Lecture 22:

Introduction to Image Search

Visual Computing Systems
CMU 15-869, Fall 2013

Are these images similar?

Pixel differences

Pixel differences

Pixel differences

Are these two web pages similar?

Visual Computing Systems

CMU 15-869 | Fall 2013

Visual computing tasks such as 2D/3D graphics, image processing, and image understanding are important responsibilities of modern computer systems ranging from sensor-rich smart phones to large datacenters. These workloads demand exceptional system efficiency and this course examines the key ideas, techniques, and challenges associated with the design of parallel systems for visual computing applications. This course is intended for graduate-level students interested in architecting efficient future graphics and image processing platforms and for students seeking to develop scalable algorithms for these platforms.

Course Description, Logistics, and Details

When We Meet

Mon/Wed 12:00 - 1:20pm (GHC 4303)
Instructor: **Kayvon Fatahalian**

Schedule

Part I: Implementing and Scheduling the Real-Time Graphics Pipeline

Sep 9 **Course Introduction + The Real-Time Graphics Pipeline**
(real-time rendering from a systems perspective)

Sep 11 **Graphics Pipeline Parallelization and Scheduling**
(characteristics of the pipeline workload, Molnar's scheduling taxonomy, trade-offs between parallelism, communication, and locality)

Sep 13 **Geometry Processing and Scheduling Under Data Amplification**
(clipping, tessellation, parallel scheduling challenges of tessellation)

Sept 16 **Visibility: Rasterization and Occlusion**
(algorithms and their fixed-function implementation, occlusion culling, anti-aliasing, frame-buffer compression)

Sep 18 **Texturing**
(anti-aliasing using the mip-map, hardware texture unit implementation, prefetching and caching policies)

Sep 23 **Texturing Part II: Texture Compression**
(hardware-friendly texture compression techniques)

Parallel Computer Architecture and Programming (CMU 15-418)

From smart phones, to multi-core CPUs and GPUs, to the world's largest supercomputers and web sites, parallel processing is ubiquitous in modern computing. The goal of this course is to provide a deep understanding of the fundamental principles and engineering trade-offs involved in designing modern parallel computing systems as well as to teach parallel programming techniques necessary to effectively utilize these machines. Because writing good parallel programs requires an understanding of key machine performance characteristics, this course will cover both parallel hardware and software design.

[Our Self-Made Online Reference]

[Policies, Logistics, and Details]

When We Meet

Tues/Thurs 9:00 - 10:20am
Baker Hall A51 (Giant Eagle Auditorium)
Instructor: **Kayvon Fatahalian**

Spring 2013 Schedule

Jan 15 **Why Parallelism?**

Jan 17 **A Modern Multi-Core Processor: Forms of Parallelism + Understanding Latency and BW**
Assignment 1 out

Jan 22 **Parallel Programming Models and Their Corresponding HW/SW Implementations**

Jan 24 **Parallel Programming Basics (the parallelization thought process)**
Assignment 1 due

Jan 29 **GPU Architecture and CUDA Programming**
Assignment 2 out

Jan 31 **Performance Optimization I: Work Distribution**

Feb 5 **Performance Optimization II: Locality, Communication, and Contention**

Feb 7 **Parallel Application Case Studies**

Feb 12 **Workload-Driven Performance Evaluation**
Assignment 2 due
Assignment 3 out

Another example: which web page is most similar to the search query...

Are these two web pages similar?

Visual Computing Systems
CMU 15-869 | Fall 2013

Visual computing tasks such as 2D/3D graphics, image processing, and image understanding are important responsibilities of modern computer systems ranging from sensor-rich smart phones to large datacenters. These workloads demand exceptional system efficiency and this course examines the key ideas, techniques, and challenges associated with the design of parallel systems for visual computing applications. This course is intended for graduate-level students interested in architecting efficient future graphics and image processing platforms and for students seeking to develop scalable algorithms for these platforms.

Course Description, Logistics, and Details

When We Meet

Mon/Wed 12:00 - 1:20pm (GHC 4303)
Instructor: **Kayvon Fatahalian**

Schedule

Part I: Implementing and Scheduling the Real-Time Graphics Pipeline

Sep 9	**Course Introduction + The Real-Time Graphics Pipeline** (real-time rendering from a systems perspective)
Sep 11	**Graphics Pipeline Parallelization and Scheduling** (characteristics of the pipeline workload, Molnar's scheduling taxonomy, trade-offs between parallelism, communication, and locality)
Sep 13	**Geometry Processing and Scheduling Under Data Amplification** (clipping, tessellation, parallel scheduling challenges of tessellation)
Sept 16	**Visibility: Rasterization and Occlusion** (algorithms and their fixed-function implementation, occlusion culling, anti-aliasing, frame-buffer compression)
Sep 18	**Texturing** (anti-aliasing using the mip-map, hardware texture unit implementation, prefetching and caching policies)
Sep 23	**Texturing Part II: Texture Compression** (hardware-friendly texture compression techniques)

Parallel Computer Architecture and Programming (CMU 15-418)

From smart phones, to multi-core CPUs and GPUs, to the world's largest supercomputers and web sites, parallel processing is ubiquitous in modern computing. The goal of this course is to provide a deep understanding of the fundamental principles and engineering trade-offs involved in designing modern parallel computing systems as well as to teach parallel programming techniques necessary to effectively utilize these machines. Because writing good parallel programs requires an understanding of key machine performance characteristics, this course will cover both parallel hardware and software design.

