Lecture 8

Images as n-Dimensional Vectors

Reminders

Midterm is next week, here at 6:00.

Please be here on time.

A2 is due this Sunday (March 2) at 11:59:59.999.

Alternative office hour on Mondays at noon.

Topic 05:

Representing Images as n-Dimensional Vectors

- Template matching:
  - cross-correlation & normalized cross-correlation
- Principal component analysis
  - geometrical intuition: changing basis
  - the eigenfaces recognition algorithm
  - algorithm derivation: minimizing sample covariance

Template Matching Applications

Face detection & recognition

Fujifilm Debuts FinePix Digital Camera With Face Detection Technology

New FinePix S600fd offers breakthrough focusing technology, company also rolls out compact FinePix F20.

By PopPhoto.com Staff
July 12, 2006

Fujifilm introduces the SLR-styled FinePix S6000*fd*, the first digital camera in Fujifilm's line-up with the company's revolutionary new Face Detection Technology.

Face Detection Technology operates exactly as its name implies, identifying up to 10 faces in a framed scene. Once faces are identified and prioritized, the 6.3-megapixel FinePix S6000*fd* adjusts its focus and exposure accordingly to ensure the sharpness and clarity of human subjects in the picture, regardless of background. And since it is hardware rather than software based, Fujifilm's Face Detection Technology works in as little as 0.05 seconds, faster than similar in-camera detection systems currently on the market or soon to be available.

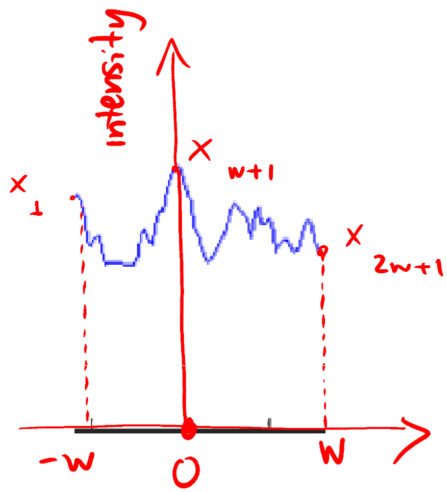
The FinePix S6000fd is the first digital camera in Fujifilm�s line-up with new Face Detection Technology.

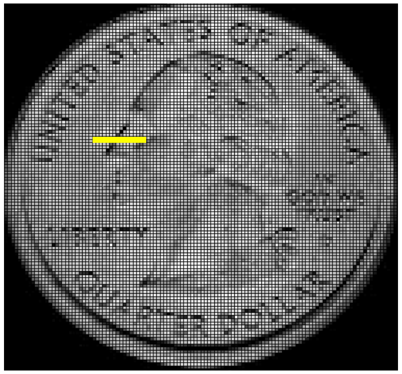
Quicker operation is said to reduce the likelihood of missed or blurry photos, frustrations often associated with digital photography. The advanced Face Detection Technology system built into Fujifilm's new FinePix S6000*fd* digital camera is based on the Image Intelligence technology found in Fujifilm's Frontier Digital Lab Systems, used by photofinishers to produce

(Felzenszwalb and Huttenlocher, IJCV 2005)

Representing Images & Patches as Vectors

As graph in 2D

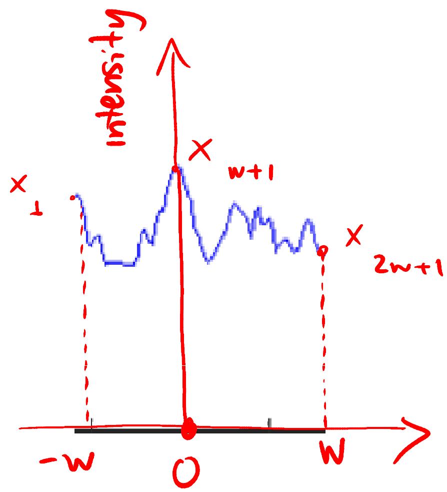


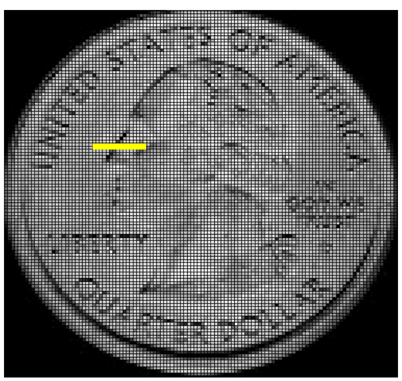


We covered some tools that can be applied to 1-D image patches represented as vectors.

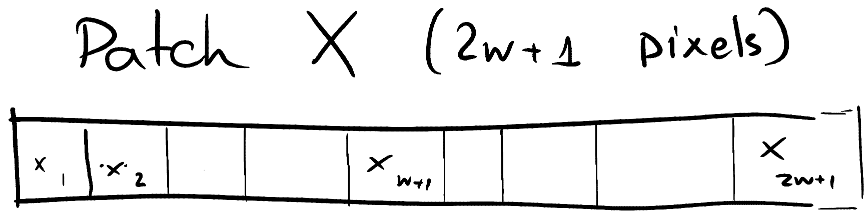
Representing Images & Patches as Vectors

As graph in 2D





For example, a patch of radius w:

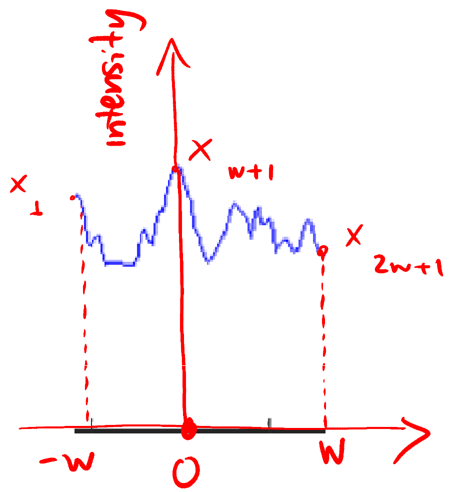


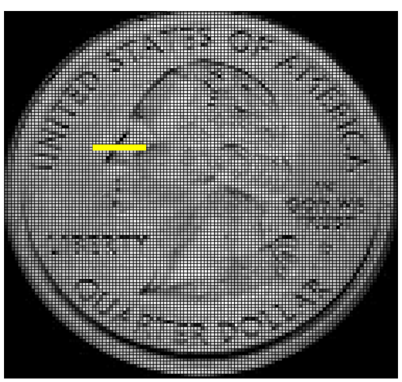
is simply a (2w+1)-dimensional (column) vector.

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_{2w+1} \end{bmatrix}$$

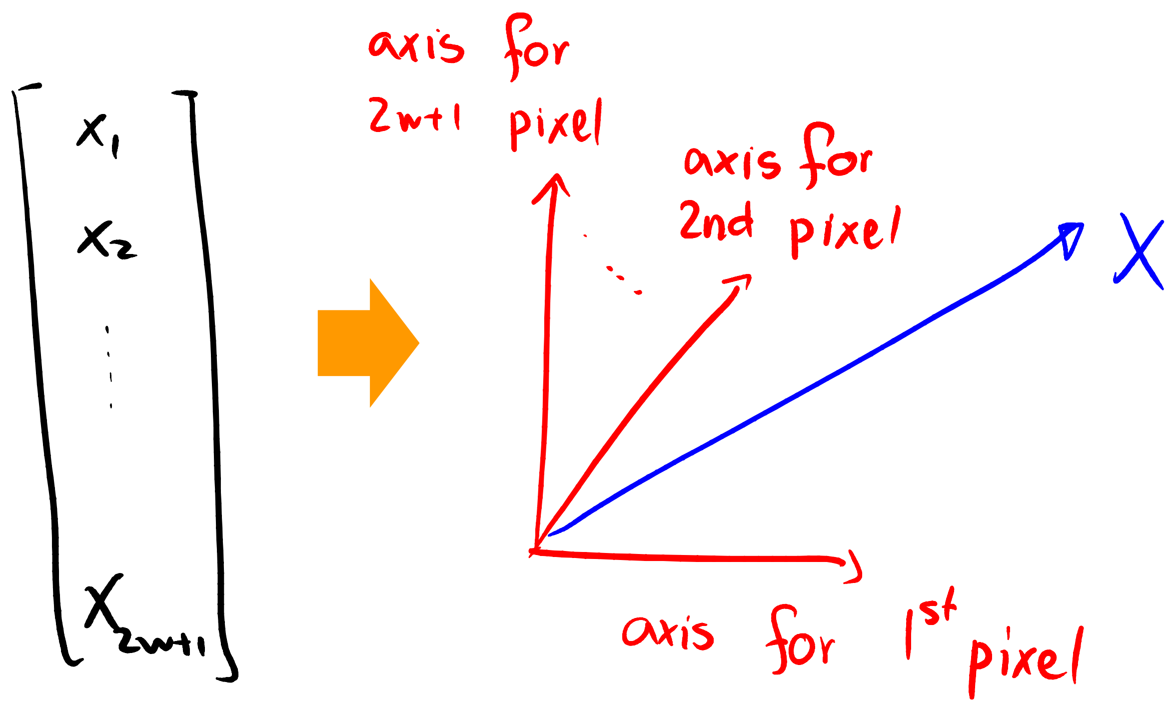
Representing Images & Patches as Vectors

As graph in 2D





And you can think of it as a point in the (2w+1)-dimensional space.

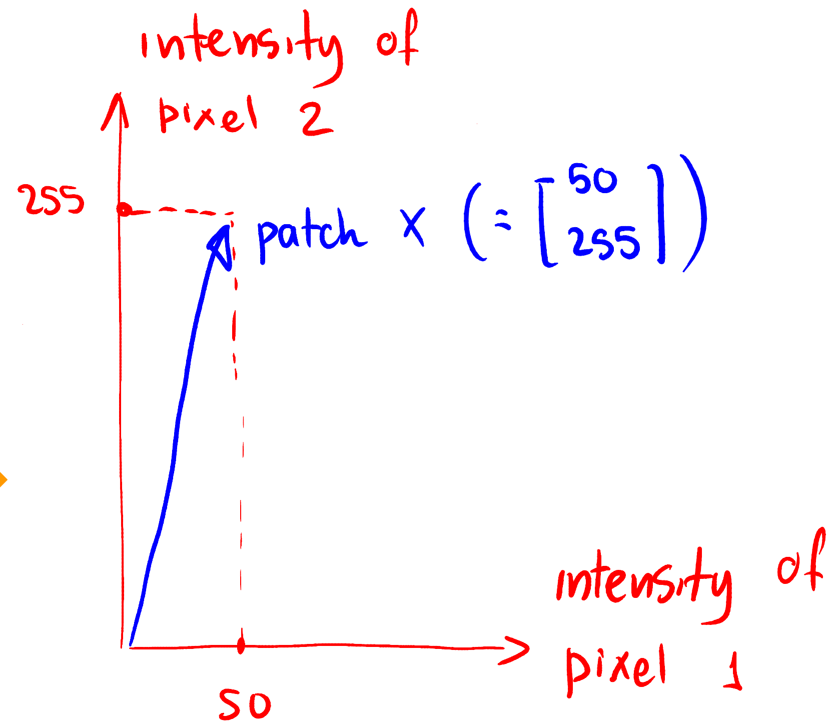


Representing Images & Patches as Vectors

A simple example is a patch x that is just a 2-dimensional vector, such as a 2 image pixel patch:

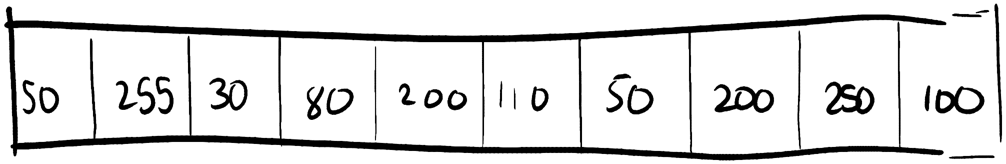
50	255

which can be represented as a point in 2D space

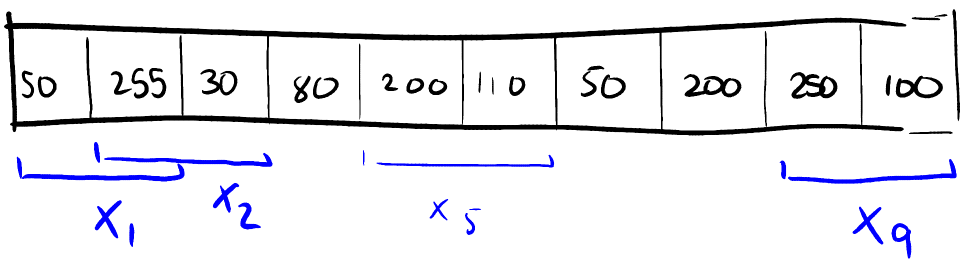


Representing Images & Patches as Vectors

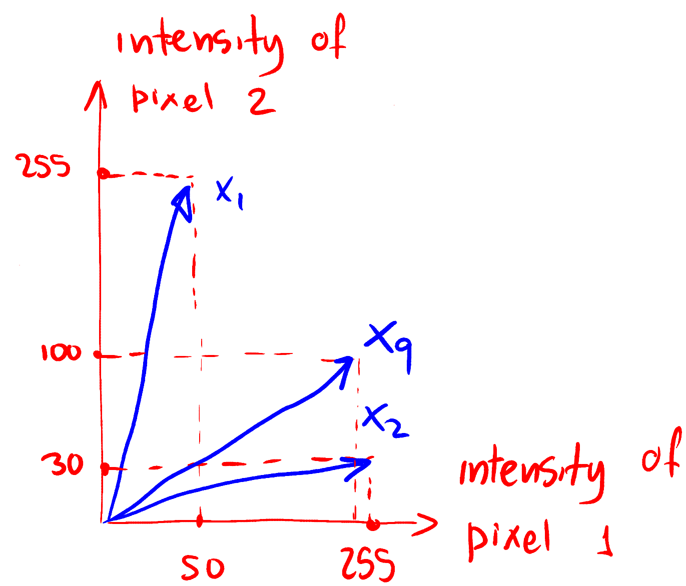
And if we think in terms of 2-pixel patches, a tiny (1x10 pixels) image like:



can be thought of as containing nine 2D patches (vectors)



with their corresponding 2D point representation:

