

Lezione 5 - Interest Point Detectors and Descriptors -

28/10/2024

Interest Point Detectors and Descriptors

è la task di object detection, trovando anche la posizione.

Il primo caso è quello dove siamo interessati a classi di oggetti. Per esempio vogliamo sapere se l'immagine contiene un'auto, qualsiasi tipo di auto. Oppure vogliamo sapere se c'è un'animale, non ci interessa che tipo di animale.

Alternativamente, vogliamo riconoscere istanze di oggetti. Vogliamo per esempio trovare un modello specifico di auto, con un colore particolare. Oppure siamo interessati ad un libro particolare, con un titolo particolare.

Object instance recognition

Ora ci concentriamo sul secondo tipo di problema.

In questo caso vogliamo sapere se l'oggetto è presente nell'immagine, e vogliamo anche localizzarlo, usando un "bounding box".



Qual è il modo più semplice in cui possiamo farlo? Sarebbe iniziare in alto a sinistra dell'immagine, e cercando per ciascuna posizione se troviamo l'oggetto

oppure no. Potremmo calcolare la differenza dei pixels, e se questa è sufficientemente bassa, possiamo dire di aver trovato l'oggetto. Quindi proveremmo tutte le possibili posizioni.

Questo approccio è molto inefficiente, e non gestisce neanche tutte le possibilità, per esempio la grandezza dell'oggetto, o la rotazione... Se vogliamo provare anche queste per ogni posizione, l'approccio diventa ancora più inefficiente.

Se mettiamo da parte l'inefficienza di questo metodo, abbiamo detto che per ogni posizione vogliamo calcolare la differenza tra l'oggetto e l'area che stiamo controllando. Qual è un buon metodo per confrontare le due aree?

Questo processo si chiama **template matching**.

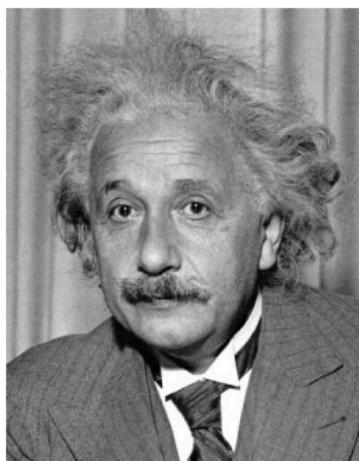
Il metodo più diretto è la

Abbiamo tolto la radice (dalla classica distanza euclidea) perché la radice è monotona quindi ri-scalerebbe i valori ma non cambierebbe le relazioni d'ordine, quindi la togliamo per risparmiare computazione.

Goal: find  in image

Method: SSD (*Sum of Squared Differences*)

$$h[m,n] = \sum_{k,l} (g[k,l] - f[m+k, n+l])^2$$

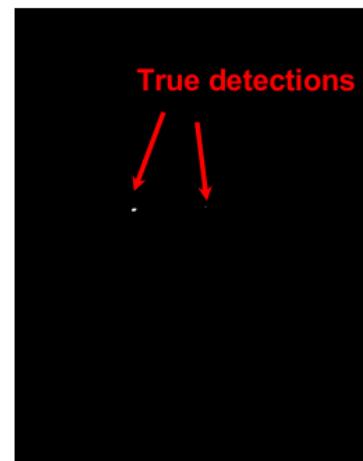


Slide credits: Hoiem

Input



1 - \sqrt{SSD}



Thresholded Image

True detections

In ciascuna posizione $[m,n]$ prendiamo il template, sottraiamo la finestra centrata in m,n

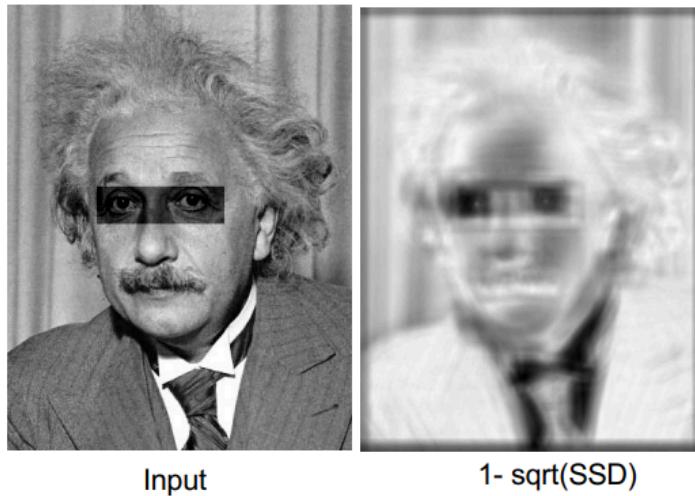
I pixels che hanno i colori più chiari sono quelli dove l'immagine è più simile. Qui abbiamo trovato 2 posizioni con pixels chiari, una più grande e una più piccola (per l'altro occhio).

Questo funziona perchè il template è stato preso direttamente dall'immagine originale.

Ora proviamo ad usare lo stesso template, ma rendendo quest'area dell'immagine più scura, di modo che il template non sia identico.

Goal: find  in image

Method: SSD
$$h[m,n] = \sum_{k,l} (g[k,l] - f[m+k, n+l])^2$$



Vediamo che ora non abbiamo più i punti bianchi, ma invece sono aree ancora più scure. Quindi non riusciamo più a trovare il nostro template. (i punti più simili sono quelli più chiari)

Supponiamo che questa è un tipo di trasformazione che vogliamo poter gestire, perchè per esempio abbiamo foto con livelli di luce diversi (giorno, notte, al chiuso, all'aperto), quindi vorremmo poter compensare.

Un metodo diverso di confrontare il template con la window è questa **normalized cross-correlation**.

Method : Normalized cross-correlation

$$h[m,n] = \frac{\sum_{k,l} (g[k,l] - \bar{g})(f[m-k, n-l] - \bar{f}_{m,n})}{\left(\sum_{k,l} (g[k,l] - \bar{g})^2 \sum_{k,l} (f[m-k, n-l] - \bar{f}_{m,n})^2 \right)^{0.5}}$$

mean template mean image patch

Qui come prima cosa centriamo il template a 0, sottraendo la media del template (g). Poi, quando andiamo in una posizione specifica dell'immagine (f), prendiamo la porzione dell'immagine attuale e sottraiamo la media f, per centrare anche a 0 questa parte.

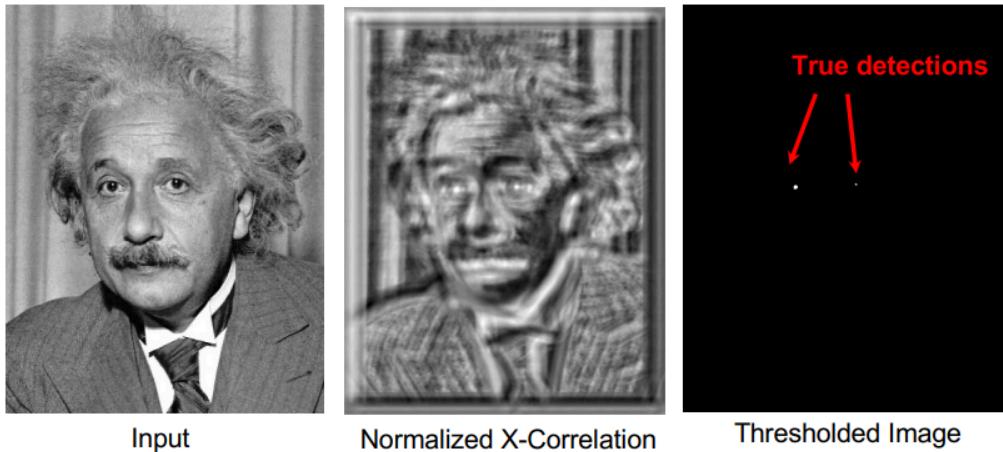
L'idea è che per ogni posizione, stiamo compensando per cambiamenti di contrasto nell'immagine. Stiamo "zero-centering" ogni window che stiamo analizzando. Quindi ogni volta che faccio un confronto locale, normalizzo l'immagine.

Nella parte sotto della formula abbiamo più o meno la stessa cosa, ed è usato per fare in modo che sia una cross-correlation, tenendo l'output tra -1 e 1. è un fattore di normalizzazione. La parte importante è quella sopra, dove facciamo lo zero-centering sul template e poi su ogni window.

Questa è ancora più pesante da computare, perchè ad ogni posizione stiamo computando la media locale per fare lo zero-centering della finestra.

Goal: find  in image

Method: Normalized cross-correlation

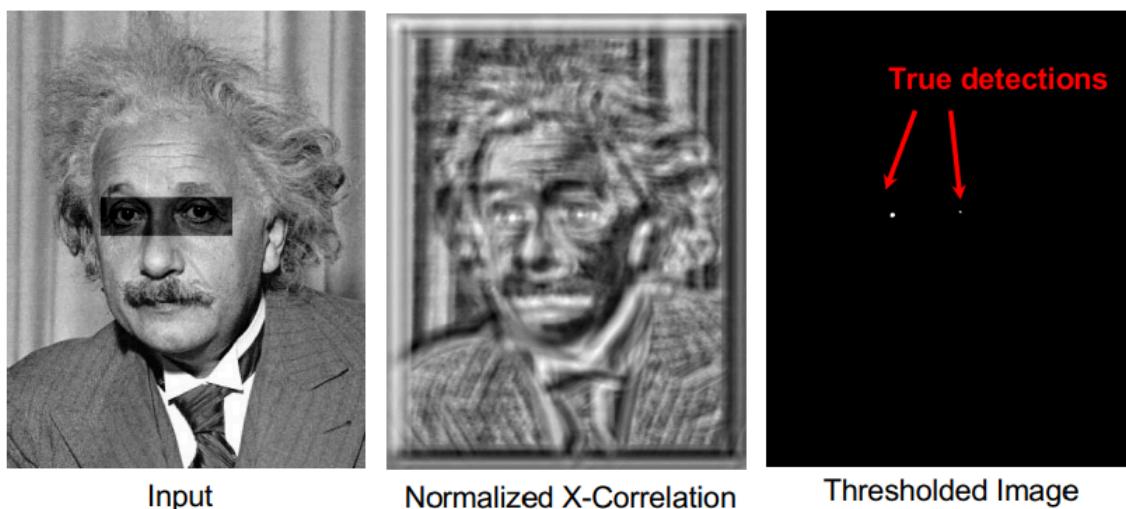


Facciamo un threshold della seconda immagine e troviamo la terza, dove ritroviamo i due puntini bianchi.