[Our Self-Made Online Reference]

[Policies, Logistics, and Details]

When We Meet

Tues/Thurs 9:00 - 10:20am
Baker Hall A51 (Giant Eagle Auditorium)
Instructor: **Kayvon Fatahalian**

Spring 2013 Schedule

Jan 15	**Why Parallelism?**
Jan 17	**A Modern Multi-Core Processor: Forms of Parallelism + Understanding Latency and BW** Assignment 1 out
Jan 22	**Parallel Programming Models and Their Corresponding HW/SW Implementations**
Jan 24	**Parallel Programming Basics (the parallelization thought process)** Assignment 1 due
Jan 29	**GPU Architecture and CUDA Programming** Assignment 2 out
Jan 31	**Performance Optimization I: Work Distribution**
Feb 5	**Performance Optimization II: Locality, Communication, and Contention**
Feb 7	**Parallel Application Case Studies**
Feb 12	**Workload-Driven Performance Evaluation** Assignment 2 due Assignment 3 out

Another example: which web page is most similar to the search query...

Google cmu visual computing systems fall 2013

Google cmu visual computing systems fall 2013

Web Images Maps Shopping More Search tools

About 262,000 results (0.89 seconds)

Kayvon Fatahalian - School of Computer Science - Carnegie Mellon ...
www.cs.cmu.edu/~kayvonf/
I am teaching 15-869: **VISUAL COMPUTING SYSTEMS** in the **Fall 2013** semester . 15-418/15-618: Parallel Computer Architecture and Programming (**Spring ...**
You've visited this page many times. Last visit: 11/7/13

[PDF] **Visual Computing Systems CMU 15-869, Fall 2013 Lecture 1:**
graphics.cs.cmu.edu/courses/.../**fall2013**content/.../gfxpipeline_slides.pdf
CMU 15-869, **Fall 2013**. Many applications driving the need for high efficiency **computing** involve **visualcomputing** tasks.

Visual Computing Systems : 15-869 Fall 2013 - Carnegie Mellon ...
kip.graphics.cs.cmu.edu/
Visual computing tasks such as 2D/3D graphics, image processing, and image understanding are important responsibilities of modern **computer systems ...**

Naive solution

Given query words: *w1 and w2*

for all documents *d* in database:

score(*d, w1, w2*) = number of occurrences of *w1 and w2* in *d*

Return top 20 results in sorted order based on score

- **Improving search:**
  - **Improve score function (return better results *)**
  - **Improve query execution time: above solution is O(N)**

*** In retrieval community: the quality of the returned results is referred to as the "performance" of the algorithm. "An algorithm performs better if it returns better results". Clearly, using the term "performance" in this way going to cause problems in this class.**

Index

To simplify, let:

`score(d,w1,w2) = 1 if d contains w1 and w2, 0 otherwise`

```
Document 0: Kayvon is teaching 15-869 today. Yay 15-869!
Document 1: 15-869 is awesome, Kayvon claims.
Document 2: Kayvon is occasionally awesome.
```

Index:

- Kayvon: 0, 1, 2
- is: 0, 1, 2
- teaching: 0
- 15-869: 0, 1
- yay: 0
- thinks: 1
- today: 0
- awesome: 1, 2
- occasionally: 2

Query: kayvon awesome

Partial result set:

kayvon: {0, 1,,2}
awesome: {1, 2}

Result:

{0,1,2} ∩ {1, 2} = {1,2}

Full inverted index

Inverted index contains one entry per word occurrence:

```
score(d,w1,w2) =
     number of occurrences of w1 or w2, if d contains w1 and w2
     0 otherwise
```

```
Document 0: Kayvon is teaching 15-869 today. Yay 15-869!
Document 1: 15-869 is awesome, Kayvon claims.
Document 2: Kayvon is occasionally awesome.
```

Index:

- Kayvon: (0,0), (1,3) (2,0)
- is: (0,1), (1,1), (2,1)
- teaching: (0, 2)
- 15-869: (0,3), (0,6), (1, 0)
- yay: (0, 5)
- claims: (1,4)
- today: (0, 4)
- awesome: (1,2), (2,3)
- occasionally: (2,2)

Query: kayvon 15-869

Partial result set:

kayvon: {(0,0), (1,3), (2,0)}
15-869: {(0,3), (0,6), (1,0)}

Result:

{0,1,2} ∩ {0, 1} = {0,1}

Ranking:

0, 1

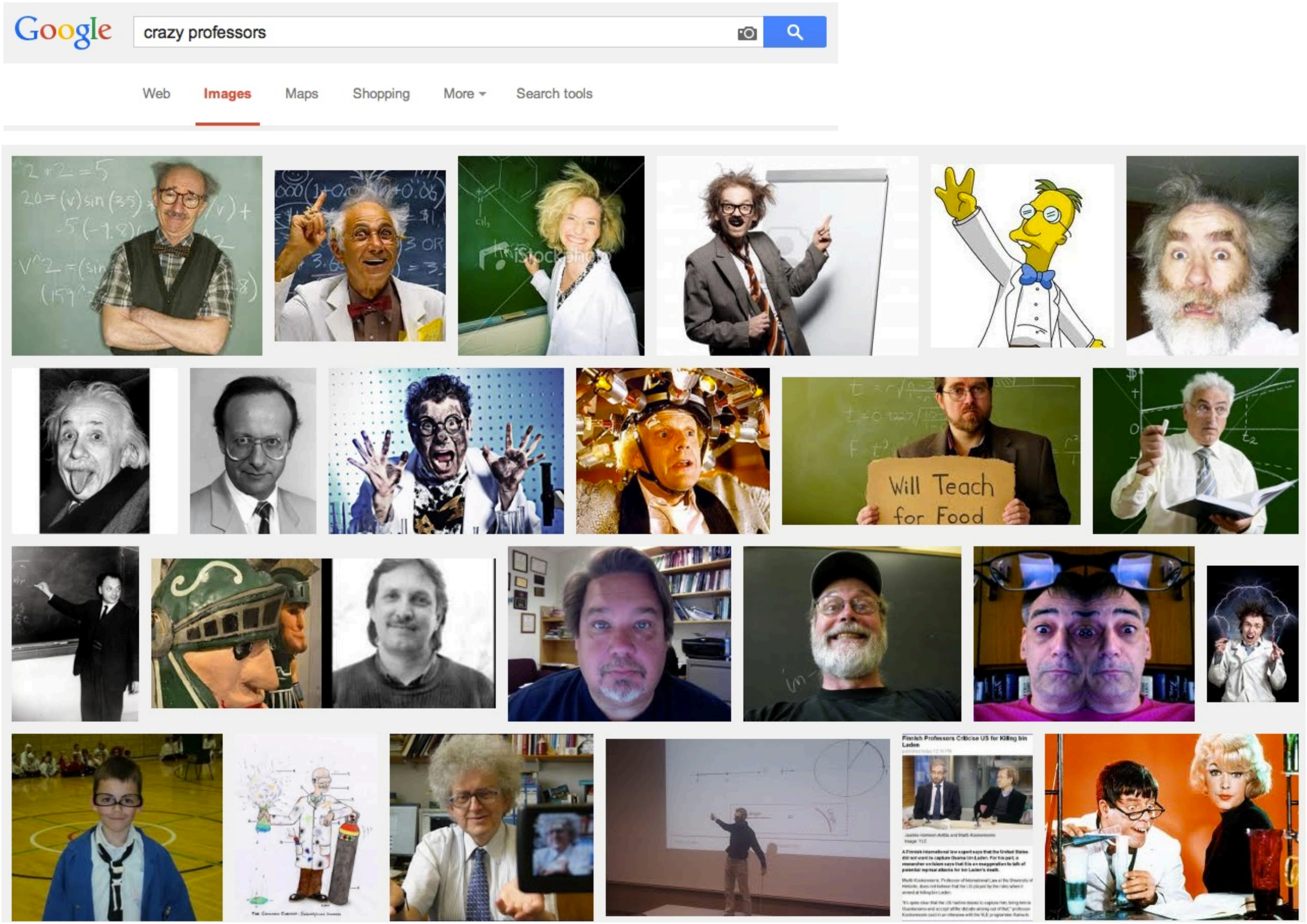
TF-IDF weighting

- TF: term frequency, the number of occurrences of a a word in a document
  - Measure of how relevant a document is for a given query word
- IDF: inverse document frequency
  - Measure of how discriminative a word is.
  - Depends on entire document collection
  - Idea: words that appear in most documents should influence score less

- tfidf_score(w, d, D) = tf(w,d) * idf(w, D)
  - tf(w,d) = number of occurrences of word w in document d
  - idf(w, D) = $\log \frac{|D|}{|\{d \in D : w \in d\}|}$ where D is the set of all documents

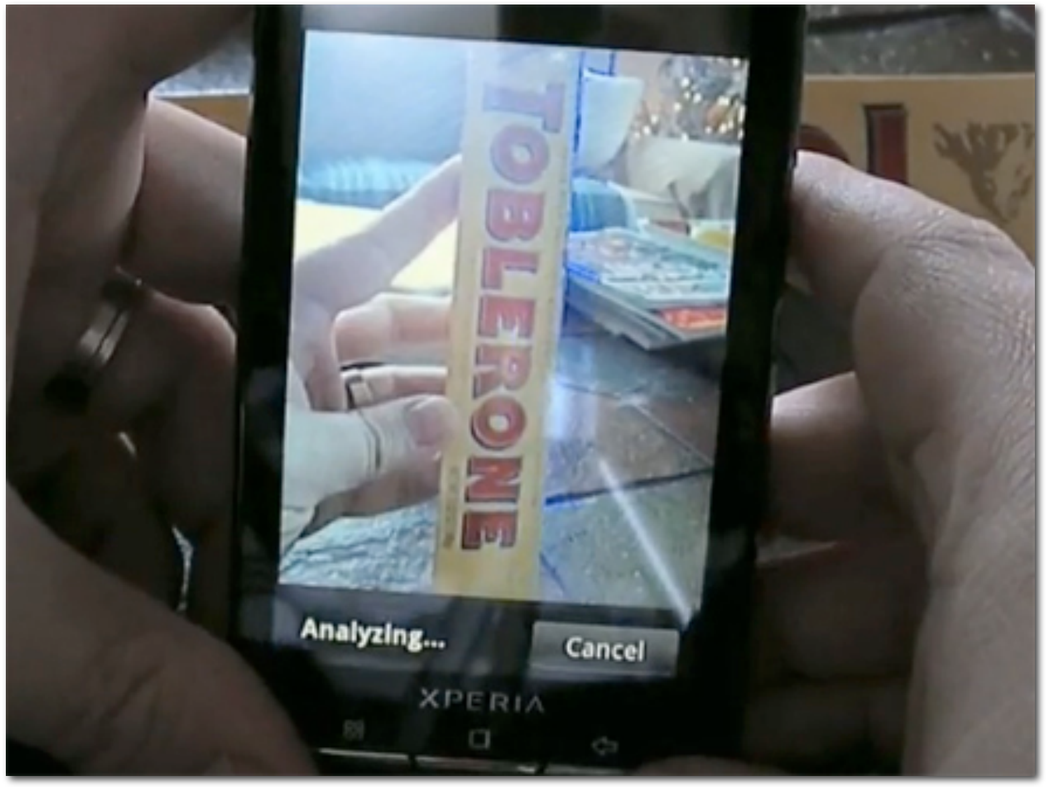
- Many variants on how to compute tf(w,d)
  - Binary: is word in document
  - Normalized frequency: number of occurrences normalized by document size (or most frequently occurring word)