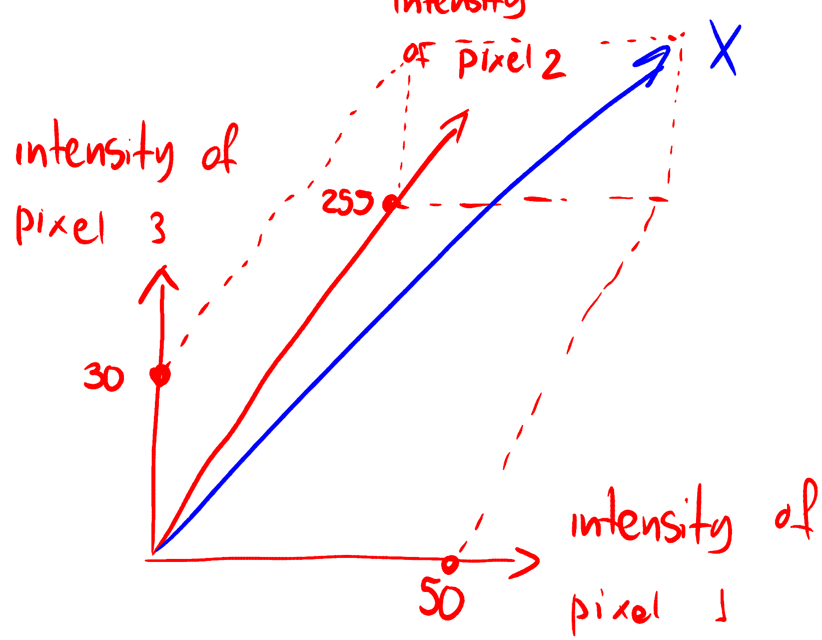


Representing Images & Patches as Vectors

Now, if patches are size 3, then a patch like

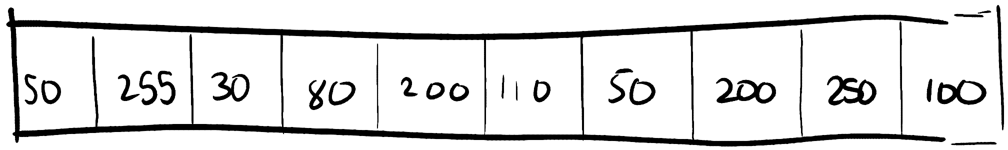
50	255	30

can be represented as a point in 3D space

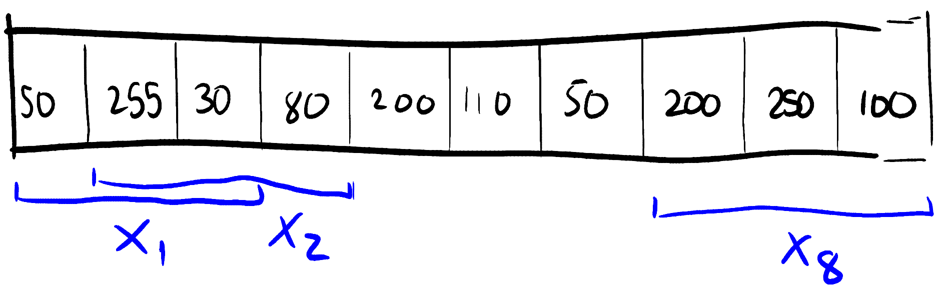


Representing Images & Patches as Vectors

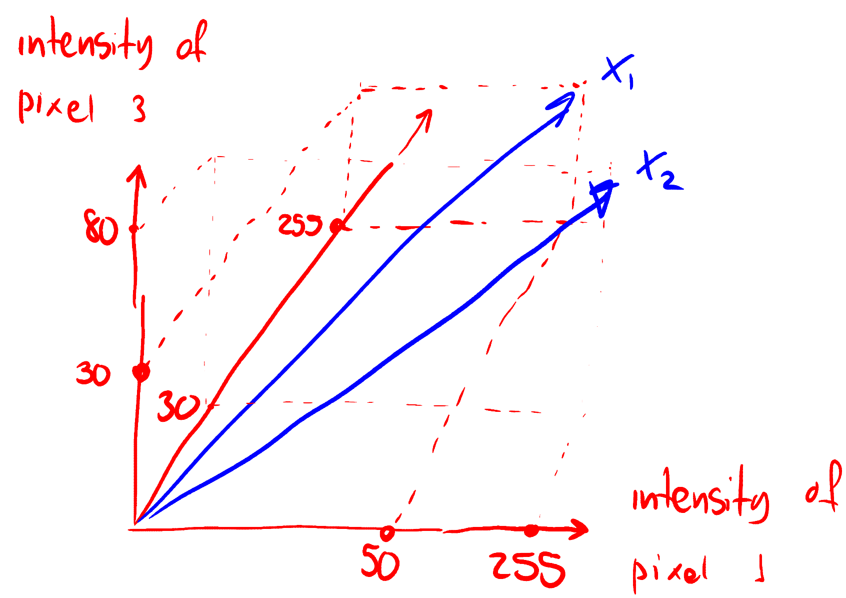
And the same tiny (1x10 pixels) image



can be thought of as containing eight (not nine) 3D vectors

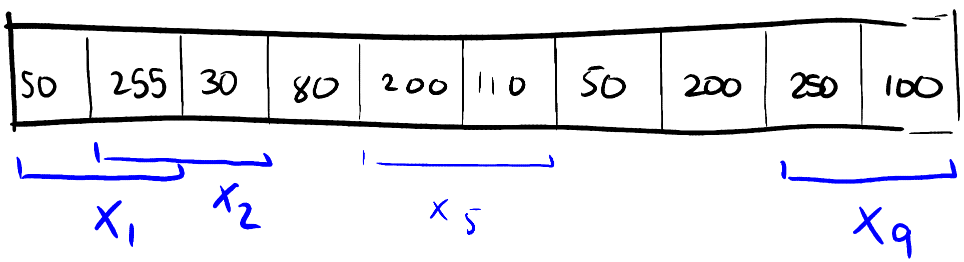


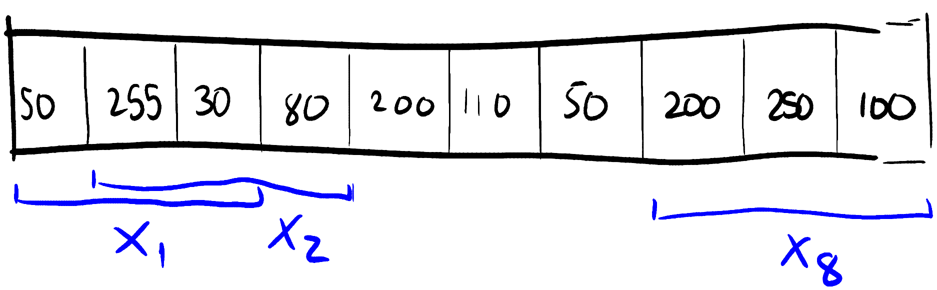
with their corresponding 3D point representation:



Representing Images & Patches as Vectors

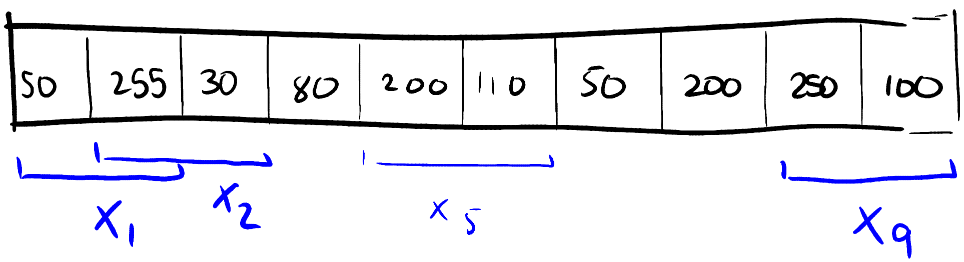
But the choice of patch size looks pretty arbitrary...

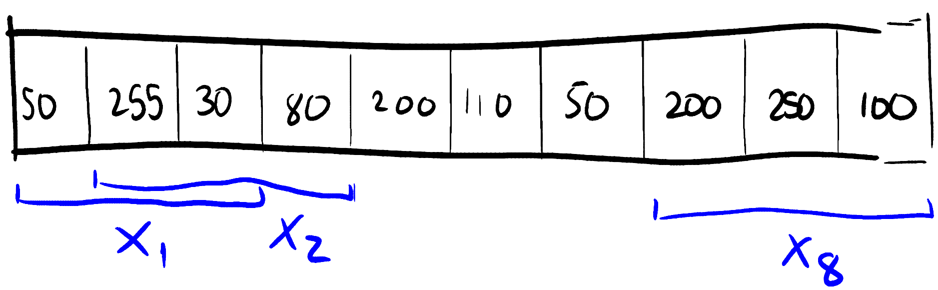




Representing Images & Patches as Vectors

But the choice of patch size looks pretty arbitrary...





And it is!

We determined which pixels go in each patch.

Representing Images & Patches as Vectors

Is there anything preventing us from creating 1D vectors from 2D patches then?

Think of the following 3x10 image:

50	255	30	80	30	100	50	200	250	100
80	200	100	50	60	30	30	60	30	90
150	90	30	80	90	100	250	100	240	

Can we think of this patch as a vector of size 9?

Representing Images & Patches as Vectors

Is there anything preventing us from creating 1D vectors from 2D patches then?

Think of the following 3x10 image:

50	255	30	80	30	100	50	200	250	100
80	200	100	50	60	30	30	60	30	90
150	90	30	80	90	100	250	100	240	

$$X_i = \begin{bmatrix} 80 \\ 30 \\ 100 \\ 50 \\ 60 \\ 30 \\ 80 \\ 90 \\ 100 \end{bmatrix}$$

Representing Images & Patches as Vectors

Is there anything preventing us from creating 1D vectors from 2D patches then?

Think of the following 3x10 image:

50	255	30	80	30	100	50	200	250	100
80	200	100	50	60	30	30	60	30	90
150	90	30	80	90	100	250	100	240	

$$X_i = \begin{bmatrix} 80 \\ 30 \\ 100 \\ 50 \\ 60 \\ 30 \\ 80 \\ 90 \\ 100 \end{bmatrix}$$

Absolutely!

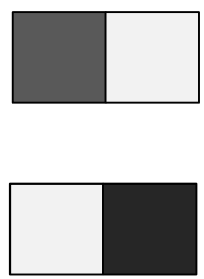
Equally arbitrary, but similar properties and interpretation.

Representing Images & Patches as Vectors

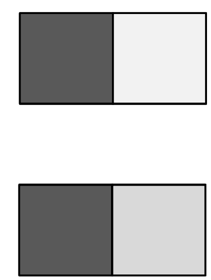
Now, why would we want to loose the absolute spatial information?

Representing Images & Patches as Vectors

Because a vector representation allows us to compute similarity between (1D or 2D) patches using simple vector operations.



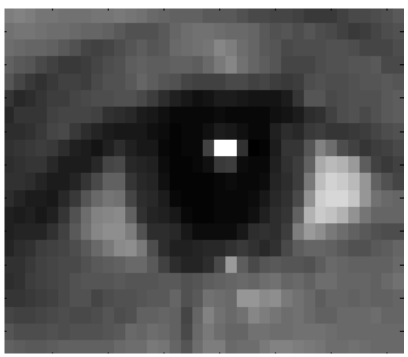
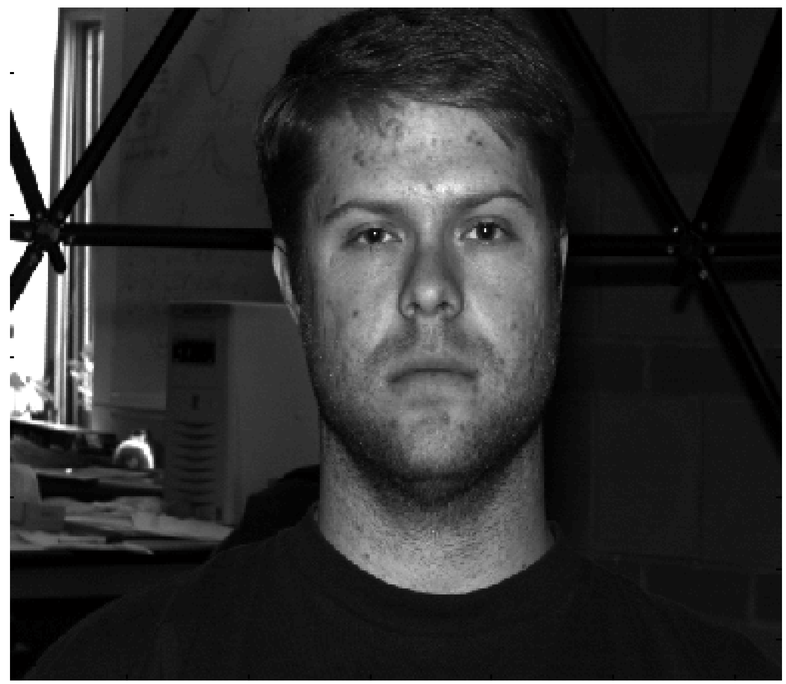
Dissimilar



