# TRECVID INSTANCE SEARCH (INS)

Shin'ichi Satoh, National Institute of Informatics
Duy-Dinh Le, University of Information Technology, Vietnam National University HCMC
Vinh-Tiep Nguyen, University of Science, Vietnam National University HCMC

## TRECVID (from TRECVID web site...)

- Workshop series from 2001 to present
- Large-scale laboratory testing for content-based video analysis and retrieval
- Forum for the
  - exchange of research ideas
  - discussion of approaches: what works, what doesn't, and why
- Aims for realistic system tasks and test collections
  - ***unfiltered data***
  - focus on relatively high-level functionality
- Provides data, tasks, and uniform, appropriate scoring procedures

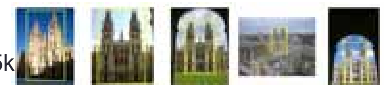
## TRECVID Instance Search (INS)

- To find "instances" of some object, person, or location in video
  - one specific object, person, or location
  - e.g., search for this particular dog
  - different manufactured objects which are indistinguishable
  - including logos
- Queries will be given as visual examples
- There exist couple of related benchmark datasets
  - Oxford Building, Paris (landmarks)
  - Flickr Logos (logos)
  - UKBench, Stanford Mobile Visual Search (specific objects)
  - etc.

## Comparison with other benchmarks

- TRECVID INS determines data first: therefore very "wild"

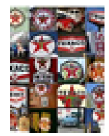

- Other benchmarks define queries first, and then collect data: therefore objects clearly appear

Ox5k


UKB

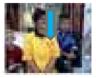

SMVS



FL32


# Data

- Collection of several hundreds hours of videos for each year
- Data should contain multiple occurrences of multiple specific objects.
- Search tasks should be reasonably difficult.



- Sound and Vision (2010): too difficult, too few repeated instances, otherwise too easy(copies)
- BBC Rushes (2011): including retakes, artificial video transformations,
- Flickr Creative Commons (2012): reasonable, but still hard to find repeated instances
- BBC EeastEnders (2013-present): drama series, “small world” many repeated instances (person, location, objects, ...)



# EastEnders’ world

Majority of episodes filmed at Elstree studios. Sometimes filmed on ‘location’.

# Task

- 2010-2015: specific object, person, or location
- 2015-present: find a specific person in a specific location

# Queries

- Couple of example images with masks
- Original videos are also given (since 2014)

Topics – segmented example images

Source Mask

Example from TV12

# Topics – 26 Objects

**Topic: True positives:**

**69** **2300**

a 'no smoking' logo

**70** **741**

a small red obelisk

**71** **31**

an Audi logo

**72** **261**

a metropolitan police logo

**73** **674**

this ceramic cat face

**74** **100**

a cigarette



# Topics – 26 Objects (cont.)

**75** **82**

a SKOE can

**76** **831**

Queen Victoria bust

**77** **31**

this dog

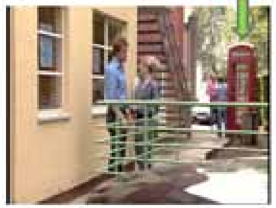
**78** **880**

A JENKINS logo

**79** **390**

this CD stand

**80** **251**



this phone booth



# Topics – 26 Objects (cont.)

**81** **213**

a black taxi

**82** **61**

a BMW logo

**83** **118**

chrome/glass cafetiere

**85** **455**

David fridge magnet

**86** **759**

these scales

**87** **25**

a VW logo



# Topics – 26 Objects (cont.)

**89** **1266**

this pendant

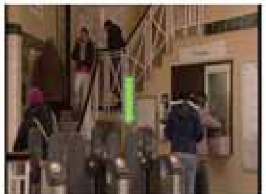
**90** **363**

this wooden bench

**91** **782**

a menu with stripes

**93** **75**

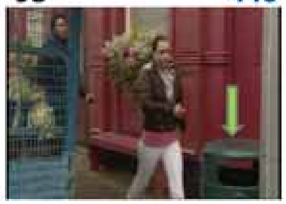


these turnstiles

**94** **171**

a tomato ketchup dispenser

**95** **440**



a public trash can

# Topics –26 Objects (cont.)

97 252

these checkerboard spheres

98 386


a P (parking automat) sign





# Topics – 4 Persons

84 32

this man

88 1605

Tamwar

92 171

this man

96 161

Aunt Sal



# NII baseline INS system

- BoVW-based simple method (ICMR2012)
- no trick, but performed very well
- This baseline works well for objects and locations (landmarks).
- This baseline software will be made public.
- “Person” queries may need other person-specific treatment (deep-based face representation, person re-identification techniques, etc.) and are outside of the scope of this baseline system