Vediamo ora cosa accade nella seconda immagine, quella scurita.

Goal: find  in image

Method: Normalized cross-correlation



Notiamo come la normalized cross-correlation riesce a trovare il template.

SSD:

- Più veloce, perchè non dobbiamo calcolare la local average per ogni finestra.
- Sensibile all'intensità della luce.

Normalized cross-correlation:

- Più lenta, perchè dobbiamo calcolare la local average per ogni finestra.
- Resistente a "local average intensity" e "local contrast".

Quindi usiamo il primo metodo se abbiamo il controllo sulla luce.

Variability

◦ Example



Di solito non sappiamo che dimensione avrà l'immagine che stiamo cercando.

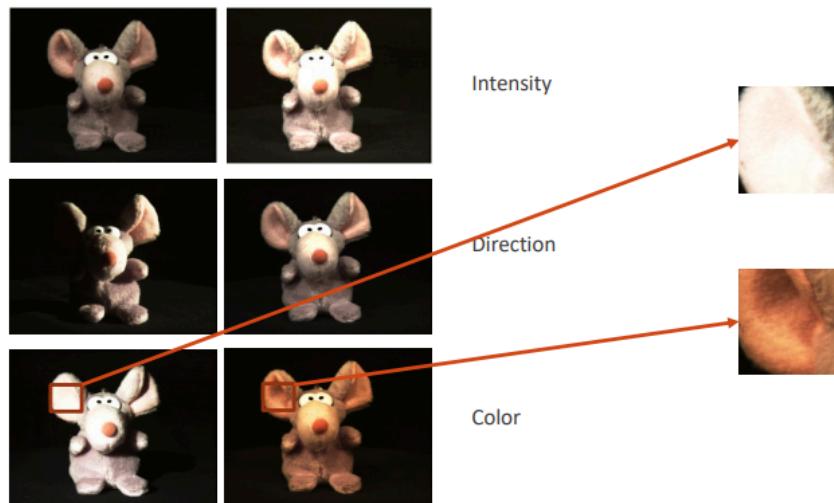
Vorremmo avere un algoritmo che può gestire la **traslazione**, ovvero dove si trova l'oggetto, la **dimensione** (scale) dell'oggetto, e la **rotazione** nell'asse verticale ed

orizzontale.

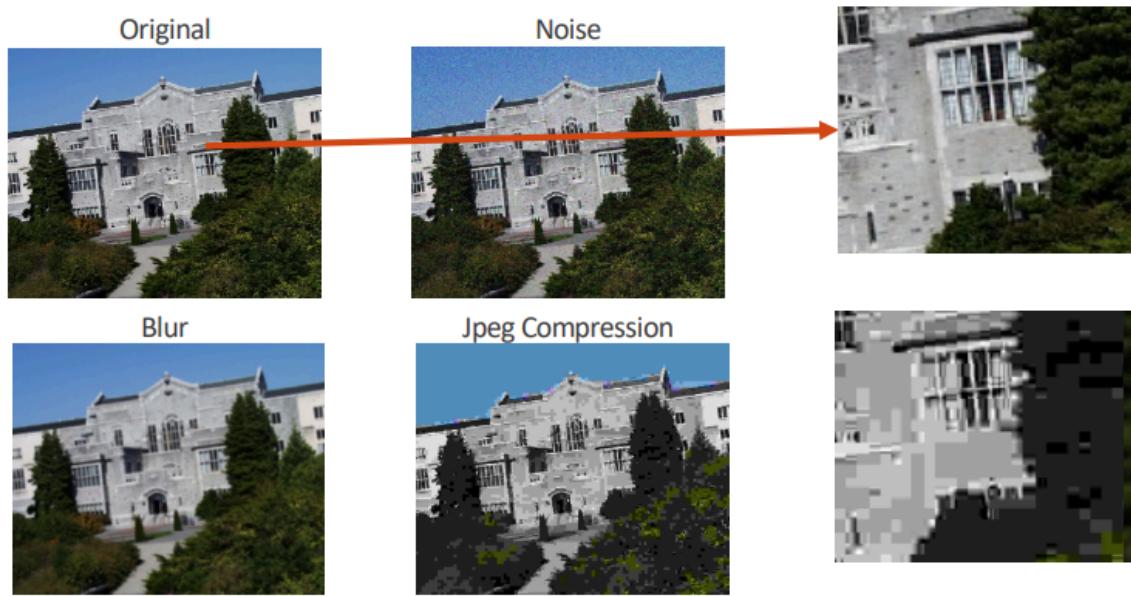


Il caso dell'auto che ruota è ancora più complicato, perché per esempio il retro dell'immagine può essere molto diverso dalla fronte. Quindi avremo bisogno di ancora più robustezza.

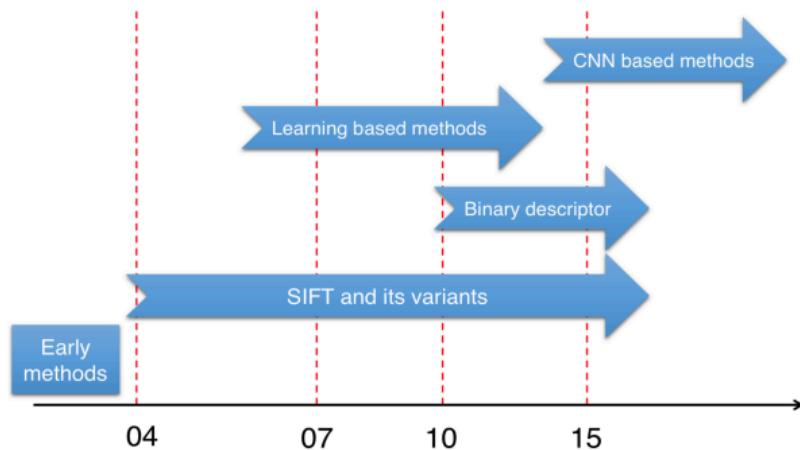
Ci sono anche altri effetti che vogliamo poter gestire.



Nelle immagini digitali abbiamo anche artefatti su cui vogliamo robustezza. Ci può essere per esempio noise perché l'immagine è stata scattata con poca luce, oppure l'immagine può essere sfocata, oppure anche gli artefatti della compressione jpeg.

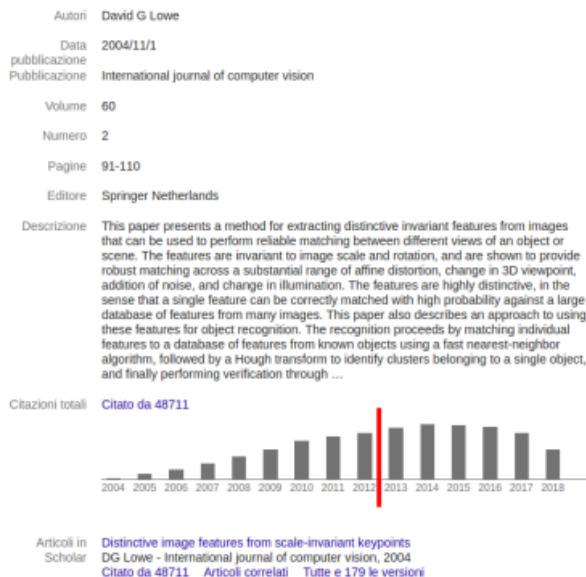


Local Descriptors Trends



SIFT Paper Citation trends

Distinctive image features from scale-invariant keypoints



- SIFT paper is among the most cited computer vision papers ever.
- But it's not as dominant as it once was.

Why Keypoints-based methods?

Keypoints Remain Relevant

- When accurate geometric recovery matters, they remain unequaled.
- They are efficient for real-time applications.
- They provide an effective way
 - to compress the information present in large images,
 - to recognize specific locations.
- The algorithms do not need to be retrained for each new application.

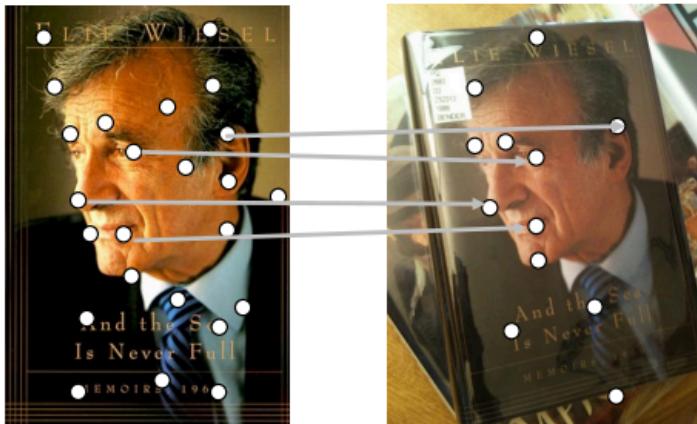
Future algorithms will combine Deep Learning and keypoint matching.

Keypoint based approaches

Un keypoint è un punto nell'immagine che è facile trovare in altre immagini (altre versioni della stessa immagine).