Similar

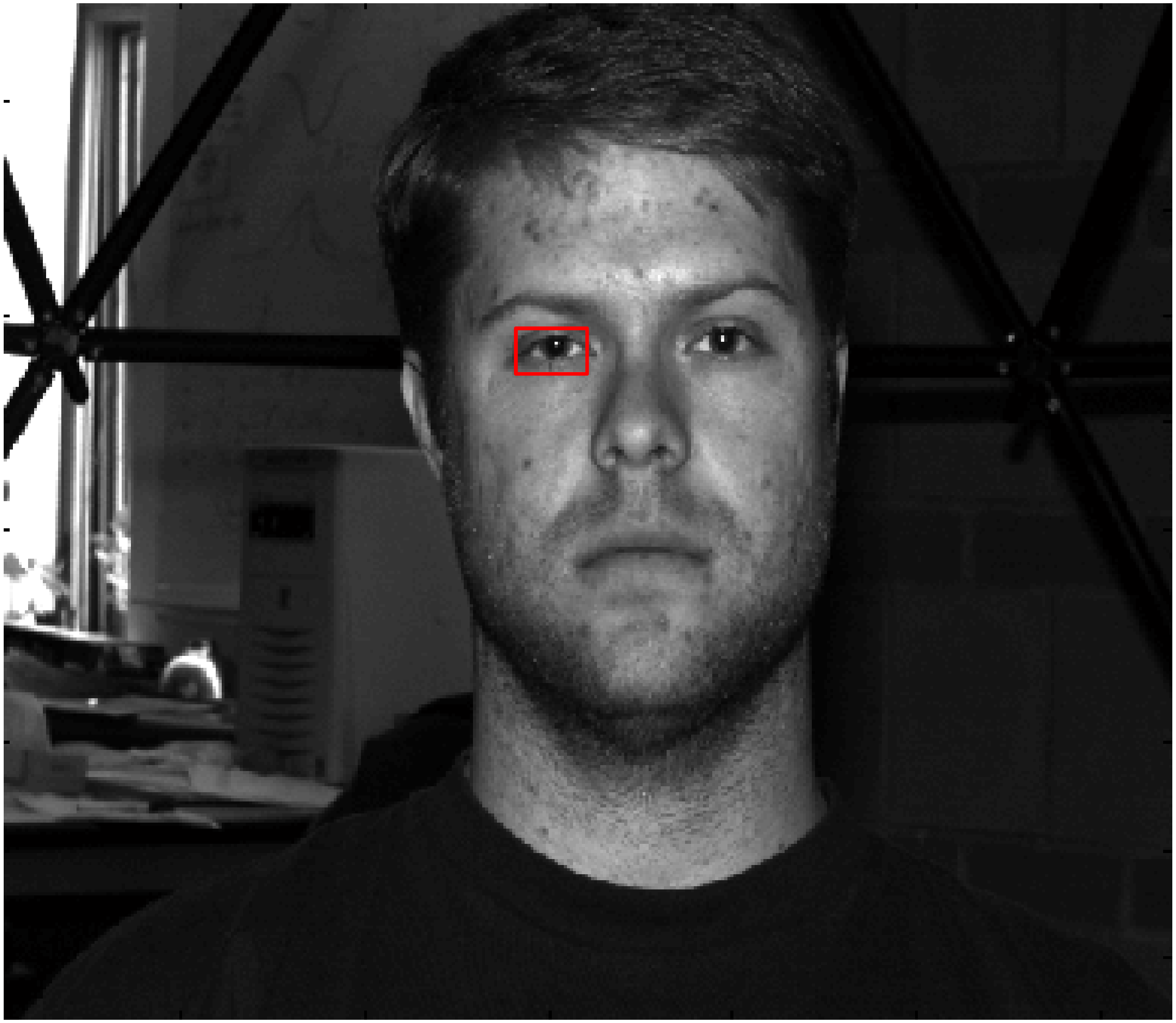
Representing Images & Patches as Vectors

Patch similarity is the foundation of an important detection procedure called template matching.
The intuition is, can we find the location of

this template  in 

Representing Images & Patches as Vectors

And get this location as the one most similar?



Representing Images & Patches as Vectors

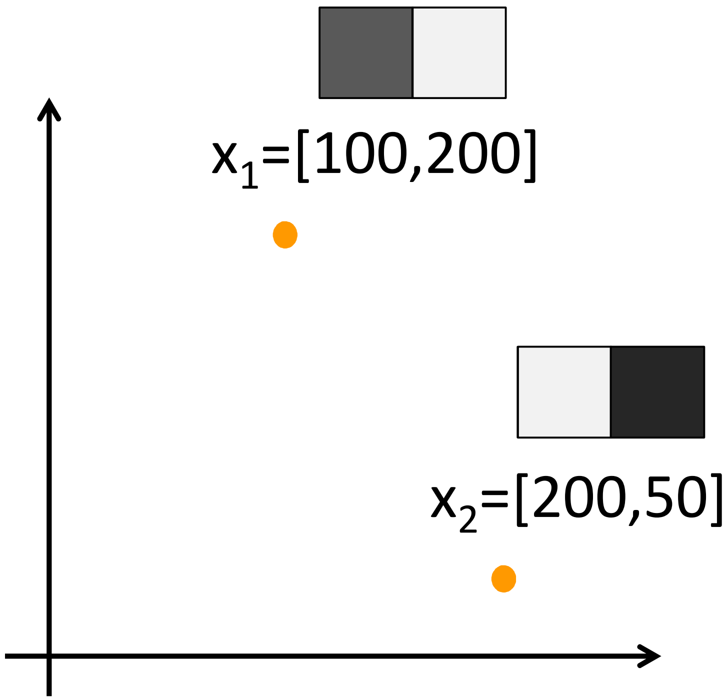
Template matching sounds useful but…

How?

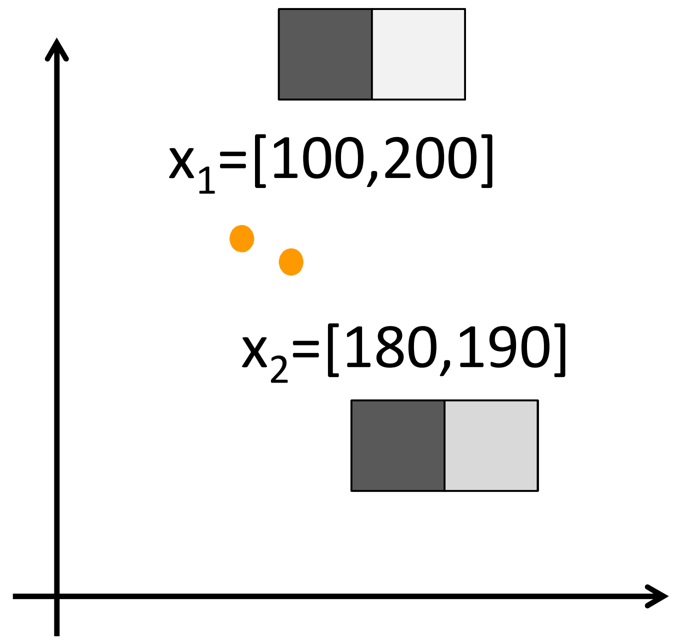
At what computational cost (in terms of memory and number of operations)?

Representing Images & Patches as Vectors

Estimating similarity between image patches.
Here are the patches from 3 slides ago, in 2D space:



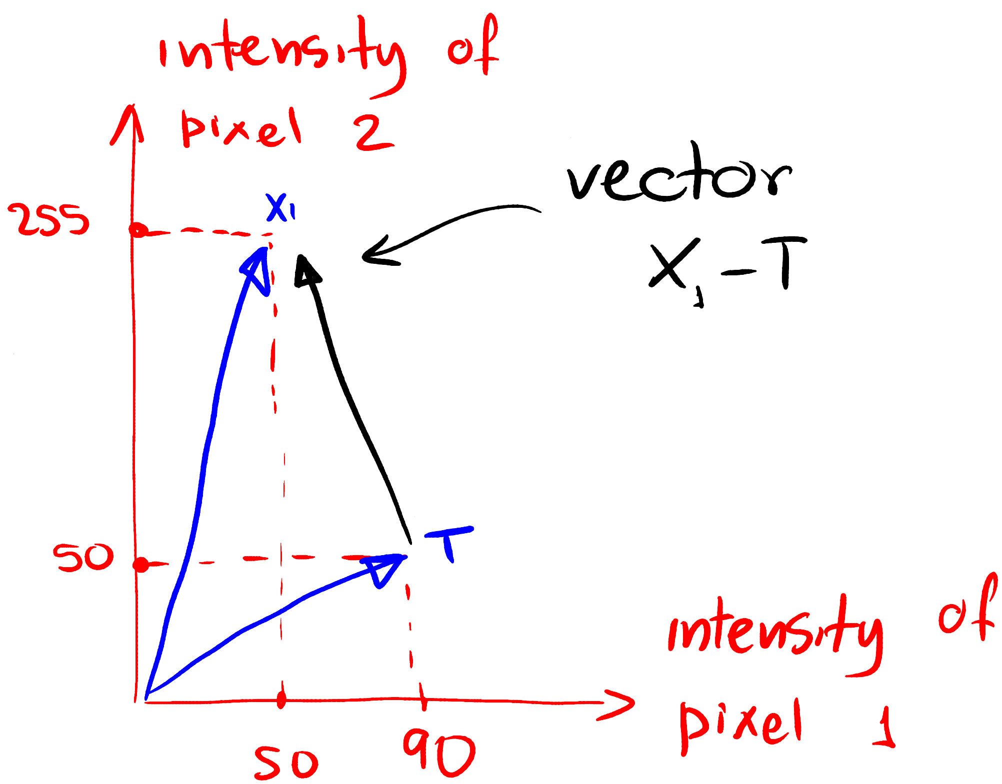
Dissimilar



Similar

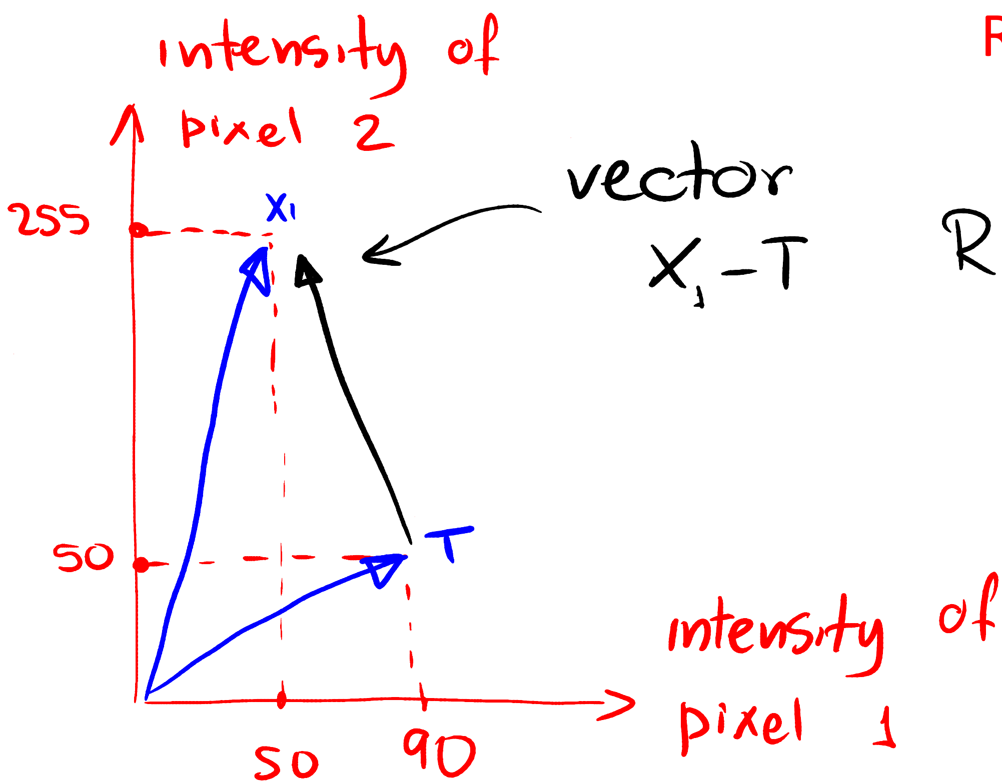
The Template Matching Problem

The goal is to find the image patch x_i that is most similar to a template T.
How about the distance?



The Template Matching Problem

Measuring distances can be done in many different ways.
Here is our Similarity function #1.



Root Mean Squared Distance

$$RMS(X_i, T) = \| X_i - T \|$$

$$= \left[(X_i - T)^T (X_i - T) \right]^{1/2}$$

The Template Matching Problem

The goal is to find the image patch x_i that is most similar to a template T. The problem can be formally written as:

Find $$\underset{x_i}{\operatorname{argmin}} \; \| x_i - T \|$$

RMS

argmin is a shorthand for “the x_i that minimizes the expression to the right”.

The Template Matching Problem

The goal is to find the image patch x_i that is most similar to a template T. The problem can be formally written as:

Find $$\underset{x_i}{\operatorname{argmin}} \ \|x_i - T\|$$

Note that efficiency can be improved by minimizing $\|x_i - T\|^2$ which minimizes in the same x_i and saves the square root computations.

This new formulation can then be written as:

$$\underset{x_i}{\operatorname{argmin}} \ (x_i - T)^T (x_i - T)$$

The Template Matching Problem

Again, note that this 1D metric is equivalent to the 2D operation that keeps the spatial relation of the template and the image.

For instance, if a patch is centered at pixel (r,c) and the template is of radius N, then the following equation computes the RMS distance between the image patch and the template.

$$\text{rms\_dist}(r,c) = \sqrt{\sum_{a=-N}^{N} \sum_{b=-N}^{N} \left(I(r+a, c+b) - T(a,b) \right)^2}$$

Template Matching Algorithm

Basic Template Matching Algorithm:

1. Define a matrix `RMS_Dist` of size equal to the image. This will hold the RMS distance at each pixel.
2. Compute $(x_i - T)^T (x_i - T)$ for each patch x_i (centered at coordinates (c, r) in the image), where it is possible to compute.
3. When RMS distances between the template and all patches have been computed, search over `RMS_Dist` to find the pixel with lowest intensity.

Template Matching Algorithm

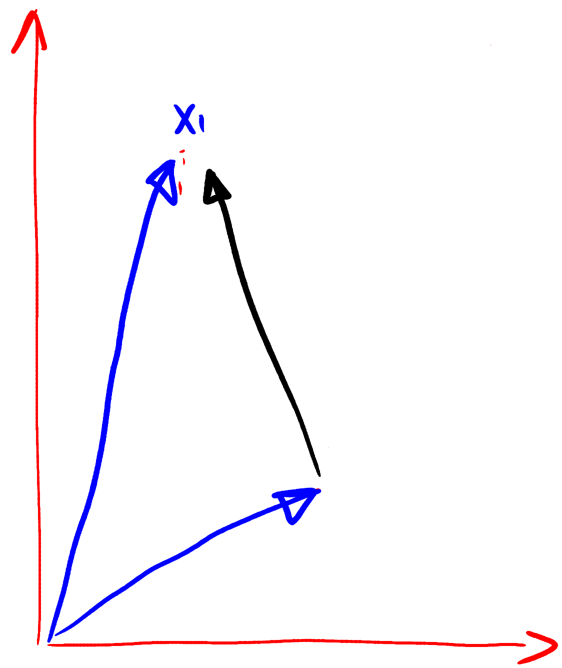
Basic Template Matching Algorithm:

1. Define a matrix `RMS_Dist` of size equal to the image. This will hold the RMS distance at each pixel.
2. Compute $(x_i - T)^T(x_i - T)$ for each patch x_i (centered at coordinates (c, r) in the image), where it is possible to compute.
3. When RMS distances between the template and all patches have been computed, search over `RMS_Dist` to find the pixel with lowest intensity.