TV2011

| Automatic | | | | | MAP |
|---|---|---|---|---|---|
| F | X | N | NII.Caizhi.HISimZ | 4 | 0.531 |
| F | X | N | NII.Caizhi.HISim | 3 | 0.491 |
| F | X | N | MCPRBUPT1 | 1 | 0.407 |
| F | X | N | MCPRBUPT2 | 2 | 0.353 |
| F | X | N | NII.SupCatGlobal | 1 | 0.340 |
| F | X | N | MCPRBUPT3 | 3 | 0.328 |
| F | X | N | TNO-SURFAC2 | 1 | 0.325 |
| F | X | N | vireo_f | 1 | 0.312 |
| F | X | N | vireo_b | 2 | 0.309 |
| F | X | N | vireo_s | 3 | 0.299 |
| F | X | N | vireo_m | 4 | 0.295 |
| F | X | N | TNO-SUREIG | 3 | 0.274 |
| F | X | N | IRIM_1 | 1 | 0.274 |
| F | X | N | IRIM_3 | 3 | 0.259 |
| F | X | N | IRIM_4 | 4 | 0.251 |
| F | X | N | JRS_VUT | 4 | 0.170 |
| F | X | N | IRIM_2 | 2 | 0.166 |
| F | X | N | NII.Chanseba | 2 | 0.115 |
| F | X | N | JRS_VUT | 3 | 0.104 |

TV2013

| Automatic | MAP |
|---|---|
| NII-AsymDis_Cai-Zhi_2 | 0.313 |
| NTT_NII_3 | 0.297 |
| NII-AvgDist_Cai-Zhi_3 | 0.276 |
| NII-GeoRerank_Cai-Zhi_1 | 0.256 |
| NTT_NII_2 | 0.256 |
| NTT_NII_1 | 0.237 |
| PKU-ICST-MIPL_1 | 0.212 |
| PKU-ICST-MIPL_3 | 0.200 |
| PKU-ICST-MIPL_4 | 0.198 |
| NTT_NII_4 | 0.198 |

TV2014

| MAP | Best run from |
|---|---|
| 0.325 | F_D_NII_2 |
| 0.304 | F_D_NU_1 |
| 0.234 | F_D_NTT_CSL_1 |
| 0.232 | F_D_PKU-ICST_2 |
| 0.227 | F_D_MediaMill_1 |
| 0.227 | F_D_BUPT_MCPRL_1 |
| 0.213 | F_D_IRIM_1 |
| 0.197 | F_D_VIREO_3 |
| 0.183 | F_D_ORAND_4 |
| 0.167 | F_D_OrangeBJ_2 |

DPM reranking

NII baseline 22.5

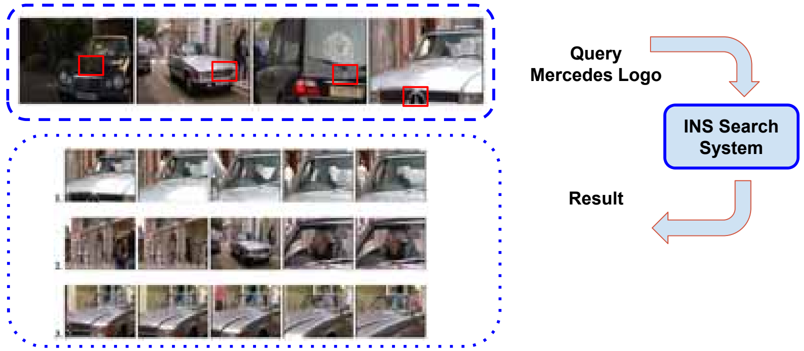
TV2015

| MAP | Top 10 runs across a |
|---|---|
| 0.453 | F_E_PKU_ICST_1 |
| 0.443 | F_E_PKU_ICST_3 |
| 0.424 | F_A_PKU_ICST_4 |
| 0.424 | F_A_NII_Hitachi_UIT_3 |
| 0.418 | F_A_NII_Hitachi_UIT_4 |
| 0.415 | F_A_NII_Hitachi_UIT_2 |
| 0.403 | F_A_BUPT_MCPRL_4 |
| 0.403 | F_A_BUPT_MCPRL_3 |
| 0.403 | F_A_BUPT_MCPRL_1 |
| 0.401 | F_A_NII_Hitachi_UIT_1 |

DPM reranking + RCNN

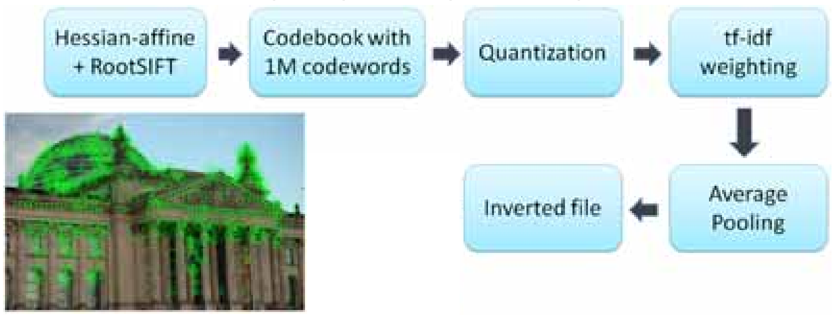
# Introduction

- KAORI-INS15 is a framework for the TRECVID-Instance Search Task developed at Video Processing Lab@NII.
- It is the baseline for the INS system ranked 1st in TRECVID-INS 2013, and TRECVID-INS 2014.
- The framework uses the BoW approach with large codebook size for fast video retrieval given a query example.