Steps principali:

- **keypoint detection**, cerchiamo questi punti importanti
- keypoint description, per ogni punto calcoliamo la descrizione, ovvero l'aspetto del keypoint
- **matching of similar keypoints**, tra le due immagini matchiamo quelli più simili
- **similarity score based on matching keypoints**, in base per esempio a quanti keypoints matchati abbiamo trovato



Steps

- Keypoint detection
- Keypoint description
- Matching of similar keypoints
- Similarity score based on matching points

- Use of Interest points (keypoints)
- Do not need to try all the combinations of transformations.

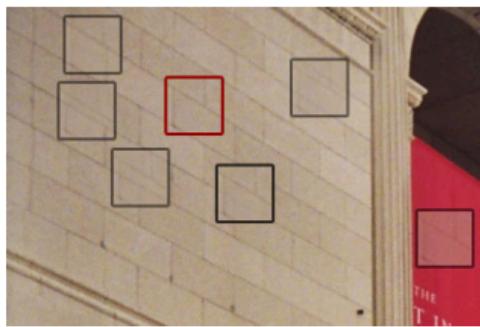
Harris corner detector

Uno dei metodi più semplici è l'**Harris corner detector**.

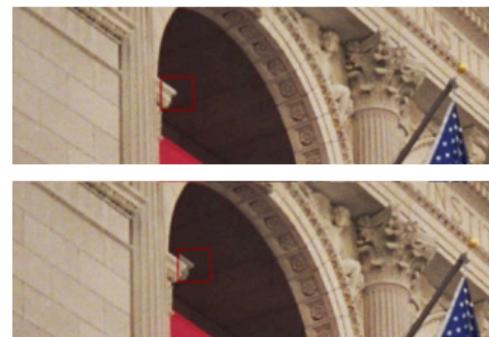
I corners sono migliori degli edges. Gli angoli sono buoni keypoints perché sono più facili da trovare.

- Corners are better than edges!
- How to define a corner?
- Harris et al. [1] → Patches that generate a large variation when moved around

Example of a **bad** corner



Example of a **good** corner



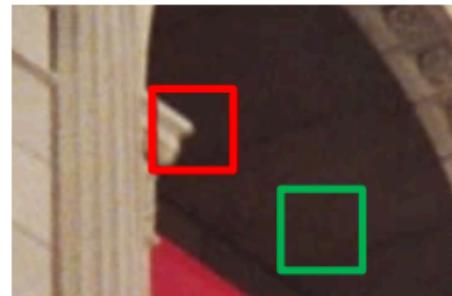
Un corner è un "patch" che genera una grande variazione quando è spostato.

Patches that generate a large variation when moved around



$$E(u, v) = \sum_{x,y} w(x, y)[I(x + u, y + v) - I(x, y)]^2$$

- E is the difference between the original and the moved window.
- u is the window's displacement in the x direction
- v is the window's displacement in the y direction
- $w(x, y)$ is the window at position (x, y) . This acts like a mask. Ensuring that only the desired window is used.
- I is the intensity of the image at a position (x, y)
- $I(x+u, y+v)$ is the intensity of the moved window
- $I(x, y)$ is the intensity of the original window



Questo metodo ci da un modo di trovare questi keypoints, che è il primo punto di quelli sopra. Dovremo usare altri metodi per gli altri punti.

SIFT

SIFT (Scale Invariant Features Transform) oltre a trovare i keypoints, ci da anche una descrizione di questi che è robusta a: scale, rotation, illumination, viewpoint.

- SIFT [1] (Scale Invariant Features Transform)
- Author: David Lowe. University of British Columbia –Canada

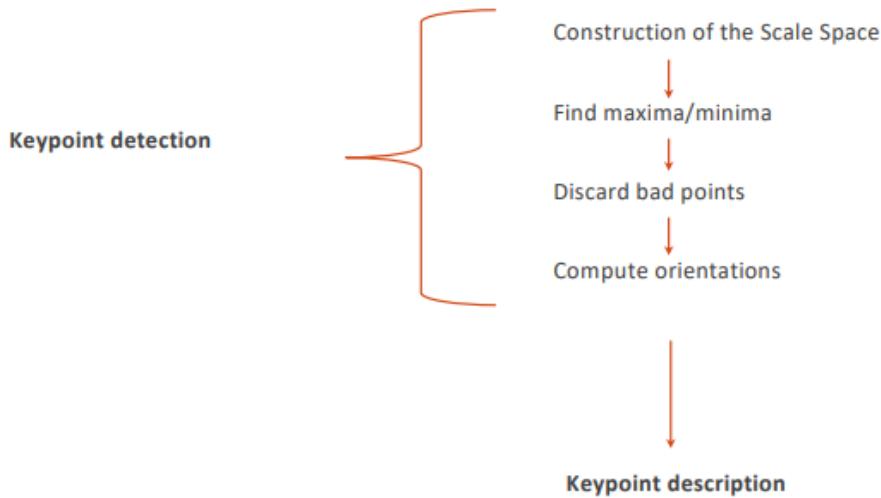
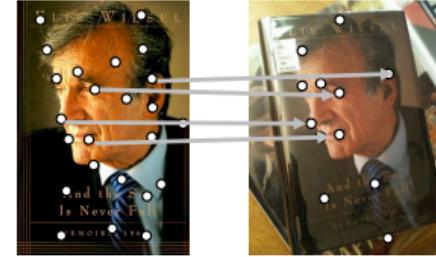


Main difference with Harris detector:

- Not only a Keypoint Detector but also a **Keypoint Descriptor**

Robust to:

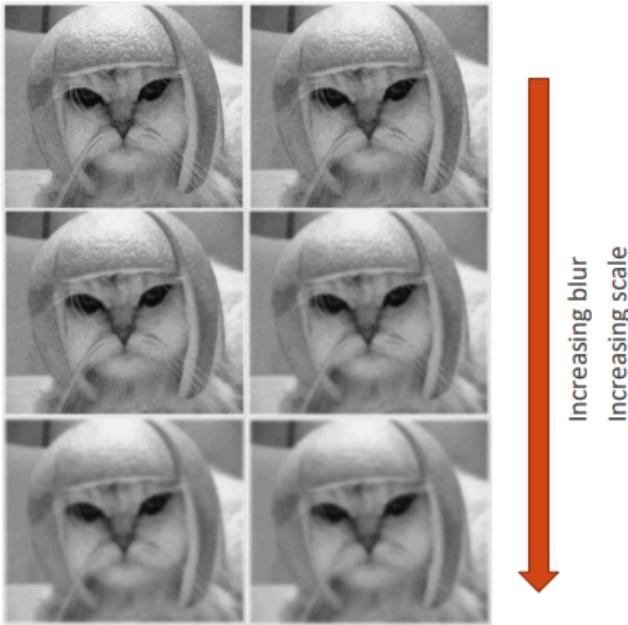
- Scale
- Rotation
- Illumination
- Viewpoint



Scale space

L'idea è che man mano che aumenti la distanza tra la camera a l'oggetto, l'oggetto diventerà più sfocato, quindi si perdono i dettagli dell'immagine.

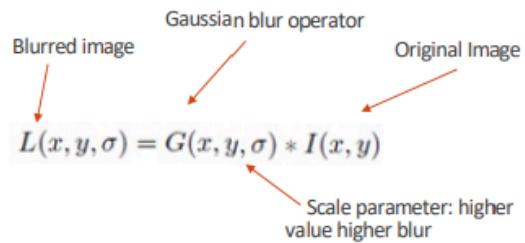
Viene usato questo metodo, tramite una sfocatura gaussiana, che sfoca l'immagine a diversi livelli, creando più varianti.



◦ Needed to obtain **Scale Invariance**

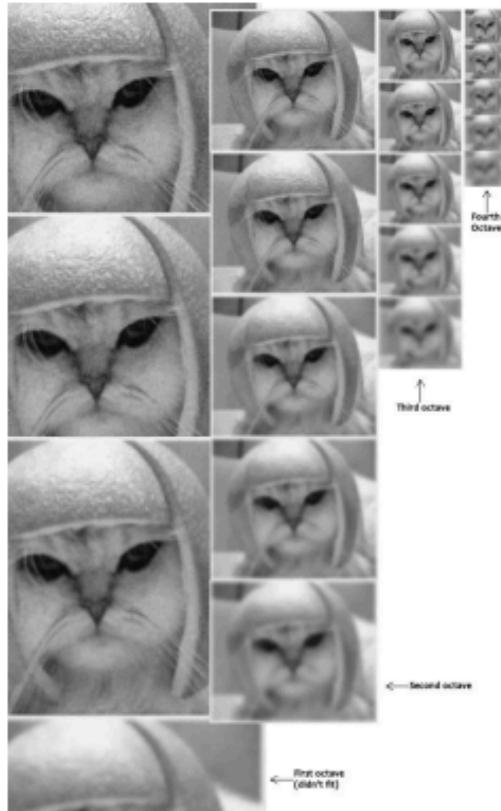
◦ Higher scales = higher blur

◦ Higher blur = less details



$$G(x, y, \sigma) = \frac{1}{2\pi\sigma^2} e^{-(x^2+y^2)/2\sigma^2}$$

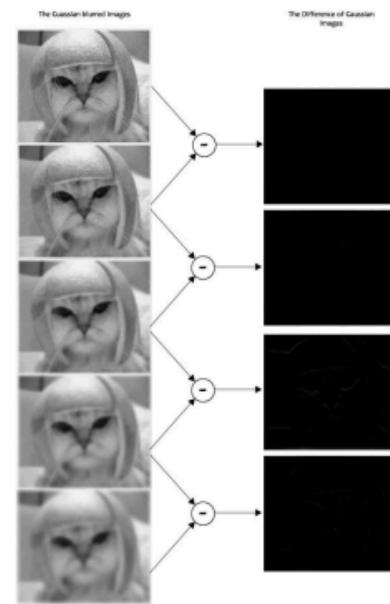
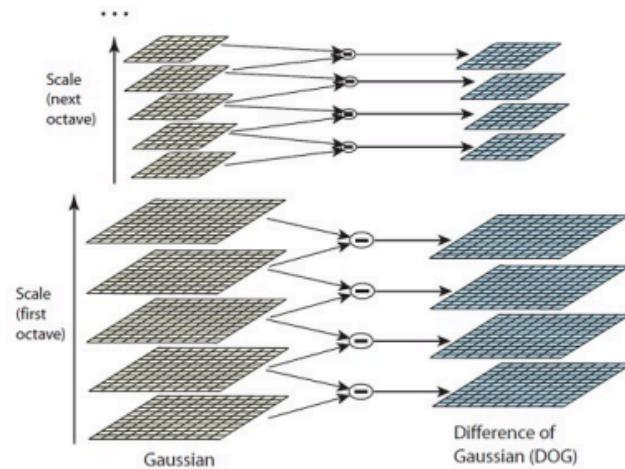
Quindi abbiamo l'immagine, abbiamo il filter kernel. Un metodo per avere più sfocatura è avere un filter kernel più grande, che però è più costoso computazionalmente. L'idea è quella di ridurre la grandezza dell'immagine, ed applicare lo stesso filter kernel (al posto di aumentare il filtro gaussiano come prima).