What is the problem with this distance metric?

Template Matching Algorithm

The problem with this distance metric.

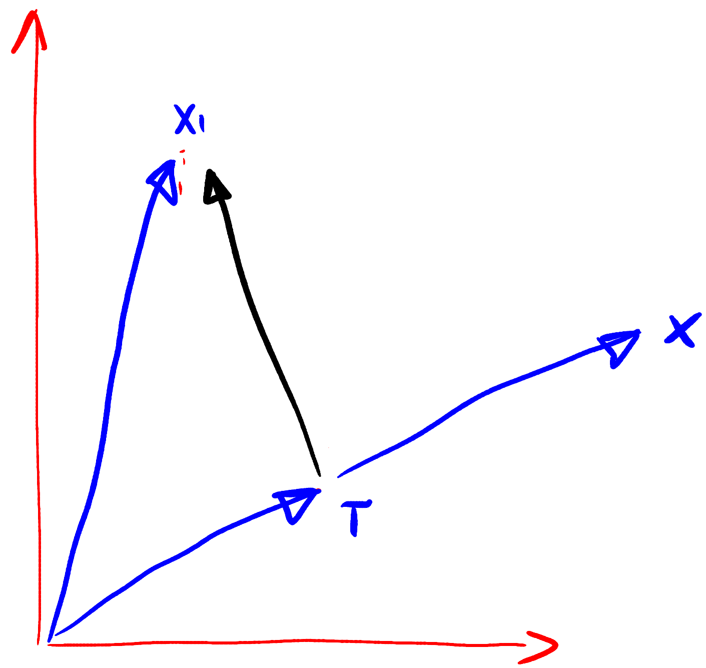


$$\left[(x_i - T)^T (x_i - T)\right]^{1/2}$$

Thoughts?

Template Matching Algorithm

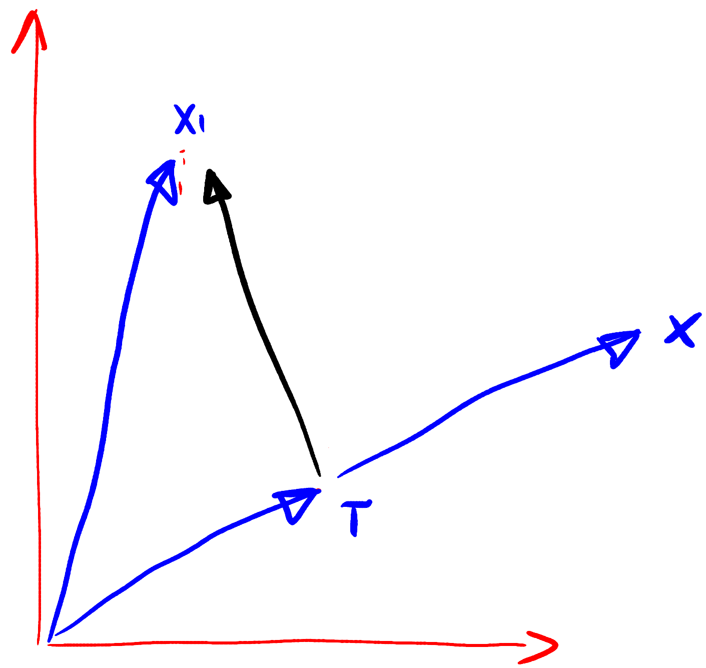
The problem with this distance metric.



What is the distance from T to this new patch x?

Template Matching Algorithm

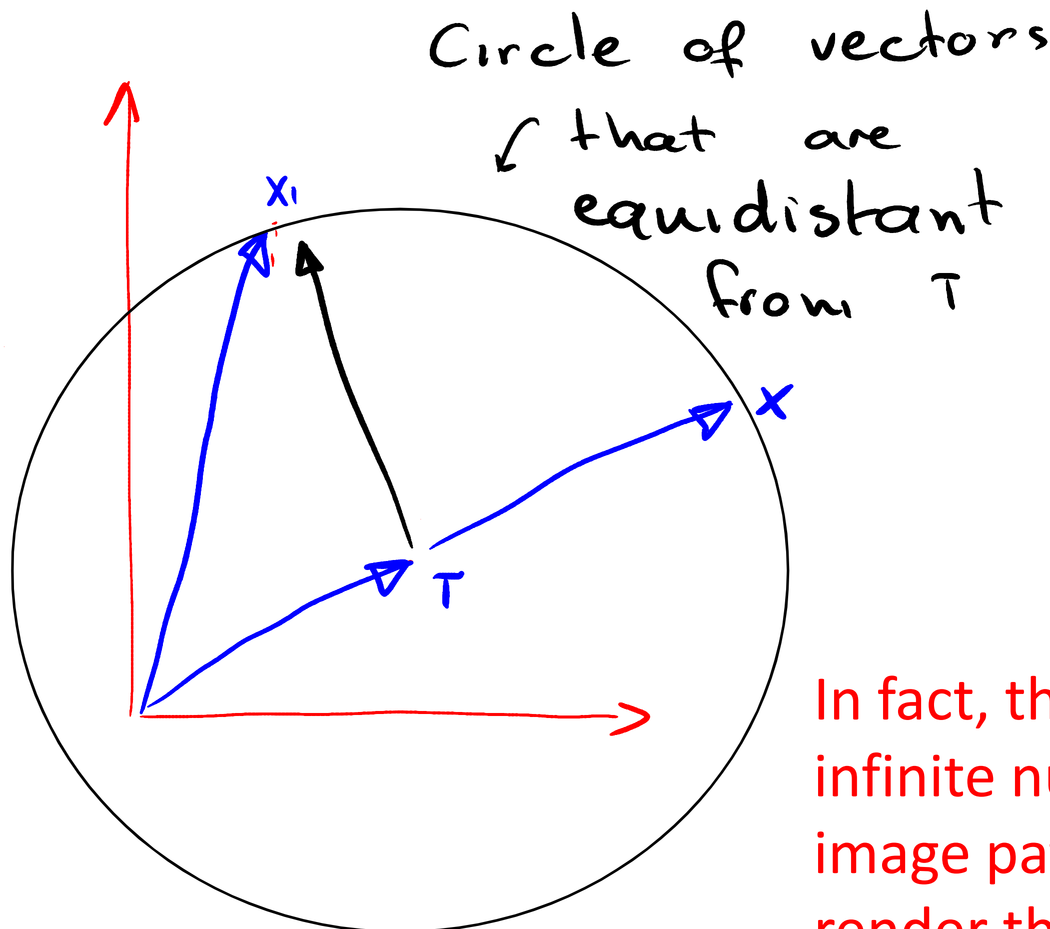
The problem with this distance metric.



The same!

Template Matching Algorithm

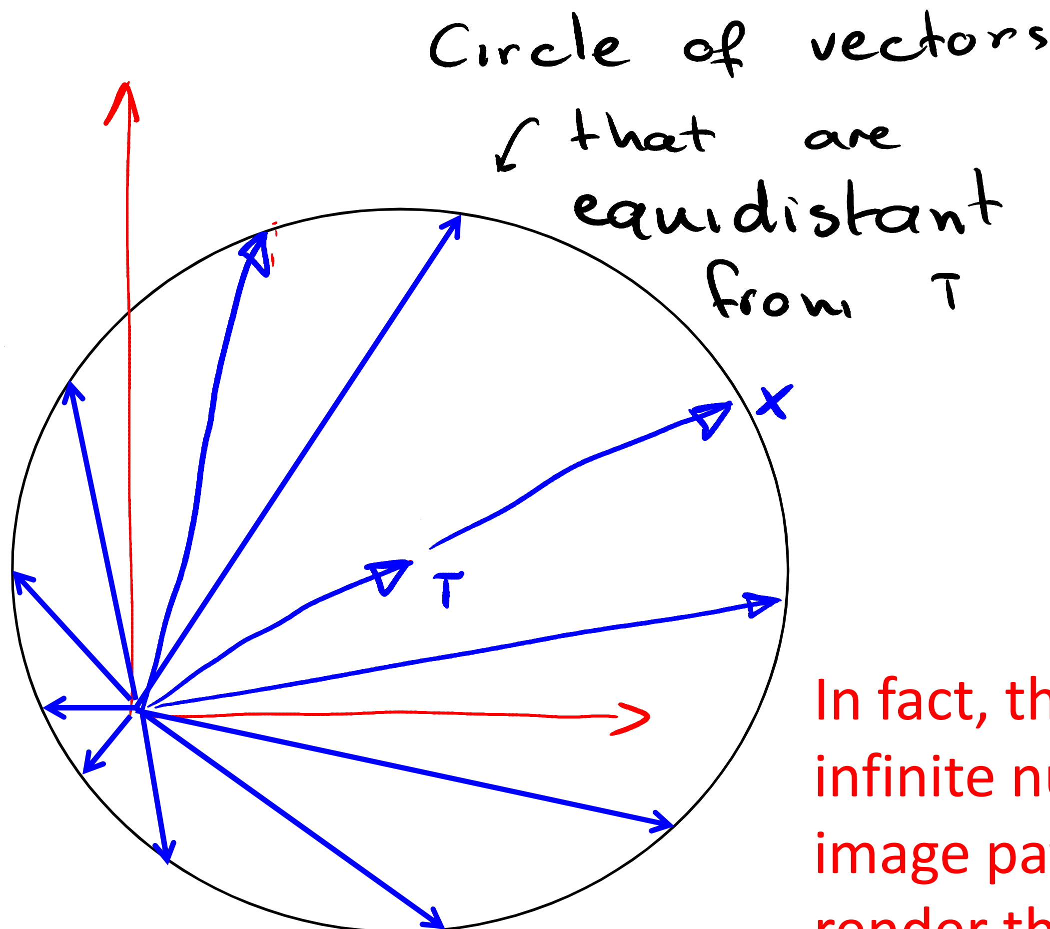
The problem with this distance metric.



In fact, there is an infinite number of image patches that render the same distance.

Template Matching Algorithm

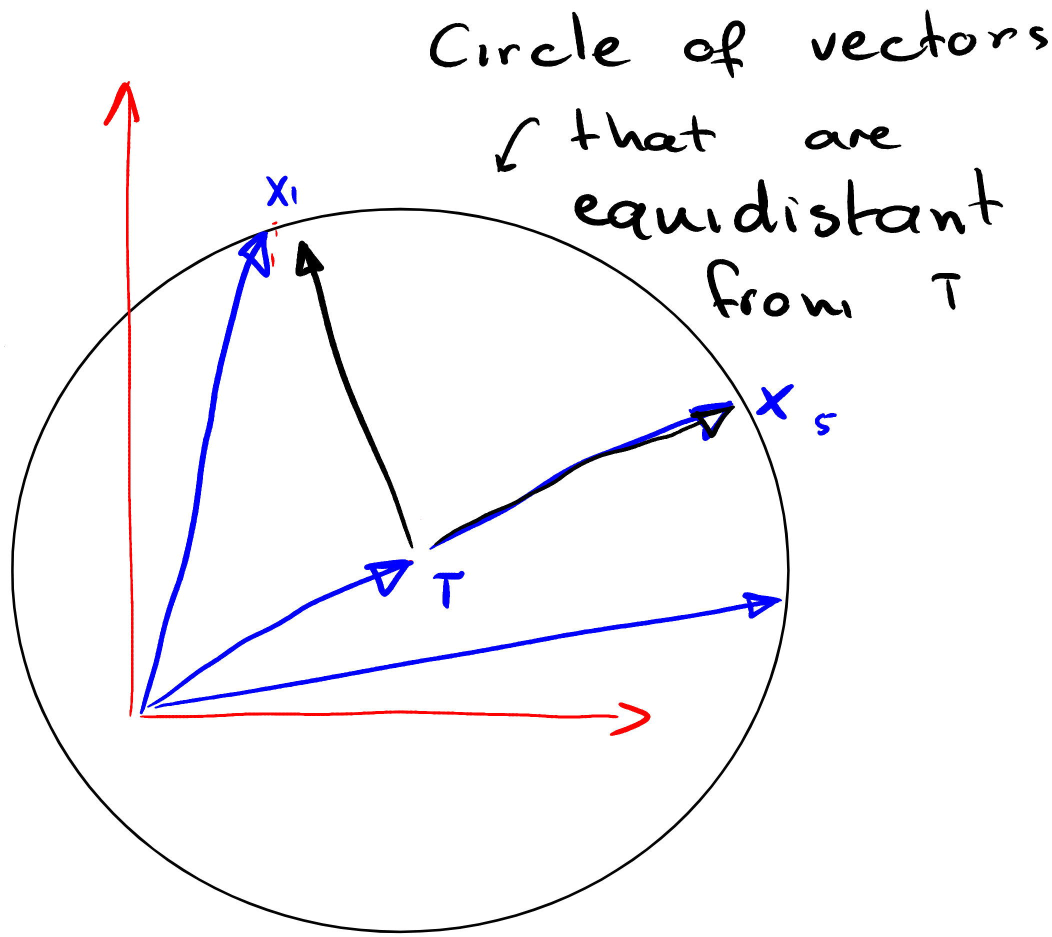
But there is a problem with this distance metric.



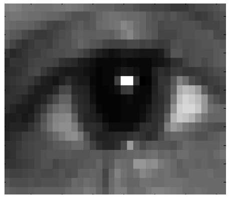
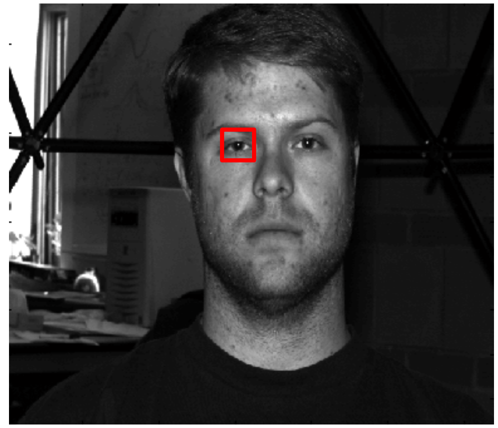
In fact, there is an infinite number of image patches that render the same distance.

