

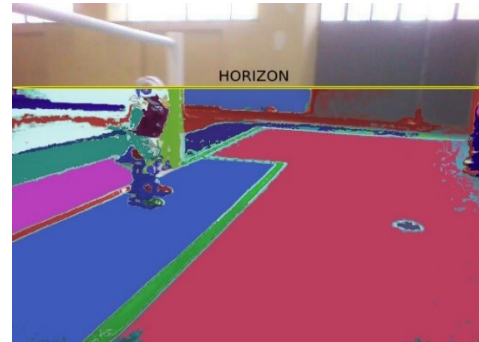
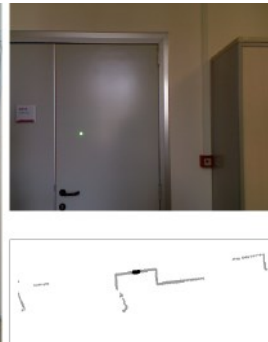
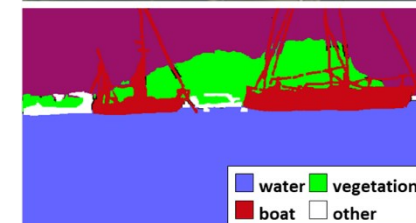
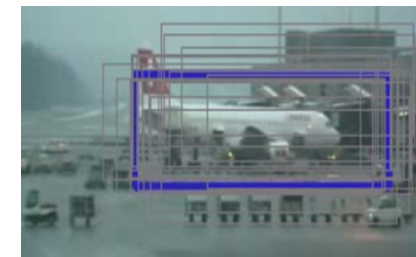


**UNIVERSITÀ DEGLI STUDI
DELLA BASILICATA**

Corso di Sistemi Informativi
A.A. 2018/19

Feature Matching

Docente
Domenico Daniele Bloisi

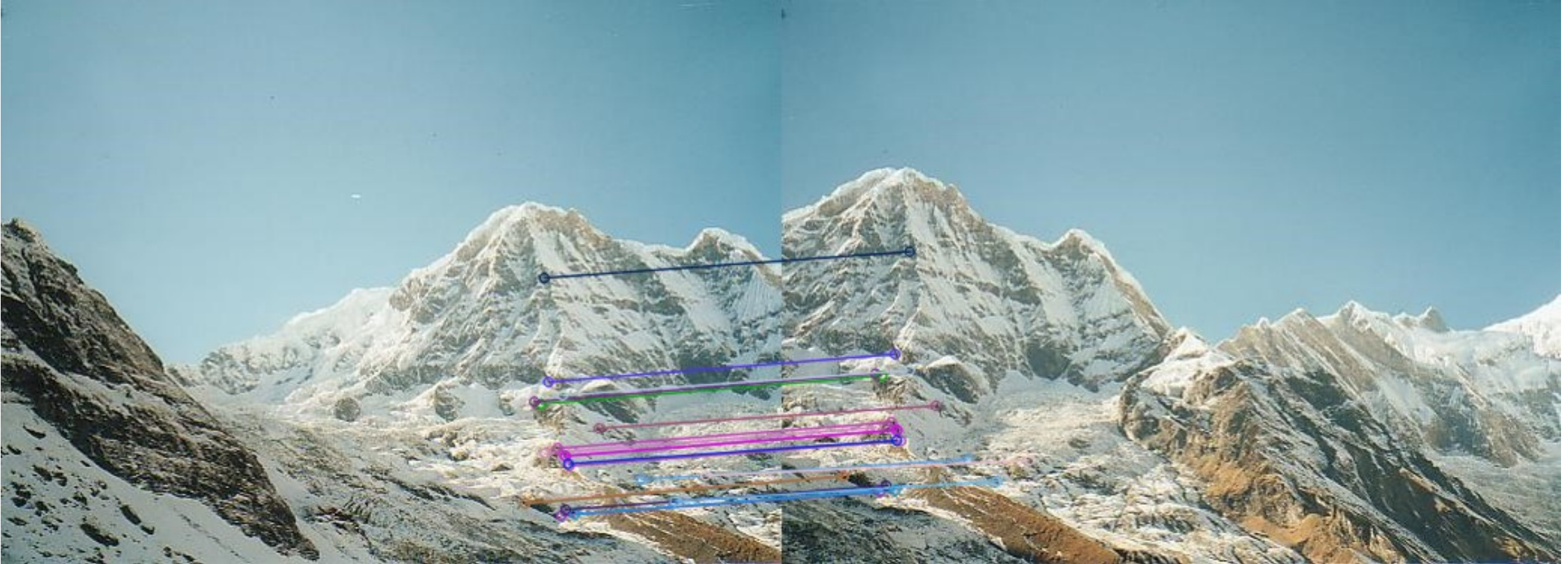


Aprile 2019

Riferimenti

- Queste slide sono adattate da
Noah Snavely - CS5670: Computer Vision
["Lecture 5: Feature descriptors and matching"](#)
["Lecture 7: Transformations and warping"](#)
- I contenuti fanno riferimento ai capitoli 3 e 4 del libro
"Computer Vision: Algorithms and Applications"
di Richard Szeliski, disponibile al seguente indirizzo
<http://szeliski.org/Book/>

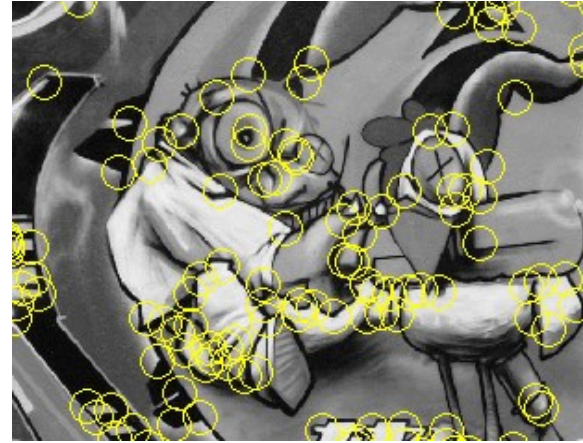
Problem: Feature matching



Recap

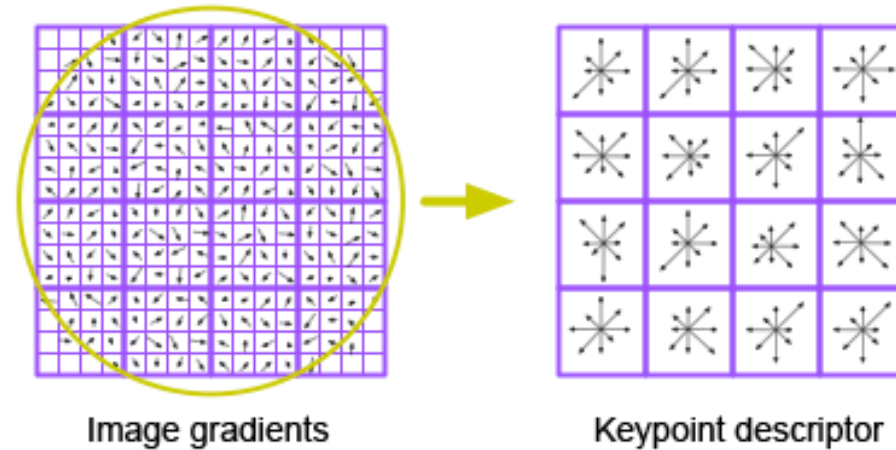
Keypoint detection: repeatable and distinctive

- Corners, blobs, stable regions
- Harris

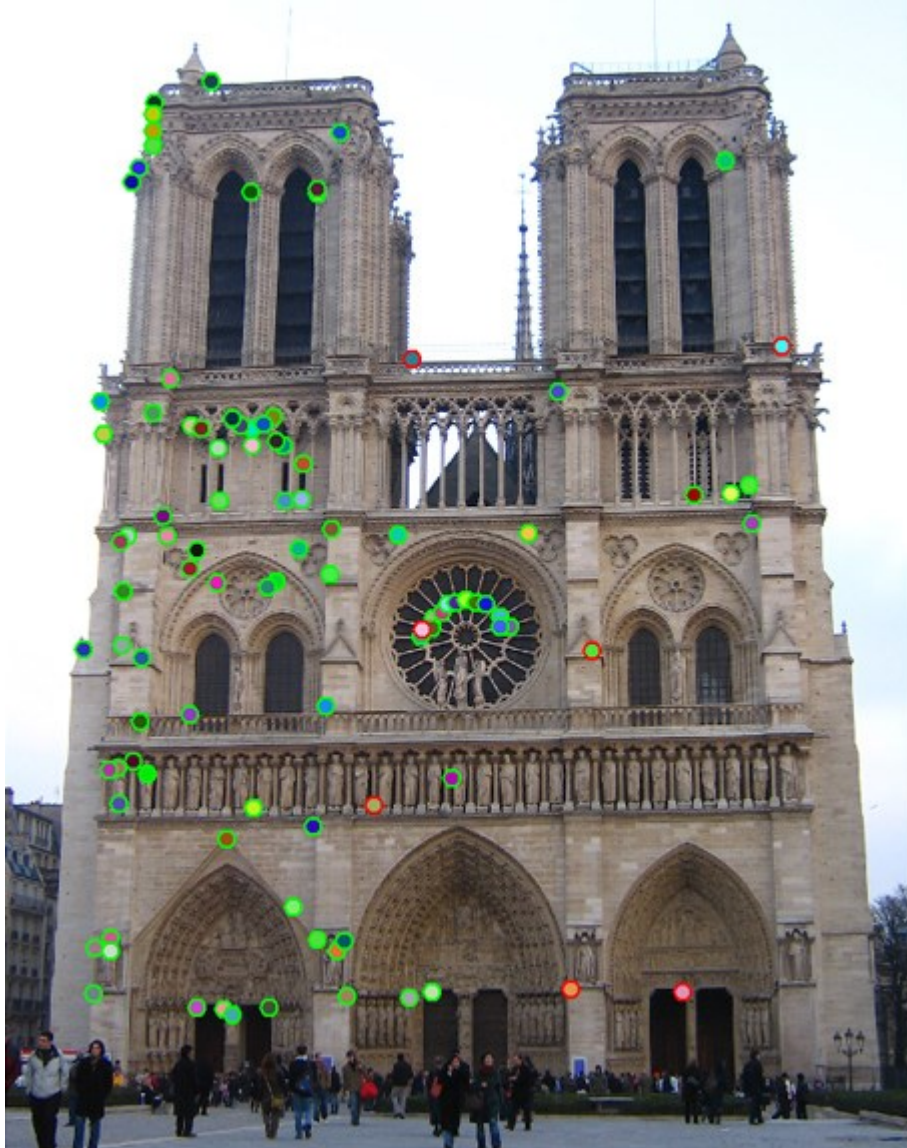


Descriptors: robust and selective

- spatial histograms of orientation
- SIFT and variants are typically good for stitching and recognition



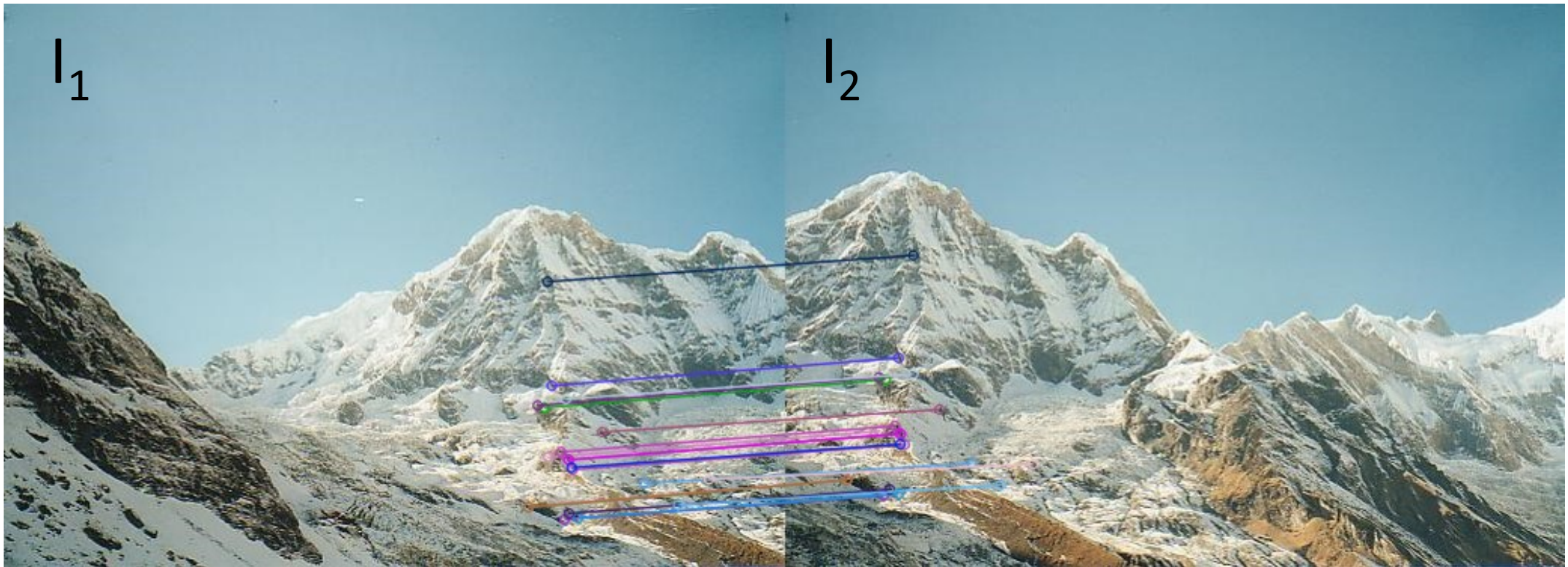
Which features match?