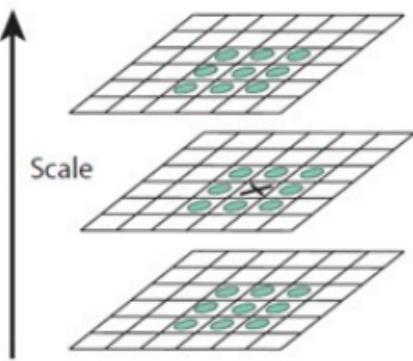
- Scales and Octaves
- Usually 4 octaves and 5 blur levels
- Reduces computations

Quando abbiamo computato lo scale space, calcoliamo le differenze tra 2 immagini consecutive, ottenendo una nuova rappresentazione: la **difference of gaussian (DOG)**.

- Used to detect edges and corners
- Laplacian of Gaussian (LoG) approximation
- Very fast and efficient (only subtraction)



Vogliamo trovare punti di massimo/minimo in questa nuova rappresentazione



- X is the current pixel
- Green circles are the considered neighbours
- 26 checks.
- X is a **keypoint** if it is the greatest of all its neighbours
- Usually there is no need to check all the neighbours.
Discard happens after few checks.

Poi andiamo a ridurre i keypoints facendo dei controlli:

- To be removed
- Low Contrast Points
 - Check for the value of pixel in the DoG image
 - If value is under threshold the point is rejected
 - Edges
 - Compute gradients in the two directions.
 - Gradient in one direction is just a difference of pixel values

Flat region		
40	40	40
41	45	42
40	41	40

Abs difference
 $41 - 40 = 1$
 $42 - 41 = 1$

Small Small

Edge		
20	20	20
40	45	40
40	40	40

Big Small

Corner		
20	20	20
20	45	40
20	40	40

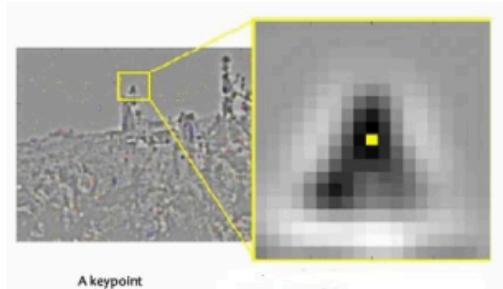
Big Big

Searching for this!

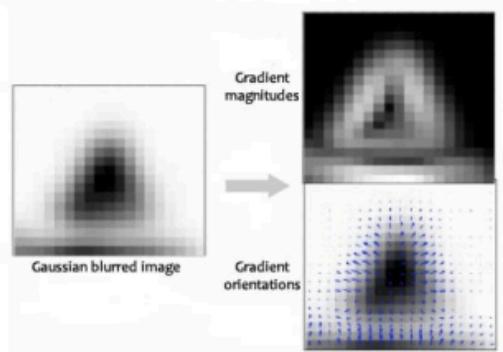
Vogliamo escludere le regioni piatte e gli edges. Ci interessano gli angoli (differenza elevata in entrambe le direzioni).

Keypoint orientation

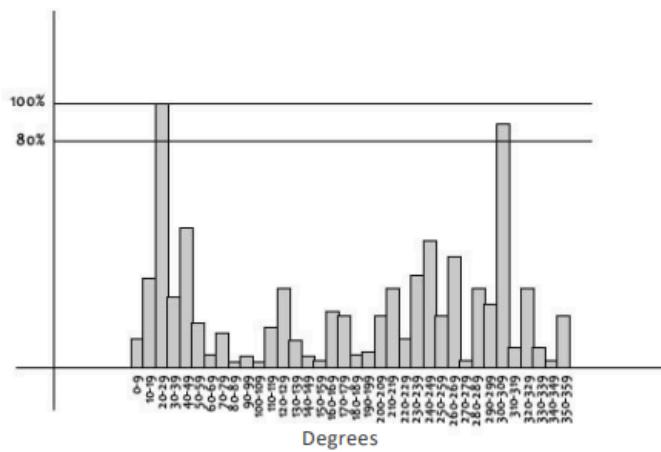
Se questo è un keypoint, calcoliamo l'orientazione, costruendo un istogramma delle orientazioni intorno al punto. Andiamo ad assegnare la direzione più popolata dell'istogramma come direzione principale.



- Goal: achieve **rotation invariance**
- Histogram of Orientations
- Weighted by Gradient magnitudes



Gradient magnitude Gradient orientation Gaussian blurred image
 $m(x, y) = \sqrt{(L(x + 1, y) - L(x - 1, y))^2 + (L(x, y + 1) - L(x, y - 1))^2}$
 $\theta(x, y) = \tan^{-1}((L(x, y + 1) - L(x, y - 1)) / (L(x + 1, y) - L(x - 1, y)))$

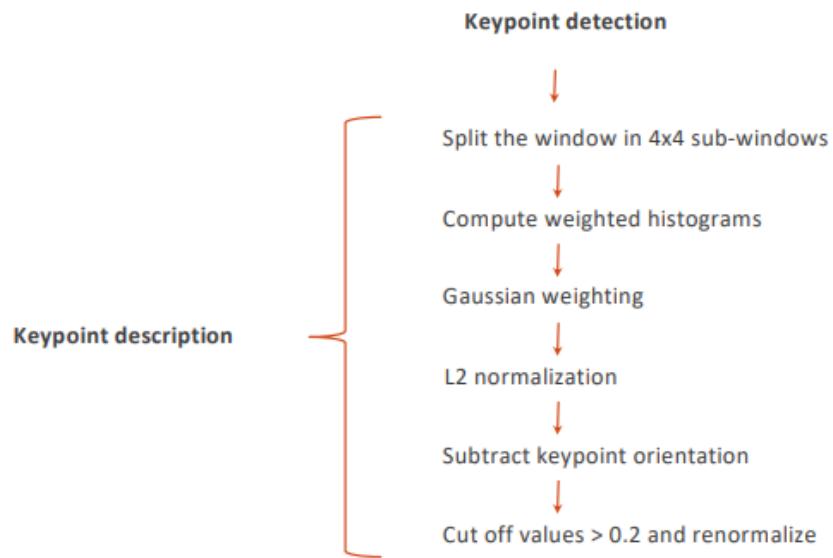


- 36 bin (10 degrees per bin)
- Dominant orientation given by the highest peak
- Peaks higher than the 80% of the highest are kept (the keypoint is split, i.e. duplicated).
- New keypoint has the same scale and location but different orientation

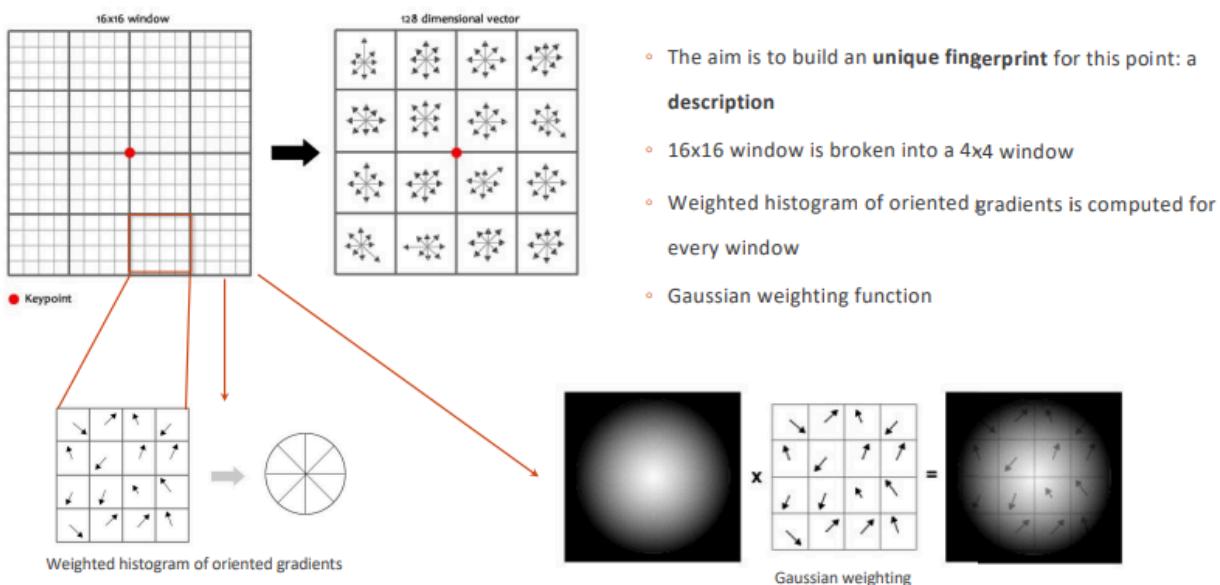
L'idea è che dopo aver assegnato la direzione all'immagine, andiamo a normalizzare questo punto mettendo l'orientazione a 0 gradi (che punta verso destra). A questo punto possiamo confrontare due keypoints per vedere se sono simili o no.

keypoint description

Questo era il keypoint detection. Ora vogliamo poter descrivere il keypoint, per poter confrontare quelli dell'immagine. L'idea è quella di usare gli istogrammi.