Template Matching Algorithm

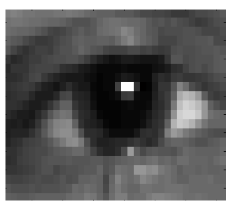
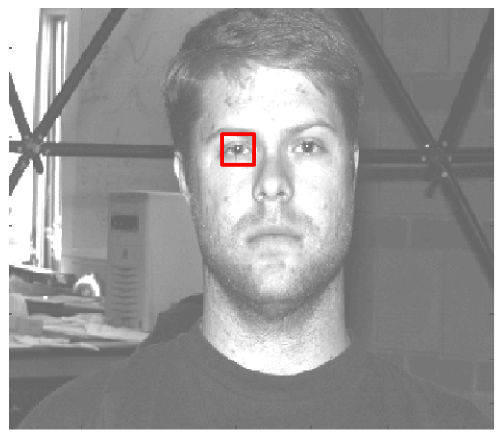
But there is a problem with this distance metric.



Representing Images & Patches as Vectors

RMS of  in  would be small

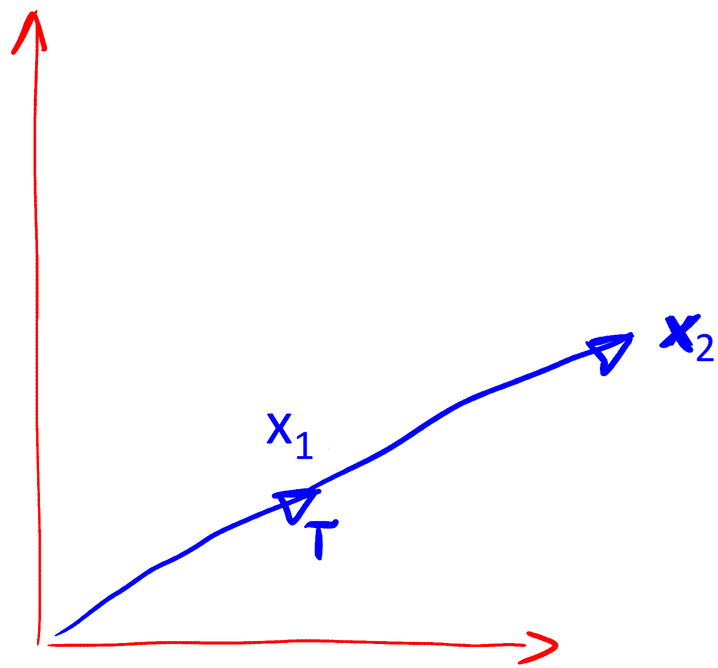
I_1

RMS of  in  would be large

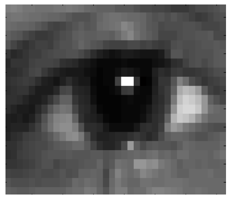
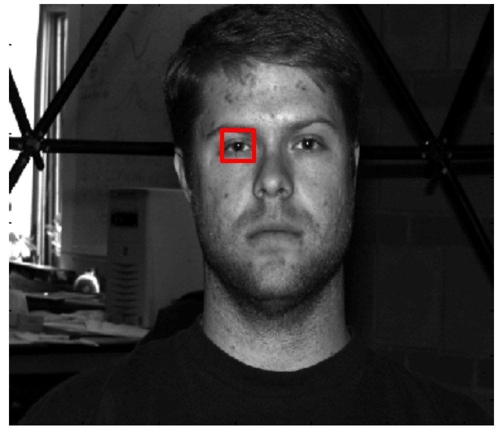
I_2

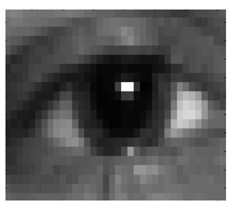
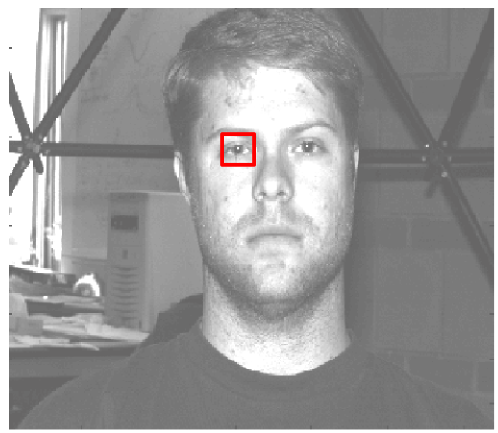
Template Matching Algorithm

I_2 is just a scaled version of x_1, but the RMS(x_2,T) is much bigger than RMS(x_1,T) (which is almost zero)



Representing Images & Patches as Vectors

RMS of  in  would be small

RMS of  in  would be large

RMS cannot distinguish between patches that are just scaled versions of T, from other patches that differ in other ways.

Representing Images & Patches as Vectors

Is there anything we can do?

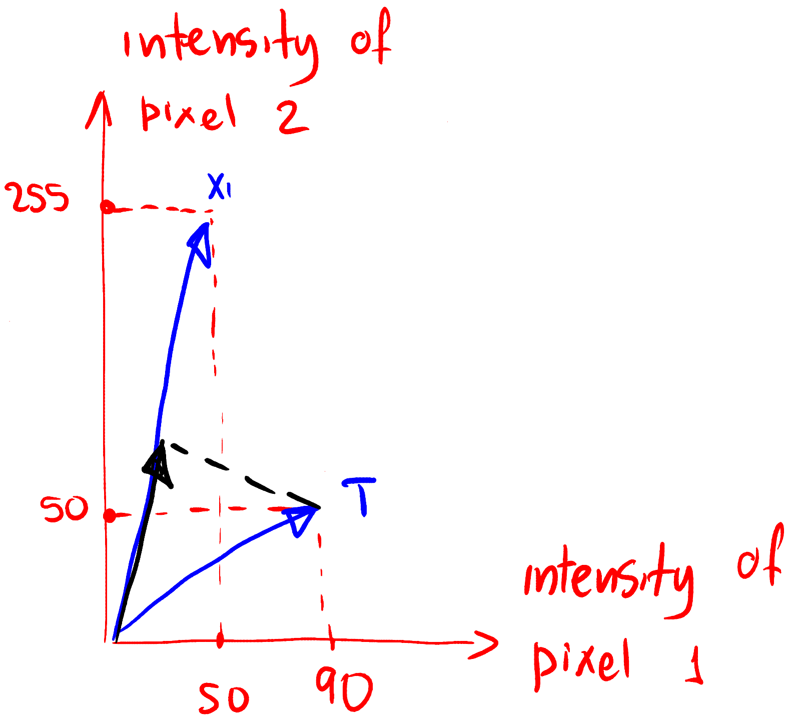
Topic 05:

Representing Images as n-Dimensional Vectors

- Template matching:
  - cross-correlation & normalized cross-correlation
- Principal component analysis
  - geometrical intuition: changing basis
  - the eigenfaces recognition algorithm
  - algorithm derivation: minimizing sample covariance

The Template Matching Problem

Measuring distances can be done in many different ways.
Similarity function #2.



Cross-correlation

$$cc(x_i, T) = x_i^T \cdot T$$

i.e. the dot product between the two vectors

The projection from one onto the other.

The Template Matching Problem

Properties of cross-correlation $CC(x_i, T)$

Depends on the lengths of x_i and T (still ☹)

From all x_i of the same length, the CC is maximum when x_i and T have the same direction.

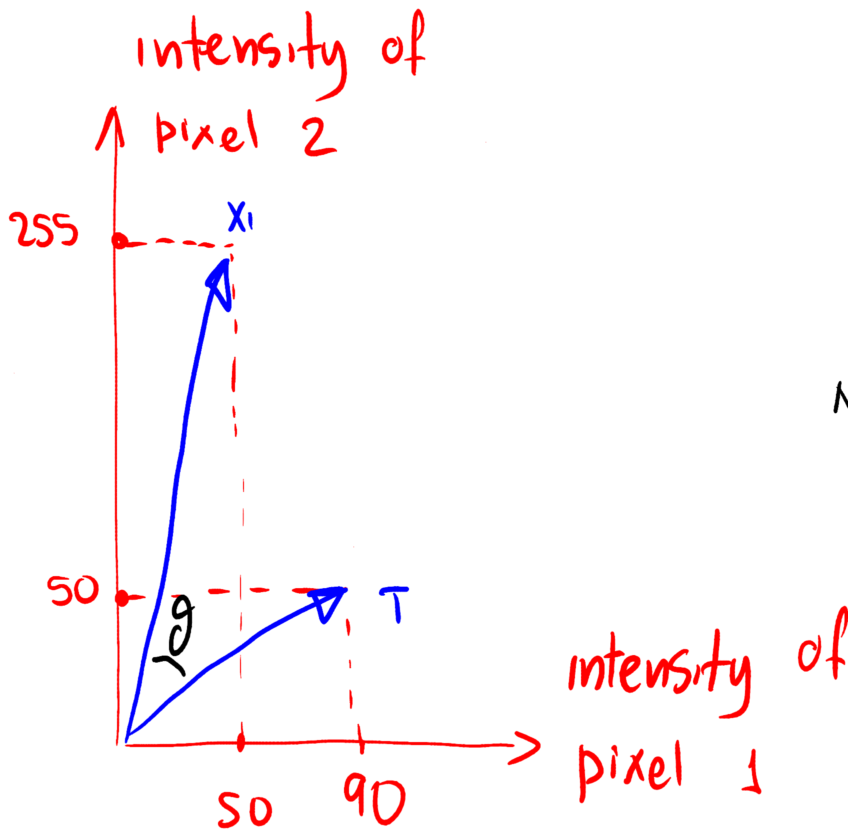
CC is zero when x_i and T are orthogonal (most dissimilar!)

$$CC(x_i, T) = X_i^T \cdot T$$

$$= \|X_i\| \cdot \|T\| \cdot \cos\theta$$

The Template Matching Problem

Measuring distances can be done in many different ways.
Similarity function #3.



Normalized Cross-Correlation

$$NCC(x_i, T) = \frac{x_i^T \cdot T}{\|x_i\| \cdot \|T\|}$$

$$= \cos\theta$$

The cosine of the angle between the two vectors.

The Template Matching Problem

Properties of Normalized Cross-Correlation:

Independent of lengths of x_i and T (finally ☺)

Maximum ($NCC(x_i, T)=1$) when the intensities of x_i and T are the same, up to a scale factor

Minimum ($NCC(x_i, T) = 0$) when x_i and T are orthogonal (most dissimilar).

$NCC(x_i, T) = CC(x_i, T)$ when x_i and T are unit vectors.

2D Template Matching Using CC & NCC

Note that Cross-Correlation and Normalized Cross-Correlation can be computed as a 2D sum

$$cc\_dist(r,c) = \sum_{a=-1}^{1} \sum_{b=-1}^{1} I(r+a,\, c+b)\, T(a,b)$$

row 0 →

50	255	90
80	200	100
150	90	30

↑ col 0

row r →

50	255	30	80	30	100	50	200	250	100
80	200	100	50	60	30	30	60	30	90
150	90	30	80	90	100	250	100	240	

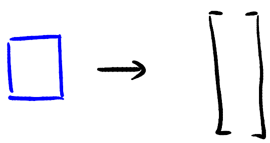
↑ col c

Q: What is the 2D sum expression for NCC ?

2D Template Matching Using CC & NCC

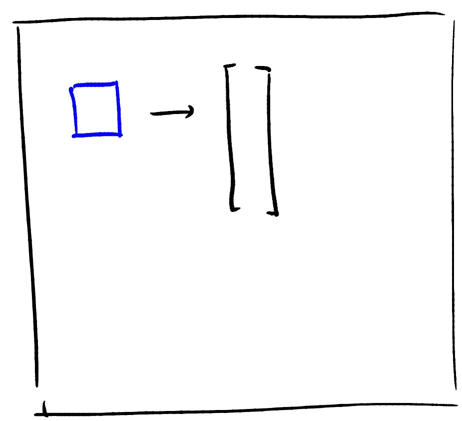
Applying this procedure to the entire image

Template T (M pixels)



T = M-dimensional column vector

Image I (N patches)



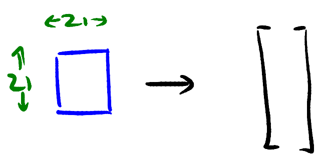
Each image patch = M-dimensional column vector

What is the computational complexity?