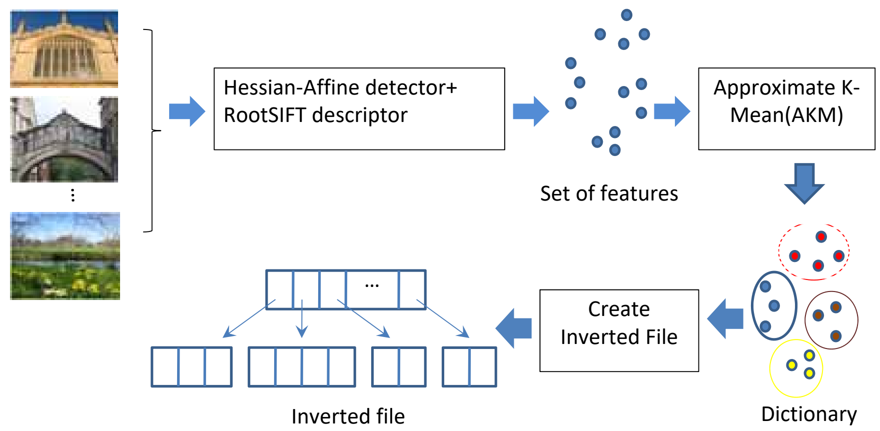

# Method Overview

- Keypoint detector: Hessian-Affine.
- Descriptor: RootSIFT.
- Codebook size: 1M.
- Quantization: Hard assignment.
- Others: tf-idf weighting, average pooling, inverted index.

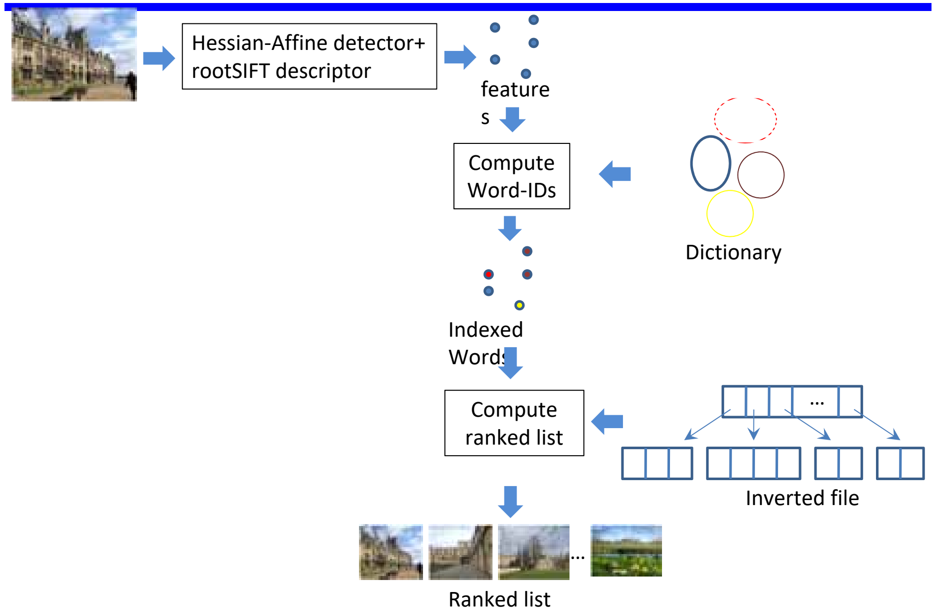



*Ref: Three Things Everyone Should Know to Improve Object Retrieval*, Relja Arandjelović and Andrew Zisserman (CVPR 2012)
http://www.robots.ox.ac.uk/~vgg/publications/2012/Arandjelovic12/presentation.pdf

# Method Overview - Offline Processing



# Method Overview - Online Searching




# External Libraries

- Keypoint detector + SIFT descriptor
  - Reference: http://kahlan.eps.surrey.ac.uk/featurespace/web/
  - Download (Linux version): http://kahlan.eps.surrey.ac.uk/featurespace/web/desc/compute_descriptors_64bit.ln
- Clustering: AKM → FASTANN + FASTCLUSTER
  - Reference: http://www.robots.ox.ac.uk/~vgg/software/fastanncluster/
  - Download: http://www.robots.ox.ac.uk/~vgg/software/fastanncluster/ or https://github.com/philbinj
  - Installation guide: http://www.robots.ox.ac.uk/~vgg/software/fastanncluster/fastann/README.txt and http://www.robots.ox.ac.uk/~vgg/software/fastanncluster/fastcluster/README.txt
- VLFeat 0.9.18
  - Download: http://www.vlfeat.org/download.html
- NII-KAORI-INS15
  - http://www2.satoh-lab.nii.ac.jp/users/ledduy/nii-kaori-ins15/ (trecvid/niitrec)
- MATLAB is needed. And others (Python, ...)

# Data Organization

- Working dir → containing all keyframe images, features, metadata, and results for one experiment (i.e. one dataset → DB = oxford5k)
  - → nii-kaori-ins15/experiments
- Image dataset
  - → nii-kaori-ins15/experiments/oxford5k/images_test
- Metadata
  - → nii-kaori-ins15/experiments/oxford5k/meta/lst_images.mat
  - → generated by create_list_images.m
- Feature
  - raw → nii-kaori-ins15/experiments/oxford5k/feature/hesaff_rootsift_noangle_mat
  - BoW → nii-kaori-ins15/experiments/oxford5k/feature/hesaff_rootsift_noangle_cluster