Features matching

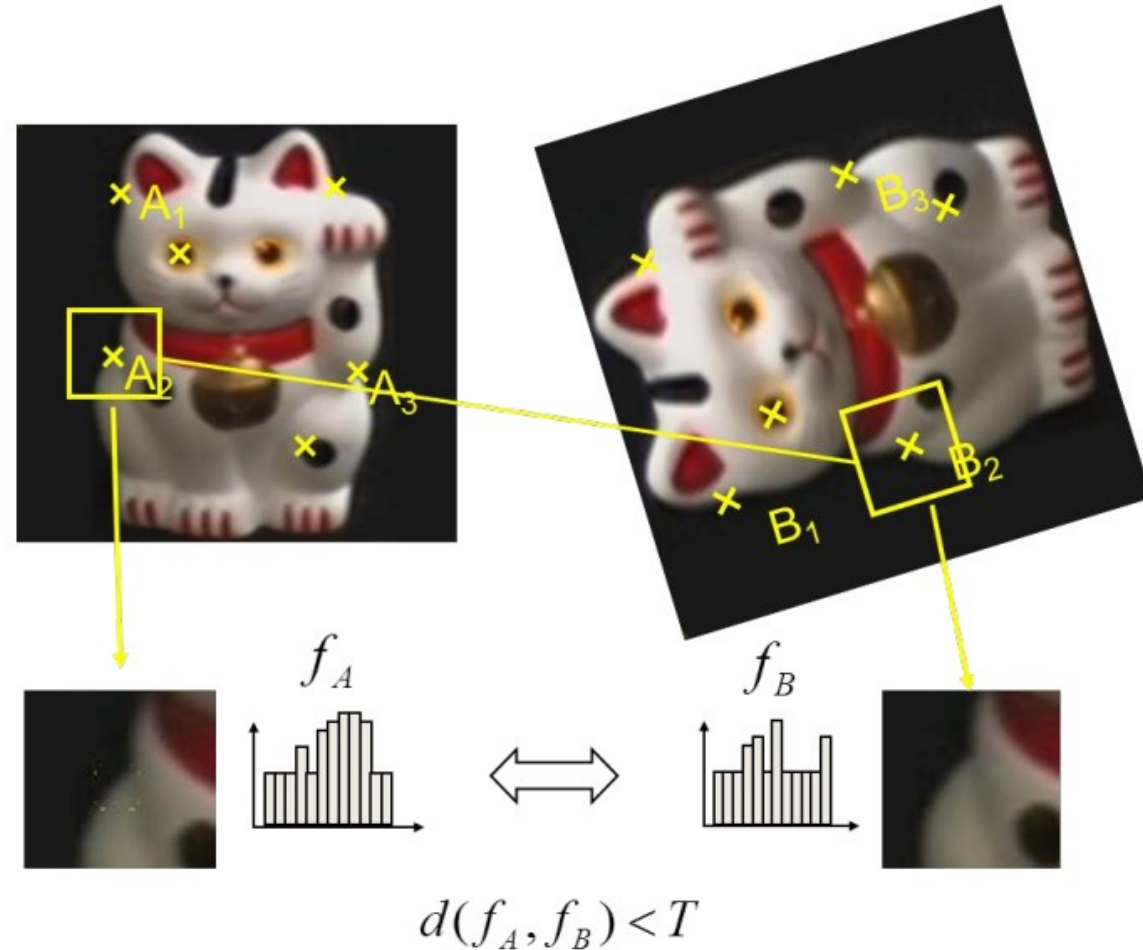
Given a feature in I_1 , how to find the best match in I_2 ?

1. Define distance function that compares two descriptors
2. Test all the features in I_2 , find the one with min distance



Overview of point feature matching

1. Detect a set of distinct feature points
2. Define a patch around each point
3. Extract and normalize the patch
4. Compute a local descriptor
5. Match local descriptors



Distance between descriptors

- L_1 distance (SAD):

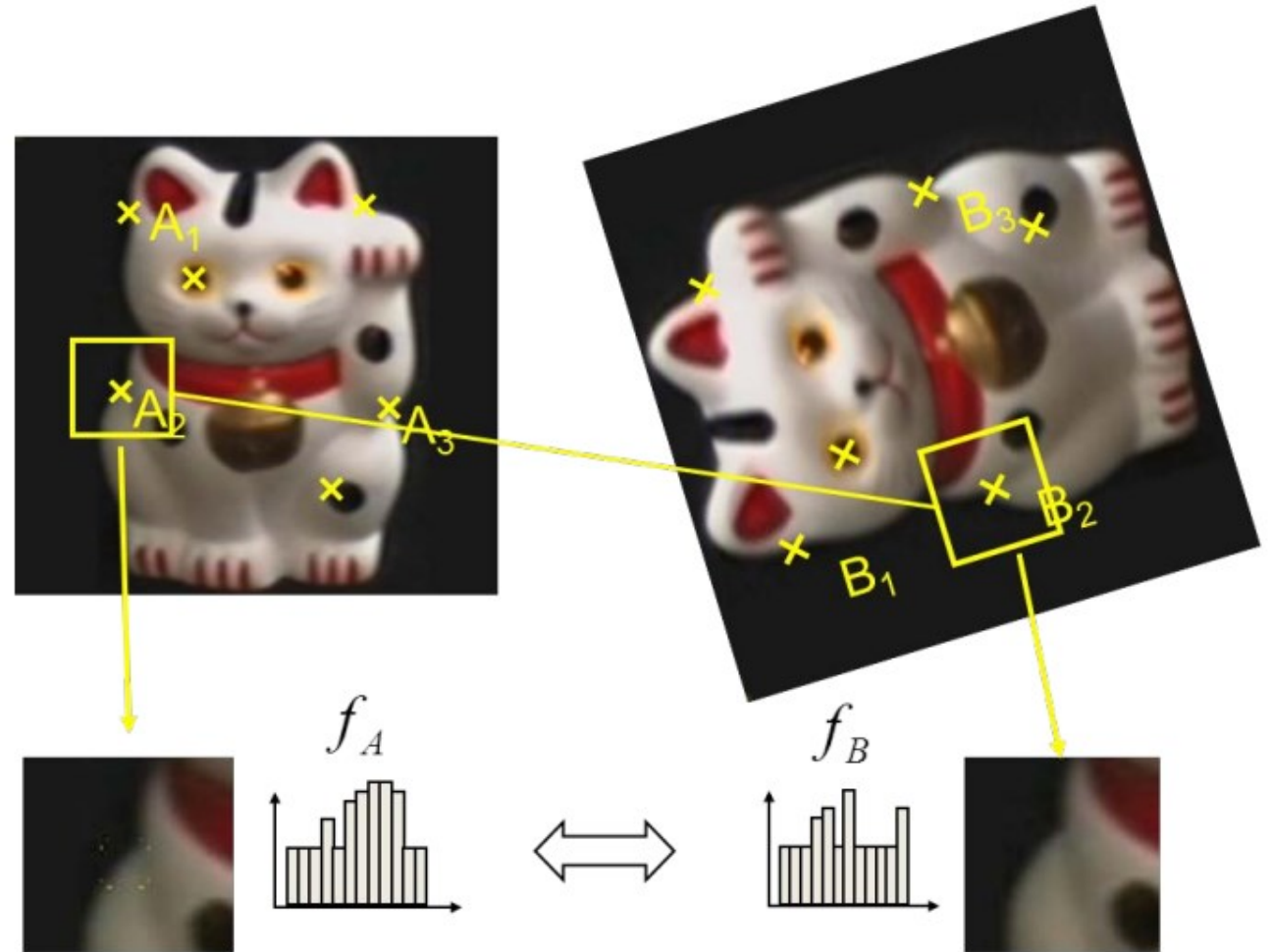
$$d(f_a, f_b) = \sum |f_a - f_b|$$

- L_2 distance (SSD):

$$d(f_a, f_b) = \sum (f_a - f_b)^2$$

- Hamming distance:

$$d(f_a, f_b) = \sum \text{XOR}(f_a, f_b)$$

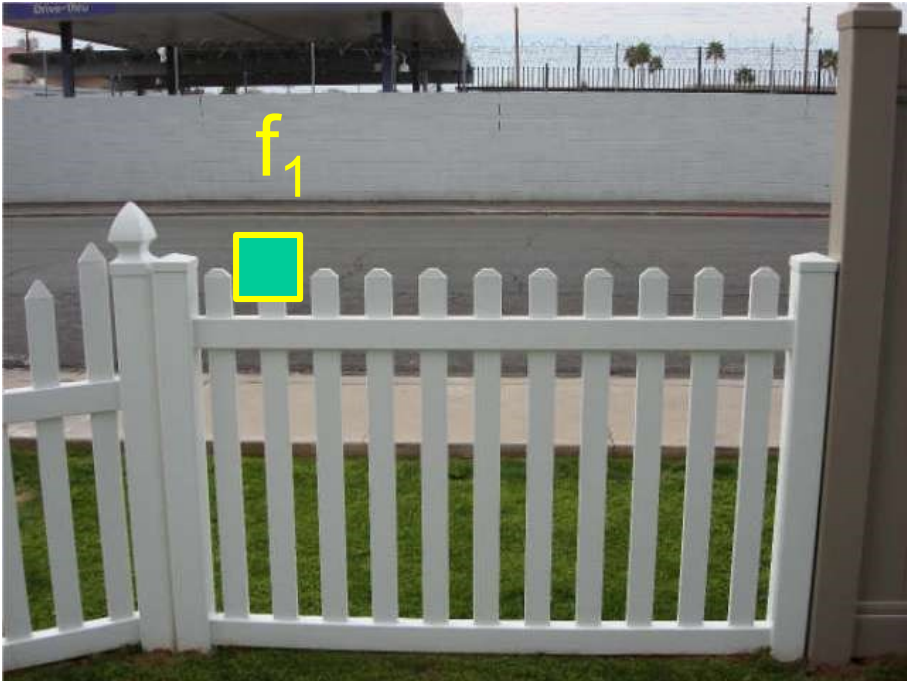


$$d(f_A, f_B) < T$$

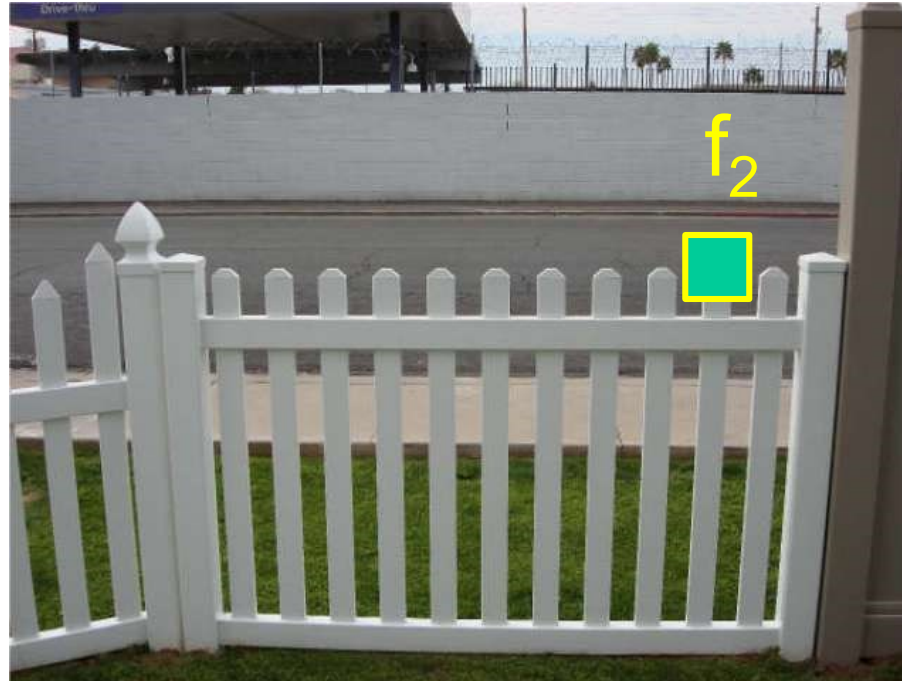
Features distance: SSD

How to define the difference between two features f_1, f_2 ?