2D Template Matching Using CC & NCC

For instance match a template of 21x21 to an image of 1000x1000

Template T (M pixels)

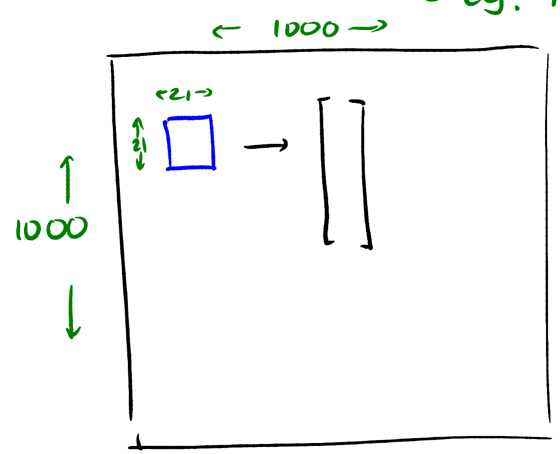


eg. $M = 21^2 = 441$

T = M -dimensional column vector

Image I (N patches)

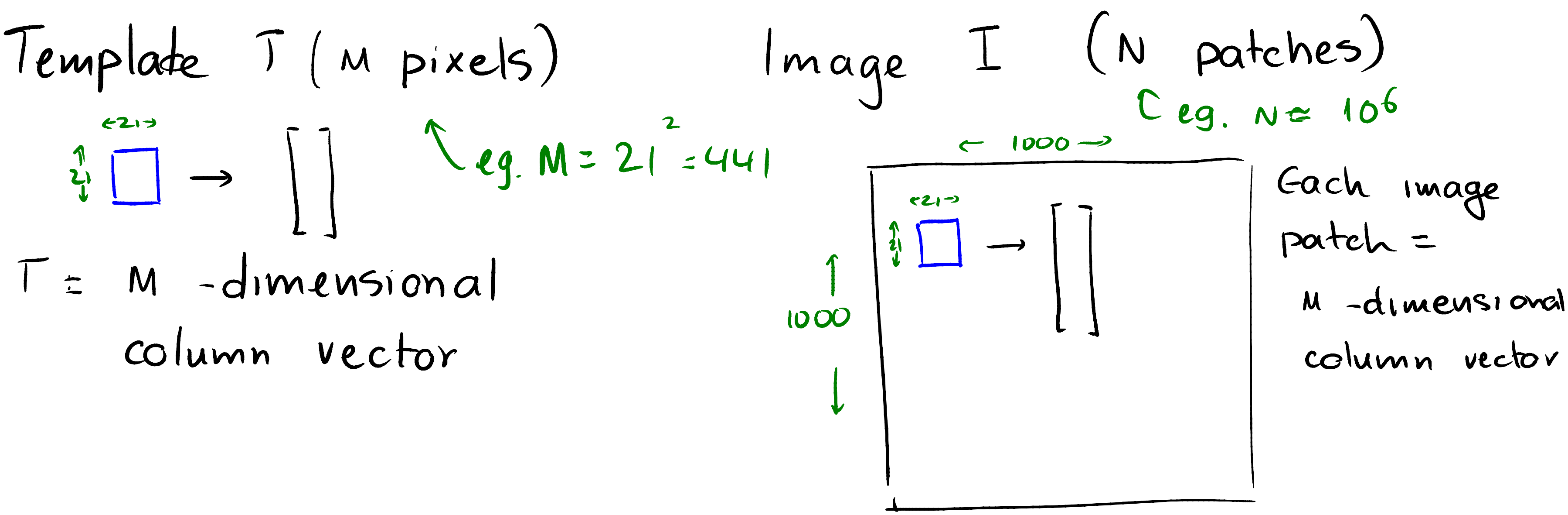
(eg. $N \approx 10^6$



Each image patch = M -dimensional column vector

2D Template Matching Using CC & NCC

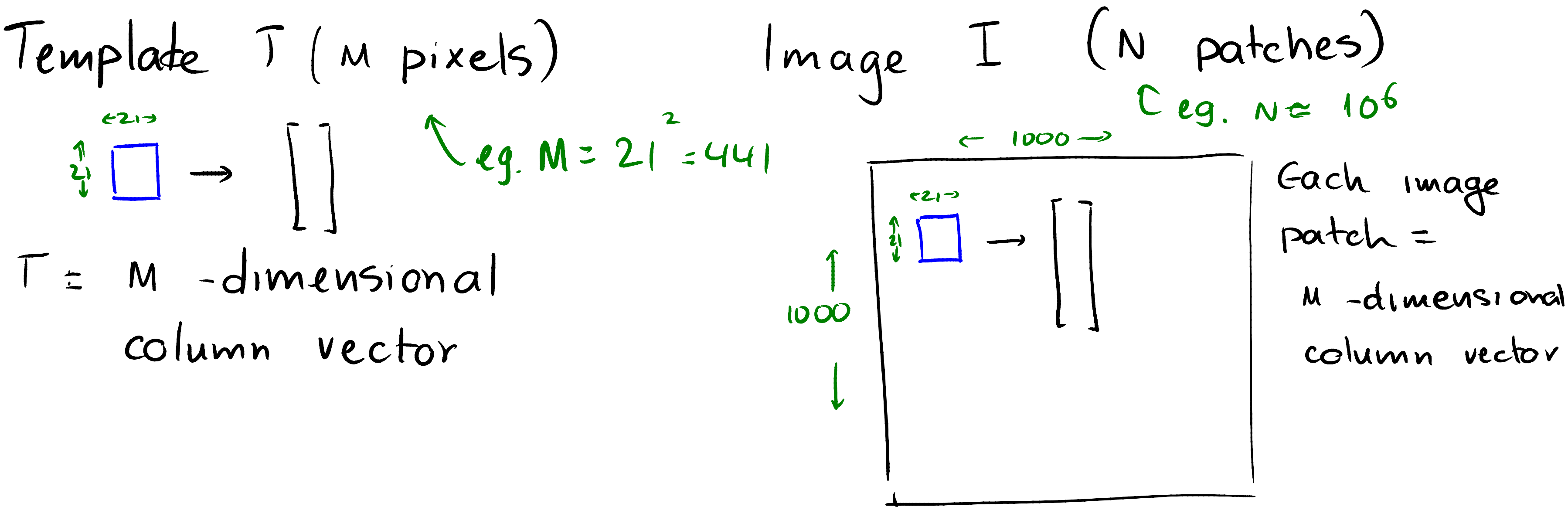
For instance match a template of 21x21 to an image of 1000x1000



If we use CC as the distance metric, we do M multiplications and M-1 additions per pixel in the image I.

2D Template Matching Using CC & NCC

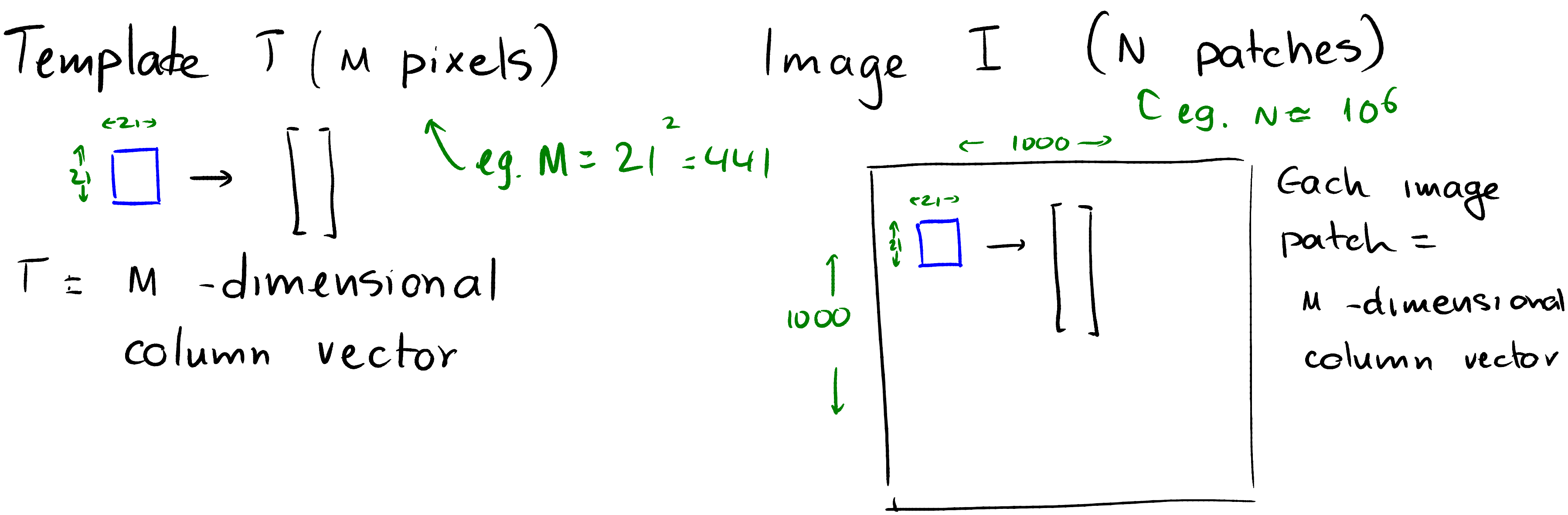
For instance match a template of 21x21 to an image of 1000x1000



The complexity when using CC is in the order of O(MN) operations for the entire image.

2D Template Matching Using CC & NCC

For instance match a template of 21x21 to an image of 1000x1000



The complexity of NCC is also O(MN), with only some more products and additions to normalize the patch (x_i) vectors.

2D Template Matching Using CC & NCC

For instance match a template of 21x21 to an image of 1000x1000

Template T (M pixels)

21

21

eg. $M = 21^2 = 441$

T = M-dimensional column vector

Image I (N patches)

eg. $N \approx 10^6$

1000

1000

21

21

Each image patch = M-dimensional column vector

These are over 1 billion operations!

2D Template Matching Using CC & NCC

Matching a template of 21x21 to an image of 1000x1000 requires around 1 billion operations.

Is there a way to represent x_i and T with d<<M to improve efficiency?

Taking O(MN) down to O(dN)

(with d = 5, as opposed to d = 441, for instance)

Template Matching: Computational Issues

This problem is called Dimensionality Reduction

and using it can lead to speed-ups of orders of magnitude!

Template Matching: Computational Issues

Demo!

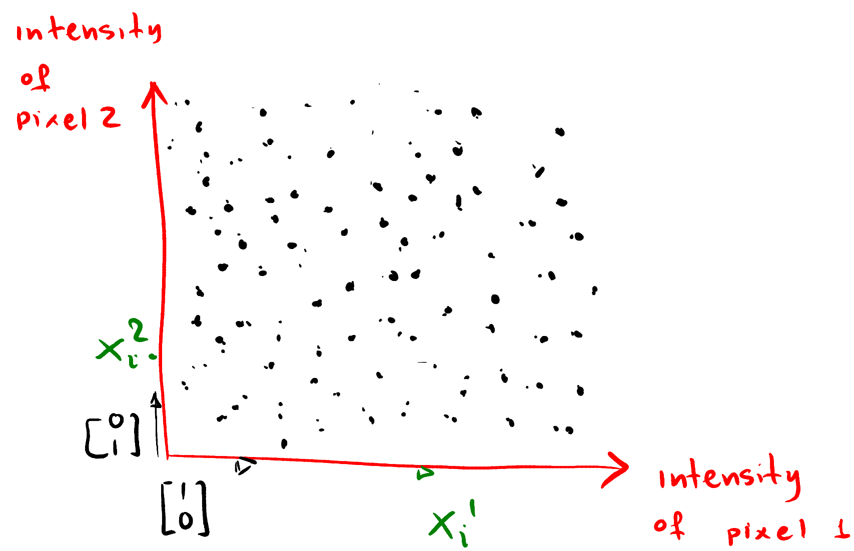
Topic 05:

Representing Images as n-Dimensional Vectors

- Template matching:
  - cross-correlation & normalized cross-correlation
- Principal component analysis
  - geometrical intuition: changing basis
  - the eigenfaces recognition algorithm
  - algorithm derivation: minimizing sample covariance

Linear Dimensionality Reduction: Basic Intuition

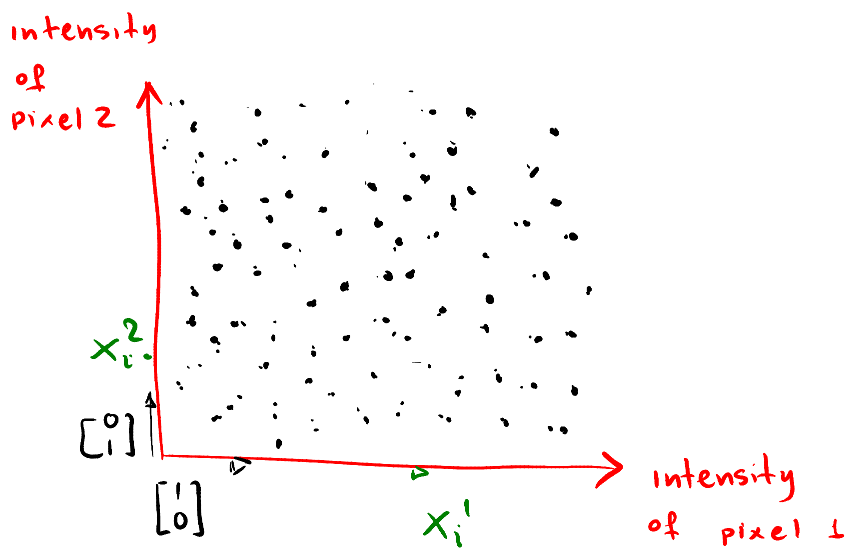
Lets look at an example to develop some intuition about dimensionality reduction.



Imagine a set of 2-pixel patches whose x_1 and x_2 values are uncorrelated

Linear Dimensionality Reduction: Basic Intuition

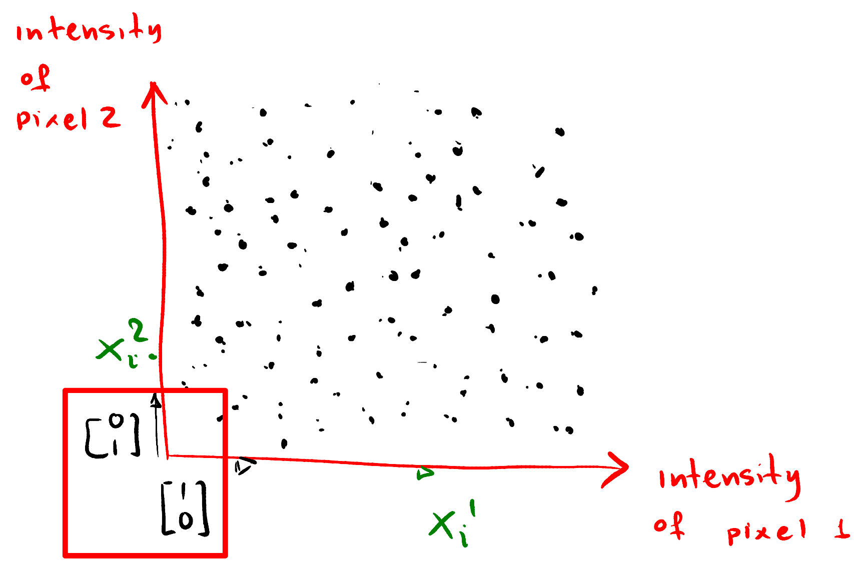
Lets look at an example to develop some intuition about dimensionality reduction.



The coordinate of each of these patches can be determined given two basis vectors.