# Component 1: Feature Extraction

- Input: a set of keyframes of a dataset (e.g. oxford5k)
  - keyframes, eitheir in jpg or png format, stored in images_test/*.jpg|*.png
  - list of keyframes of the dataset, stored in meta/lst_images.mat (generated by → create_list_images.m)
- Output: a set of raw feature files, one file for one keyframe stored in .mat.
  - raw features, stored in feature/hesaff_rootsift_noangle_mat
- Workflow → extract_hesaffine_rootsift_noangle4image.m
  - Extract keypoints and SIFT descriptor → Param: *-hesaff -sift -noangle*
  - Compute RootSIFT (loading data using vl_ucbread)
  - Save data - *one feature file (.mat) for one keyframe*, each item in the file is feature descriptors of each keyframe.
- Can be run in parallel by controlling startShotID and endShotID.
- Processing time for oxford5K (5,063 images)
  - → 5 hours (3.76 secs/keyframe - 1,024x768).
  - → total feature points: 24.46M → 4,832 feature points/keyframe.

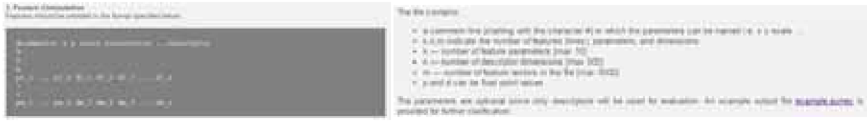
## Component 1: Feature Extraction

- Processing time: 3.76 secs/keyframe (1,024x768). oxford5k → 5 hours

```
[LOOP] - ###5063 [1 - 5063] - [worcester_000198.jpg]
detector: hesaff
descriptor:

computing features in image /net/per610a/export/das1
jpg
Hessian-Laplace(affine) interest points 4612
total time: 3.53333 user 3.38333 system 0.15
saving 4612 features in output file: /tmp/hesaff_roo
[DONE] - Total keyframes: [5063]
. Total keypoints: [24464227]
. Current average speed: [3.7574]    /frame
[ledduy@per900c code]$
```

- Misc
  - for visualization: http://www.robots.ox.ac.uk/~vgg/research/affine/detectors.html
  - for file format: http://kahlan.eps.surrey.ac.uk/featurespace/web/



## Component 2: Codebook Construction

- Input: a set of feature files, each feature file corresponding to a keyframe image.
- Output: a codebook
- Processing time depends on number of features, codebook size, iterations and processors
  - Sampling features: 10 mins → 24,464,227 feature points (all)
  - One iteration: 10 mins → 10 hours (24.46M features clustered to 1M words with 50 iterations using 24 processors)
- Workflow → sampling_feat4clustering_vgg_hesaff.m + akm.py
  - Sampling feature descriptors → Param: 100M for 1M codebook (ratio = 1:100) → sampling_feat4clustering_vgg_hesaff.m → *output format must be hdf5 (hdf5write)*
  - Run approximate k-means → Param: output, intput, nCluster=1M, nIter = 50 → akm.py
- Note: for simplification, *a pre-built codebook* can be used to skip this step → hesaff_rootsift_noangle_cluster.

## Component 2: Codebook Construction

- Server/Workstation: 24 cores.
- run_akm.sh
- ./mpirun -np 24 ./python2.7nii-kaori-

```
Warning: Cluster 761089 is empty!
    26    2.024905e+06    3.7%    0.2%   70.2%    0.3%   20.2%    5.5%     559.6s | 76.40%+-1.90%
    27    2.022611e+06    3.7%    0.2%   70.2%    0.3%   20.2%    5.5%     554.4s | 74.20%+-1.96%
Warning: Cluster 334952 is empty!
    28    2.023549e+06    3.7%    0.2%   70.2%    0.3%   20.2%    5.5%     553.3s | 76.00%+-1.91%
Warning: Cluster 626168 is empty!
    29    2.020875e+06    3.7%    0.2%   70.2%    0.3%   20.2%    5.5%     557.7s | 72.20%+-2.00%
Warning: Cluster 936525 is empty!
    30    2.020403e+06    3.7%    0.2%   70.3%    0.3%   20.2%    5.5%     564.9s | 75.80%+-1.92%
    31    2.019965e+06    3.7%    0.2%   70.3%    0.3%   20.2%    5.4%     565.2s | 75.20%+-1.93%
Warning: Cluster 367849 is empty!
    32    2.018603e+06    3.7%    0.2%   70.3%    0.3%   20.1%    5.4%     537.5s | 75.00%+-1.94%
Warning: Cluster 480095 is empty!
Warning: Cluster 492427 is empty!
    33    2.017563e+06    3.7%    0.2%   70.5%    0.3%   20.0%    5.4%     544.1s | 74.20%+-1.96%
    34    2.016044e+06    3.7%    0.2%   70.5%    0.3%   20.0%    5.4%     543.1s | 72.40%+-2.00%
Warning: Cluster 332116 is empty!
    35    2.015974e+06    3.7%    0.2%   70.6%    0.3%   19.9%    5.4%     543.4s | 74.00%+-1.96%
    36    2.014151e+06    3.7%    0.2%   70.6%    0.3%   19.9%    5.4%     559.7s | 77.20%+-1.88%
    37    2.013753e+06    3.7%    0.2%   70.6%    0.3%   19.8%    5.4%     407.2s | 73.80%+-1.97%
    38    2.013400e+06    3.6%    0.2%   70.5%    0.5%   19.8%    5.4%     895.6s | 73.40%+-1.98%
    39    2.013006e+06    3.5%    0.2%   70.3%    1.5%   19.2%    5.4%    1550.3s | 71.60%+-2.02%
    40    2.011307e+06    3.5%    0.2%   70.3%    1.4%   19.3%    5.4%     705.9s | 77.60%+-1.86%
Warning: Cluster 279879 is empty!
    41    2.010909e+06    3.5%    0.2%   70.4%    1.4%   19.3%    5.4%     528.9s | 77.80%+-1.86%
Warning: Cluster 420103 is empty!
    42    2.009433e+06    3.5%    0.2%   70.4%    1.4%   19.2%    5.4%     524.7s | 75.20%+-1.93%
    43    2.008883e+06    3.5%    0.2%   70.5%    1.4%   19.2%    5.4%     527.5s | 73.40%+-1.98%
```