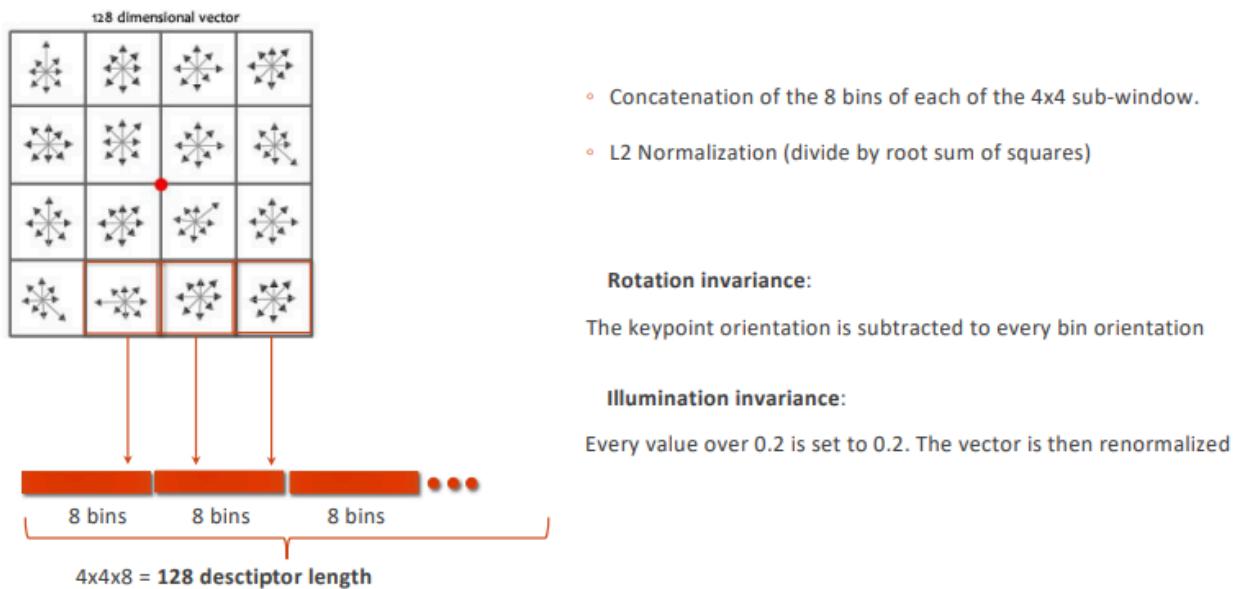


Abbiamo il keypoint, prendiamo i pixel intorno, dividiamo in window 4×4 , per ciascuna finestra calcoliamo “weighted histogram of oriented gradients” (i gradienti dell’immagine, calcolando un istogramma lungo 8 direzioni principali).



Guardando alle differenze, creo robustezza rispetto alla quantità di luce che c’è nell’immagine, sto dicendo che un pixel è più chiaro o più scuro dell’altro. Questo è il motivo per il quale uso il gradiente per fare le direzioni.

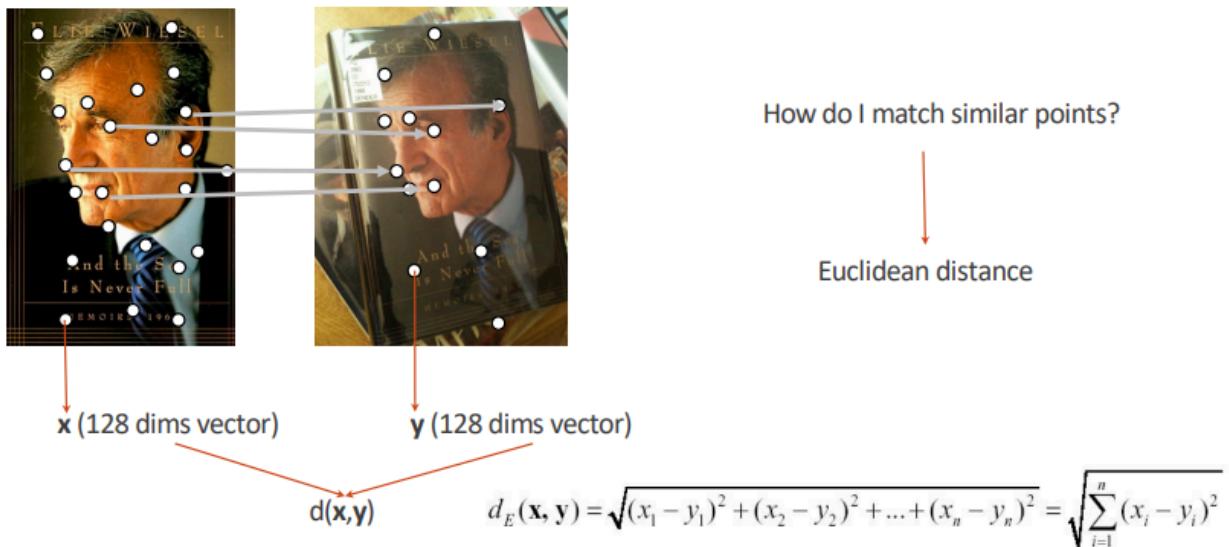
Usiamo solo 8 bin per i 360°, quindi questa risoluzione molto grossolana ci dà altre invarianti.



Per costruire un descriptor di un keypoint, concateniamo i bins di ciascun istogramma di ciascuna finestra, arrivando a 128 valori, che sono la fotografia del nostro punto.

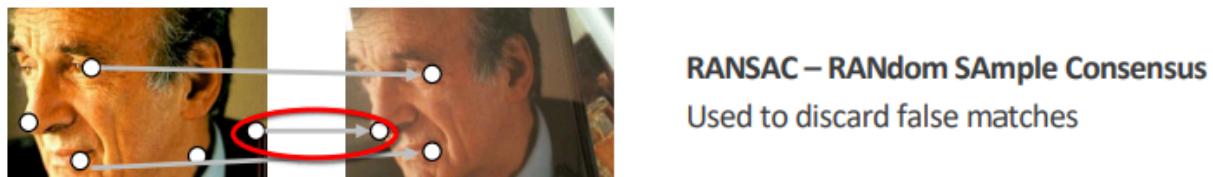
Keypoint matching

Quindi ciascun keypoint sia sull'immagine originale che sull'altra è un vettore di 128 dimensioni. Si calcola la distanza euclidea tra 2, e lo facciamo per tutti, cercando quali sono le coppie migliori, con distanza più bassa. Consideriamo matching pairs i punti che hanno un threshold sopra un tot.



Vogliamo trovare una trasformazione dai punti a sinistra a quelli di destra, sapendo che ci potrebbe essere del rumore (match sbagliati), per essere robusti.

Questo viene fatto usando RANSAC (RANdom SAMple Consensus):

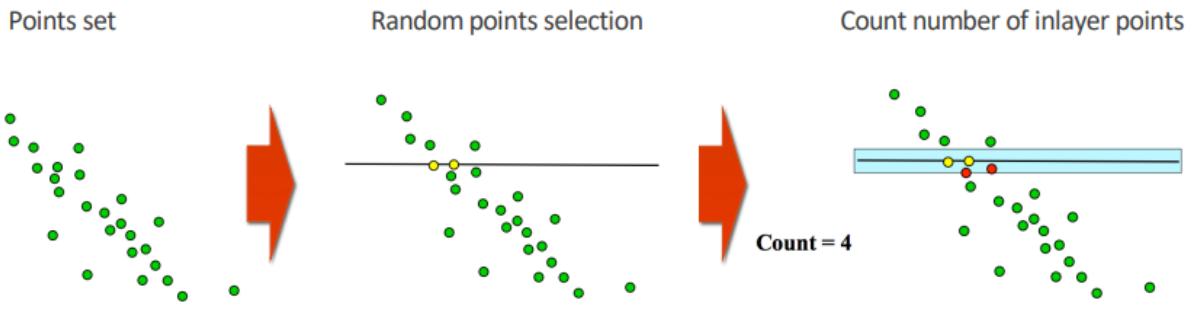


Algorithm outline:

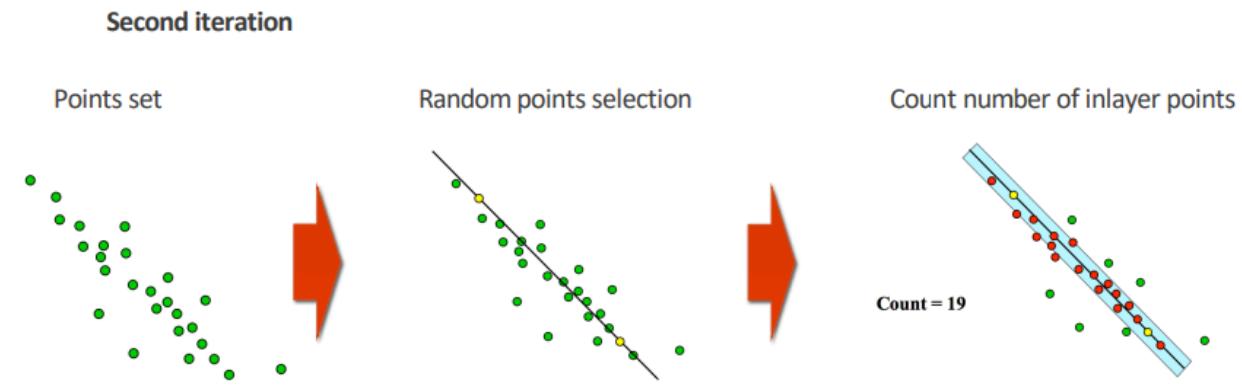
1. Select at least four feature pairs (at random)
 2. Compute homography H (exact)
 3. Compute inliers where $SSD(p', Hp) < \epsilon$
 4. Keep largest set of inliers
 5. Re-compute least-squares H estimate on all of the inliers
- } Iterate multiple times

Ransac seleziona il numero minimo di punti (in base a se vogliamo la traslazione, anche la rotazione...), decidiamo una soglia di tolleranza per verificare la consistenza geometrica, e vediamo quanti punti seguono la trasformazione.