- Simple approach: L_2 distance, $||f_1 - f_2||$
i.e., sum of square differences (SSD) between entries of the two descriptors
- can give small distances for ambiguous (incorrect) matches
i.e., does not provide a way to discard ambiguous (bad) matches



I_1

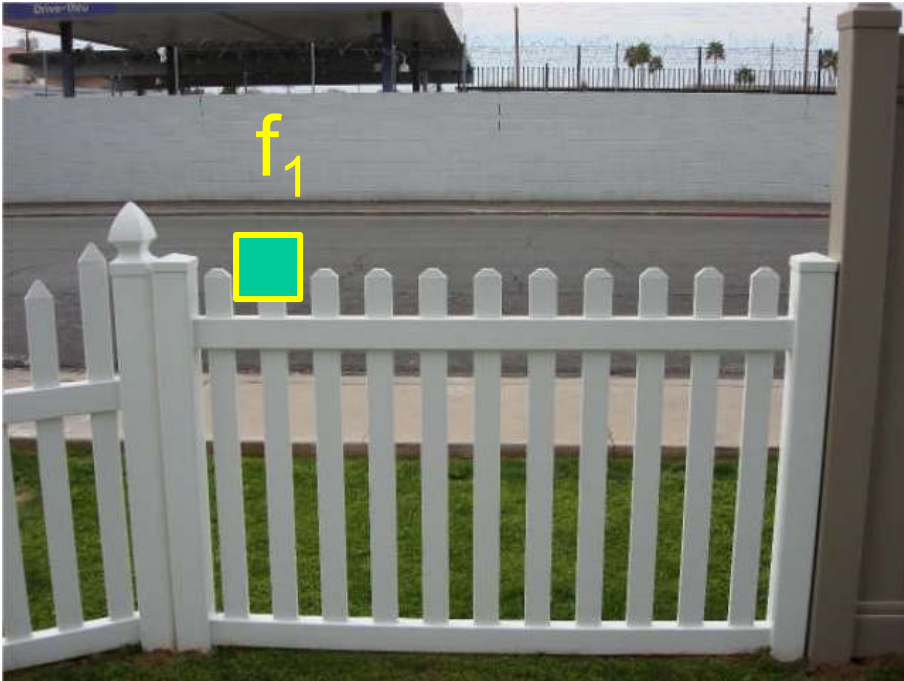


I_2

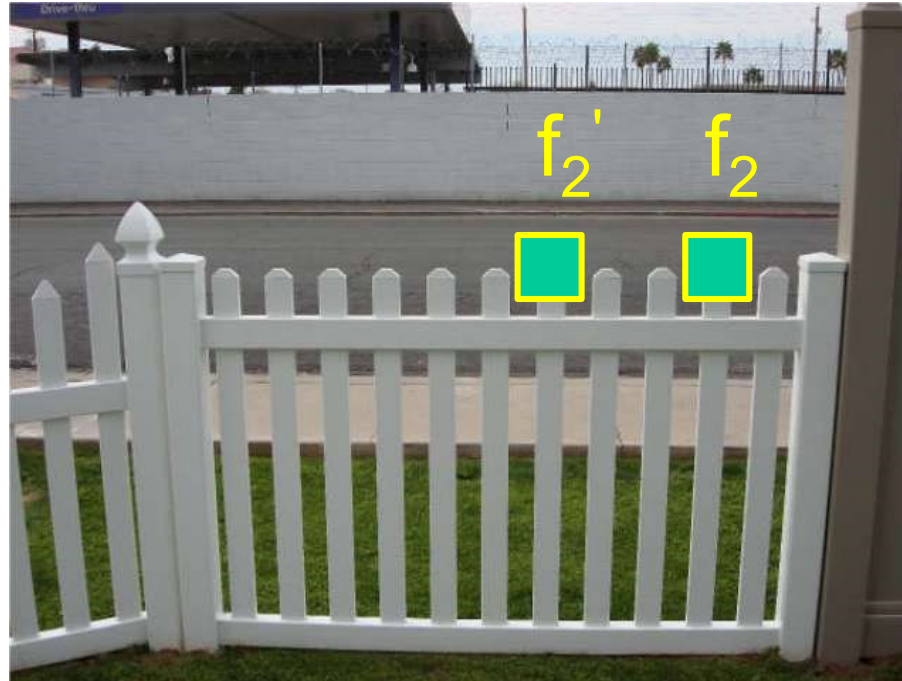
Features distance: Ratio of SSDs

How to define the difference between two features f_1, f_2 ?

- Better approach: ratio distance = $\text{SSD}(f_1, f_2) / \text{SSD}(f_1, f_2')$
 - f_2 is best SSD match to f_1 in I_2
 - f_2' is 2nd best SSD match to f_1 in I_2
 - An ambiguous/bad match will have ratio close to 1
 - Look for unique matches which have low ratio

