Local Difference Binary for Ultrafast and Distinctive Feature Description

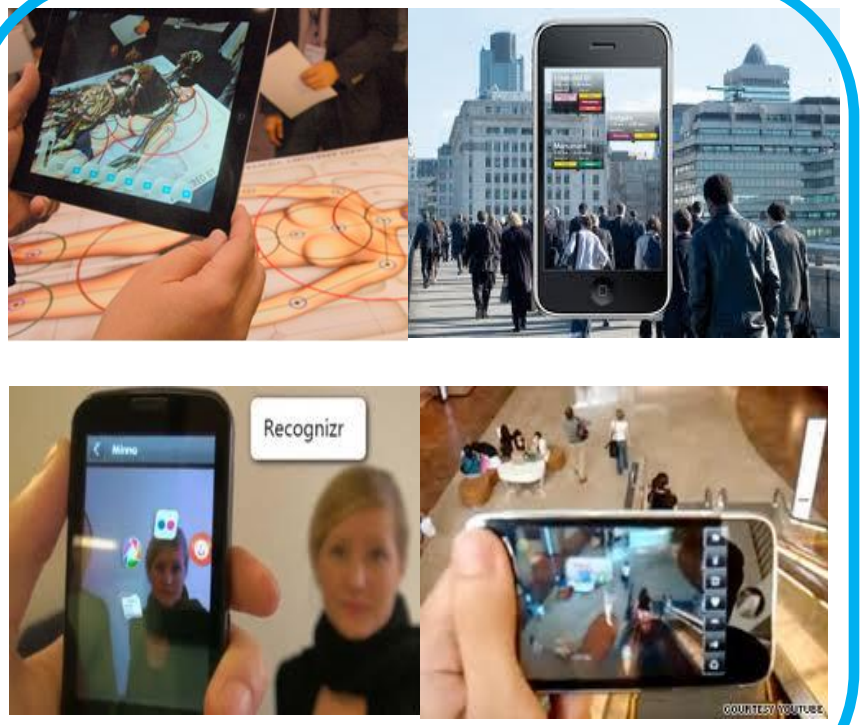
Xin Yang, K.-T. Tim Cheng

IEEE Trans. on Pattern Analysis and Machine Intelligence, 2014, January

*Source code has been released and published on ACM MM 2014 open source software competition