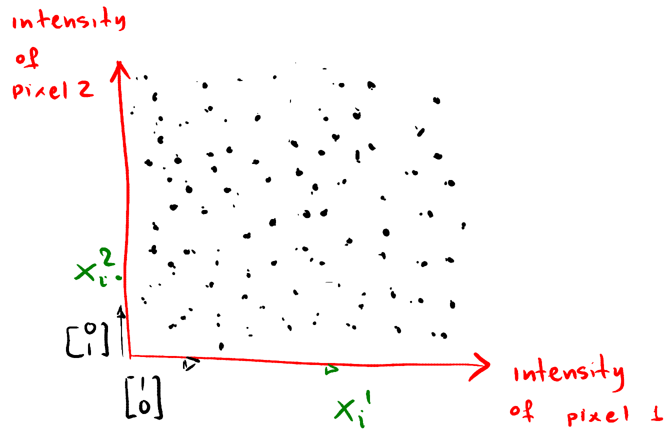
Linear Dimensionality Reduction: Basic Intuition

Lets look at an example to develop some intuition about dimensionality reduction.



I chose unit vectors that aligned with the x and y axis, but any two (non-parallel) vectors could have been used.

Linear Dimensionality Reduction: Basic Intuition



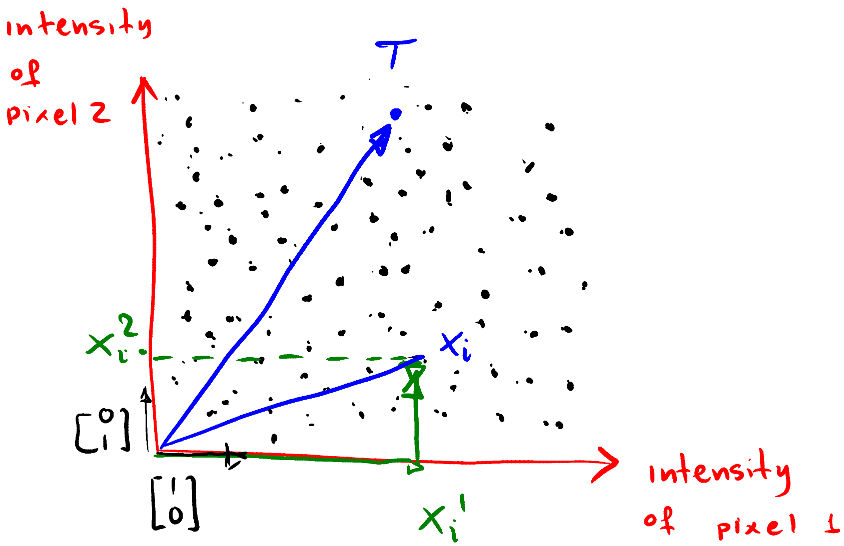
In this setting, a point x_i can be written as:

basis vectors

$$x_i = \begin{bmatrix} x_i^1 \\ x_i^2 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} x_i^1 + \begin{bmatrix} 0 \\ 1 \end{bmatrix} x_i^2$$

coordinates

Linear Dimensionality Reduction: Basic Intuition



And the CC (T, x_i) as

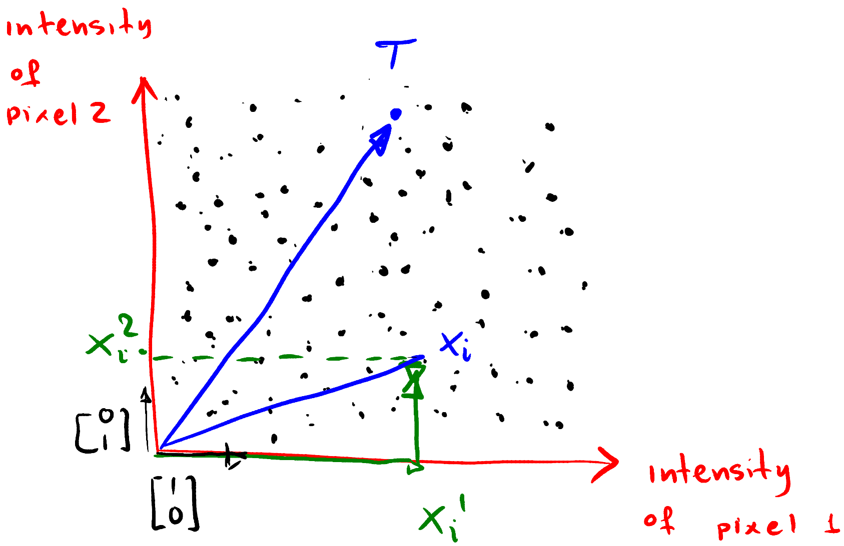
$$CC(T, x_i) = T^T \cdot x_i$$

unconstrained

$$= T^T \left(\begin{bmatrix}1\\0\end{bmatrix} x_i^1 + \begin{bmatrix}0\\1\end{bmatrix} x_i^2 \right)$$

$$= \left(T^T \begin{bmatrix}1\\0\end{bmatrix} \right) x_i^1 + \left(T^T \begin{bmatrix}0\\1\end{bmatrix} \right) x_i^2$$

Linear Dimensionality Reduction: Basic Intuition



And the CC (T, x_i) as

$$CC(T, x_i) = T^T x_i$$

$$= T^T\left(\begin{bmatrix}1\\0\end{bmatrix}\underbrace{x_i^1}_{\text{unconstrained}} + \begin{bmatrix}0\\1\end{bmatrix}\underbrace{x_i^2}_{\text{unconstrained}}\right)$$

$$= \left(T^T\begin{bmatrix}1\\0\end{bmatrix}\right)x_i^1 + \left(T^T\begin{bmatrix}0\\1\end{bmatrix}\right)x_i^2$$

Notice both x_i^1 and x_i^2 are relevant

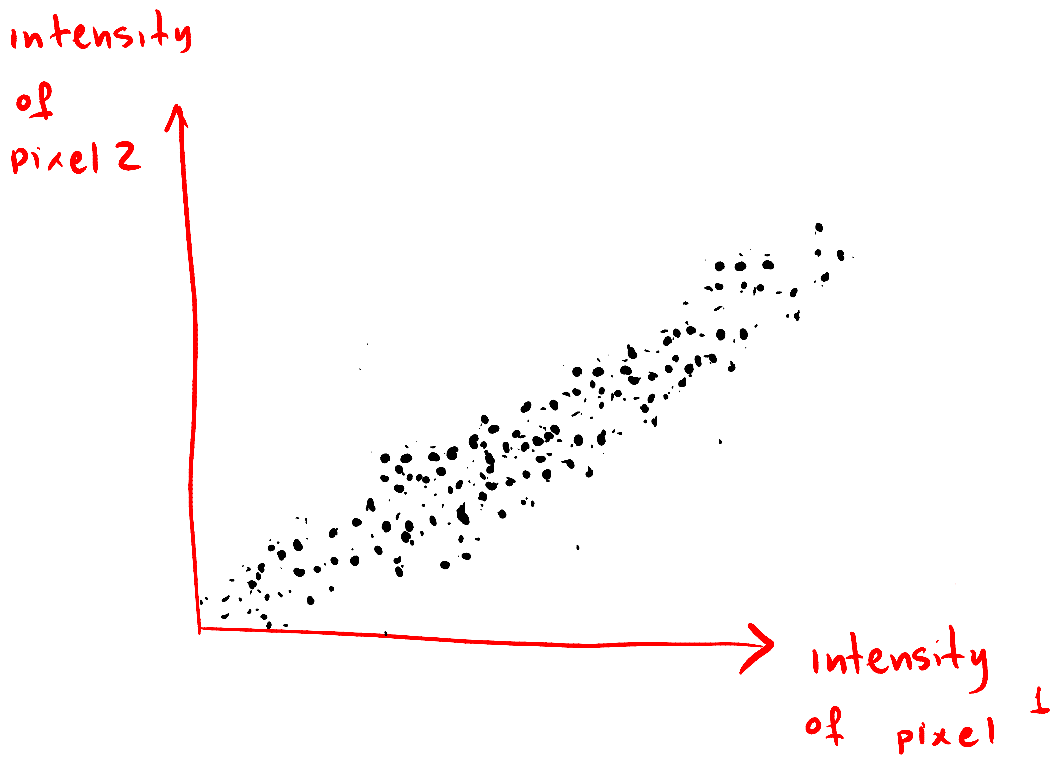
Linear Dimensionality Reduction: Basic Intuition

Now imagine that pixels in a patch are correlated.

Let’s not imagine, but look at some actual data!

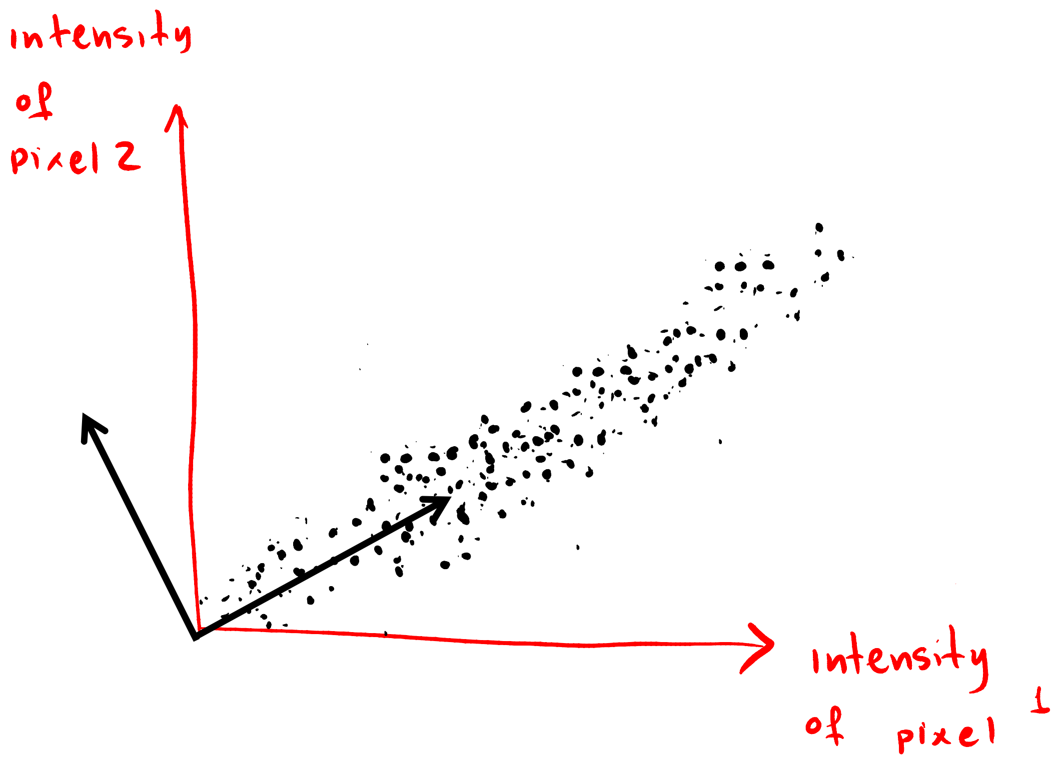
Linear Dimensionality Reduction: Basic Intuition

If pixel intensities are correlated, as in:

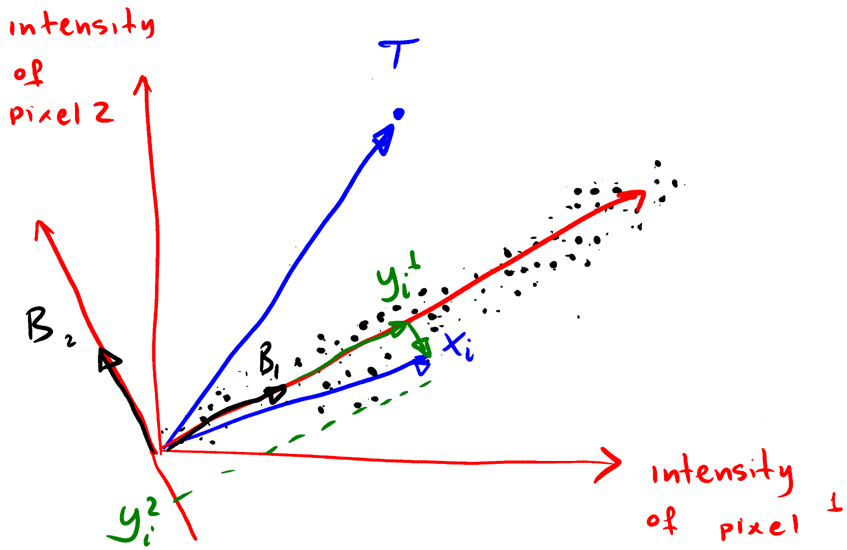


Linear Dimensionality Reduction: Basic Intuition

Then we can use different basis vectors with interesting properties, for instance assume the basis vectors in black.