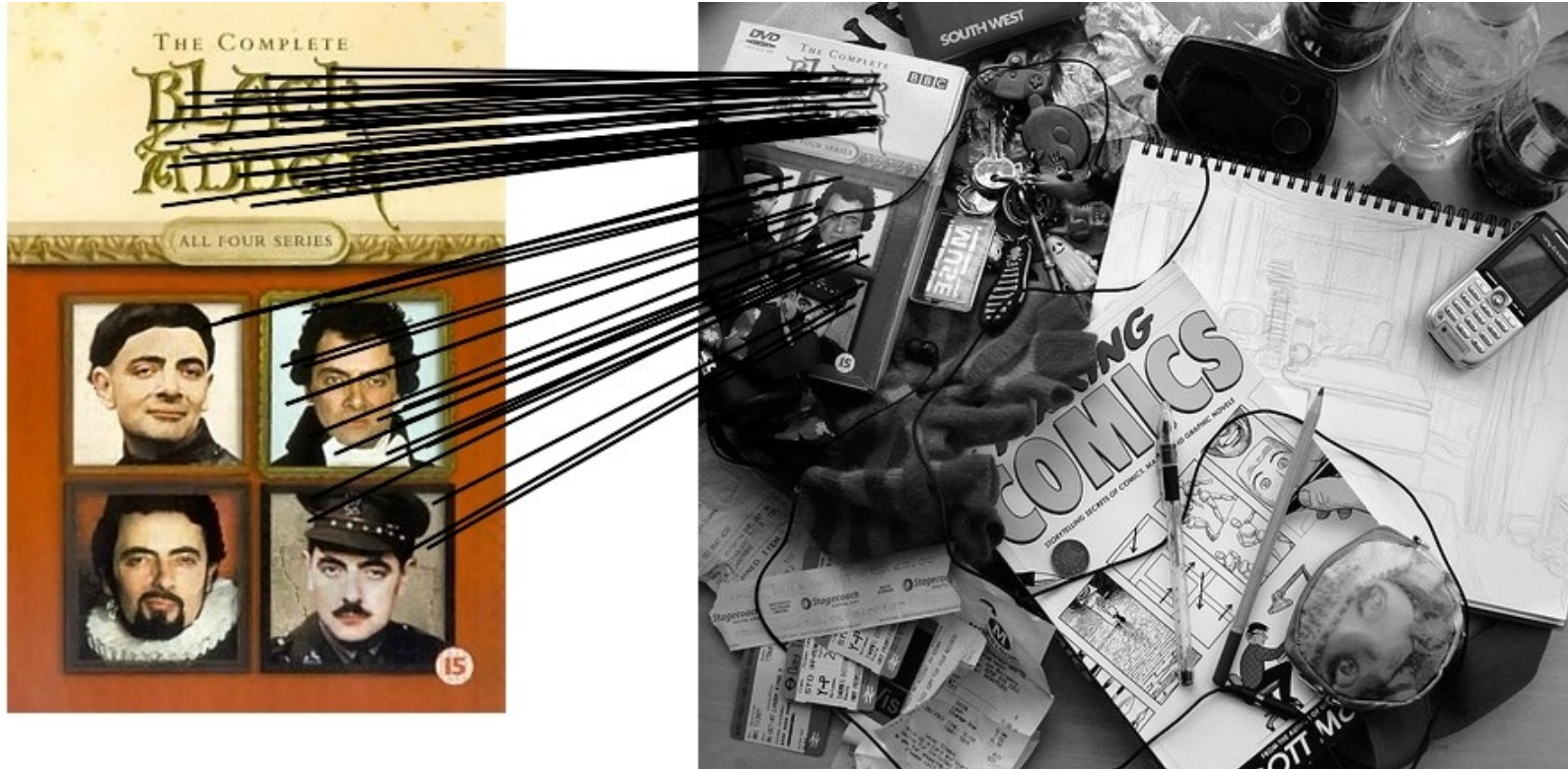


I_1



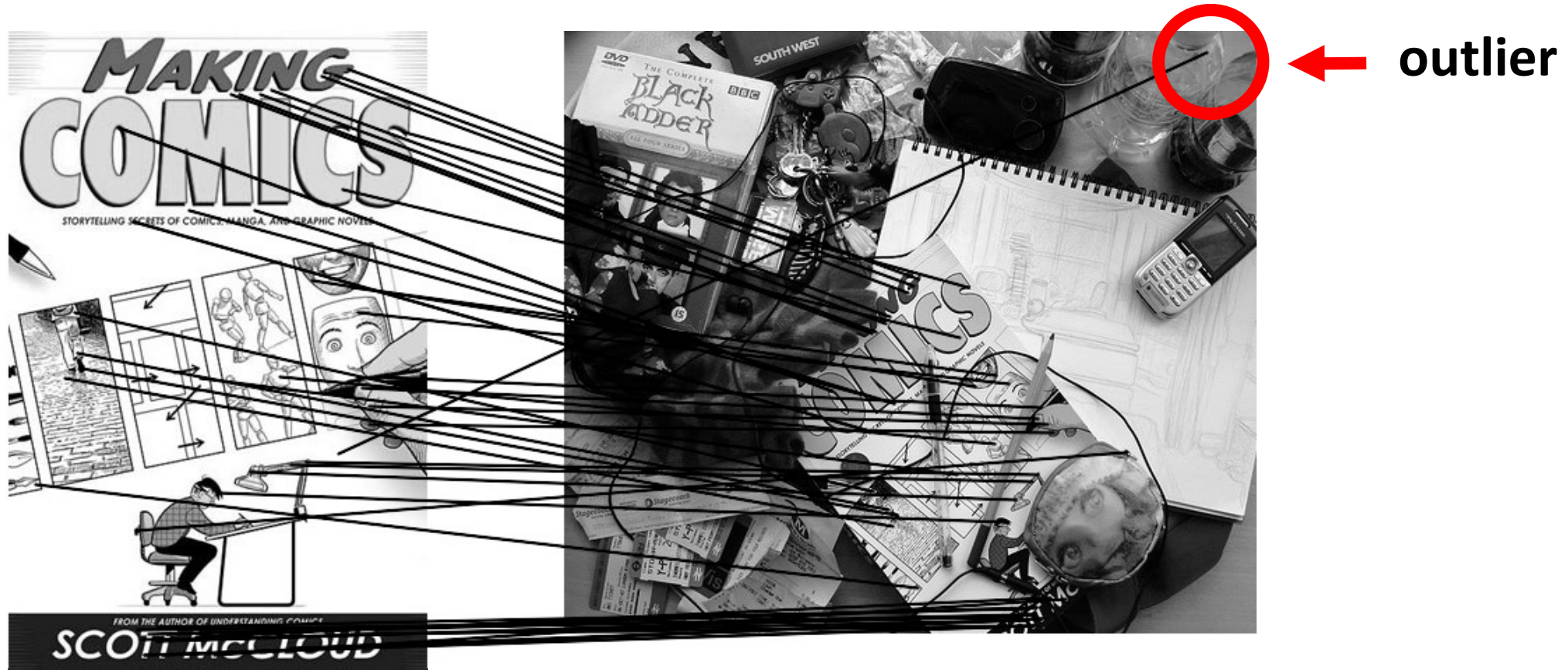
I_2

Feature matching example



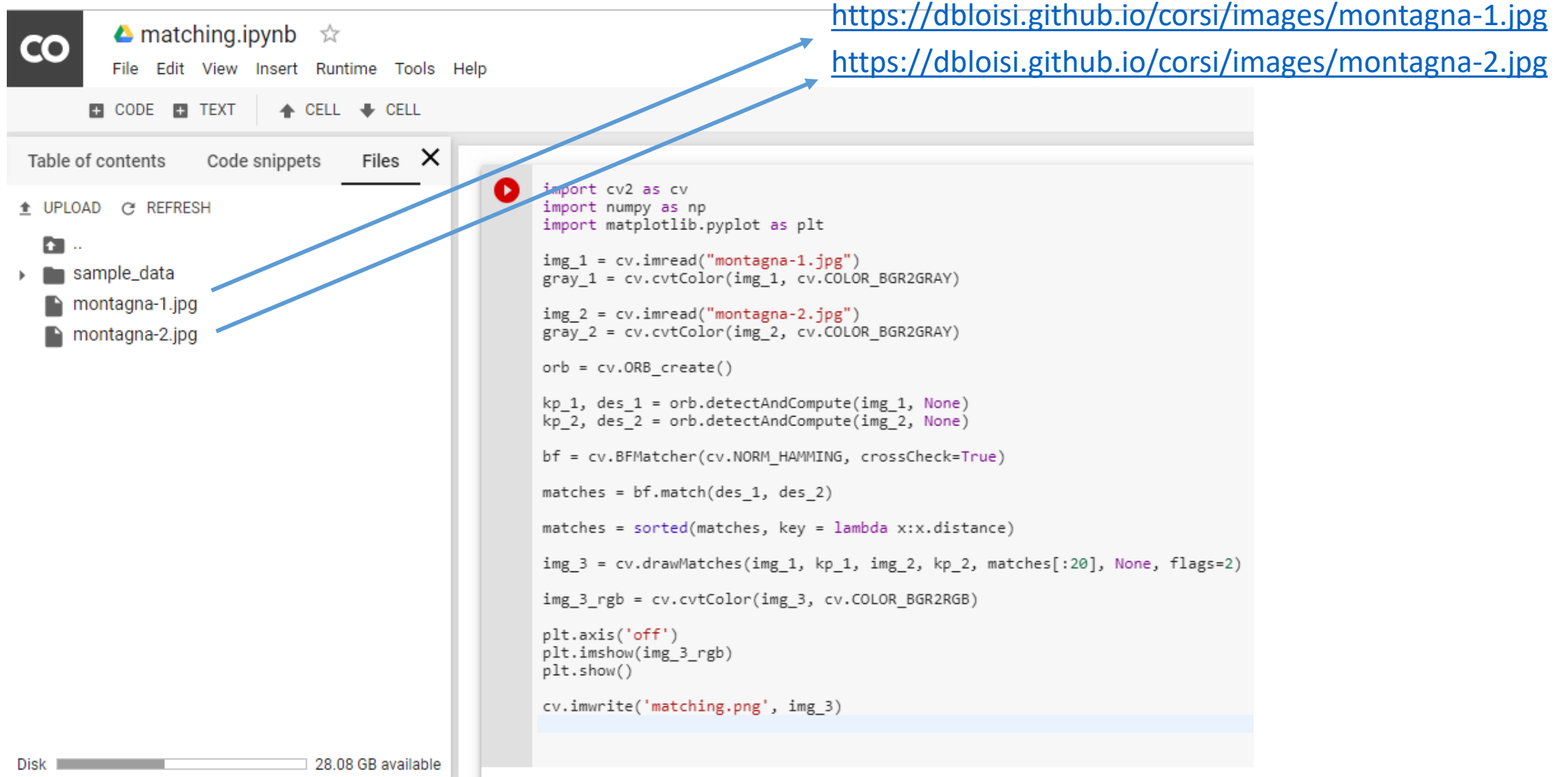
58 matches (thresholded by ratio score)

Feature matching example



51 matches (thresholded by ratio score)

Example in Colab



The screenshot shows a Google Colab notebook interface. The notebook is titled "matching.ipynb" and has a star icon. The top menu bar includes "File", "Edit", "View", "Insert", "Runtime", "Tools", and "Help". Below the menu bar, there are tabs for "CODE", "TEXT", "CELL", and "CELL". The left sidebar shows the "Table of contents" and "Files" sections. Under "Files", there are options to "UPLOAD" and "REFRESH". The file list includes a folder named "sample_data" and two files: "montagna-1.jpg" and "montagna-2.jpg". Two blue arrows point from these files to the code cell. The code cell contains the following Python code:

```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt

img_1 = cv.imread("montagna-1.jpg")
gray_1 = cv.cvtColor(img_1, cv.COLOR_BGR2GRAY)

img_2 = cv.imread("montagna-2.jpg")
gray_2 = cv.cvtColor(img_2, cv.COLOR_BGR2GRAY)

orb = cv.ORB_create()

kp_1, des_1 = orb.detectAndCompute(img_1, None)
kp_2, des_2 = orb.detectAndCompute(img_2, None)

bf = cv.BFMatcher(cv.NORM_HAMMING, crossCheck=True)

matches = bf.match(des_1, des_2)

matches = sorted(matches, key = lambda x:x.distance)

img_3 = cv.drawMatches(img_1, kp_1, img_2, kp_2, matches[:20], None, flags=2)

img_3_rgb = cv.cvtColor(img_3, cv.COLOR_BGR2RGB)

plt.axis('off')
plt.imshow(img_3_rgb)
plt.show()

cv.imwrite('matching.png', img_3)
```

At the bottom of the notebook, there is a "Disk" section showing a progress bar and the text "28.08 GB available".

Two blue arrows point from the file list to the code cell, indicating the source of the image files used in the code. The arrows originate from the "montagna-1.jpg" and "montagna-2.jpg" files and point to the corresponding file names in the code.

<https://dbloisi.github.io/corsi/images/montagna-1.jpg>

<https://dbloisi.github.io/corsi/images/montagna-2.jpg>

Example in Colab: ORB detection



```
import cv2 as cv
import numpy as np
import matplotlib.pyplot as plt

img_1 = cv.imread("montagna-1.jpg")
gray_1 = cv.cvtColor(img_1, cv.COLOR_BGR2GRAY)

img_2 = cv.imread("montagna-2.jpg")
gray_2 = cv.cvtColor(img_2, cv.COLOR_BGR2GRAY)

orb = cv.ORB_create()

kp_1, des_1 = orb.detectAndCompute(img_1, None)
kp_2, des_2 = orb.detectAndCompute(img_2, None)
```


Example in Colab: brute force matching

Brute-Force matcher is simple. It takes the descriptor of one feature in first set and is matched with all other features in second set using some distance calculation. And the closest one is returned.

```
bf = cv.BFMatcher(cv.NORM_HAMMING, crossCheck=True)
matches = bf.match(des_1, des_2)
matches = sorted(matches, key = lambda x:x.distance)
```

Cross check test

- Choose matches (f_a, f_b) so that
 - f_b is the best match for f_a in I_b
 - And f_a is the best match for f_b in I_a
- Alternative to ratio test



Cross check test

- Choose matches (f_a, f_b) so that
 - f_b is the best match for f_a in I_b
 - And f_a is the best match for f_b in I_a
- Alternative to ratio test



Example in Colab: sorting

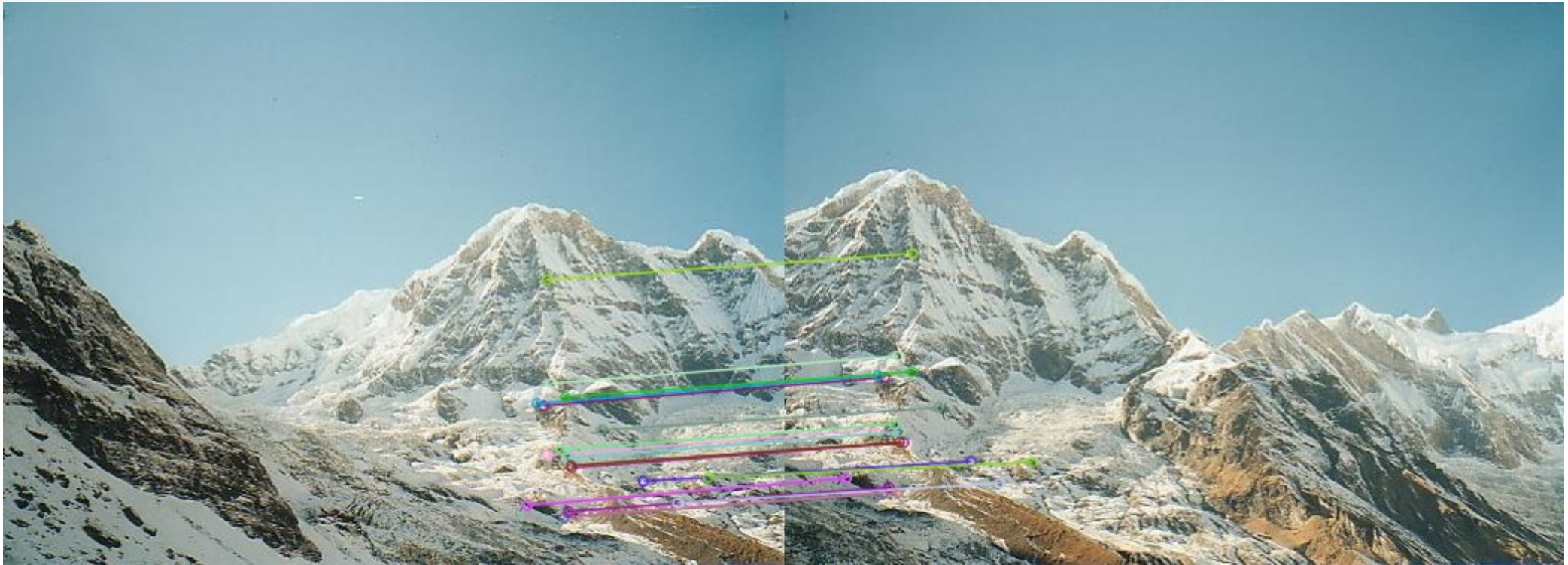
```
bf = cv.BFMatcher(cv.NORM_HAMMING, crossCheck=True)
matches = bf.match(des_1, des_2)
matches = sorted(matches, key = lambda x:x.distance)
```

Matches are sorted in ascending order of their distances so that best matches (with low distance) come to front.