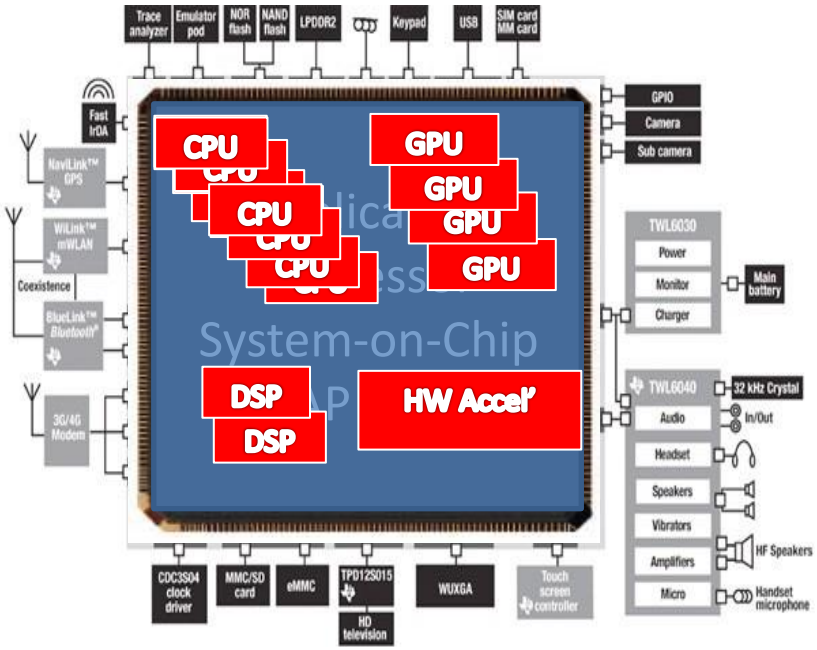
…, …

Mobile devices are growing incredibly fast these days



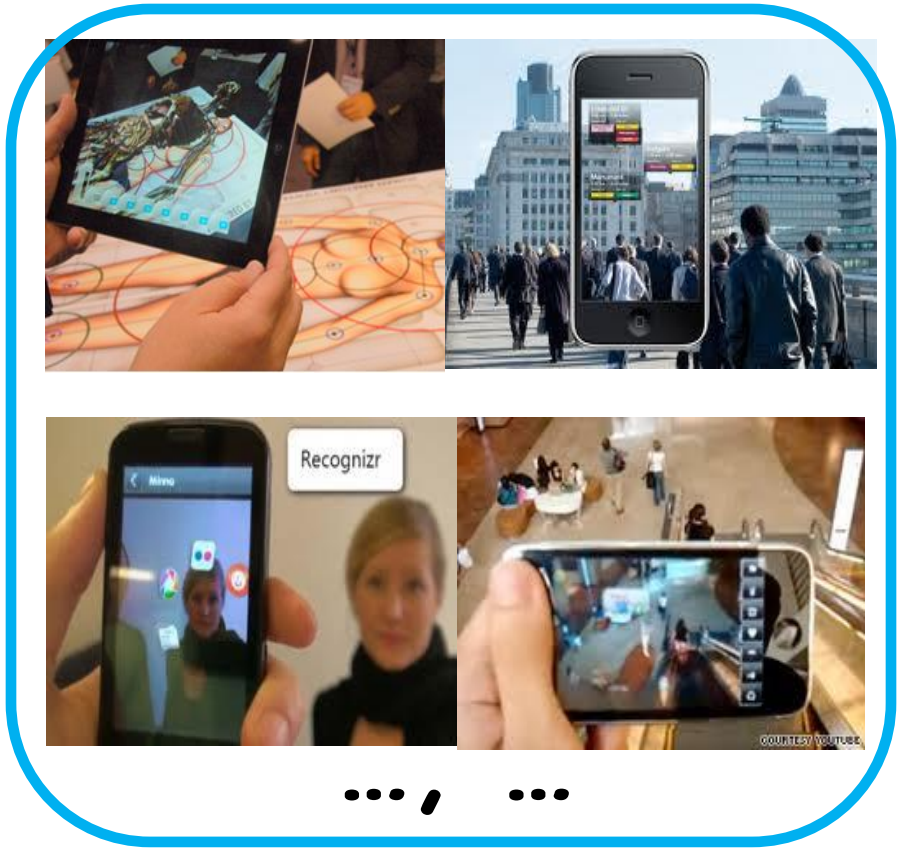
…, …

Explosive Growth of Mobile Computer Vision (CV) Apps



Samsung Exynos 5 Octa has 8 CPU cores:
4 cortex A15 + 4 cortex A7

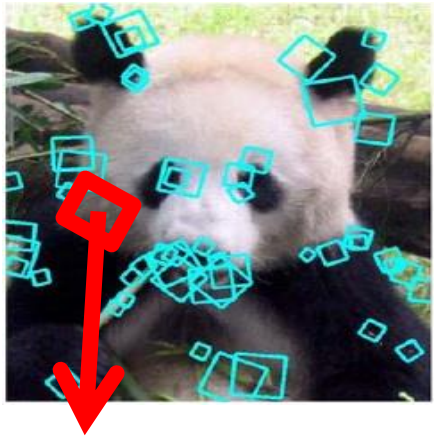
Mobile CPU: cortex A7 → A9 → A15→A17



Explosive Growth of Mobile Computer Vision (CV) Apps

Mobile devices are growing incredibly fast these days

Local Feature Representation: One of Fundamental Problems in CV



$$\vec{X} = [x_1, x_2, \ldots, x_d]$$

- Local feature extraction
  - Interest point detection
  - Interest point description
- Key challenges
  - Various image transformations
    → hard to be robust and distinctive
  - Hundreds of local features per image, involving intensive arithmetic computations →hard to run efficiently
  - Most existing algorithms are designed for x86-based PC, not for ARM-based mobile platform
    - Type of computations, data structure, memory access pattern are not suitable for mobile hardware → large runtime degradation
- Our goal
  - Ultrafast to compute and match on mobile
  - Highly distinctive → few mismatches, short runtime for post-verification
  - Compact to store

Development and Bottlenecks of Existing Local Features

< 2005	2006~2007	2008~2009	2010	2011	2012