Searching for images



Content-based image retrieval

- Take a photo, want to find webpages containing similar photos
- Take a photo, want to find information about its contents
- Take a photo, want to know what subject is



Text document retrieval

- Key idea is the breakdown into words
  - Documents that have the same words are likely to be similar
  - Words are a semantically meaningful granularity of text to latch on to

Content based image retrieval

- If we wanted to follow the text analogy, what are the words?
  - Pixels?
  - Blocks of pixels?
  - Descriptors/features computed from images?

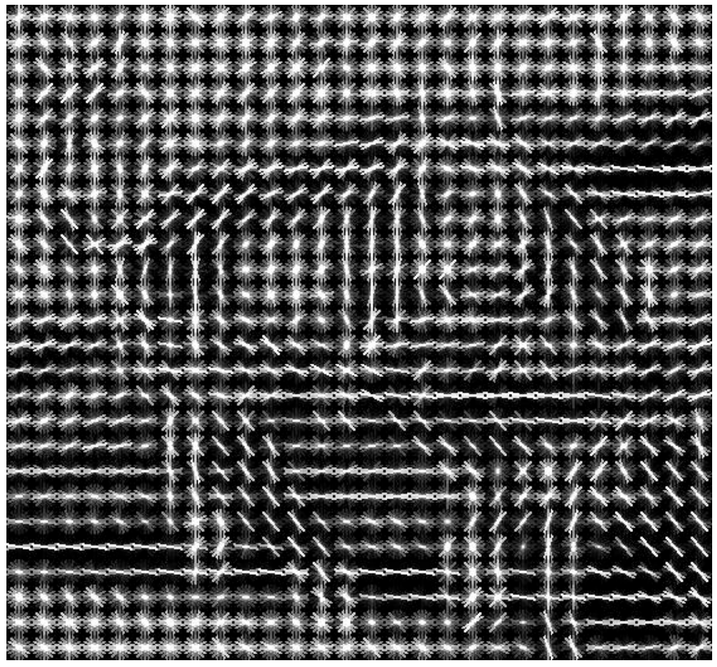
Correspondence

- Similarity is a problem of finding correspondences
  - Pictures with the same/similar objects
  - Pictures at the same place

- Want image "descriptors" such that are numerically similar descriptors correspond to meaningful similarities
  - e.g., invariant to noise, lighting, affine object transformation (rotation, translation, scale)
  - Distinctive… doesn't appear in every image

Histogram of oriented gradients (HOG) [Dalal and Triggs 05]

- Idea: local object appearance/shape is well characterized by distribution of local intensity gradients
- Gradient orientation is less sensitive to illumination change than gradient magnitude

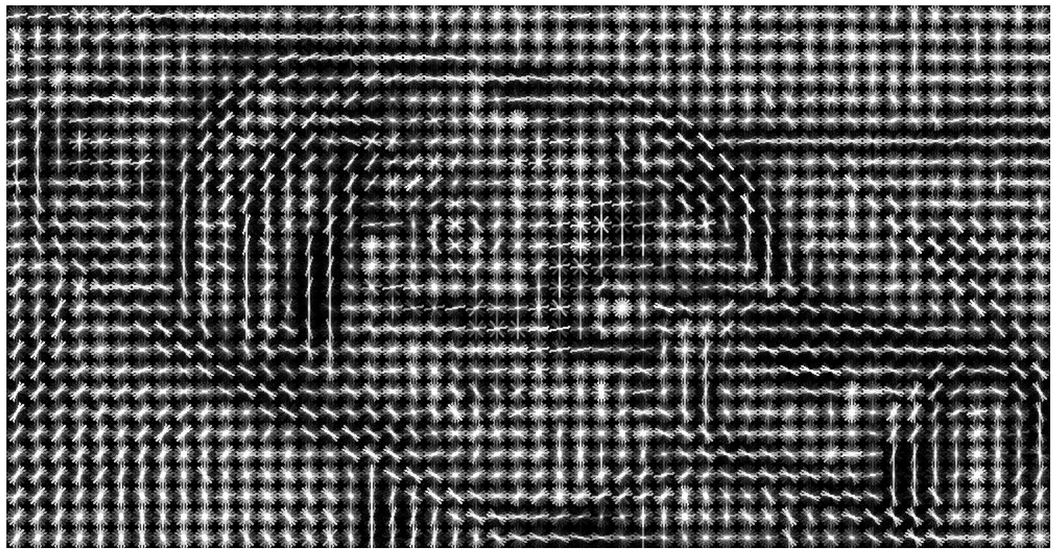


For each pixel *p* in block:

Compute gradient

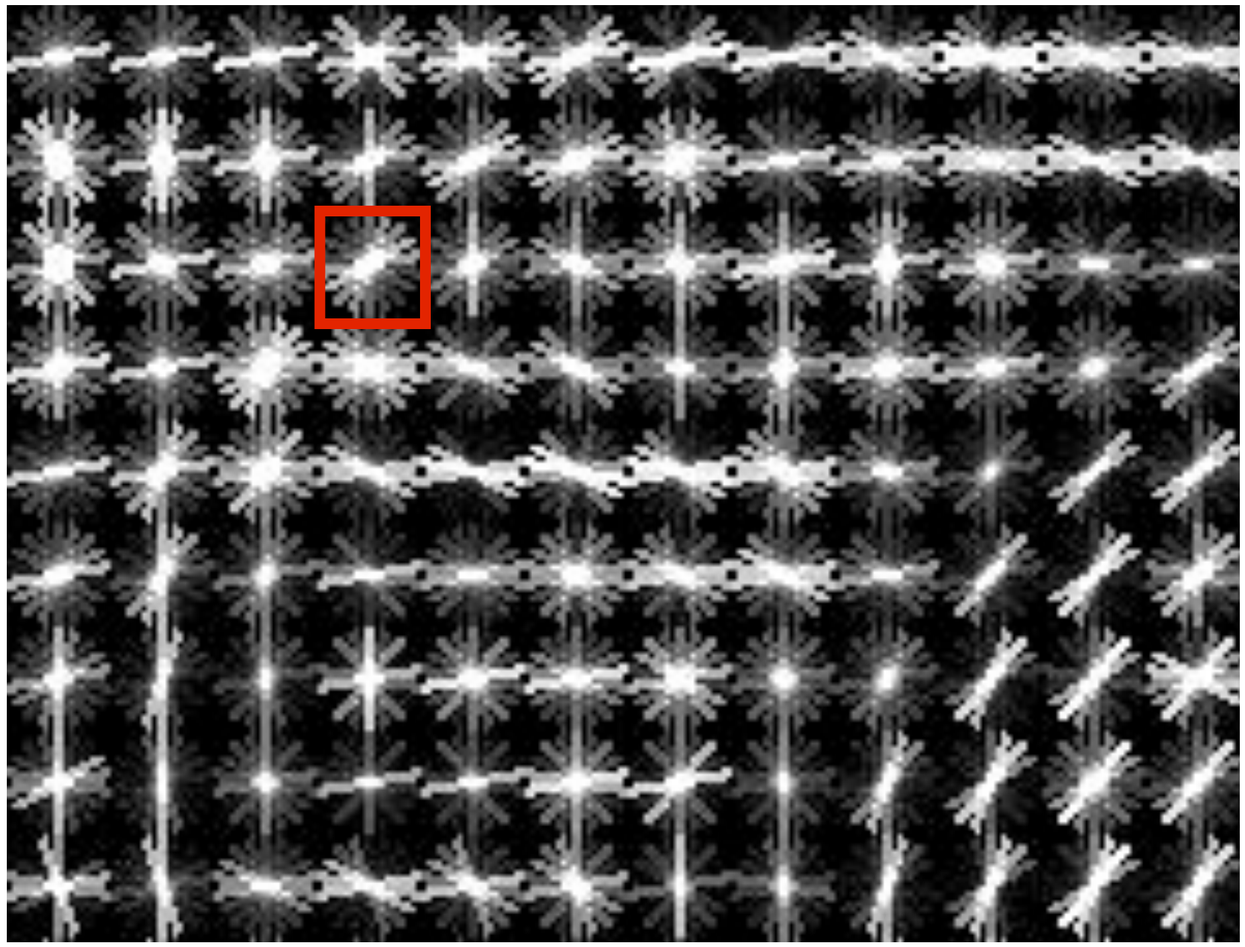
Add vote to histogram cell based on gradient orientation

(vote is weighted based on gradient magnitude and distance between p and block center)



[Image credit: Vondrick et al. ICCV 13]