Example in Colab: result

```
img_3 = cv.drawMatches(img_1, kp_1, img_2, kp_2, matches[:20], None, flags=2)
img_3_rgb = cv.cvtColor(img_3, cv.COLOR_BGR2RGB)
plt.axis('off')
plt.imshow(img_3_rgb)
plt.show()

cv.imwrite('matching.png', img_3)
```



Example in Colab: ratio test



```
orb = cv.ORB_create()

kp_1, des_1 = orb.detectAndCompute(img_1, None)
kp_2, des_2 = orb.detectAndCompute(img_2, None)

bf = cv.BFMatcher()

matches = bf.knnMatch(des_1, des_2, k=2)

# Apply ratio test
good = []
for m,n in matches:
    if m.distance < 0.7*n.distance:
        good.append([m])

img_3 = cv.drawMatchesKnn(img_1, kp_1, img_2, kp_2, good[:20], None, flags=2)

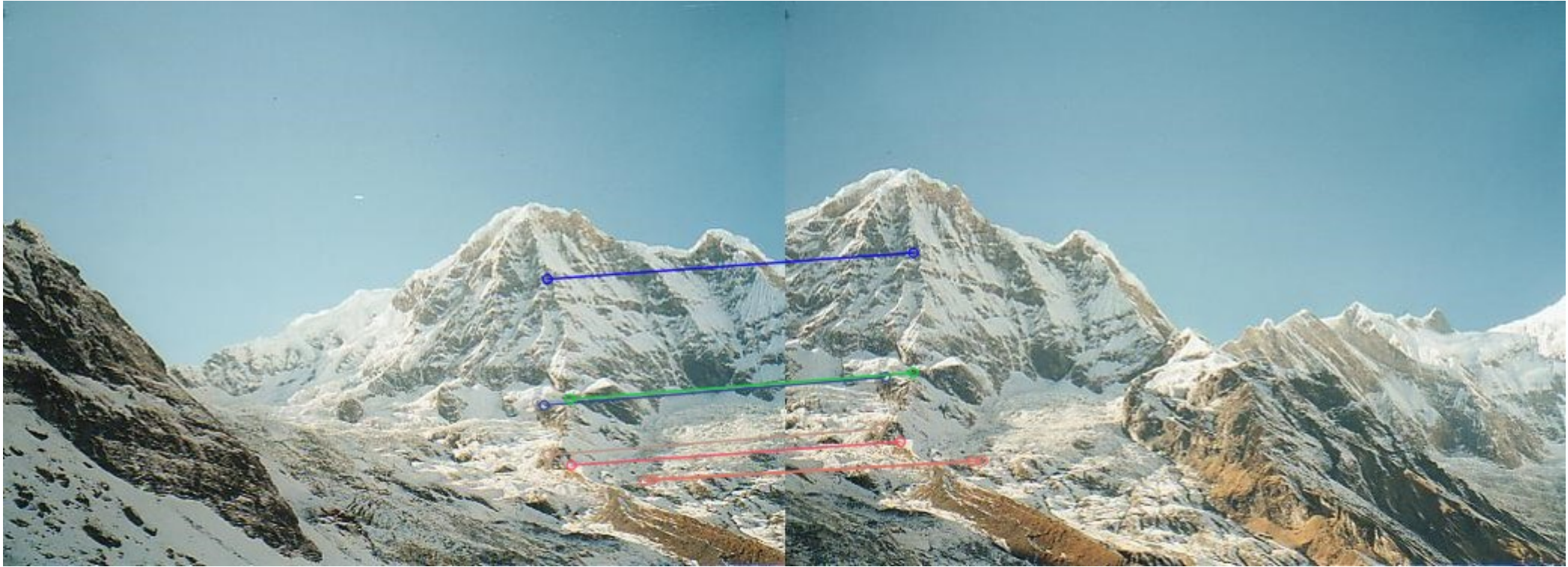
img_3_rgb = cv.cvtColor(img_3, cv.COLOR_BGR2RGB)

plt.axis('off')
plt.imshow(img_3_rgb)
plt.show()

cv.imwrite('matching.png', img_3)
```

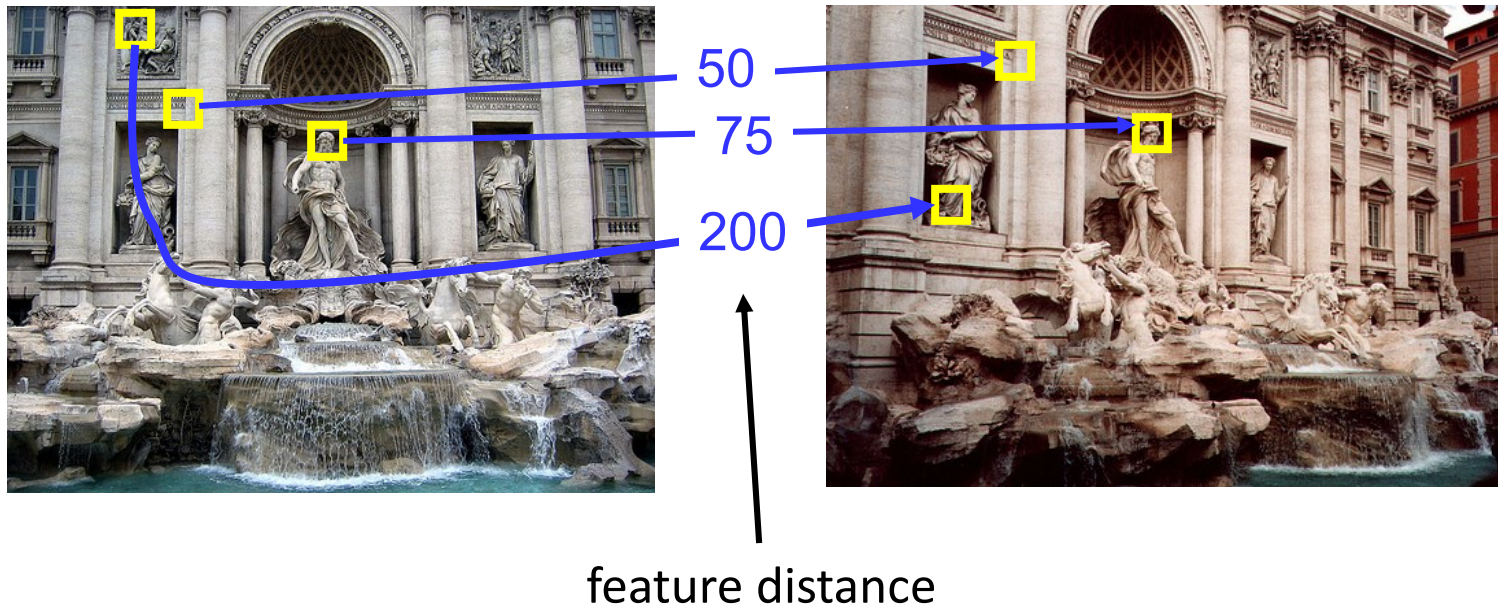
BFMatcher.knnMatch() to get k best matches.
In this example, we will take k=2 so that we
can apply ratio test

Example in Colab: ratio test results



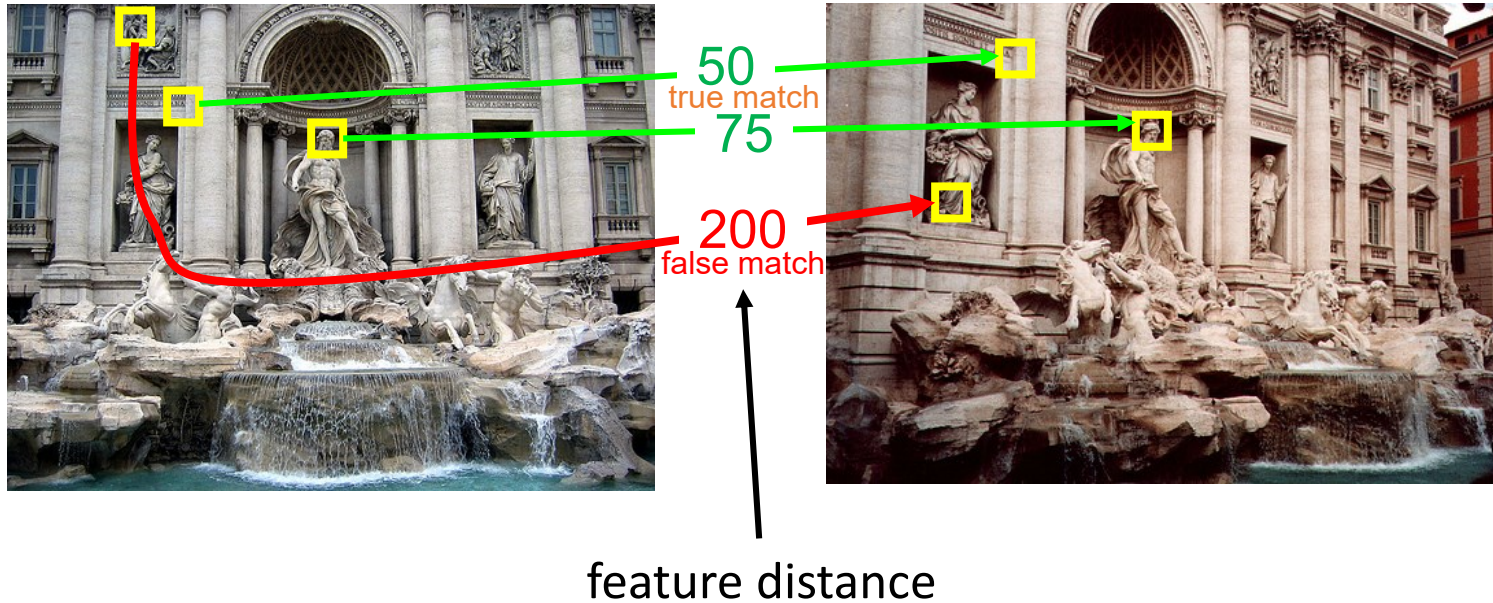
Evaluating the results

How can we measure the performance of a feature matcher?



True/false positives

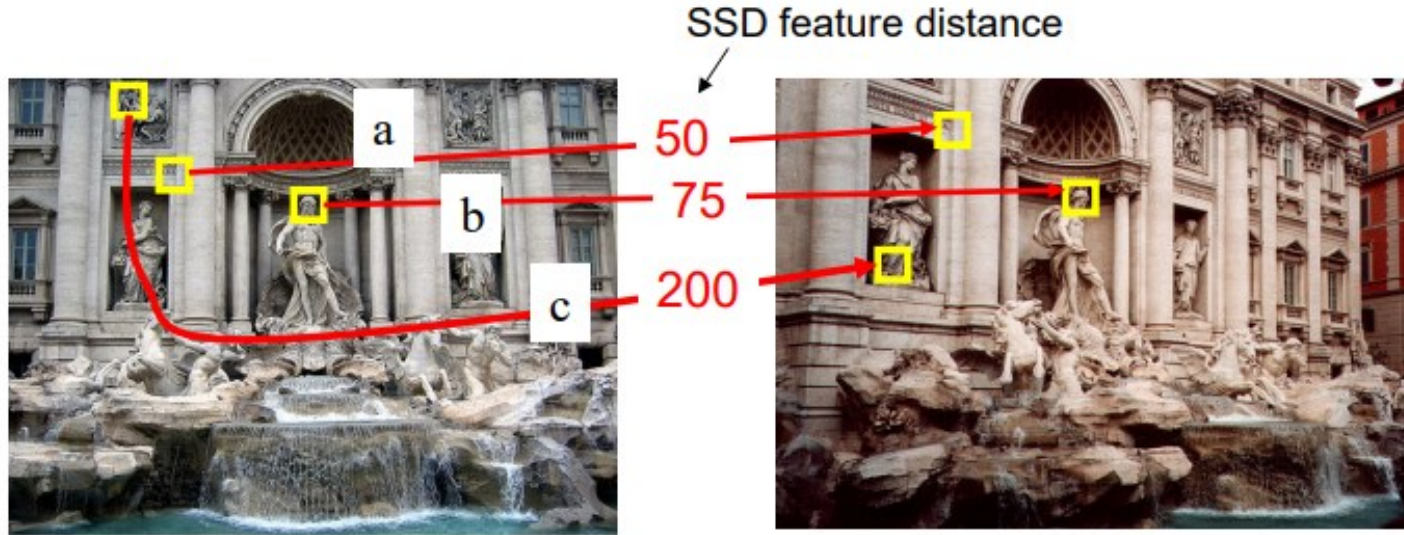
How can we measure the performance of a feature matcher?



The distance threshold affects performance

- **True positives** = # of detected matches that are correct
 - Suppose we want to maximize these—how to choose threshold?
- **False positives** = # of detected matches that are incorrect
 - Suppose we want to minimize these—how to choose threshold?