SIFT	SURF	SURFTrac	BRIEF	ORB	FREAK
PCA-SIFT	DASIY			BRISK	
GLOH	Self-Similarity				

Characteristics

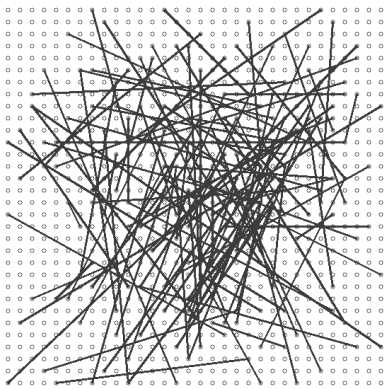
1. Great robustness &distinctiveness
2. High-dimensional real-valued vectors
   large memory cost
   →too expensive for mobile apps

Characteristics

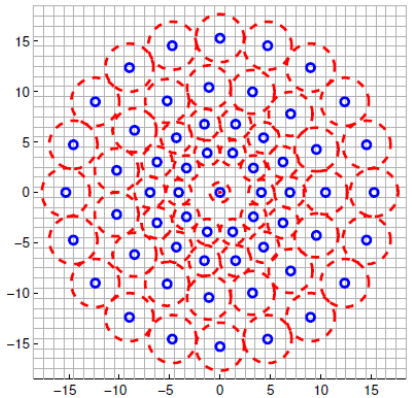
1. Fast to compute & to match
2. Small memory cost
3. Low robustness & distinctiveness
   → long post-verification time

Binary Descriptor: State-of-the-Art

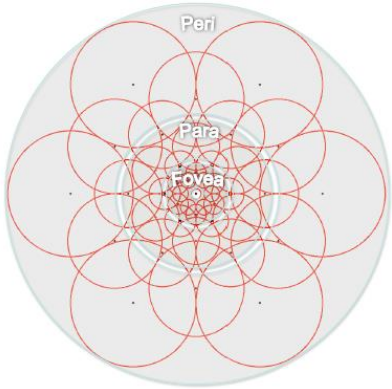
- BRIEF[ECCV10], BRISK[CVPR11] and FREAK[CVPR12]
  - Samples a list of points
  - Select pairs of points



BRIEF
256 point pairs



BRISK
~60 points → ~1770 pairs



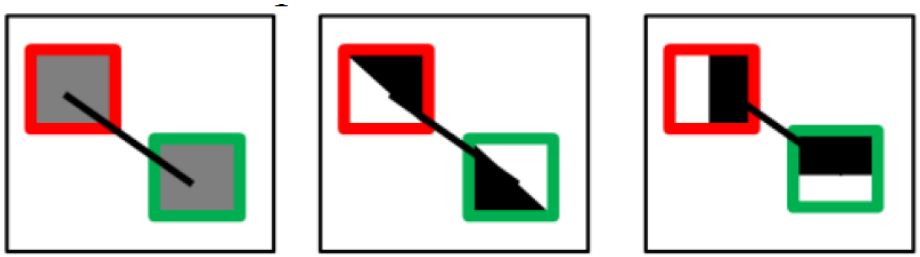
FREAK
~42 points → ~861 pairs

Fig1. Sampling Patterns of three Feature Descriptions

  - Compares pairs of smoothed points' intensities

Limitations of Existing Binary Features

- Utilize overly simplified information



- Sparse and handcrafted sampling
- Ad-hoc pairs of points selection

Low distinctiveness

↓

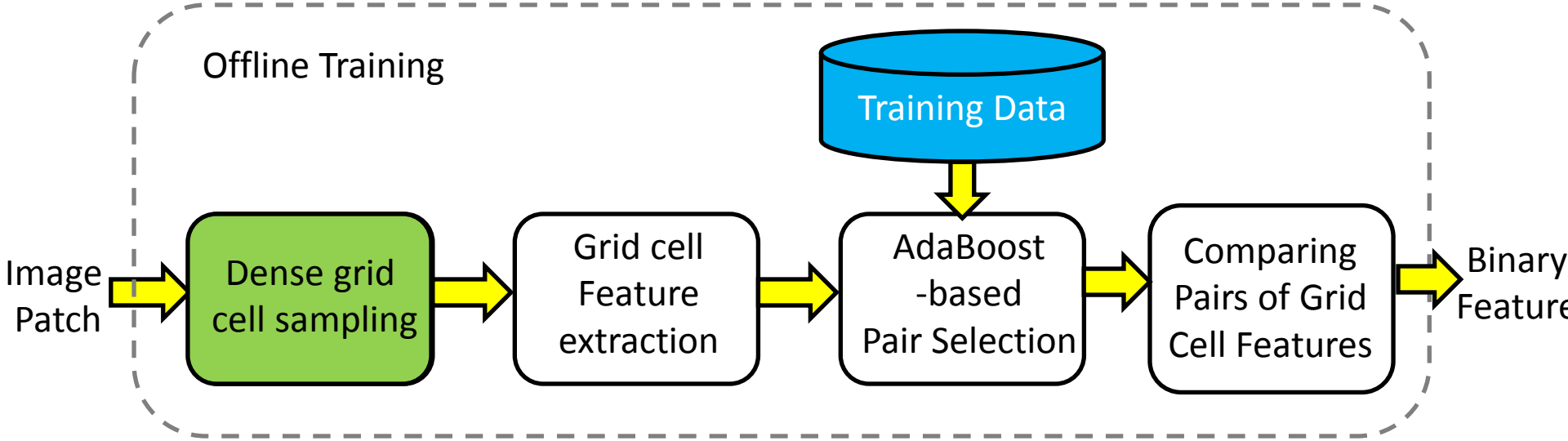
Many false matches

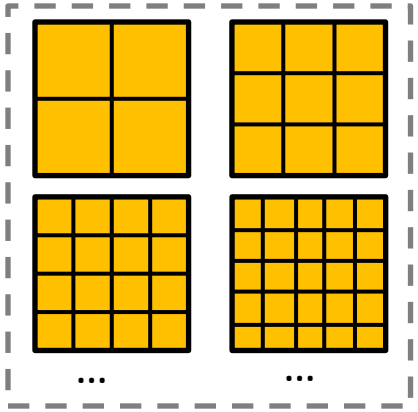
↓

Long runtime for post-processing

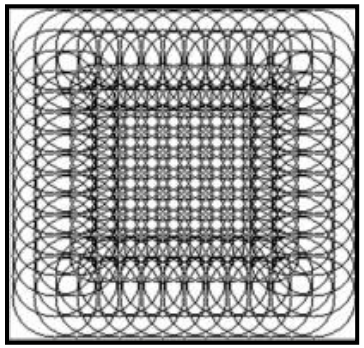
Our goal: comparable speed for feature construction,
while greater distinctiveness
→ faster matching and verification