Alla fine restituisce la trasformazione con la quale sono d'accordo il più alto numero di punti.



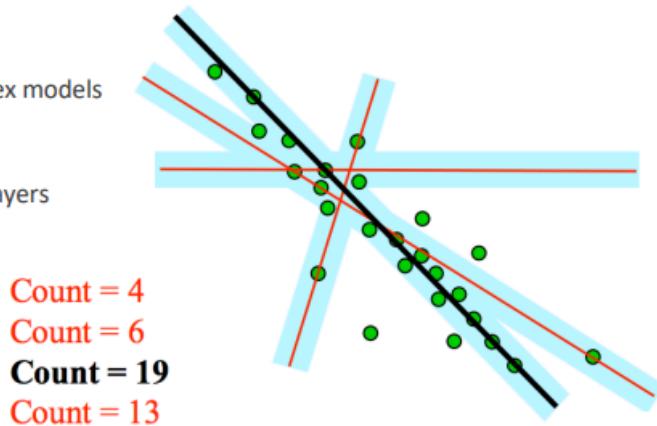
Prendiamo 2 punti, mettiamo una tolleranza, e vediamo quanti altri punti sono in questa tolleranza, troviamo per esempio questi 4 punti.



Poi prendiamo altri 2 punti random, e nella tolleranza ora troviamo 19 punti, quindi ci dimentichiamo della linea precedente e salviamo questa.

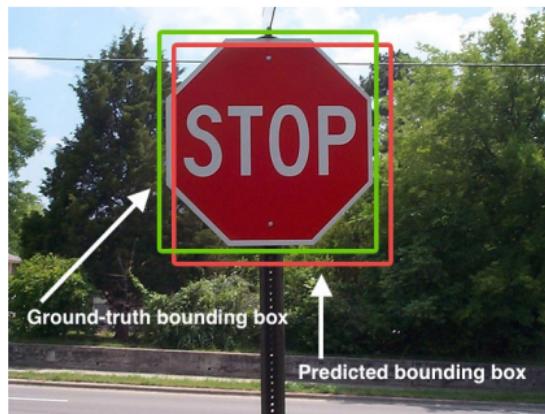
Ripetiamo fino al massimo numero di iterazioni, e troveremo la nostra trasformazione migliore. Avremo anche informazione sugli outliers. In questo modo il modello trovato non è influenzato dagli outliers.

- Choose the model with the highest number of inlayers.
- Simple example with lines.
- The same algorithm is used with more complex models (e.g. to handle affine transformations).
- Robust even if the majority of points are outliers



Object detection: evaluation metrics

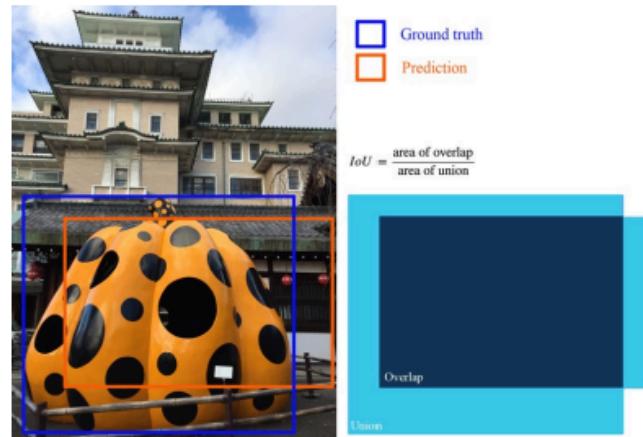
Come possiamo valutare la nostra object detection?



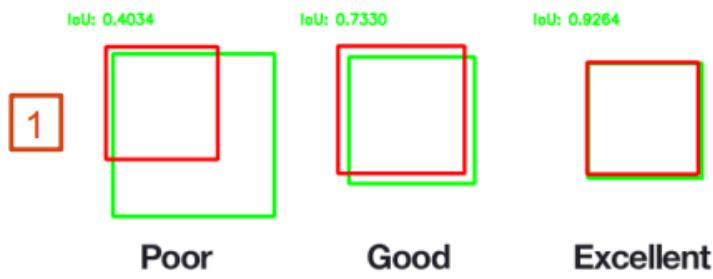
Come possiamo misurare la performance di questo bounding box?

Potremmo usare l'**intersection over union**, dividendo l'area di intersezione dei due bounding boxes per l'area della loro unione.

$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$



Avremo quindi valori più alti per risultati migliori.



1 Set a threshold (e.g. 0.5)

2 3 $TP = \text{True positive}$
 $TN = \text{True negative}$
 $FP = \text{False positive}$
 $FN = \text{False negative}$

3 $Precision = \frac{TP}{TP + FP}$

4 $Recall = \frac{TP}{TP + FN}$

Come possiamo calcolare la precision e recall? Fissiamo un threshold, di modo che ciascun risultato di IoU sarà un true positive, true negative, false positive o false negative.

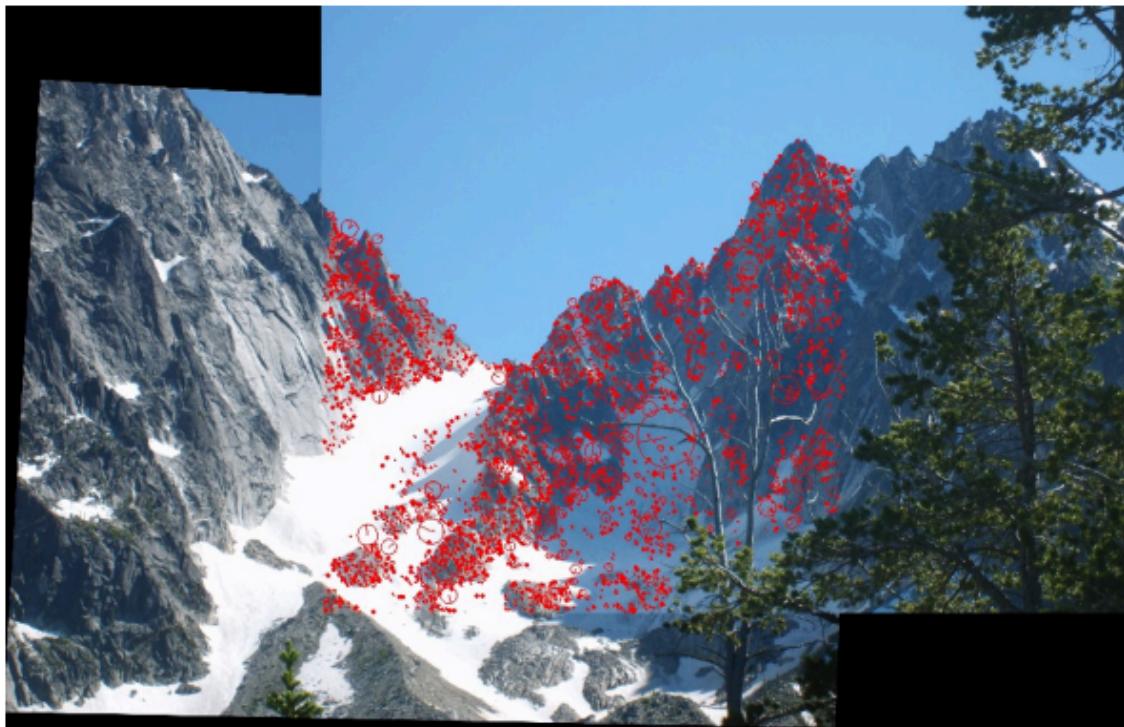
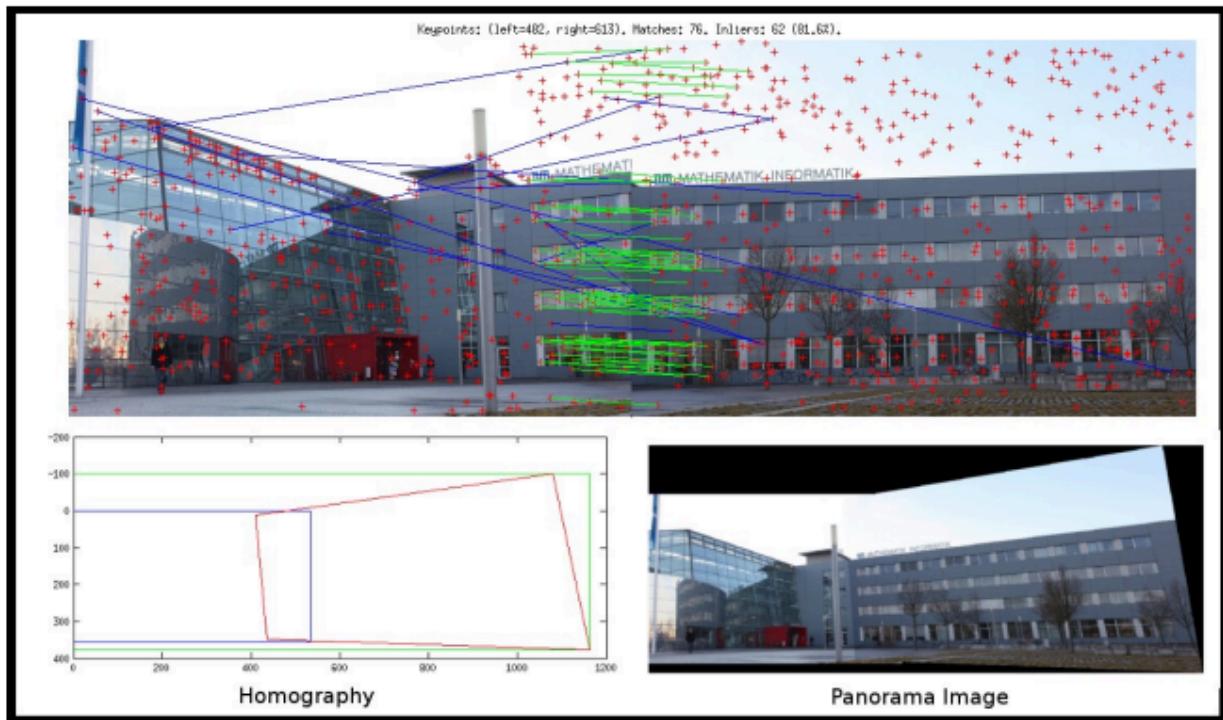
Di solito non si usa un solo threshold, ma più di uno, trovando una curva che ci dice come la precision e la recall cambiano al variare del threshold.