Effect of threshold T



Decision rule: Accept match if $SSD < T$

Example: **Large T**

$T = 250 \Rightarrow$ a, b, c are all accepted as matches

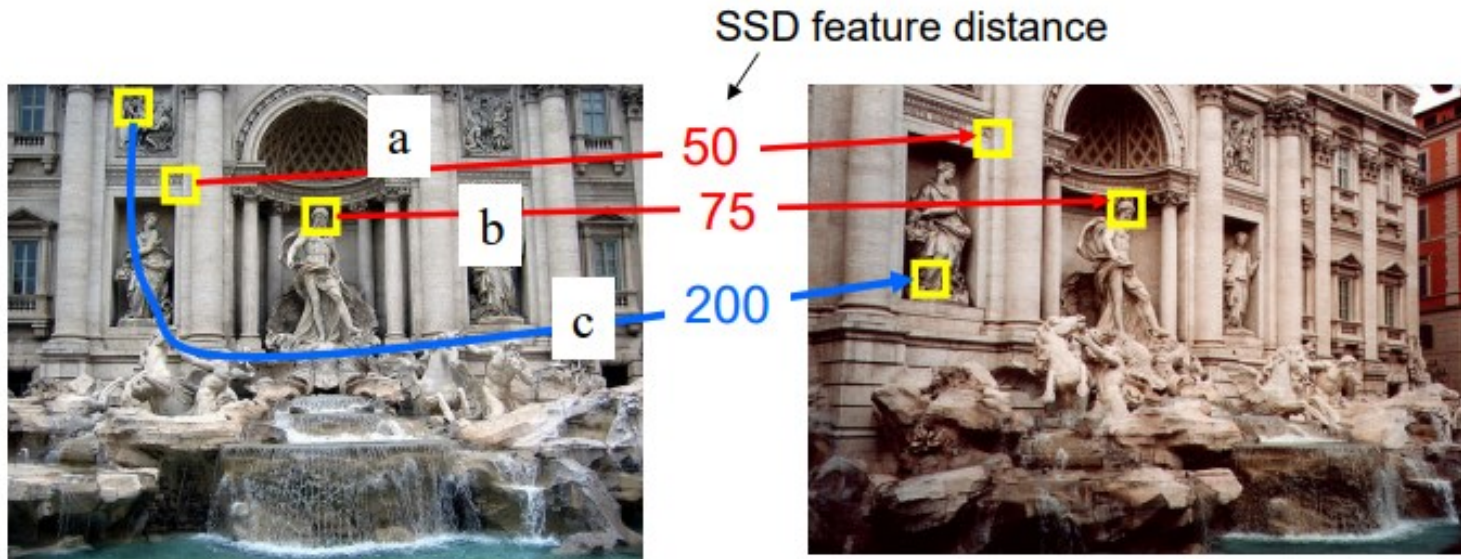
a and b are true matches (“**true positives**”)

- they are actually matches

c is a false match (“**false positive**”)

- actually not a match

Effect of threshold T



Decision rule: Accept match if $SSD < T$

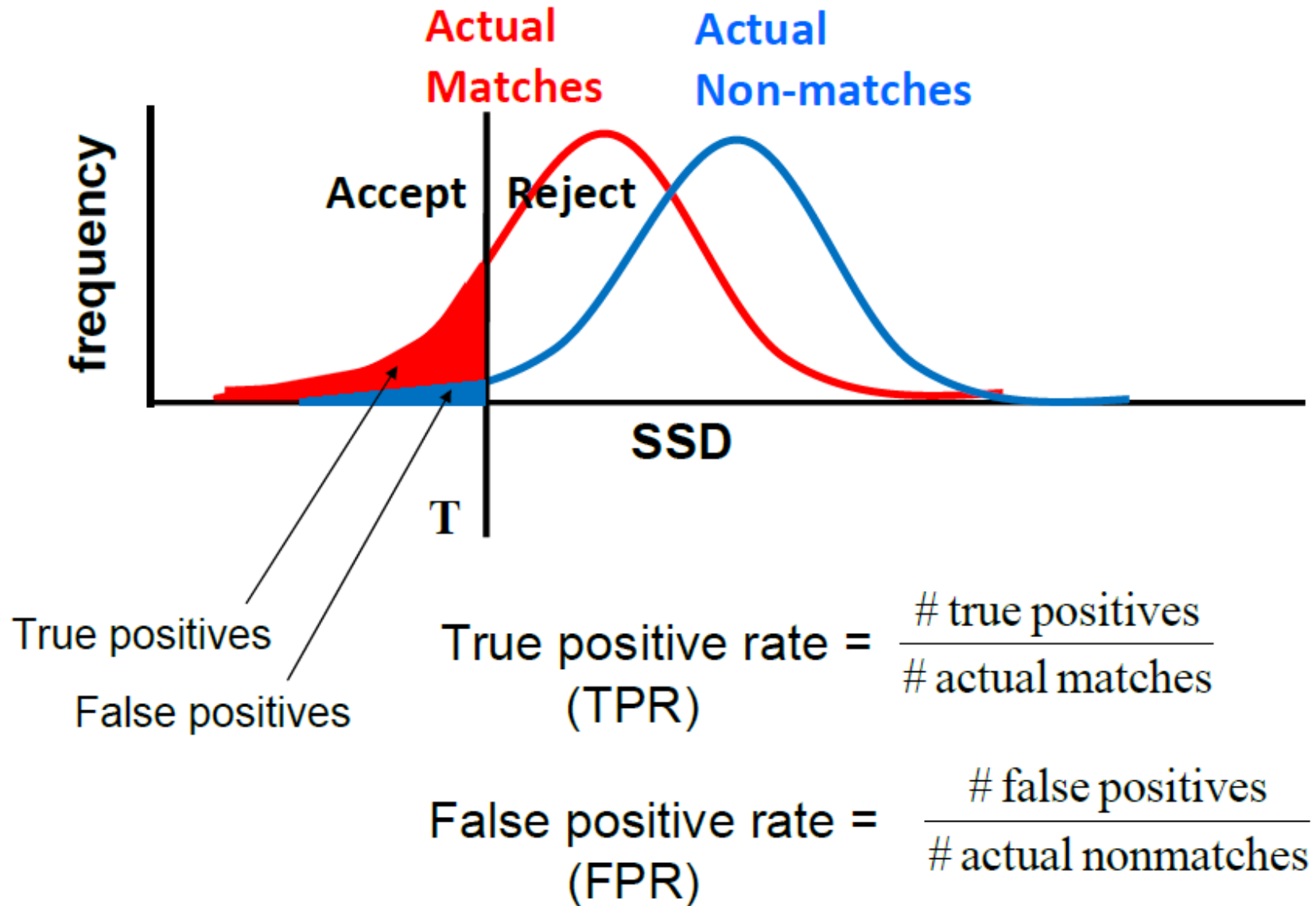
Example: **Smaller T**

$T = 100 \Rightarrow$ only a and b are accepted as matches

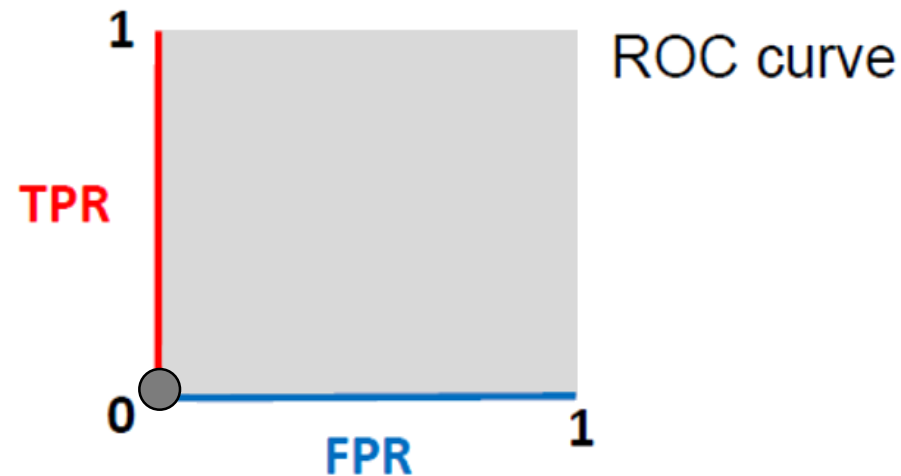
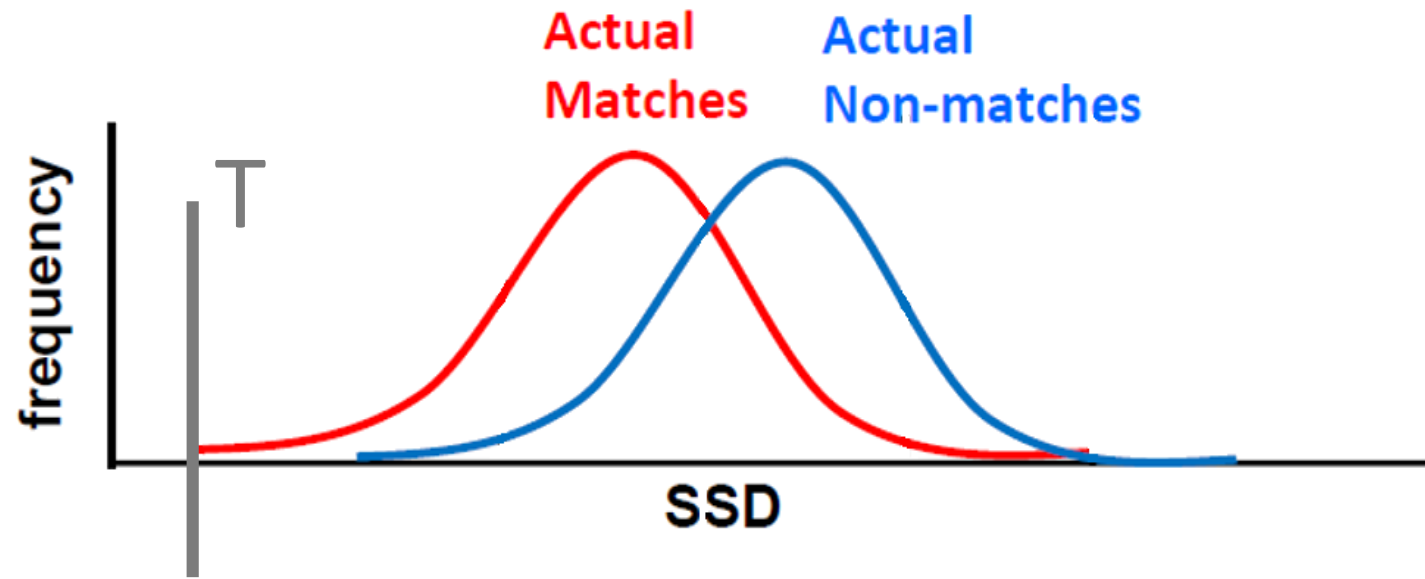
a and b are true matches (“true positives”)

c is no longer a “false positive” (it is a “true negative”)