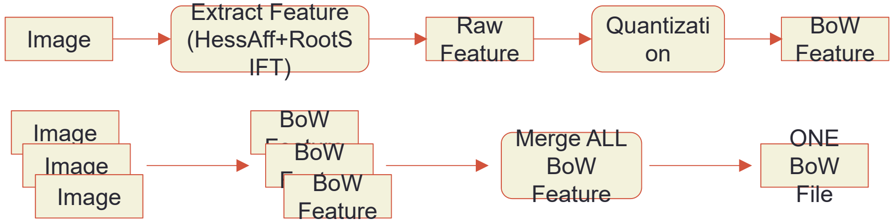
## Component 3: Feature Coding

- Input: a set of raw feature files (one keyframe → one feature file)
- Output: BoW representation for ALL keyframes (one BIG file after merging all BoW files corresponding to keyframes)
- Processing time: 4,700 secs/oxford5k
  - quantize.m: 0.9 sec/keyframe
  - merge_raw_bow_parallel.m: 235 secs
  - merge_raw_bow.m: 47 secs

- Workflow → quantize.m + merge_raw_bow_parallel.m + merge_raw_bow.m
  - quantize.m:
    - Build KDTree of cluster centers for NN search.
    - For each visual words, find k-NN (k=1 → hard assignment, k=3 → soft assignment).
    - Compute BoW for each keyframe.
  - merge_raw_bow_parallel.m
    - Merge sets of BoW into small parts.
  - merge_raw_bow_m
    - Merge parts into one BIG file.
    - Re-compute feature vector after calculating tf-idf

# Component 3: Feature Coding

- One image (.jpg) → one raw feature file (.mat) → one BoW representation file.
  - extract_hesaffine_rootsift_noangle_4image.m
  - quantize.m
- One dataset → merge ALL BoW representation files into ONE BIG file.
  - merge_raw_bow_parallel.m
  - merge_raw_bow.m

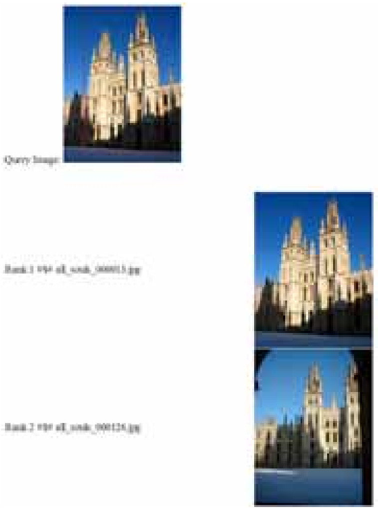



# Component 4: Inverted Index Construction

- Input: BIG BoW files of all images.
- Output: inverted index loaded into the memory.
- Processing time: 10-15 secs
- Large RAM is required
  - for hard assignment on database config:
- Workflow → load_inverted_index.m
  - Load pre-trained codebook and k-d tree → 5-10 secs.
  - Load all BOW features of dataset and build inverted index → 4-5 secs.

# Component 5: Search Process

- Input: Query image and region (x1_y1_x2_y2_imagename.jpg).
- Output: Search result in html file.
- Processing time: 6 secs/image (mainly for feature extraction - 3.7 secs and encoding - 1.5 secs)
- Workflow → process_query.m
  - Process query including: feature extraction, quantization, build BOW feature for query.
  - Search query BOW feature on inverted index structure and write ranked list to file.

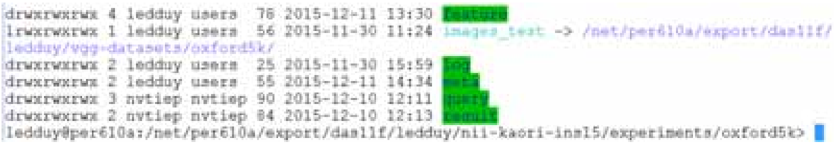


# Component 5: Web based Search Process

- Input: Query image and region (x1_y1_x2_y2_imagename.jpg).
- Output: Search result in html file.
- Processing time:
- Workflow → process_query_web.m
  - A user selects a link, upload to the server, and select query region.
  - Process query including: feature extraction, quantization, build BOW feature for query.
  - Search query BOW feature on inverted index structure and write ranked list to file.