Applications

Panorama stitching

Abbiamo 2 immagini, e vogliamo unirle per fare un'immagine panoramica.

L'homography è una trasformazione fatta per allineare il contenuto dell'immagine.



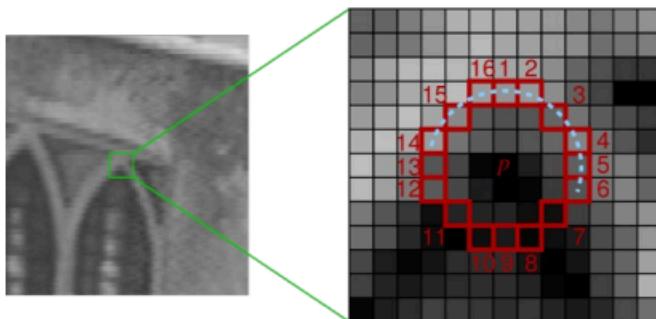
In questo caso non abbiamo più le finestre che si possono allineare facilmente.

Video stabilization

Possiamo applicare questo metodo al secondo frame, di modo da allineararlo al primo e rimuovere il tremolio.

Fin ora abbiamo visto una descrizione degli algoritmi SIFT che fino a qualche anno fa erano patented e quindi sono nate delle variazioni.

FAST – another keypoint detector



- FAST – Features from accelerated segment test
- For every pixel check the value of the neighbors
- If the value of at least N neighbors is less than the value of the center pixel, this is a corner

Tips for faster computation:

- First compare values of pixels 1,5,9,13
- At least 3 of these pixel values must be under the value of the central pixel
- If not the keypoint candidate is discarded
- Else everyone of the 16 neighbors is checked

Dense SIFT

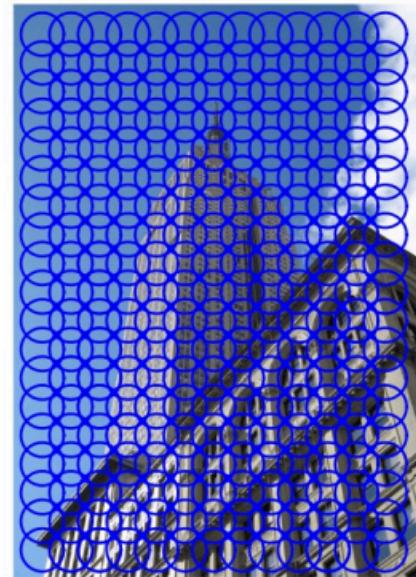
- Instead of computing keypoints use a fixed grid
- Every point in the fixed grid is described using the SIFT descriptor

PRO:

- Faster to compute (no keypoint detection)
- Can work with smooth surfaces (few corners)

CONS:

- Corners are usually more distinctive (reliable)
- Slower keypoint comparison (usually a lot more points to be compared)



L'idea è di piazzare i keypoints in un grid regolare, e calcolare il SIFT al centro di ciascuno. Quindi skippiamo la fase di trovare i keypoints, e troviamo una rappresentazione regolare.

Color descriptors

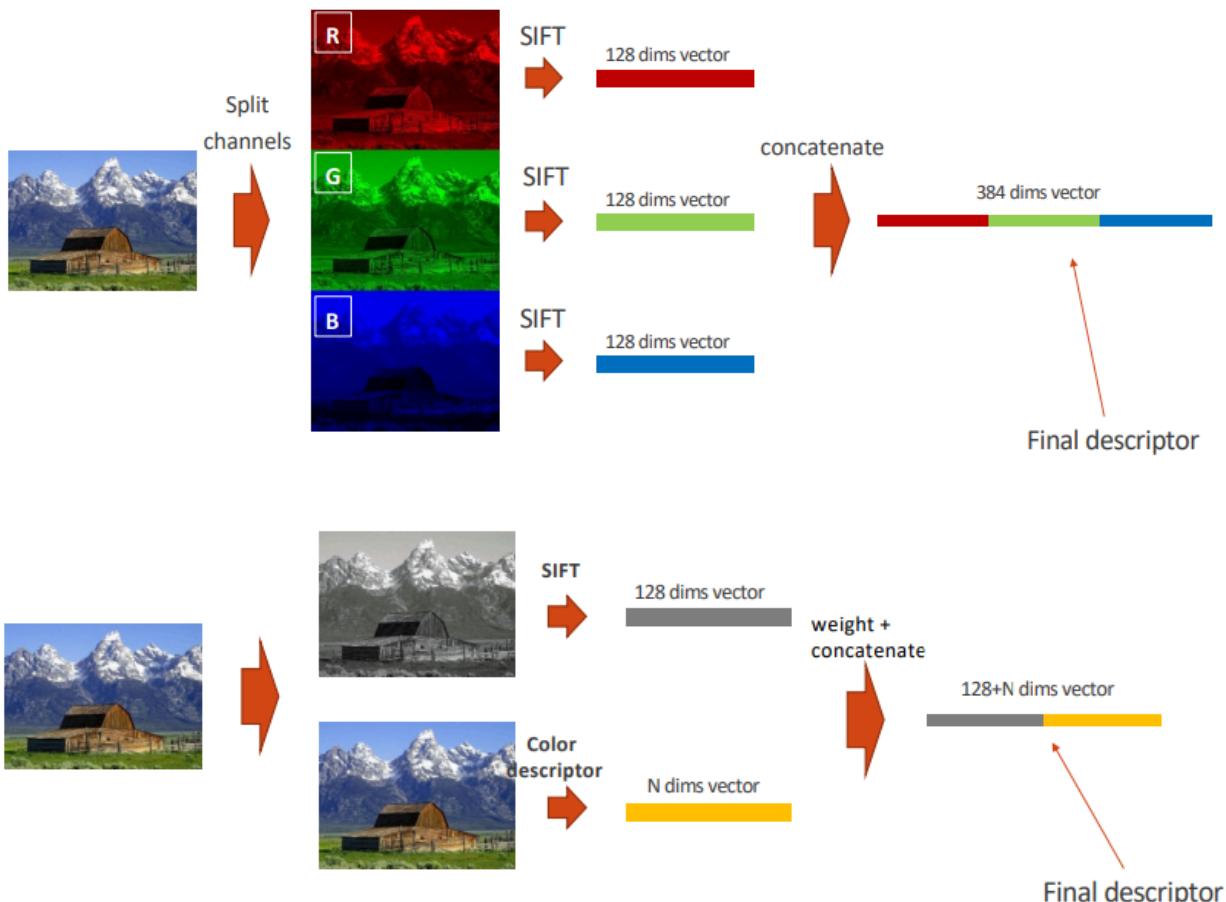
- SIFT does not use color information
- It just works on grayscale images
- Some objects categories need color information to be distinguished
- Two ways of mixing the color and shape information:
 - Late Fusion approaches
 - Early Fusion approaches



Supermarket Milan Dataset

I descrittori SIFT lavorano su grayscale, possiamo quindi aggiungere anche il colore per estenderlo.

Early Fusion approaches



- N depends on the type of descriptor
- Example: simple histogram on color-space
- Before the concatenation of color and shape descriptor usually the two components are weighted
- Possibility to give more importance (higher weight) to color or shape information

Si applica SIFT indipendentemente su ciascun canale RGB, trovando un descrittore che è 3 volte l'originale.

Bag of Visual-Words

L'idea è che abbiamo molto testo che vogliamo descrivere in un modo compatto.