HOG visualization close up

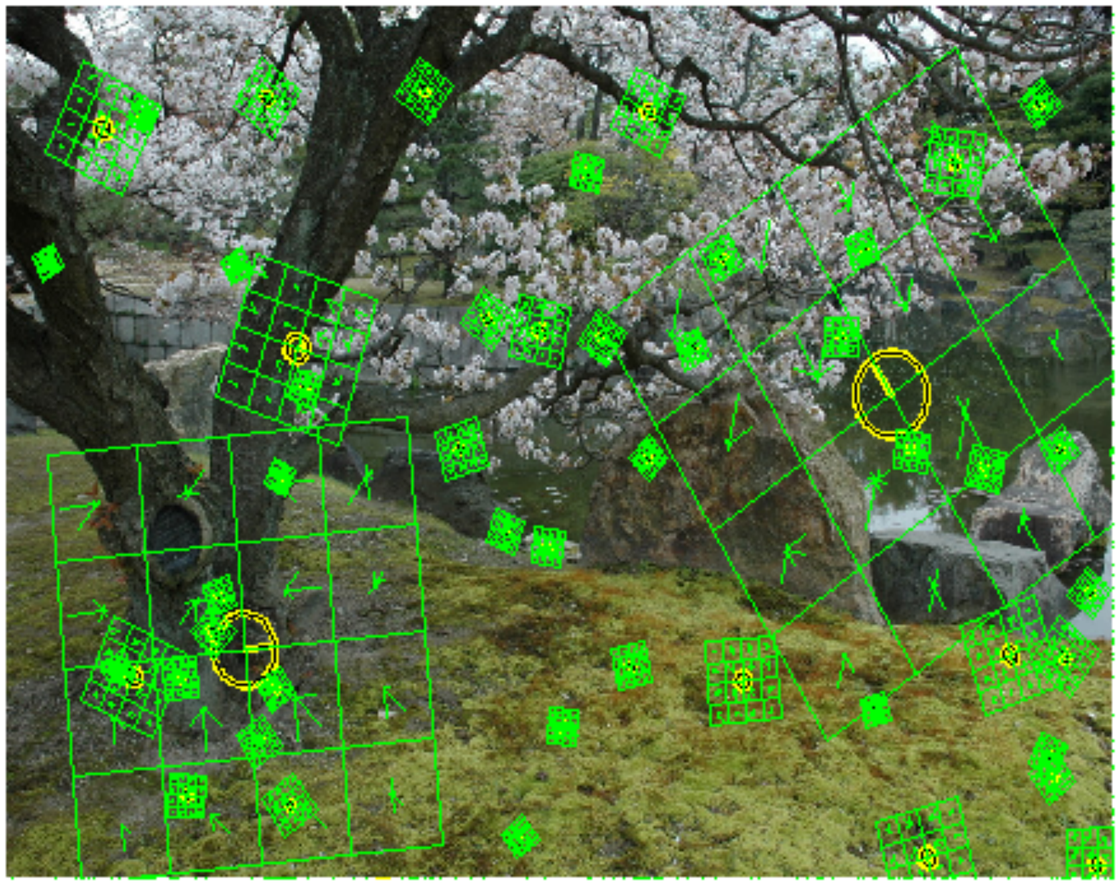


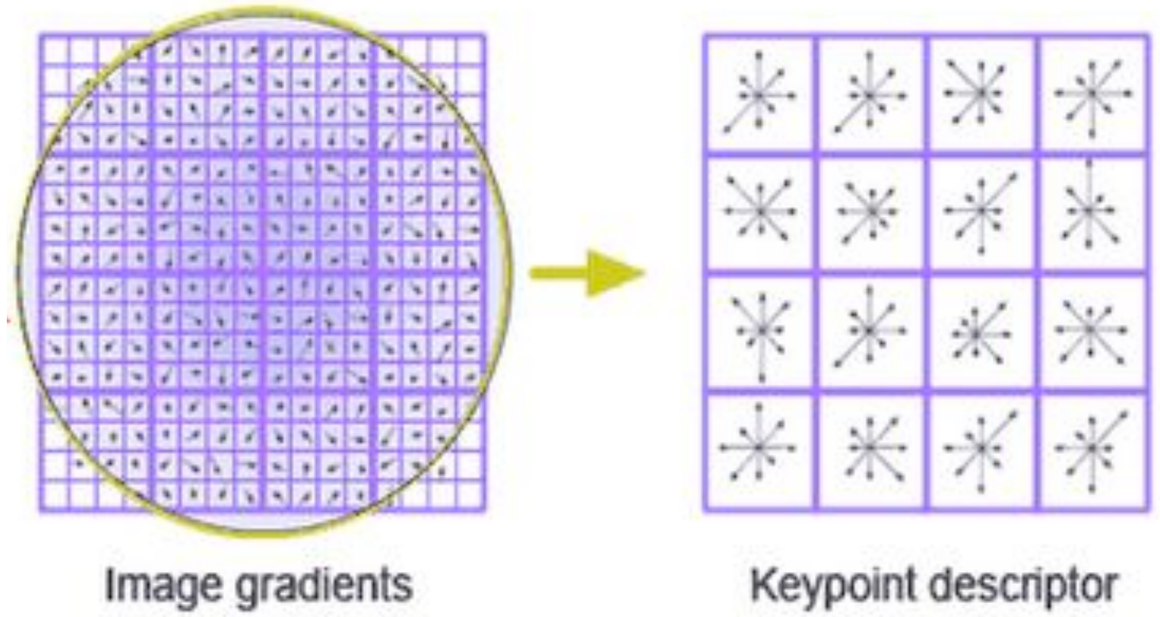
Visualizing magnitude of each histogram cell as a line

(Direction of line is at a right angle to the corresponding gradient orientation)

SIFT

- Interest-point-based, orientation of gradients descriptor
- Find interest points (location in image, scale, and orientation)
- Compute 128-element descriptor for interest points





Pool gradient samples from 4x4 window into 8-bin histogram

Stack 4x4 grid of histograms to get full descriptor

Figure credits:
R. Bandara, Codeproject
Chen, Kong, Oh, Sanan, Wohlberk 09

Aside: feature extraction workload

- Discussion: series of local image processing operations
  - Multi-scale: local convolution to create gaussian/laplacian pyramid (mip-map)
  - Keypoint identification: e.g.. corner detector: strong gradients in two directions
  - Gradient computation: local differences