LDB: Local Difference Binary Extraction Framework

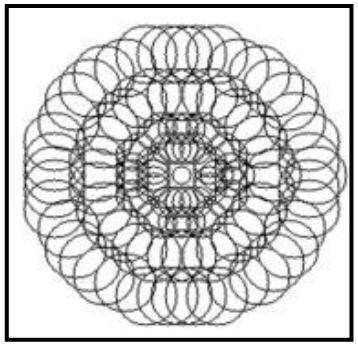




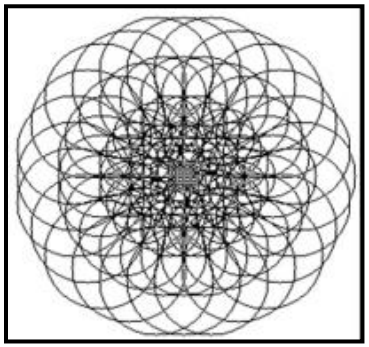
Multiple gridding



Uniform Sampling



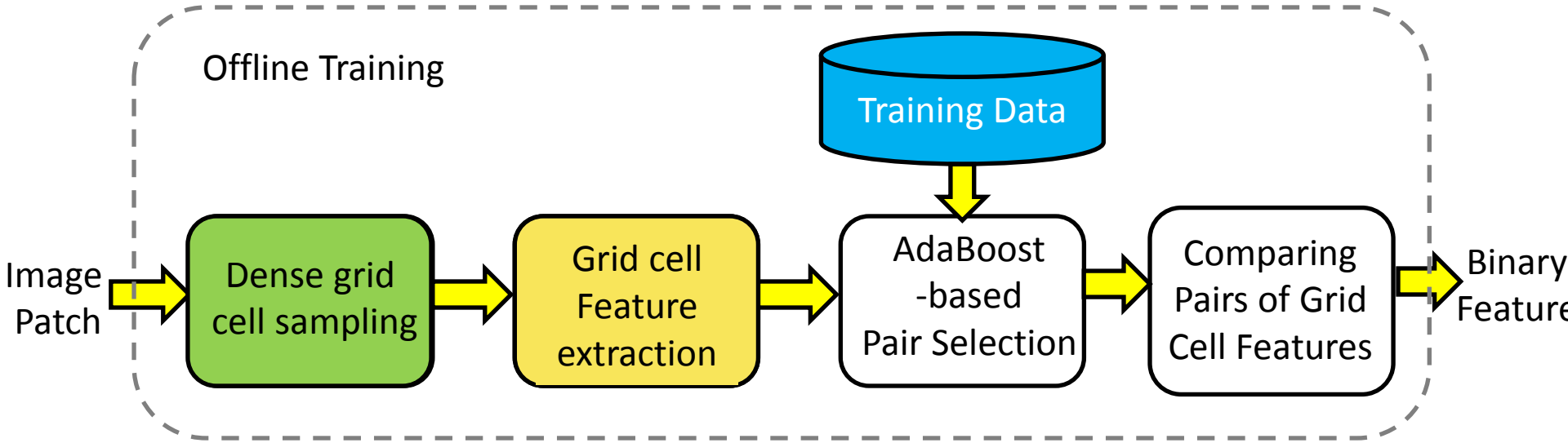
Circular Sampling



Retinal Sampling

Sample 169~220 grid cells

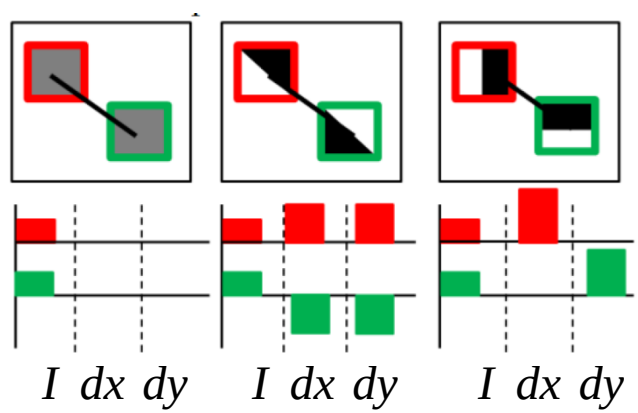
LDB Extraction Framework



$$I_{avg}(i) = \frac{1}{m_i}\sum_{k=1\sim m_i} Intensity(k)$$
$$d_x(i) = Gradient_x(i)$$
$$d_y(i) = Gradient_y(i)$$

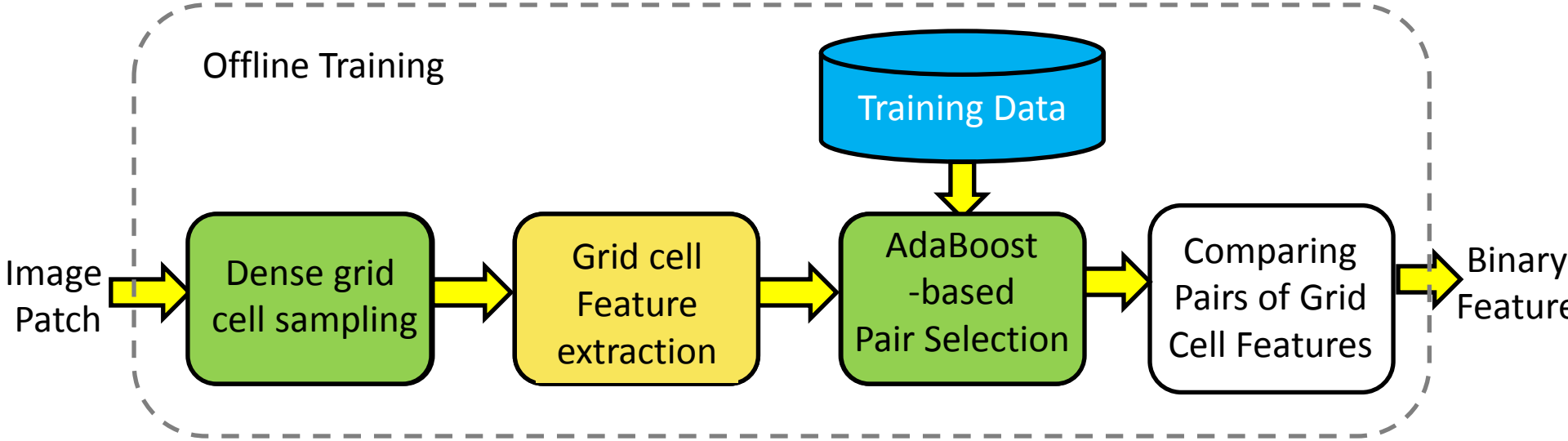
$$F_i \in \{I_{avg}(i),\ d_x(i),\ d_x(j)\}$$

$$\tau(F_i, F_j) = \begin{cases} 1 & \text{if } F_i > F_j \\ 0 & otherwise \end{cases}$$