## Practice - Step 0 - Preparation

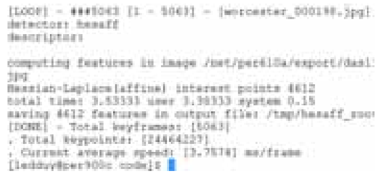
- Create a directory structure
  - *nii-kaori-ins15/code*: source code
  - *nii-kaori-ins15/experiments/oxford5k* (DB = oxford5k).
- Copy images of the test dataset into one dir→ all images in one dir.
  - *nii-kaori-ins15/experiments/oxford5k/images_test*



```
drwxrwxrwx 4 ledduy users  76 2015-12-11 13:30 feature
lrwxrwxrwx 1 ledduy users  56 2015-11-30 11:24 images_test -> /net/per610a/export/das11f/
ledduy/vgg-datasets/oxford5k/
drwxrwxrwx 2 ledduy users  25 2015-11-30 15:59 log
drwxrwxrwx 2 ledduy users  55 2015-12-11 14:34 meta
drwxrwxrwx 3 nvtiep nvtiep 90 2015-12-10 12:11 query
drwxrwxrwx 2 nvtiep nvtiep 84 2015-12-10 12:13 result
ledduy@per610a:/net/per610a/export/das11f/ledduy/nii-kaori-ins15/experiments/oxford5k>
```

## Practice - Step 1 - Feature Extraction (5 hours)

- Run raw feature extraction → Hessian-Affine keypoint detectors + RootSIFT
  - extract_hesaffine_rootsift_noangle_4image.m
  - lst_images.mat is generated.
- Output .mat files are located in feature dir
  - *nii-kaori-ins15/experiments/oxford5k/feature/hesaff_rootsift_noangle_mat*
- One .mat file → RootSIFT descriptor of feature points detected by Hessian-Affine keypoint detectors (~4,800 points/image).
- Processing time: 3.76 secs/image → 5 hours to finish.
- Can be run in parallel to reduce the processing time.



```
[LOOP] - ###5063 [1 - 5063] - [worcester_000198.jpg]
detector: hesaff
descriptor:

computing features in image /net/per610a/export/das1
jpg
Hessian-Laplace(affine) interest points 4612
total time: 3.53333 user 3.38333 system 0.15
saving 4612 features in output file: /tmp/hesaff_roo
[DONE] - Total keyframes: [5063]
, Total keypoints: [24464227]
, Current average speed: [3.7578] ms/frame
[ledduy@per900c code]$
```

## Practice - Step 2 - Codebook Generation

- Run samping feature
  - sampling_feat4clustering_vgg_hesaff.m
  - akm.py
- Sampling features: 100M for 1M codebook → 10 mins.
- AKM clustering
- Or use pre-built codebook.

## Practice - Step 3 - Feature Encoding (1.5 hours)

- Run quantization
  - quantize.m
- Processing time: 1.5 secs/image → 1.5 hours.

# Practice - Step 4 - Merge BoW (10 mins)

- Run 2 files sequentially
  - merge_raw_bow_parallel.m
  - merge_raw_bow.m
- Processing time: 10 mins.

# Practice - Step 5 - Build and Load Inverted Index

- Run building inverted index
  - load_inverted_index.m
- Processing time: 1-2 mins.

# Practice - Step 6 - Process Query

- Run query processing
  - process_query.m
- Processing time: 8 secs/image.
  - raw feature extraction: 3.76 secs,
  - feature encoding: 1.5 secs.
  - search:
  - write2output file: 2 secs.

# Experiments on Oxford Building dataset

- Oxford Building Dataset contains
  - 5062 images capture at Oxford (Oxford 5K)
  - And ~100K distractor images (Oxford 105K)
  - 55 queries with ground truth
- MAP for all queries: 65.64 (Oxford5K) and 59.44 (Oxford105K)

| | Ox5k | Ox105k |
|---|---|---|
| Triemb | 56.0 | 50.2 |
| SMK | 74.9 | - |
| ASMK | 78.1 | - |
| CroW | 59.2 | 51.6 |
| R-MAC | 66.9 | 61.6 |
| Ours | 65.6 | 59.4 |
| Ours (tuned) | 82.8 | 75.7 |

## Experiments on TRECVID Instance Search

- TRECVID Instance Search (INS) organized annually by NIST
- The dataset (from 2013 until now) contains:
  - ~ 244 videos from the BBC EastEnders program
  - ~ 300 GB in storage
  - ~ 464 hours in duration
- Query types:
  - Object
  - Person
  - Location
  - Compound of person and location (from 2016)

## TRECVID INS Query examples

| Easy topics | Difficult topics |
|---|---|
| • Simple visual context<br>• Stationary target<br>• Planar, rigid objects | • Small target<br>• Moving target: differences in camera angle, location<br>• Non-planar, non-rigid |

## Experiments on TRECVID Instance Search

- Trying with many detectors, descriptors and distance metrics


...