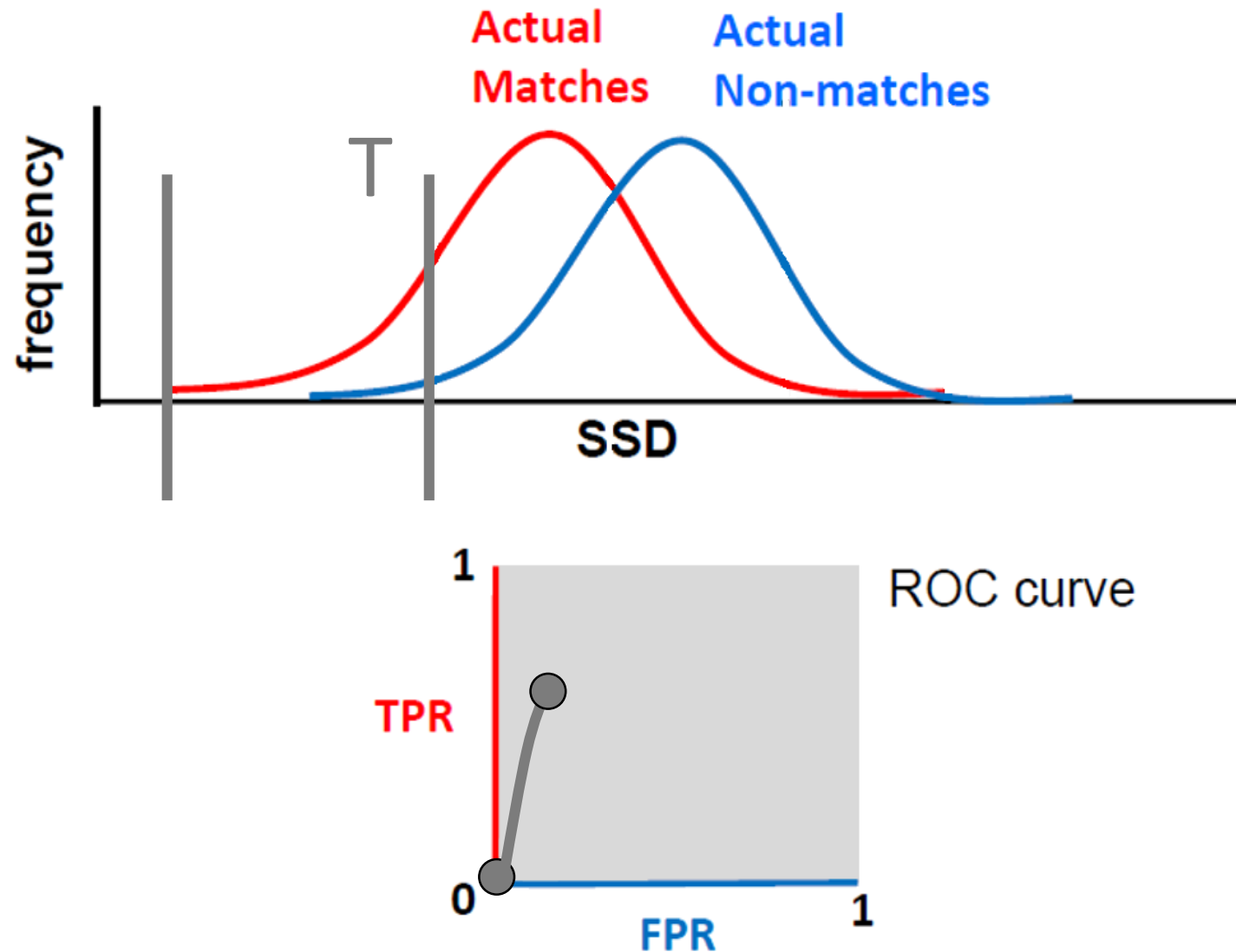
True positives and false positives



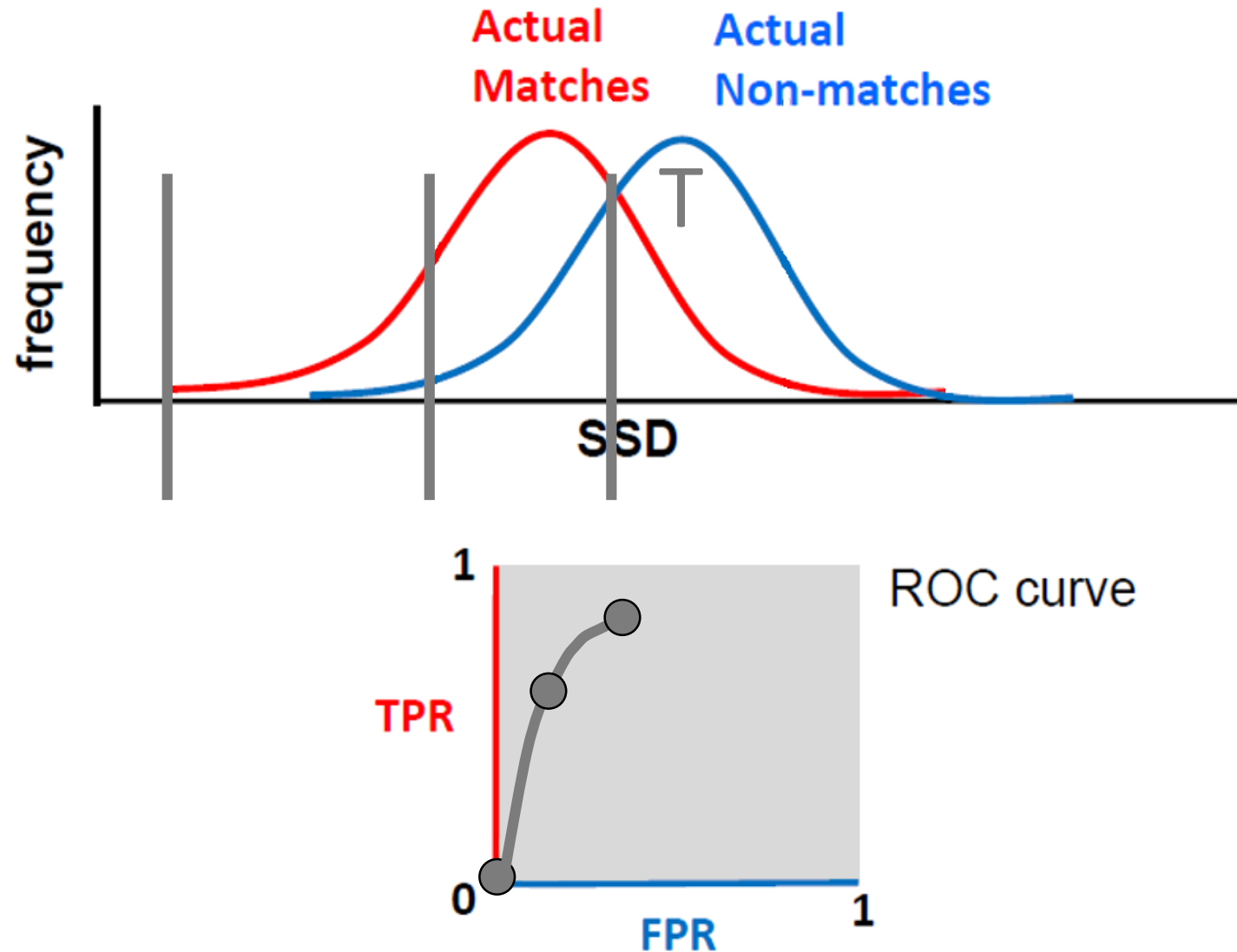
Receiver Operating Characteristic (ROC) curve



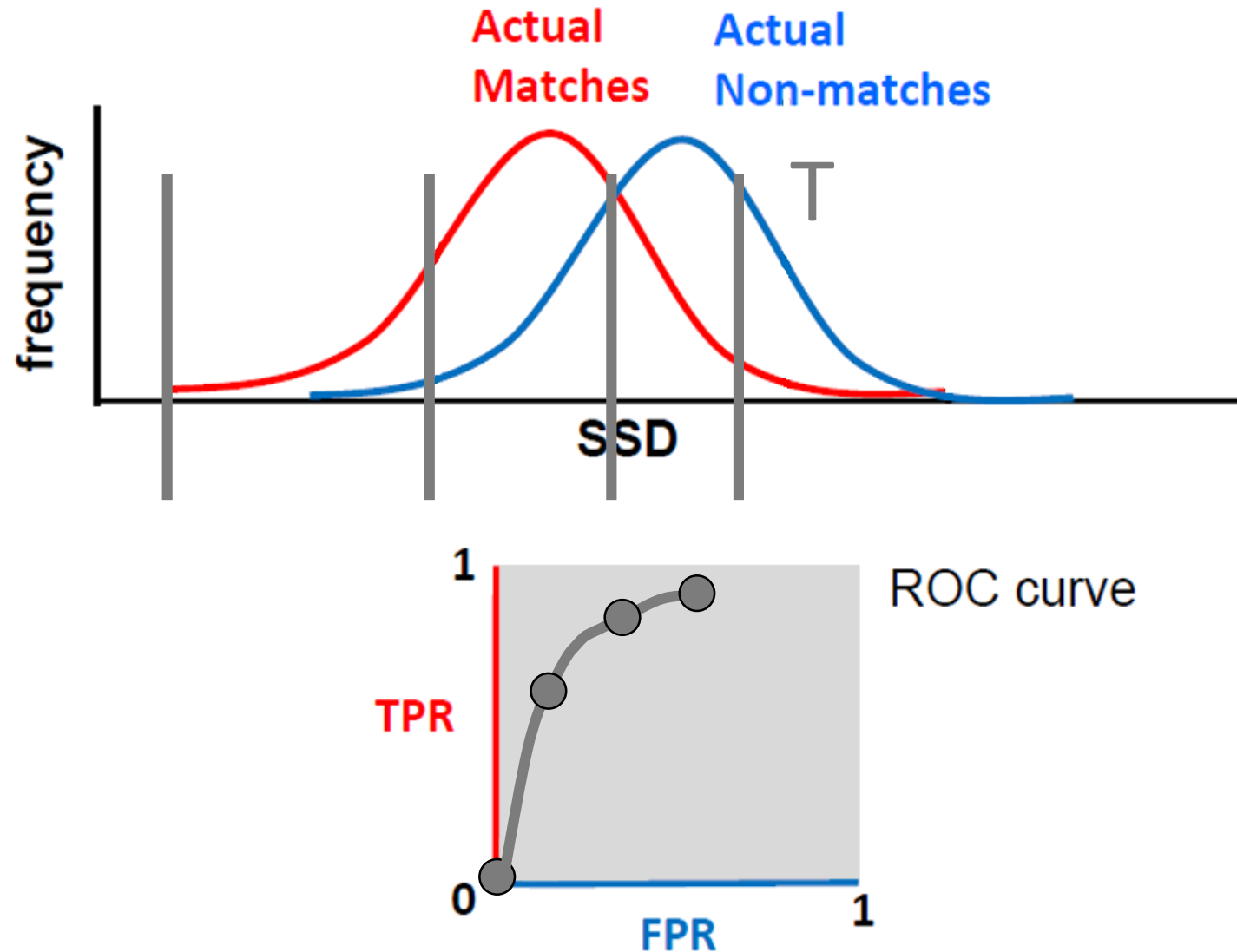
Receiver Operating Characteristic (ROC) curve



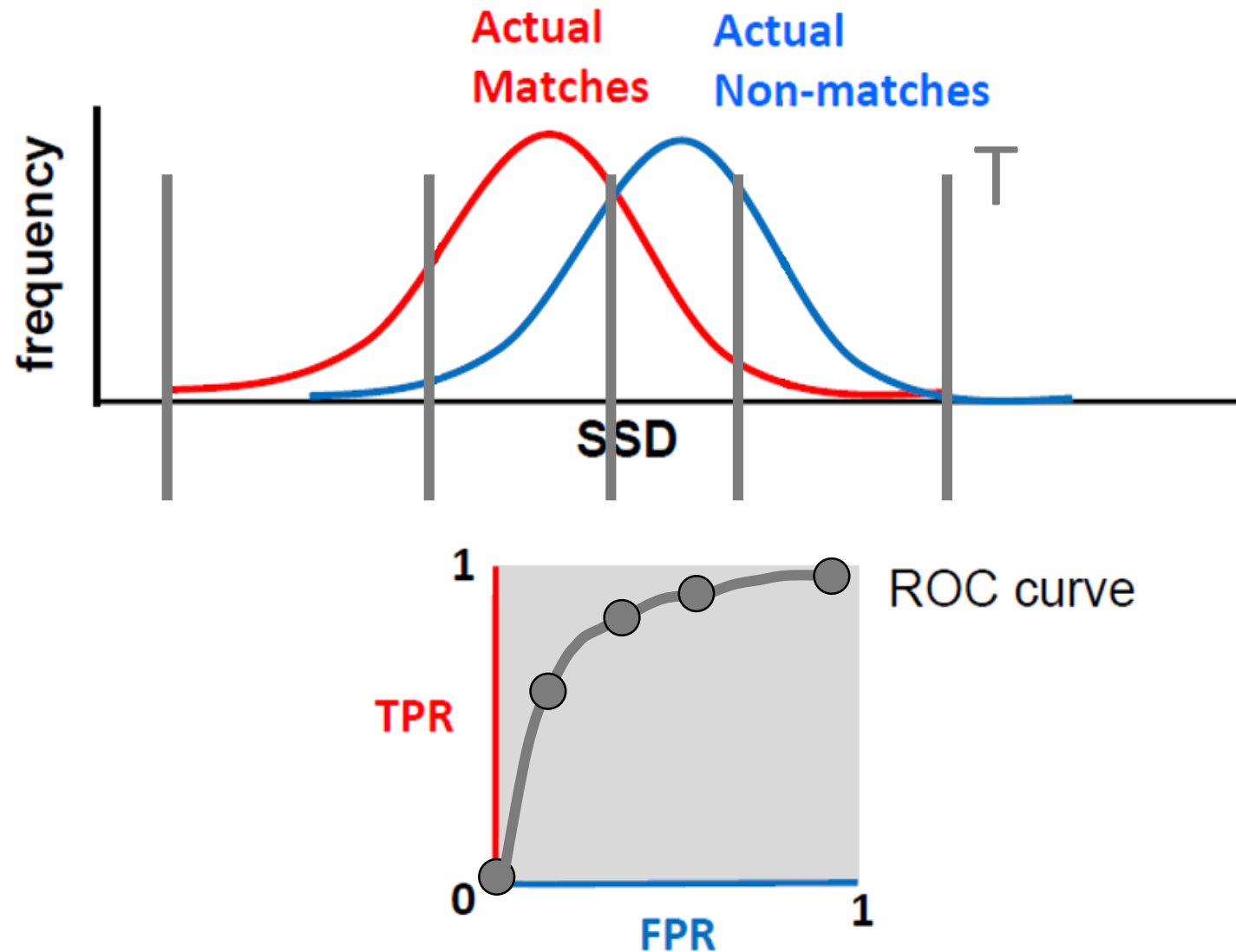
Receiver Operating Characteristic (ROC) curve