Grid cell features can be efficiently calculated via integral image

LDB Extraction Framework

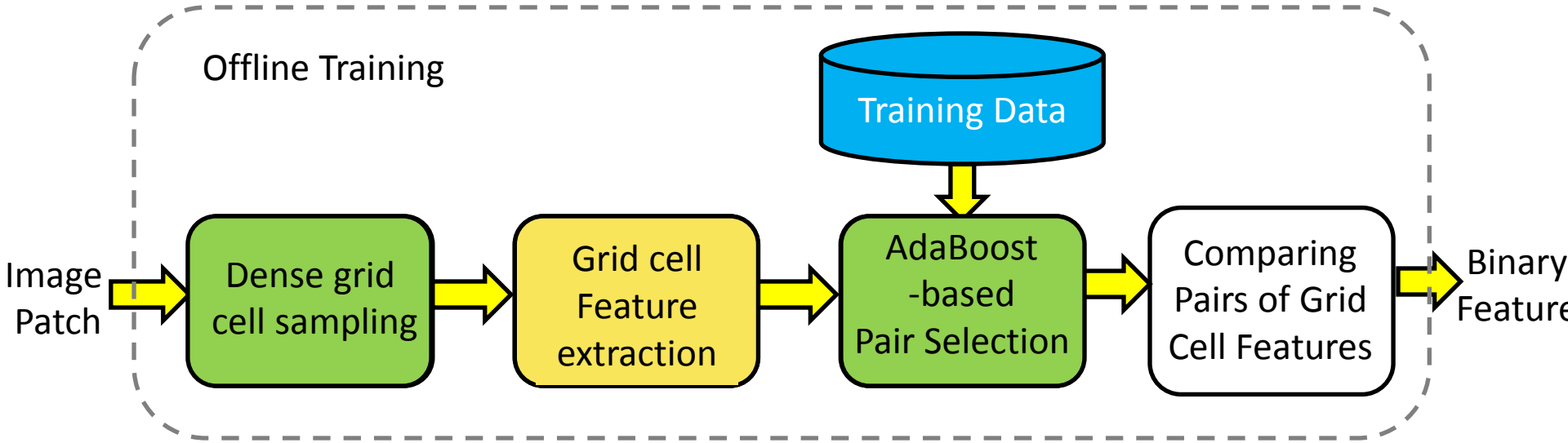


Total number of pairs of grid cell features is > 42,588!
21X more than BRISK[CVPR'11],
45X more than FREAK[CVPR'12],
165X more than BRIEF[ECCV'10] and ORB[ICCV'11],

How to select 256 (<0.05%) pairs to form a binary feature?

- Maximize (minimize) distance between mismatched (matching) patches
- Minimize correlations between pairs

LDB Extraction Framework



Conventional AdaBoost for bit selection

1. Construct weak classifier as

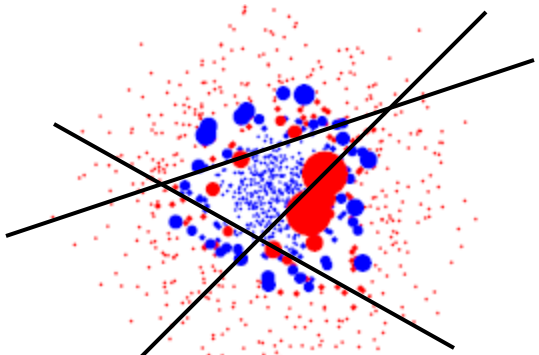
$$h_m(x) = h_m(p_a, p_b) = \begin{cases} 1 & \tau(F_{i_m}(p_a),\ F_{j_m}(p_a)) = \tau(F_{i_m}(p_b),\ F_{j_m}(p_b)) \\ -1 & \tau(F_{i_m}(p_a),\ F_{j_m}(p_a)) \neq \tau(F_{i_m}(p_b),\ F_{j_m}(p_b)) \end{cases}$$

2. Define object function as $L = \sum_{n=1}^{N} e^{-\frac{1}{2} y_n \sum_{m=1}^{M} \alpha_m h_m(x_n)}$

3. Select a bit which can minimize misclassification error ε_m

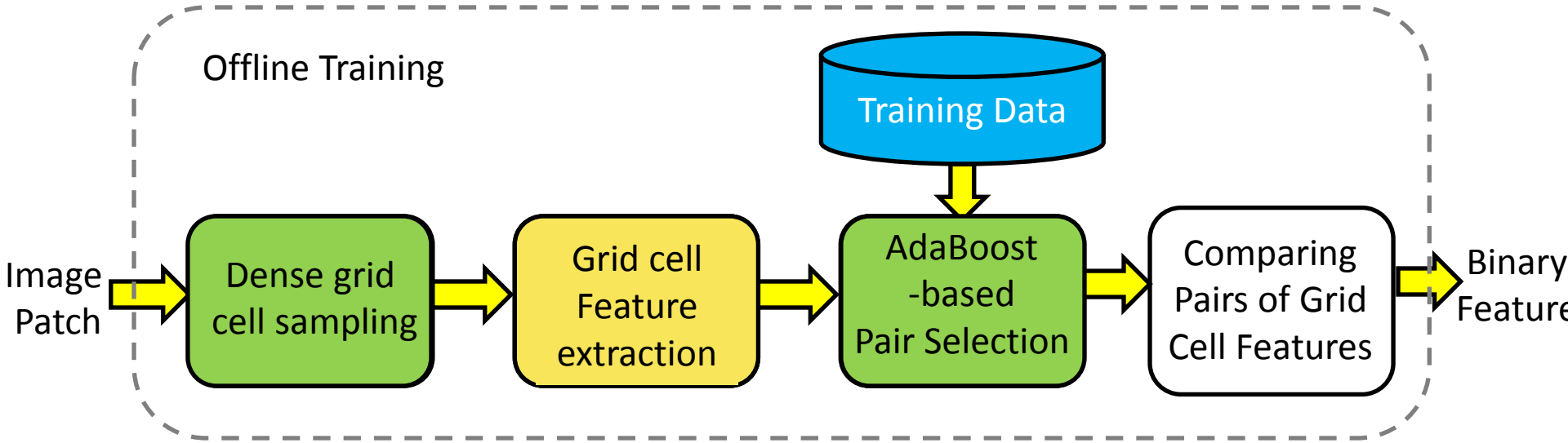
4. Weight the classifier and re-weight training samples

Blue points: matches
Red points: mismatches



$$\varepsilon_m = \frac{\sum_n \omega_n^{(m)} I\left(h_m(x_n) \neq y_n\right)}{\sum_n \omega_n^{(m)}}$$