President George W. Bush Speech in 2001
ADDRESS TO THE JOINT SESSION OF THE 107TH CONGRESS
 UNITED STATES CAPITOL
 WASHINGTON, D.C. SEPTEMBER 20, 2001

Mr. Speaker, Mr. President Pro Tempore, members of Congress, and fellow Americans: In the normal course of events, Presidents come to this chamber to report on the state of the Union. Tonight, no such report is needed. It has already been delivered by the American people. We have seen it in the courage of passengers, who rushed terrorists to save others on the ground — passengers like an exceptional man named Todd Beamer. And would you please help me to welcome his wife, Lisa Beamer, here tonight. We have seen the state of our Union in the endurance of rescuers, working past exhaustion. We have seen the unfurling of flags, the lighting of candles, the giving of blood, the saying of prayers — in English, Hebrew, and Arabic. We have seen the decency of a loving and giving people who have made the grief of strangers their own. My fellow citizens, for the last nine days, the entire world has seen for itself the state of our Union — and it is strong. Tonight we are a country awakened to danger and called to defend freedom. Our grief has turned to anger, and anger to resolution. Whether we bring our enemies to justice, or bring justice to our enemies, justice will be done. I thank the Congress for its leadership at such an important time. All of America was touched on the evening of the tragedy to see Republicans and Democrats joined together on the steps of this Capitol, singing "God Bless America." And you did more than sing; you acted, by delivering \$40 billion to rebuild our communities and meet the needs of our military. Speaker Hastert, Minority Leader Gephardt, Majority Leader Daschle and Senator Lott, I thank you for your friendship, for your leadership and for your service to our country. ... (to be continued)

- We want to represent the topic of the document in a compact way
- Orderless document representation: frequencies of words from a dictionary Salton & McGill (1983)

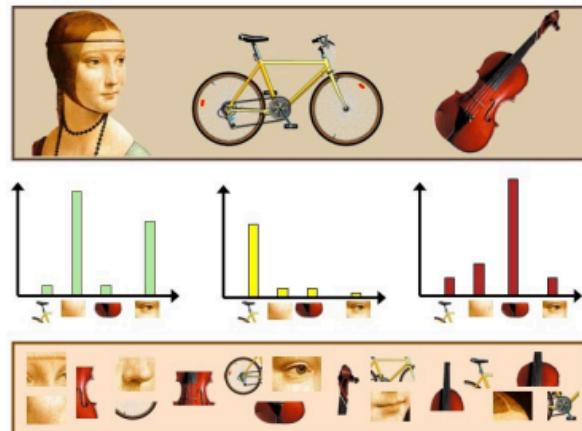
Abbiamo un vocabolario, contiamo quante volte queste parole appaiono nel testo. Usando questo, possiamo classificare un modello.

L'idea per le immagini è simile, decomponiamo l'immagine in words, contiamo quante volte compaiono, e usiamo quella distribuzione per classificare.

- Fei Fei et al. [1]
- From text analysis
- Document representation as a bag of important keywords
- An object is represented as a bag of visual words (patches)
- Allows the recognition of a large collection of objects



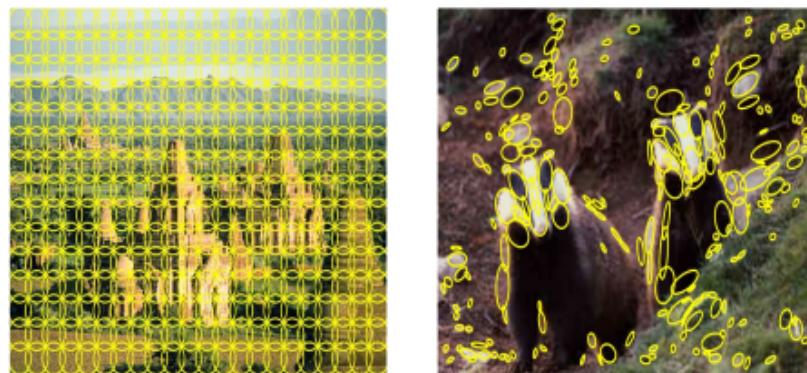
- a. Extract local features
- b. Learn "visual vocabulary"
- c. Quantize local features using visual vocabulary
- d. Represent images by frequencies of "visual words"



Per ogni immagine abbiamo una distribuzione (istogramma) rispetto alle 4 features.

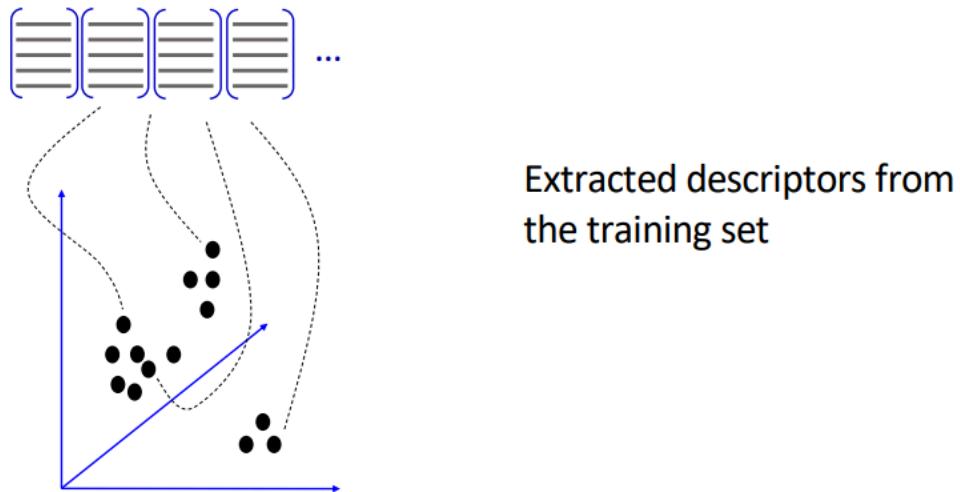
Local features extraction

- Sample patches and extract descriptors

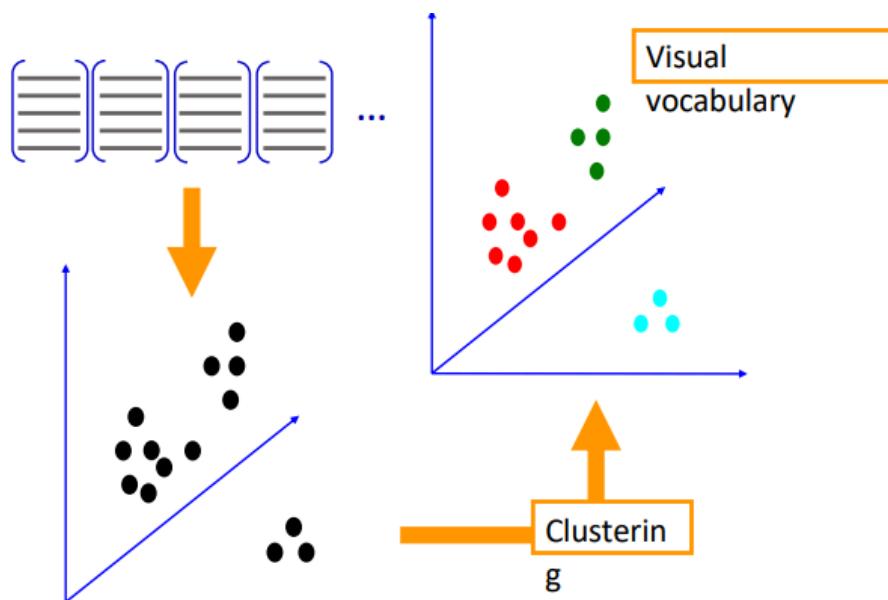


Usiamo per esempio il dense SIFT per estrarre i descriptors.

L'immagine ci darà dei keypoint descriptors, ognuno dei quali sarà lungo 128. Ogni immagine ci darà una serie di questi.



Andiamo quindi a mettere insieme questi descriptors nel grafico e poi andiamo ad applicare clustering in questo spazio a 128 dimensioni.



Selezioniamo come "visual words" il centroide di ciascun cluster.

Quando abbiamo il vocabolario, torniamo all'immagine e contiamo quante occorrenze abbiamo, facciamo una distribuzione su un'istogramma.

Esempio di visual vocabulary:



Per ciascuna immagine calcoliamo la dense SIFT, troviamo quelle a destra, quindi applichiamo clustering. Troviamo words e tutte quelle che fanno parte dello stesso cluster.

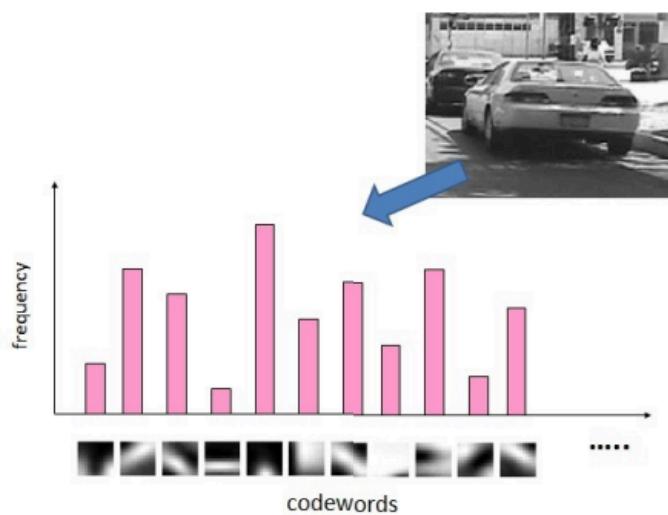
Questa è la fase di training

Cosa succede quando abbiamo una nuova immagine di test?

Calcoliamo la dense SIFT representation ad ogni posizione, proiettiamo quest'informazione nello spazio che contiene i clusters, e in base a di che cluster fa parte, sappiamo che tipo di feature è.

Test:

1. Extract descriptors from the image
2. For each descriptor extracted compute its nearest neighbor in the dictionary
3. Build a histogram of length k where the i^{th} value is the frequency of the i^{th} dictionary word
4. Classify the histogram with a classifier (e.g. Nearest Neighbor, SVM or Naive Bayes)



Questo approccio potrebbe essere usato anche per object detection, nello specifico per oggetti che cambiano forma.

Works with **Deformable Objects**!
Example: tracking