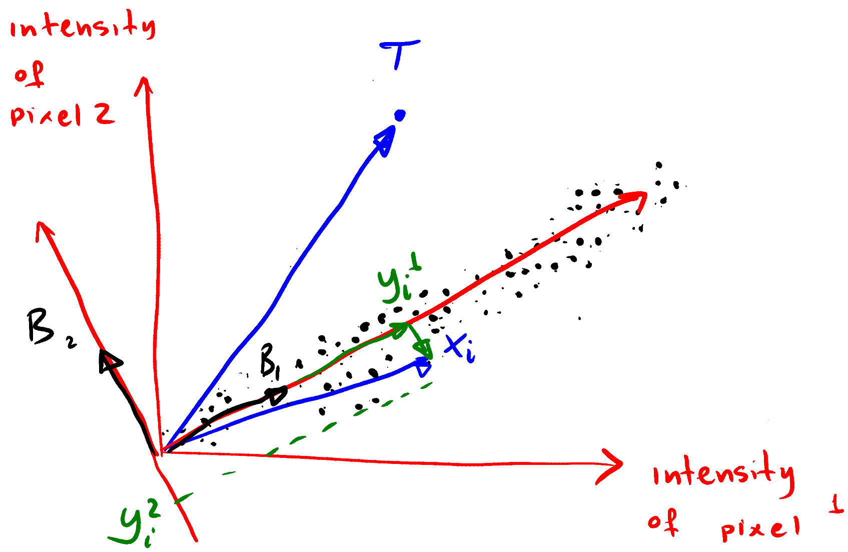
Linear Dimensionality Reduction: Basic Intuition



Now, note that when pixel intensities are correlated, it is possible to express a patch in terms of basis vectors where only a few of the coordinates are significant (not close to zero):

$$x_i = B_1\, y_i^1 + \underbrace{B_2\, y_i^2}_{\text{Close to zero}}$$

Linear Dimensionality Reduction: Basic Intuition



$$x_i = B_1\, y_i^1 + \underbrace{B_2\, y_i^2}_{\text{Close to zero}}$$

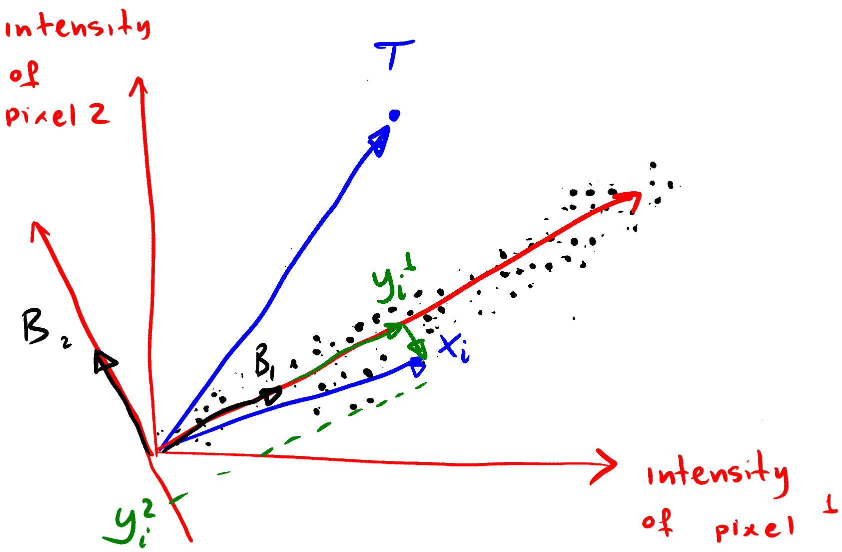
And when this is true, then:

$$x_i = y_i^1 \cdot B_1 + y_i^2 B_2 \simeq y_i^1 B_1$$

$$CC(T, x_i) = T^T \cdot X_i$$

$$= \underset{\substack{\uparrow \\ \text{unconstrained}}}{y_i^1}\,(T^T B_1) + \underset{\substack{\uparrow \\ \text{near } 0}}{y_i^2}\,(T^T B_2) \simeq y_i^1 (T^T B_1)$$

Linear Dimensionality Reduction: Basic Intuition



$$x_i = B_1\, y_i^1 + \underbrace{B_2\, y_i^2}_{\text{Close to zero}}$$

And when this is true, then: $cc(T, x_i) \simeq y^1_i\,(T^T B_1)$

Compared to: $cc(T, x_i) = \left(T^T \begin{bmatrix}1\\0\end{bmatrix}\right) x_i^1 + \left(T^T \begin{bmatrix}0\\1\end{bmatrix}\right) x_i^2$

Note that when pixels intensities are related, the choice of basis vectors can make a big difference in computational complexity.

Dimensionality Reduction by Principal Component Analysis

In summary:

We now know that **carefully chosen basis** vectors can represent image patches of correlated pixels much **more efficiently**

And we also know that in “**natural images**”, pixel intensities inside each patch are **highly correlated**.

We can exploit these two pieces of knowledge to do template matching much more efficiently.

Dimensionality Reduction by Principal Component Analysis

Algorithm:

1) Find the optimal set of basis vectors B_1, B_2, … , B_m. These basis are often called the Principal Components.

2) Compute patch coordinates in that basis

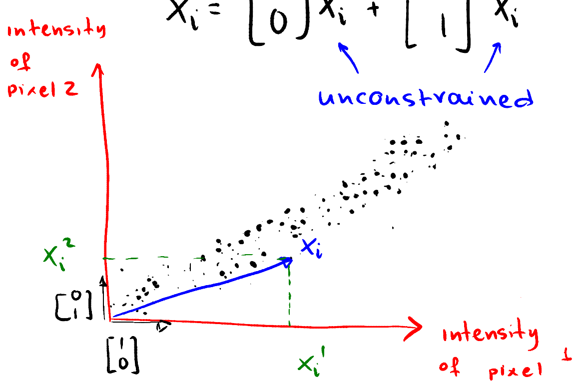
3) Discard the axes with near zero coordinates for all patches.

Changing the Basis: Matrix Notation

Keep in mind that the goal is to go from this

$$X_i = \begin{bmatrix} 1 \\ 0 \end{bmatrix} X_i^1 + \begin{bmatrix} 0 \\ 1 \end{bmatrix} X_i^2$$

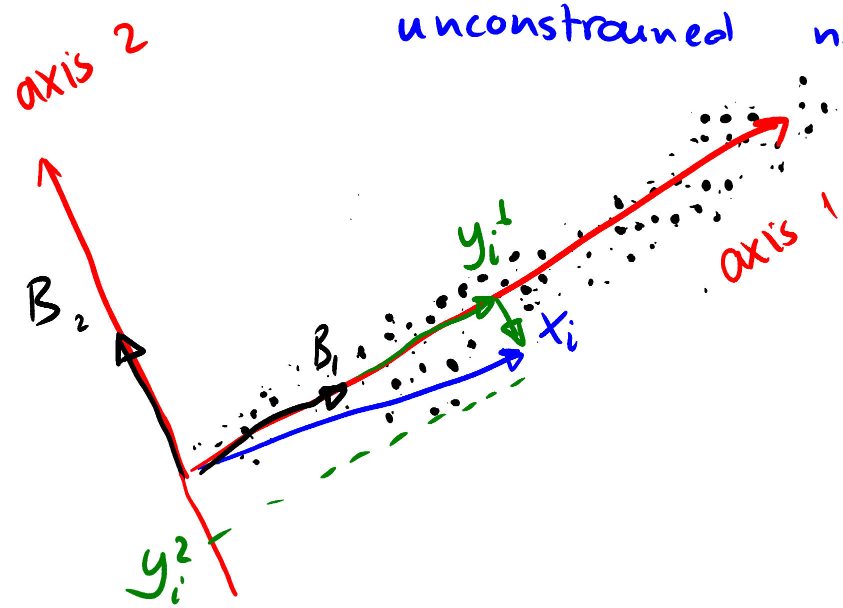
unconstrained



to this

$$X_i = B_1 \cdot y_i^1 + B_2 \cdot y_i^2$$

unconstrained near 0

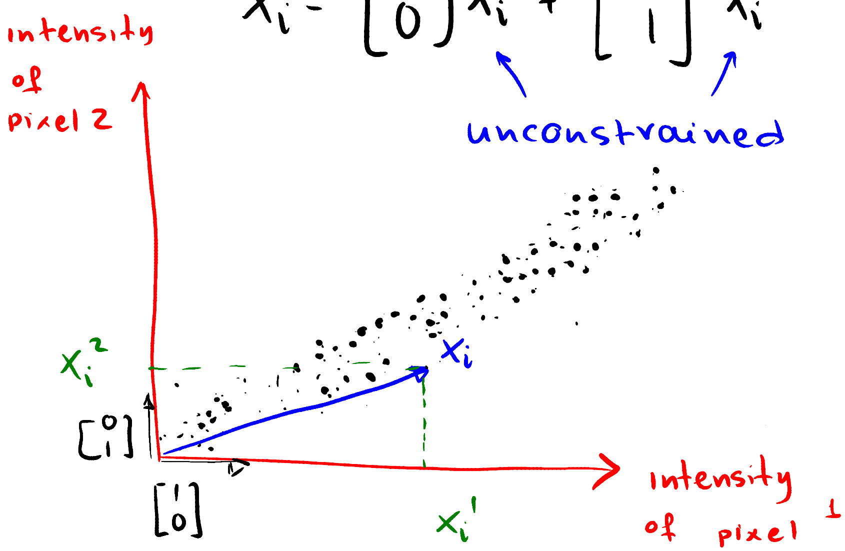


Changing the Basis: Matrix Notation

In the original case, the basis matrix is the identity matrix.

$$X_i = \begin{bmatrix} 1 \\ 0 \end{bmatrix} x_i^1 + \begin{bmatrix} 0 \\ 1 \end{bmatrix} x_i^2$$

unconstrained



In matrix notation (i^{th} patch):

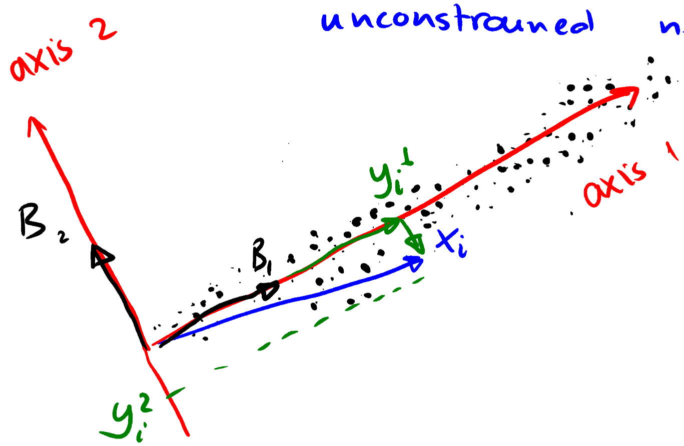
$$X_i = \underbrace{\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}}_{\text{basis matrix}} \underbrace{\begin{bmatrix} x_i^1 \\ x_i^2 \end{bmatrix}}_{\text{coordinate vector}}$$

Changing the Basis: Matrix Notation

In the alternative representation, the basis B_i are the transformations that take new coordinates y_i, to reconstruct the original data x_i.

$$x_i = B_1 \cdot y_i^1 + B_2 \cdot y_i^2$$

unconstrained   near 0



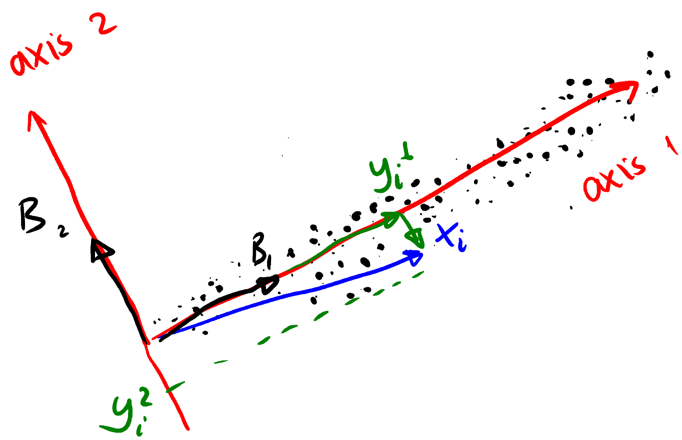
$$x_i = \begin{bmatrix} B_1 & B_2 \end{bmatrix} \begin{bmatrix} y_i^1 \\ y_i^2 \end{bmatrix}$$

All N patches

$$\begin{bmatrix} x_1 & x_2 & \cdots & x_N \end{bmatrix} = \begin{bmatrix} B_1 & B_2 \end{bmatrix} \begin{bmatrix} y_1^1 & y_2^1 & \cdots & y_N^1 \\ y_1^2 & y_2^2 & \cdots & y_N^2 \end{bmatrix}$$

Changing the Basis: Matrix Notation

The same is true for M-Dimensional patches: The reconstructed data X_i is the basis B times the new representations (Y_i).



$$\begin{bmatrix} X_1 & X_2 & \cdots & X_N \end{bmatrix} = \begin{bmatrix} B_1 & B_2 & \cdots & B_M \end{bmatrix} \cdot \begin{bmatrix} y_1^1 & y_2^1 & \cdots & y_N^1 \\ y_1^2 & y_2^2 & \cdots & y_N^2 \\ \vdots & \vdots & & \vdots \\ y_1^M & y_2^M & \cdots & y_N^M \end{bmatrix}$$

Changing the Basis: Matrix Notation

The same is true for M-Dimensional patches: The reconstructed data X_i is the basis B times the new representations (Y_i).

$$\begin{bmatrix} X_1 & X_2 & \cdots & X_N \end{bmatrix} = \begin{bmatrix} B_1 & B_2 & \cdots & B_M \end{bmatrix} \cdot \begin{bmatrix} y_1^1 & y_2^1 & & y_N^1 \\ y_1^d & y_2^d & & y_N^d \\ \hline y_1^{d+1} & y_2^{d+1} & & y_N^{d+1} \\ \vdots & \vdots & & \vdots \\ y_1^M & y_2^M & & y_N^M \end{bmatrix}$$

But crucially, many of these coefficients will be close to zero and can be ignored.

Changing the Basis: Matrix Notation

Eliminating these coefficients leaves us with a d-Dimensional approximation:

$$\begin{bmatrix} X_1 & X_2 & \cdots & X_N \end{bmatrix} = \begin{bmatrix} B_1 & B_2 & \cdots & B_d \end{bmatrix} \cdot \begin{bmatrix} y_1^1 & y_2^1 & & y_N^1 \\ y_1^d & y_2^d & & y_N^d \end{bmatrix}$$

Note only *d* basis are used now, not M
(and d << M)

Changing the Basis: Matrix Notation

Finding these (not so) magical Basis in 4 steps:
Input: matrix X, and desired dimension *d*
Output: Basis vectors $B_1, B_2, \ldots B_d$

1) Compute the average patch $\boxed{\bar{X} = \frac{1}{N} \Sigma x_i}$

2) Subtract the average patch from each X_i $\boxed{Z_i = X_i - \bar{X}}$

3) Define the matrix $Z = [z_1, z_2, \ldots, z_n]$

4) $[B_1, B_2, \ldots B_d]$ = the eigenvectors of the matrix ZZ^T with the *d* largest eigenvalues.

Notes on the dimensions of these matrices

The matrix Z is defined as the concatenation of n column-vectors of size M, as in:

$$Z = [z_1, z_2, \ldots, z_n]$$

The size of Z is therefore [M x n].

The dimension of ZZ^T is [M x M] (noting that its size is independent of the number of data points (n)). So, ZZ^T is square and of size equal to the dimension of one point.

The dimensionality reduction basis **B** are the first *d* eigenvectors **B** = [b_1, b_2, … B_d] of the matrix ZZ^T. The size of **B** is [M x d].