LDB Extraction Framework



Problems: 1. each weak classifier (bit) has a weight → floating point feature vector

$$D(p_a, p_b) = \sum_{m=1}^{M} \alpha_m h_m(p_a, p_b)$$

2. rapid over-fitting, i.e. more bits, more training data do not decrease error rate

Modify AdaBoost object function *L* by introducing a variable β

$$L = \sum_{n=1}^{N} e^{-\frac{\beta}{2} y_n \sum_{m=1}^{M} \alpha_m h_m(x_n)} \qquad \alpha_m = \frac{1}{\beta} \ln\left(\frac{1-\varepsilon_m}{\varepsilon_m}\right) \qquad \omega_n^{(m)} \propto \omega_n^{(m-1)} e^{-\frac{\beta}{2} y_n \alpha_{m-1} h_{m-1}(x_n)}$$

Impact of β

Randomly choose 5,000 pairs of matching patches and 20,000 pairs of non-matching patches from Liberty dataset*

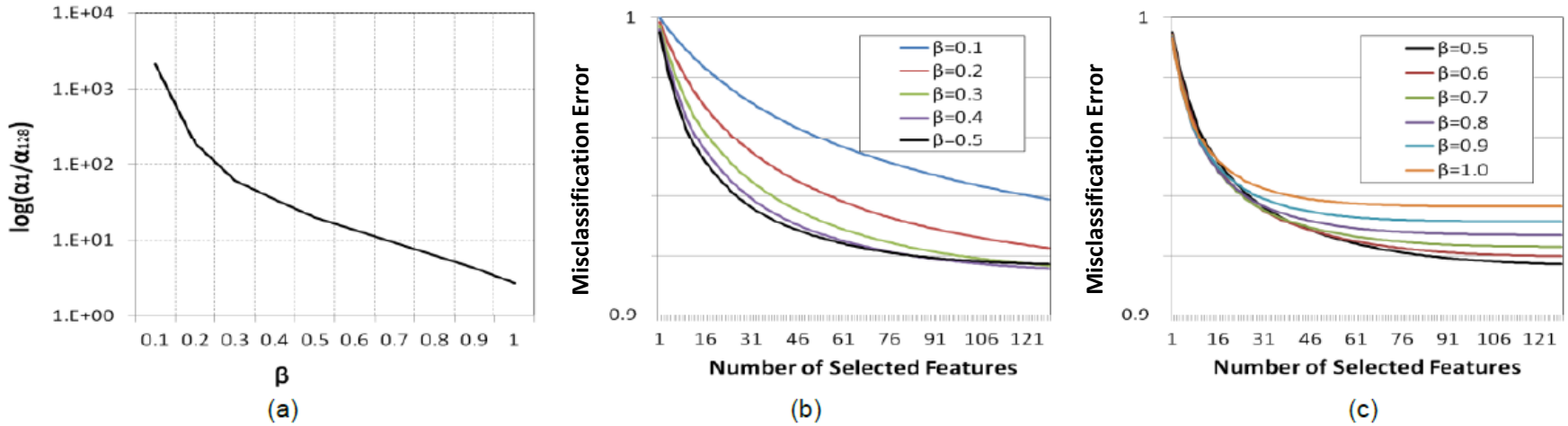


Figure 6: (a) Weight ratio α_1/α_{128} (i.e. weight gap) as a function of the tunning knob β. Exponential loss function L as a function of number of selected features (b) when β/2 = 0.1~0.5 and (c) when β/2 = 0.5~1.0. Note that the exponential loss function L is normalized such that the largest entry of L is equal to 1.

$\beta = 0.5$ provides small gap between α and minimum misclassification error

* Winder, S., Hua, G., and Brown, M., Picking the Best DAISY, In Proc. CVPR'09

Evaluation: Invariance to Transformations

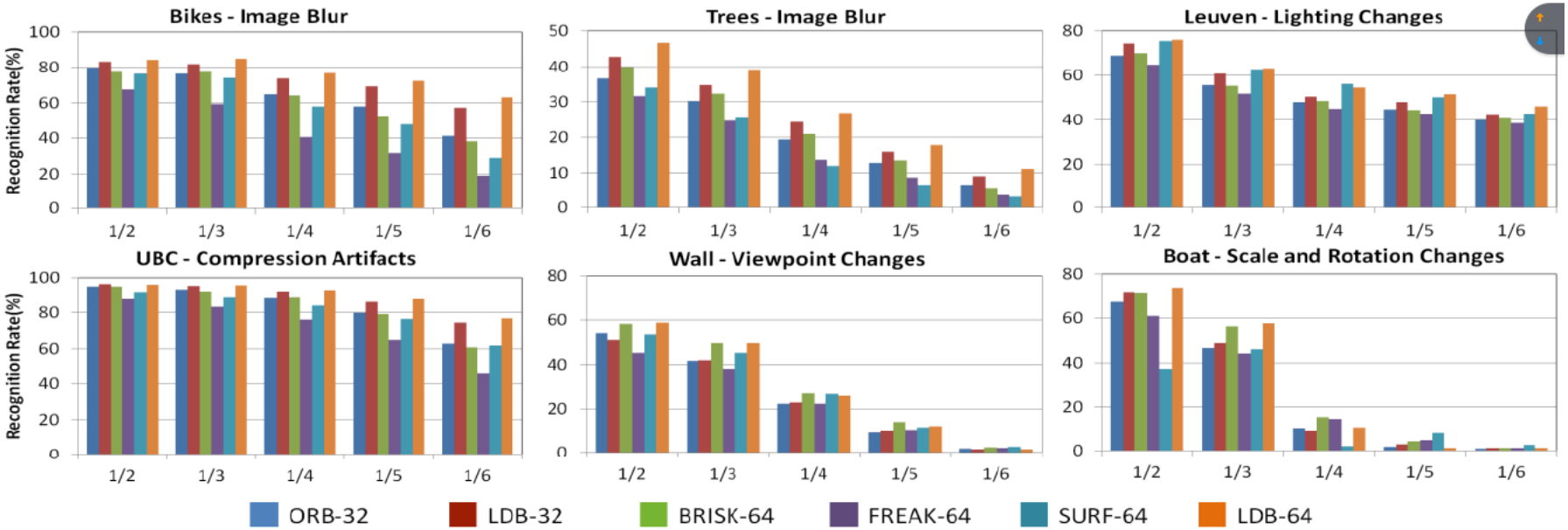
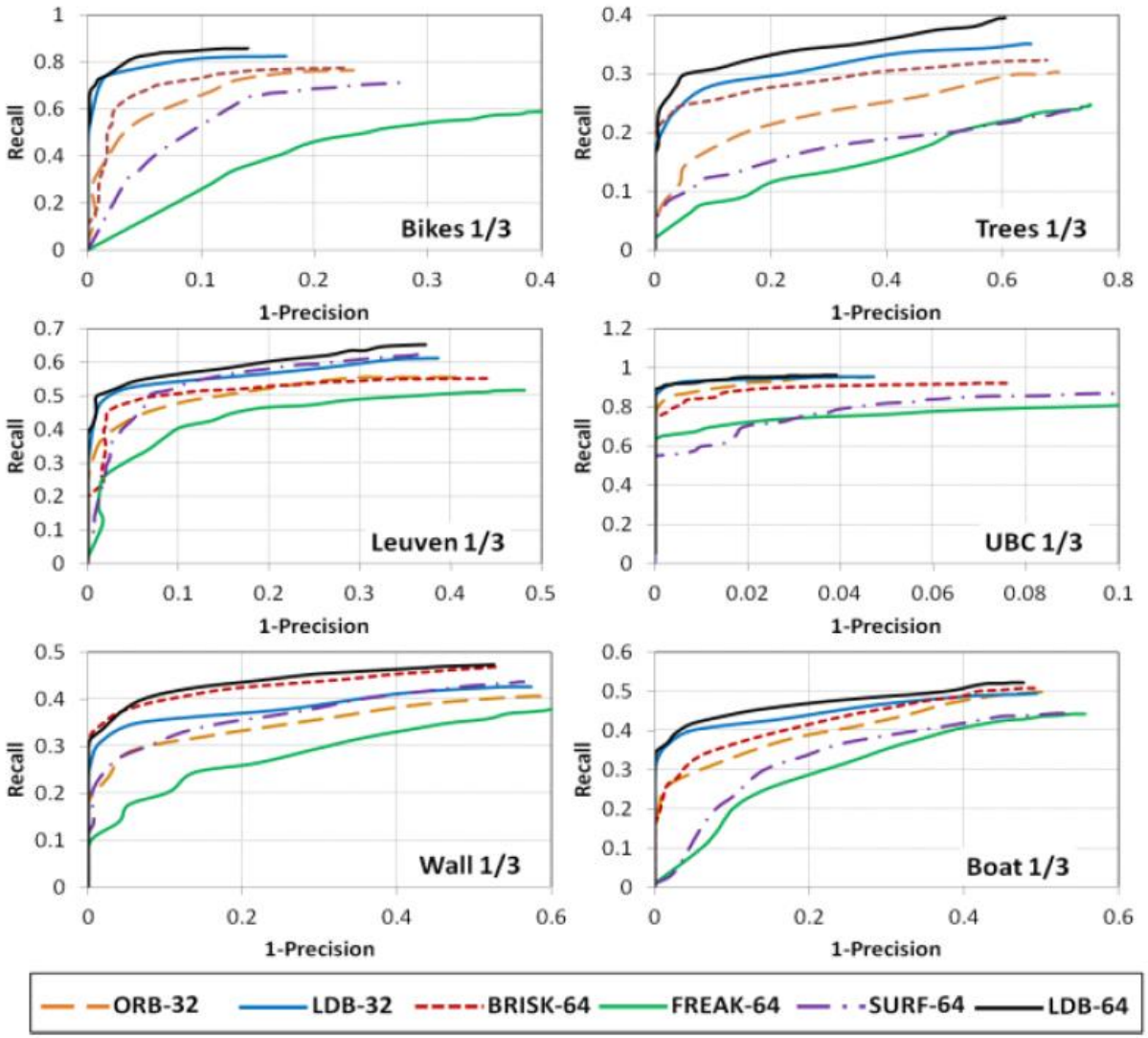