Visual words

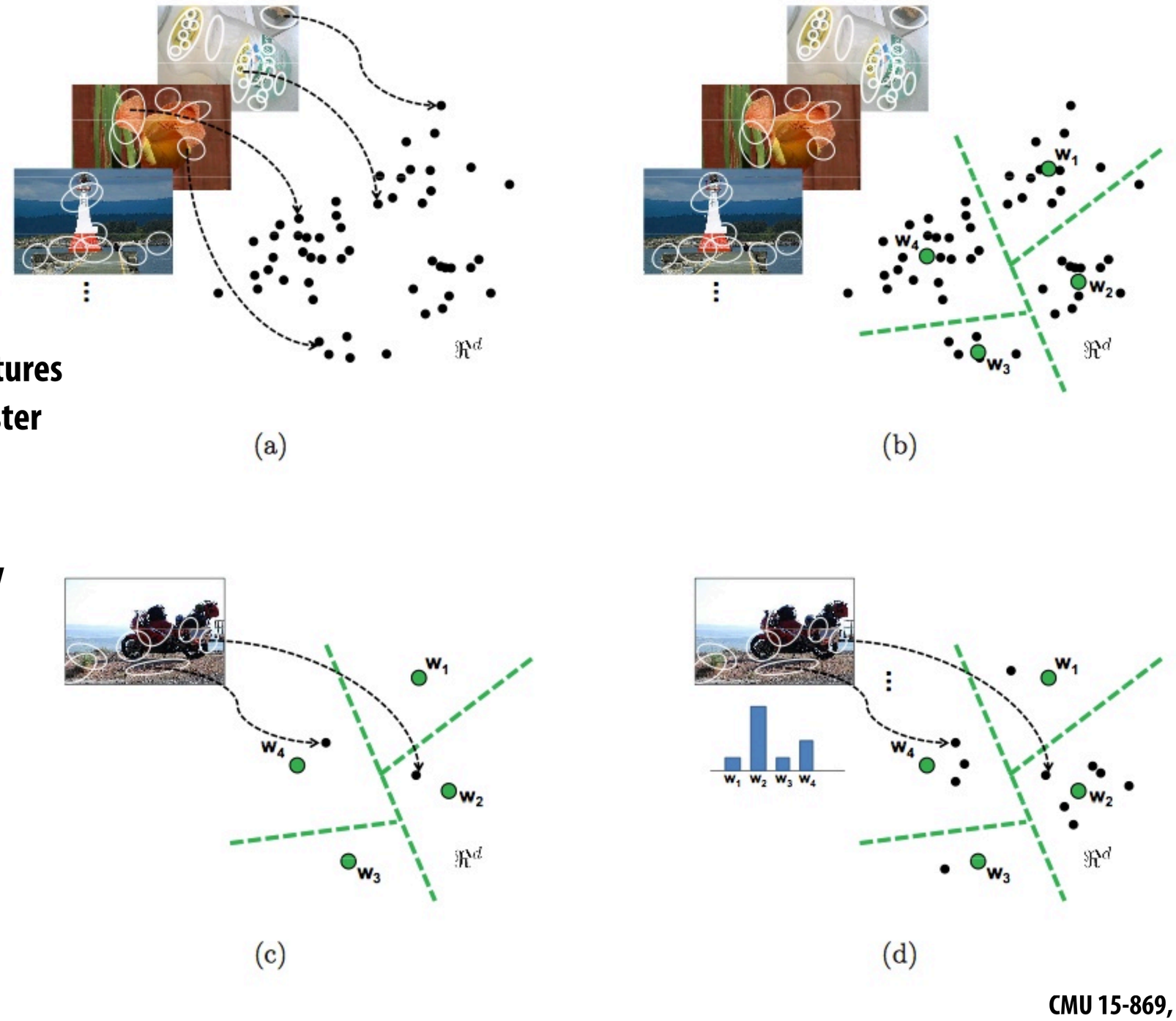
- Text document is made up of words (discrete values)
- Features are points in continuous high-dimensional space
- Construct visual words from features

(A) Features in images

(B) Compute vocabulary from all features by clustering: represent each cluster by mean (or median) feature

(C) Bin (discretize) image features by assignment to closest cluster.

(D) Image is now represented as a histogram of visual words!



Bag of words

- **Bag of words (BOW) descriptor:**
  - **Image descriptor is a histogram of word occurrences**
  - **Very sparse vector**

0	0	0	1	0	1	0	4	0	0	0	0	0	8	9	3	0	0	0	. . .

