lie near top of image

---

Original images

Feature histograms:
- Level 3
  - $= \mathcal{I}_3$
- Level 2
  - $= \mathcal{I}_2$
- Level 1
  - $= \mathcal{I}_1$
- Level 0
  - $= \mathcal{I}_0$

Total weight (value of pyramid match kernel): $\mathcal{I}_3 + \frac{1}{2}(\mathcal{I}_2 - \mathcal{I}_3) + \frac{1}{4}(\mathcal{I}_1 - \mathcal{I}_2) + \frac{1}{8}(\mathcal{I}_0 - \mathcal{I}_1)$
Texture

- SVD Matlab demo
- Texture
- Bag-of-words
- (Spatial) pyramid matching