Figure 3: Recognition rate obtained by LDB, ORB, BRISK, FREAK and SURF for the six image sequences of VggAffine Dataset.

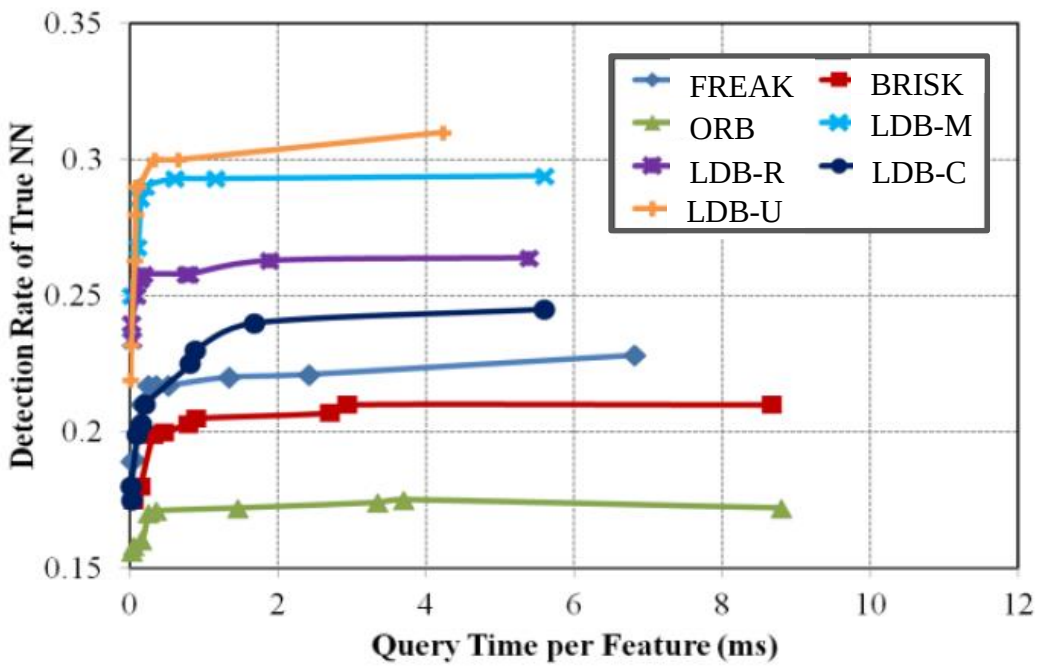
Increasing illumination changes

Evaluation: Invariance to Transformations



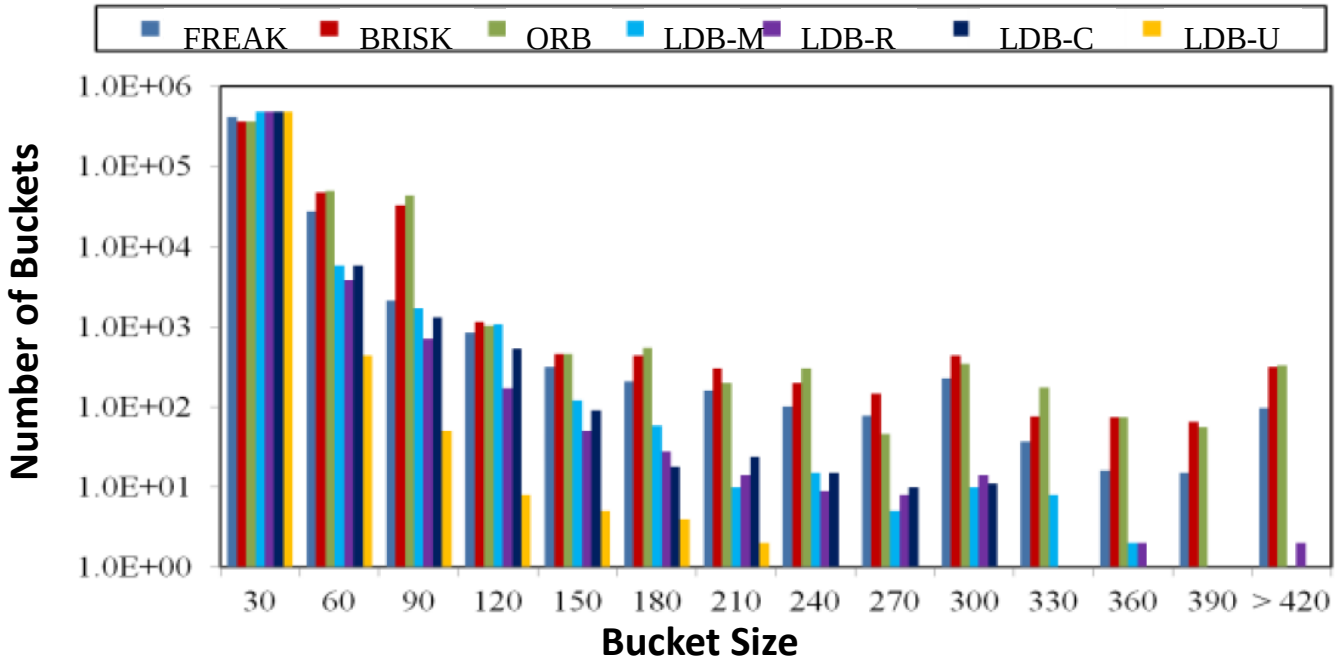
Evaluation: Scalable Matching

- Locality Sensitive Hashing (LSH) for scalable matching
  - Hash function: a subset of bits from the binary string
  - Construct hash tables with ~2.3M features, change hash key size and number of probes to adjust detection rate and query time
  - 1000 queries, 256-bit binary feature



Evaluation: Scalable Matching

- Locality Sensitive Hashing (LSH) for scalable matching
  - Hash function: a subset of bits from the binary string
  - Construct hash tables with ~2.3M features, change hash key size and number of probes to adjust detection rate and query
  - 1000 queries, 256-bit binary feature



Evaluation for Mobile Object Recognition

- Recognize 228 planer objects (FXPAL: EMM dataset)
- Runtime is record on Motorola Xoom1 tablet, cortex A9, 1GHz
- 228 query images, 500 features/image and 2.3M features in hash tables

(a) Database Images

(b) Query Images

(c) Database Images

(d) Query Images

Table 1. Query Time (ms) Comparison

Descriptor	32bits	64bits	128bits	256bits
FREAK	103	130	332	365
BRISK	98	196	224	277
ORB	75	149	222	231
LDB-M	56	72	100	141
LDB-U	**37**	**40**	**53**	**71**

2X ~5.1X speedup

Evaluation: Mobile Object Recognition

Table 2. Runtime (ms) for Constructing 256-bit Binary Descriptors

Descriptor	FREAK	BRISK	ORB	LDB-M	LDB-U
Time(ms)	61	120	63	70	77

Table 3. Recognition Rate (Top1 accuracy) Comparison

Descriptor	32bits	64bits	128bits	256bits
FREAK	21.1	60.8	85.5	92.5
BRISK	17.6	52.4	82.5	89.4
ORB	7.5	45.4	72.2	86.3
LDB-M	25.1	69.6	**90.7**	92.7
LDB-U	**28.6**	**76.2**	89.0	**93.8**

Performance Comparison for Recognizing 1140 images on Google Nexus 4

Descriptor	Detection Rate (%)	Precision (%)	Construction Time (ms)	Recognition Time (ms)	Memory Usage (MB)
BRISK [2011]	97.7	99.4	0.034	7.64	168
FREAK [2012]	98.3	99.5	0.108	24.65	168
SURF [2006]	83.8	92.9	1.488	-	466
LDB-M	98.6	99.5	0.143	4.76	168

LDB achieves the highest detection rate and precision

LDB is **10.4X** faster to construct than SURF-64

LDB-M is **1.6X** faster than BRISK and **5.2X** faster than FREAK for recognition

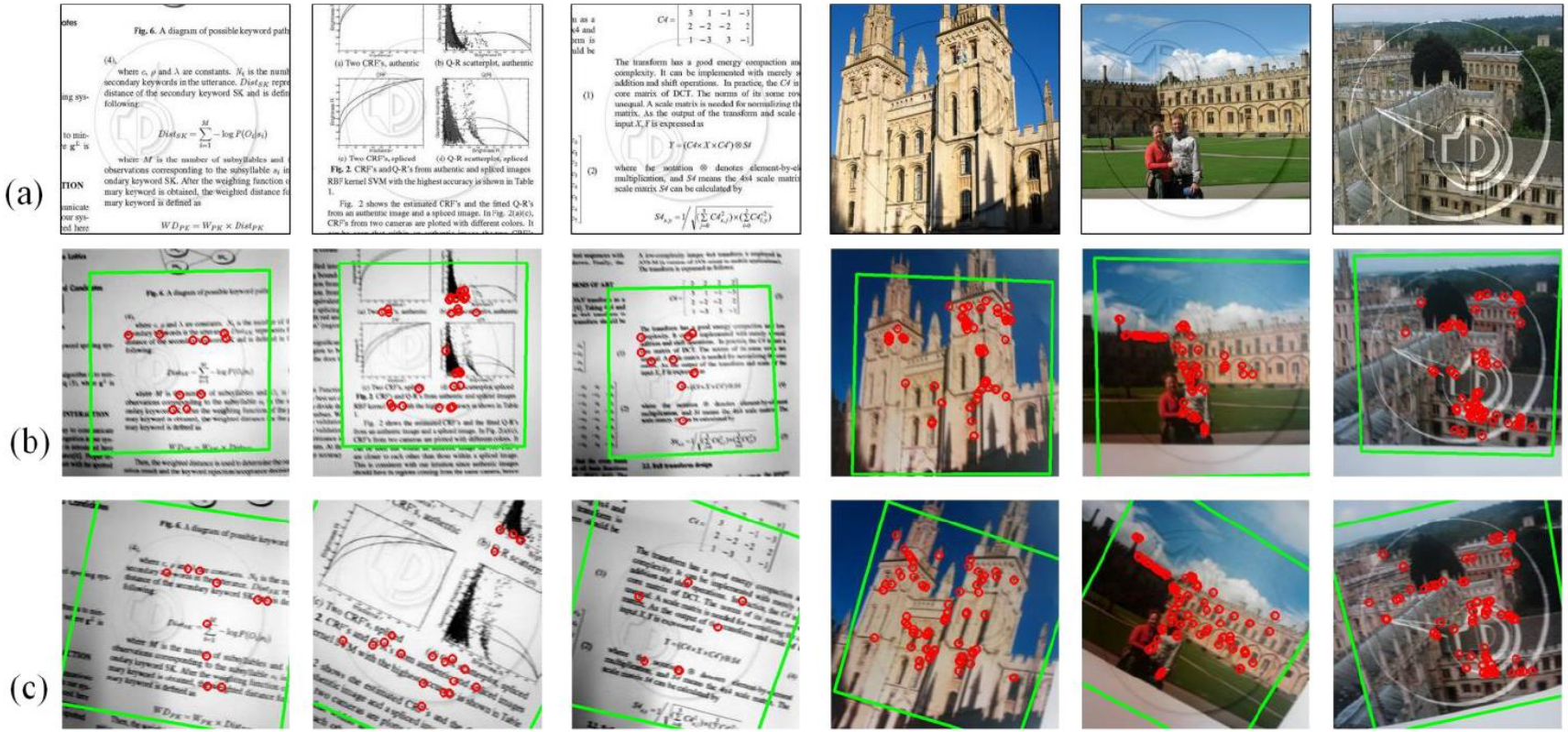


Figure 12: (a) Exemplar images from the *Document-Natural* database, (b) and (c) recognized query images with scaling and rotations, respectively. We use 64-bit LLDB-U in this experiment. The green rectangles illustrate the pose between the camera and the recognized object, and red dots denote the matched points.