- **Given query image descriptor *q*, compute score for database image *d*:**
  - **Example: dot product of normalized query descriptor and DB image descriptor:**

$$\text{score}(q,d) = \frac{q \cdot d}{\|q\|\|d\|}$$

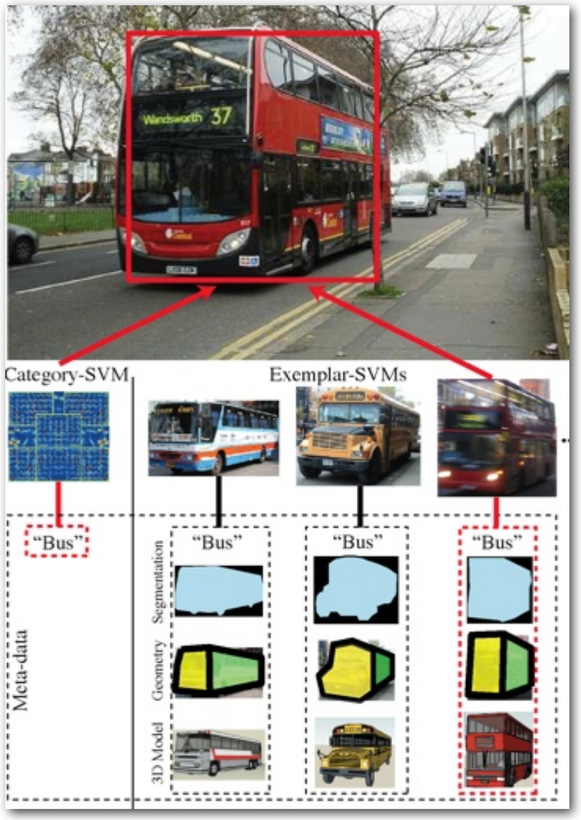
  - **Better: weight descriptor elements by visual word IDF values**
  - **Many alternatives:**
    - **e.g., histogram intersection: $\min(q_i, d_i)$ rather than product**

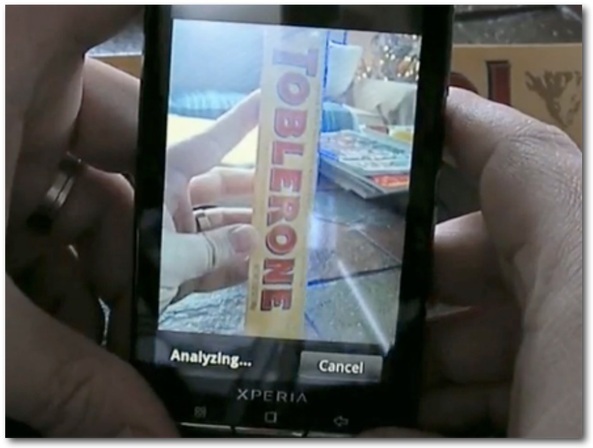
Summary

- **Image search using bag of words descriptors and an inverted index acceleration structure:**
  1. **Compute features for image collection**
  2. **Build vocabulary (visual words) via clustering features in collection**
  3. **Compute inverted index:**
     - **For each visual word, index stores list of images with word, plus the tf-idf weight for that word in that image:** `tfidf_score(w, d, D) = tf(w,d) * idf(w, D)`
  4. **For each query image:**
     - **Compute BOW descriptor**
     - **Use inverted index to find candidate set of similar images**
     - **Compute score between query and candidate images (e.g., dot product of descriptors)**
     - **Rank results by score**

Why image retrieval is important

Retrieval as building block for class of vision applications





Object Detection [Malisiewicz 11]

Movement prediction [Yuen 11]

Novelty Detection [Aghazadeh 11]

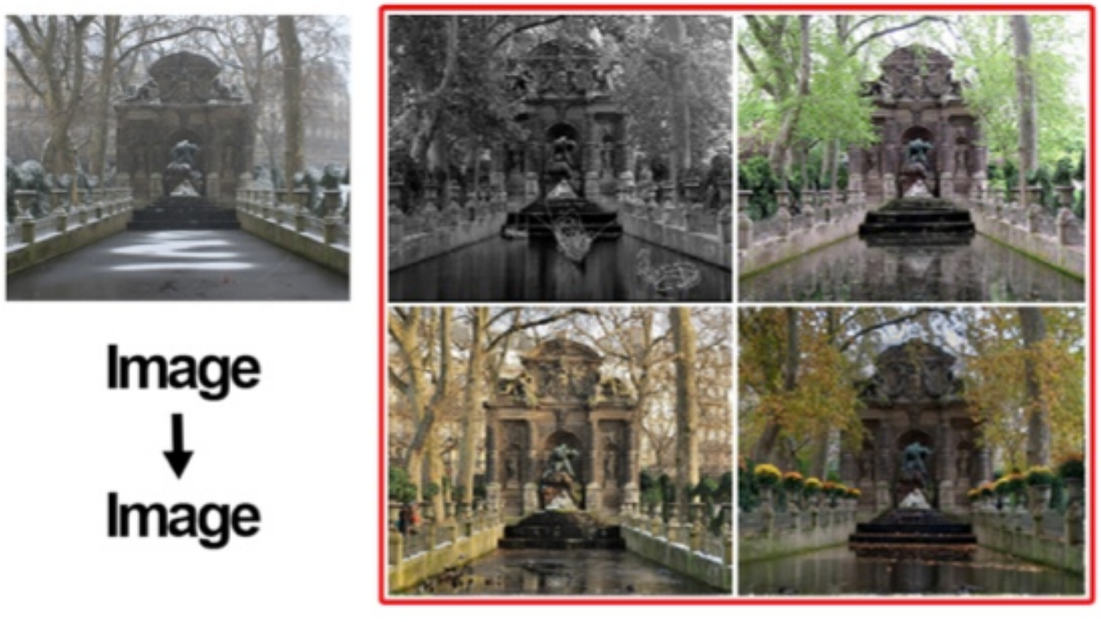


Image Matching [Shrivastava 11]