## Experiments on TRECVID Instance Search

- Detector config plays an important role in our baseline system

| Detector | Descriptor | MAP |
|---|---|---|
| Harris-Laplace | rootSIFT w/o angle | 27.17 |
| **Hessian-Affine (Surrey)** | **rootSIFT w/o angle** | **29.56** |
| Hessian-Affine (Perdoch) | rootSIFT w/o angle | 24.37 |
| MSER | rootSIFT w/o angle | 16.78 |
| **Average fusion** | | 31.31 |

## rootSIFT vs Color SIFT

| Detector | Descriptor | MAP |
|---|---|---|
| Hessian-Affine | root SIFT w/o angle | 29.56 |
| Hessian-Affine | Color SIFT w/o angle | 18.37 |
| MSER | rootSIFT w/o angle | 16.78 |
| MSER | Color SIFT w/o angle | 14.10 |

⇒ In average, color SIFT does not improve the performance

## Experiments on TRECVID Instance Search

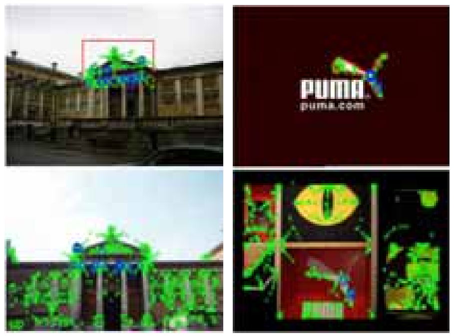
- Comparing symmetric distance with asymmetric one

| Detector | Descriptor | Distance | MAP |
|---|---|---|---|
| Hessian-Affine | root SIFT w/o angle | **$L_1$** | 28.13 |
| Hessian-Affine | root SIFT w/o angle | **$L_2$** | 28.83 |
| Hessian-Affine | root SIFT w/o angle | **asymmetric** | 29.56 |

⇒ asymmetric distance is better than symmetric ones (L1, L2), especially for small queries

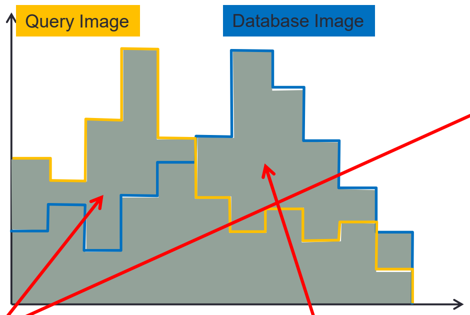
## Asymmetric dissimilarity

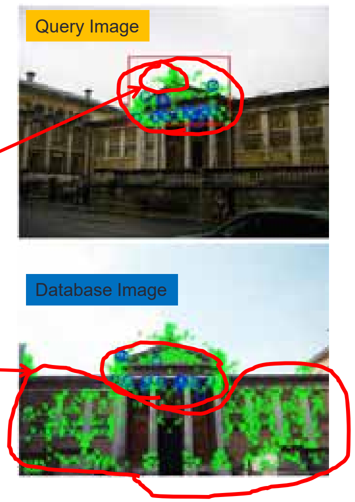
- There is inherent asymmetry between query and database images for object image retrieval
- Object region in query tends to be large or is explicitly indicated
- On the other hand, object regions in database images are not necessarily large and background regions may be large
- Typically used similarity metrics such as histogram intersection and Minkowski distances do not take this fact into account




Cai-Zhi Zhu, Hervé Jégou, and Shin'ichi Satoh, ``Query-Adaptive Asymmetrical Dissimilarities for Visual Object Retrieval,'' International Conference on Computer Vision (ICCV2013), 2013.

## Asymmetric dissimilarity



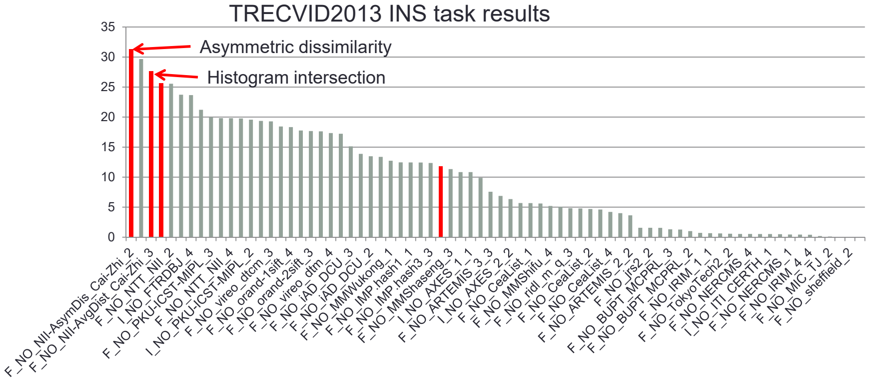

Minkowski distance (e.g. Euclid distance)
Measures difference
Histogram intersection
Measures common part

A part exists in query but not in database image
The smaller the better

A part exists in database image but not in query
Cannot be avoided

**Table. 1.** Symmetrical $L_1$ vs. (query-adaptive) asymmetrical $\delta_1$. The asymmetrical methods are compatible with an inverted index.

| | |
|---|---|
| Symmetrical | $\ell_1(\mathbf{Q}_i, \mathbf{T}_j) = \left\| \frac{\mathbf{Q}_i}{\|\mathbf{Q}_i\|_1} - \frac{\mathbf{T}_j}{\|\mathbf{T}_j\|_1} \right\|_1$ |
| Asymmetrical | $\delta_1(\mathbf{Q}_i, \mathbf{T}_j, w) = \|\mathbf{T}_j\|_1 - w \,\|\min(\mathbf{Q}_i, \mathbf{T}_j)\|_1$ |
| Query-adaptive | $\delta_1(\mathbf{Q}_i, \mathbf{T}_j, \alpha) = \|\mathbf{T}_j\|_1 - \alpha \frac{\sum_{j=1}^{N} \|\mathbf{T}_j\|_1}{\sum_{j=1}^{N} \|\min(\mathbf{Q}_i, \mathbf{T}_j)\|_1} \|\min(\mathbf{Q}_i, \mathbf{T}_j)\|_1$ |