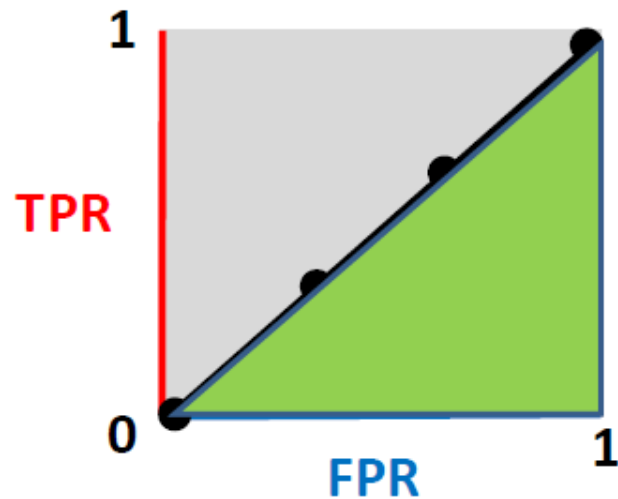
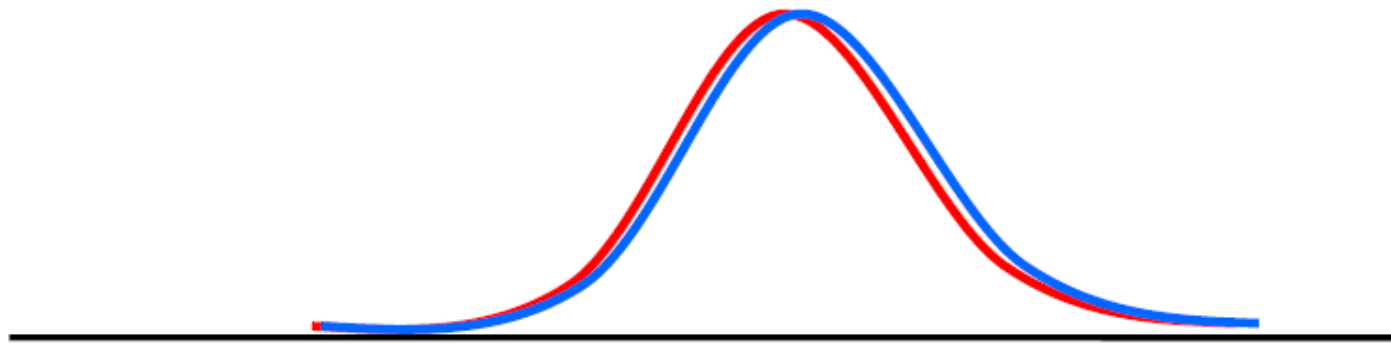
Receiver Operating Characteristic (ROC) curve



Receiver Operating Characteristic (ROC) curve



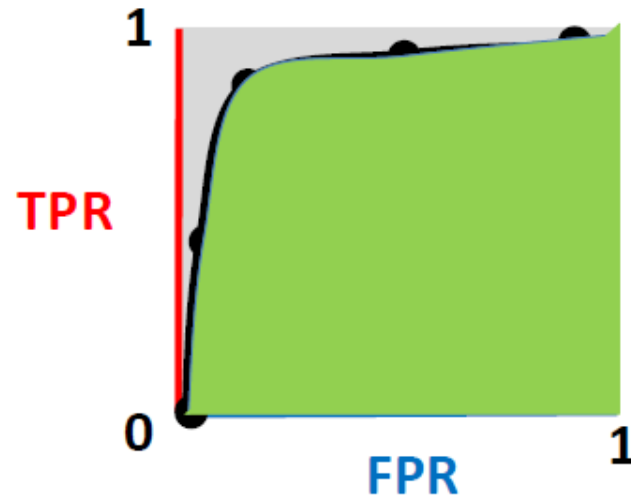
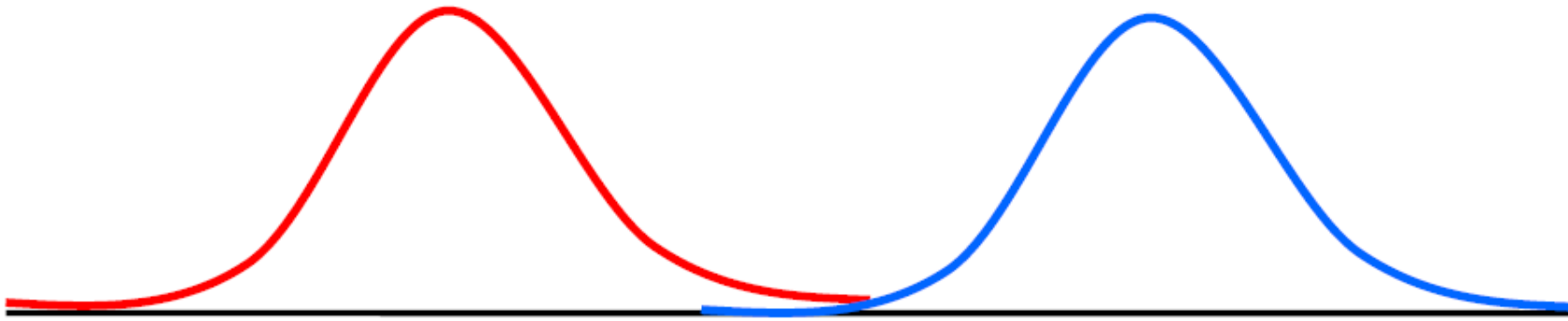
If the features selected were bad...



Area ≈ 0.5

Adapted from a slide by Shin Kira

If the features selected were good...

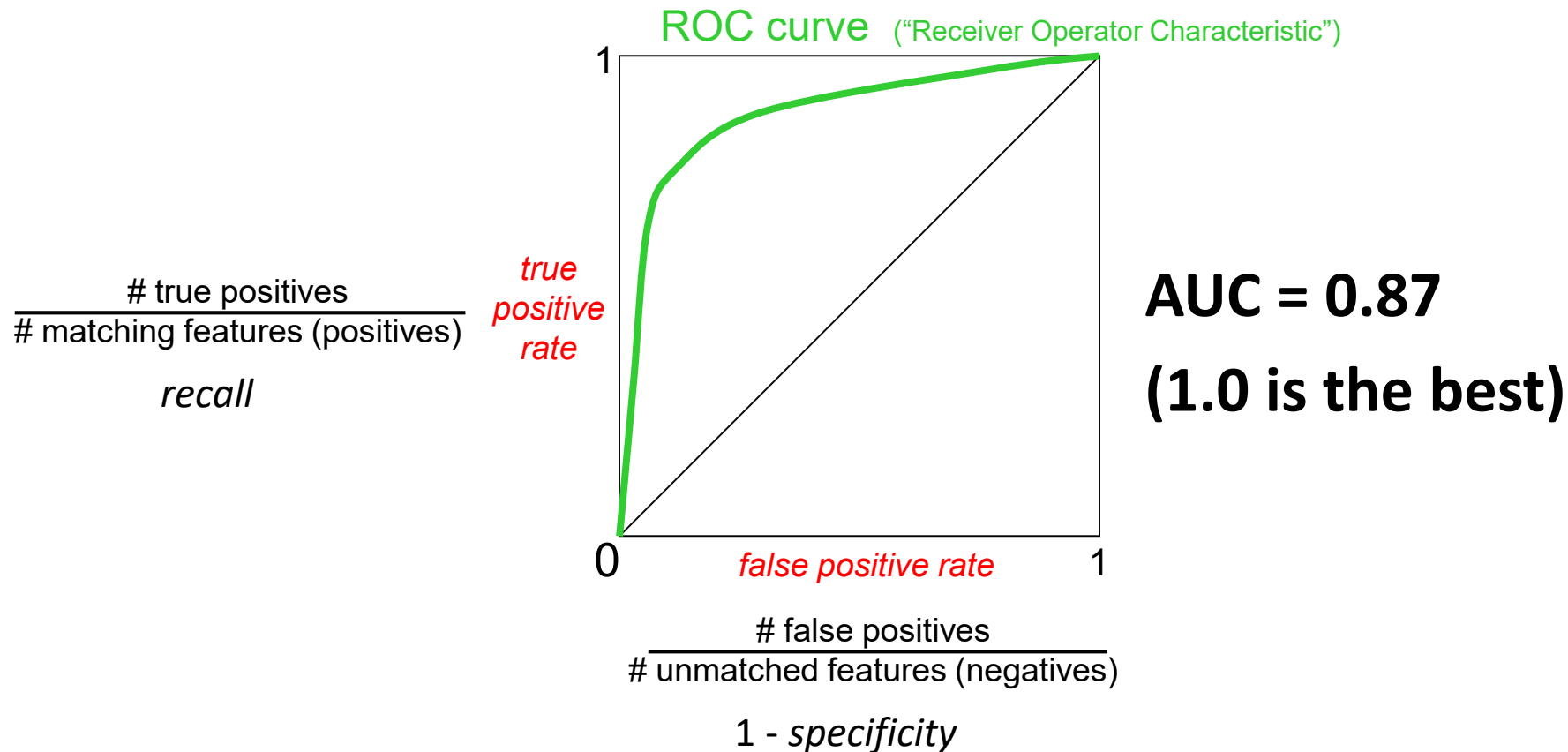


Area ≈ 1.0

Adapted from a slide by Shin Kira

Area under the curve

Single number: Area Under the Curve (AUC)



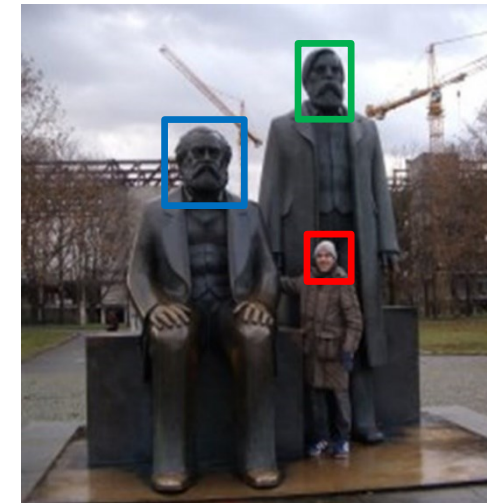
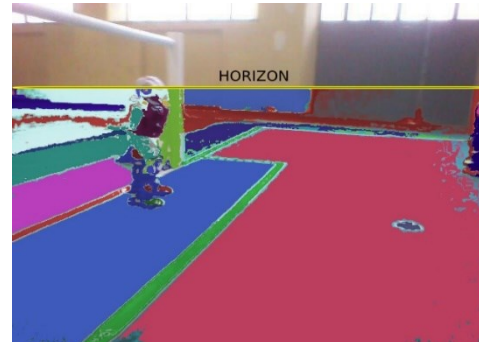
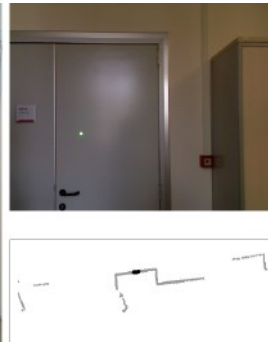
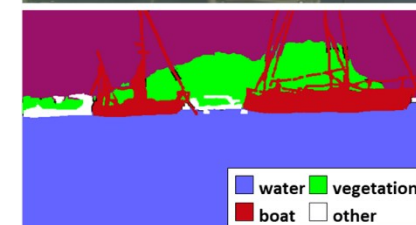
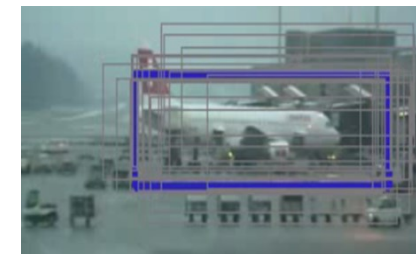


**UNIVERSITÀ DEGLI STUDI
DELLA BASILICATA**

Corso di Sistemi Informativi
A.A. 2018/19

Feature Matching

Docente
Domenico Daniele Bloisi



Aprile 2019