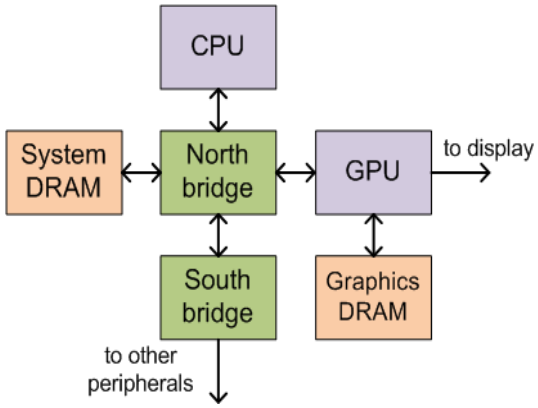
Project Website:

http://lbmedia.ece.ucsb.edu/research/binaryDescriptor/web_home/web_home/index.html

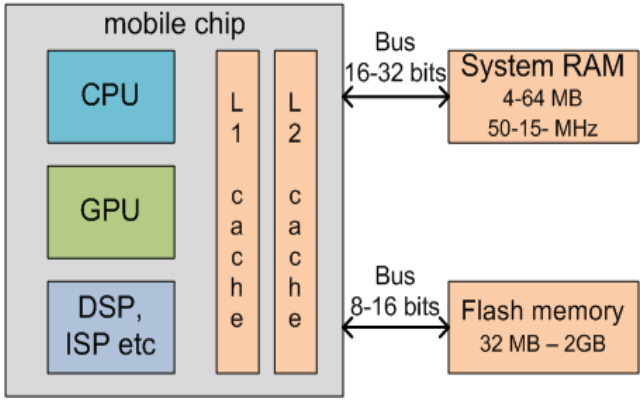
*Source code has been released and published on ACM MM 2014 open source software competition

Challenges and Opportunities of Running CV Apps on Mobile Devices

- My research interest: performance & energy enhancement of vision apps on mobile devices



- Most robust, high quality CV algorithms are
  - Compute-/memory-intensive
  - Designed to run on powerful computing systems (e.g. PC/server), not on mobile handheld devices
- Mobile handheld devices and PC use different processors
  - Different processor micro-architecture, system memory bus architecture, memory bandwidth, etc.
- Mobile devices are equipped with various sensors

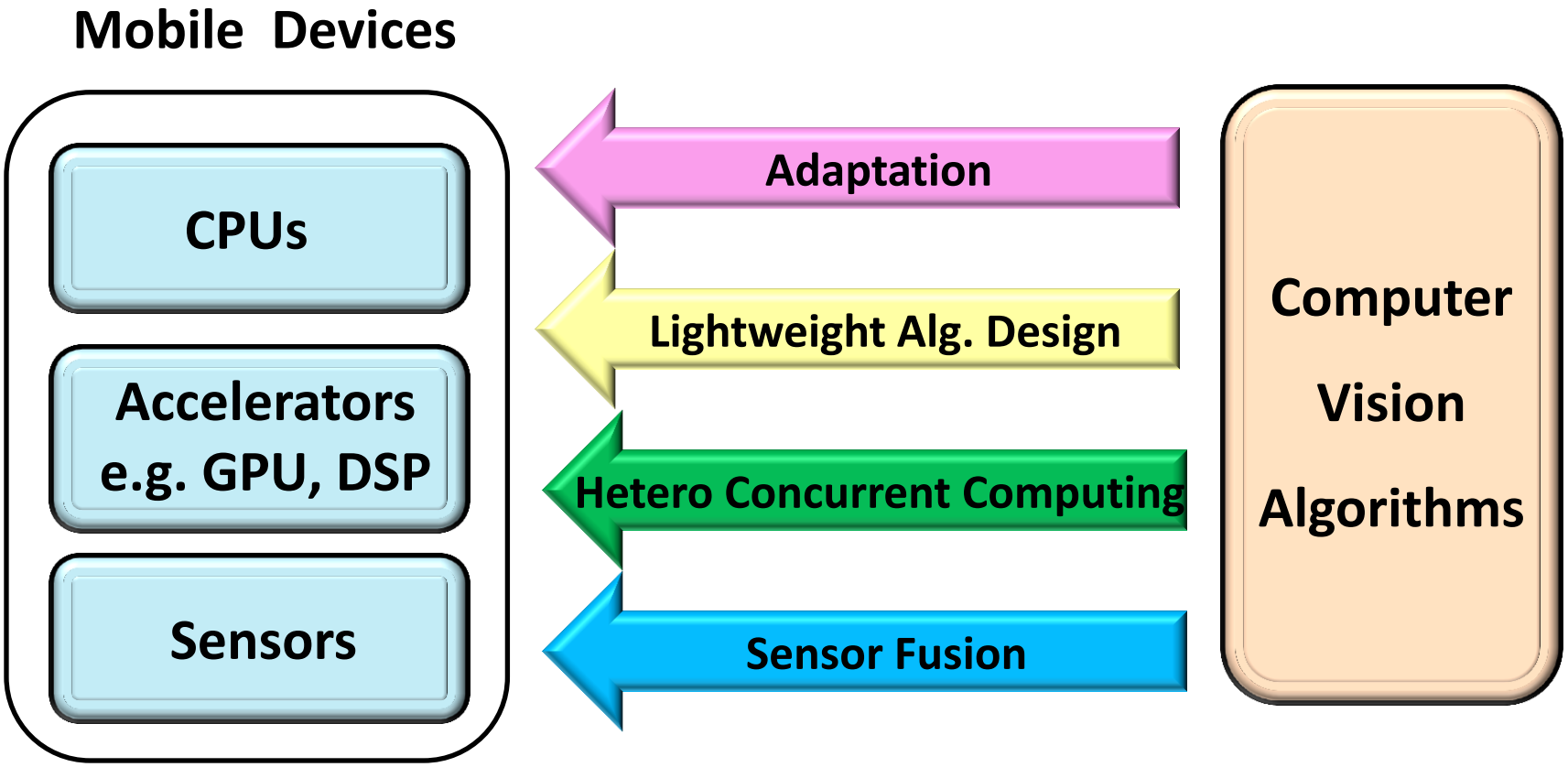


Conventional Vision vs. Mobile Vision

Algorithm design

Algorithm design
+ Unique mobile hardware constraints
+ Energy consumption
+ Unique resources equipped on mobile handhelds
+ Various mobile environments

Strategies of Improving Performance and Energy for Mobile Vision Apps



Thanks! & Questions?