# Symmetric vs Asymmetric distance

| Small Query | L2 | Asym |
|---|---|---|
|  | 71.45 | 79.15 |
|  | 4.35 | 10.01 |

| Small Query | L2 | Asym |
|---|---|---|
|  | 21.17 | 36.70 |
|  | 66.7 | 77.95 |

# Some bad cases when using BOW

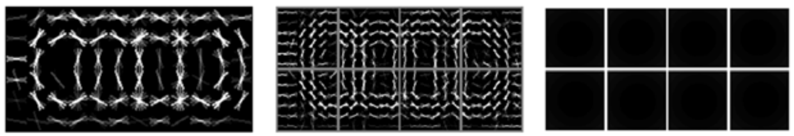
- BOW model gives bad performance when searching on
  - Small objects
  - Irrelevant object with similar shape


Irrelevant objects with similar shape or texture. Query objects are marked by red boundary. Light blue lines are visual word matches after spatial reranking

# Using DPM to rerank

- Deformable Part Models (DPM) is an algorithm for object detection. It was designed to handle
  - Small object
  - Partial occluded object
  - Deformable object

Positive example of a query topic (Audi logo). Negative images from Google Images with “things” keywords



DPM model visualization of Audi logo

# DPM reranking with average fusion

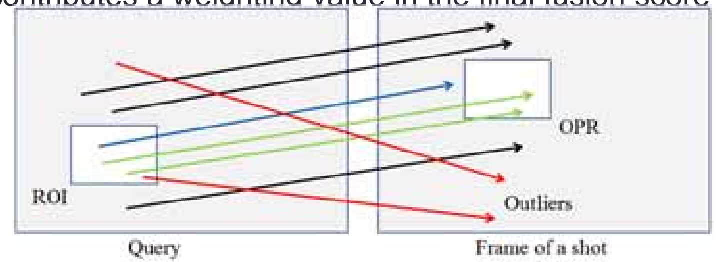
- Experiment on TRECVID INS2013 and INS2014

| Config | Description | INS2013 | INS2014 |
|---|---|---|---|
| BOW baseline | The baseline with BOW model | 29.56 | 25.01 |
| DPM reranking | Using DPM to rerank top 10K shots of BOW | 19.98 | 21.23 |
| BOW+DPM | Simple average fusion of BOW and DPM | 32.89 | 28.21 |

⇒ Average fusion of BOW and DPM improves the performance

# Advanced fusion of BOW and DPM

- We propose a new fusion score to make agreement between BOW and DPM. Each type of visual word match contributes a weighting value in the final fusion score




$$S_{new} = (1+N_d)^2(1+N_{fg}-N_d)\log_2(2+N_{bg})(w_1 S_{BOW} + w_2 S_{DPM})$$

where:

$N_d$ : number of shared words of foreground inside bounding box (green lines)
$N_{fg}$ : number of shared word of foreground (both blue and green lines)
$N_{bg}$ : number of shared word of background (black lines)
$w_1$ : weight of BOW score
$w_2$ : weight of DPM score

# Using DPM to rerank

- We fine tuned on many configurations to find the best formula

| ID | Fomular | MAP |
|---|---|---|
| 1 | $(1+N_d)^2(1+N_{fg}-N_d)(\log_2 N_{bg})Score_{base}$ | 27.7013 |
| 2 | $N_d^2(N_{fg}-N_d)(\log_2 N_{bg})(Score_{base}+1)$ | 27.4773 |
| 3 | $N_d^2(N_{fg}-N_d)(\log_2 N_{bg})Score_{base}^2$ | 25.7851 |
| 4 | $N_d^2(N_{fg}-N_d)(\log_2 N_{bg})Score_{base}^3$ | 26.0116 |
| 5 | $N_d^3(N_{fg}-N_d)(\log_2 N_{bg})Score_{base}$ | 28.8457 |
| 6 | $(N_{fg})(N_{fg})(\log_2 N_{bg})Score_{base}$ | 27.7235 |
| 7 | $N_d(N_{fg}-N_d)(\log_2 N_{bg})Score_{base}$ | 27.6632 |
| 8 | $N_d^2(N_{fg}-N_d)(\log_2 N_{bg})Score_{base}$ | 29.2378 |
| 9 | $N_d^3(N_{fg}-N_d)^2(\log_2 N_{bg})Score_{base}$ | 28.4936 |
| 10 | $N_d^2(N_{fg}-N_d)^2(\log_2 N_{bg})Score_{base}$ | 27.9780 |
| 11 | $N_d^3(N_{fg}-N_d)^{1/2}(\log_2 N_{bg})Score_{base}$ | 28.3208 |
| 12 | $N_d^2(N_{fg}-N_d)^{1/2}(\log_2 N_{bg})Score_{base}$ | 28.9218 |
| 13 | $N_d^{5/2}(N_{fg}-N_d)(\log_2 N_{bg})Score_{base}$ | 28.7754 |
| 14 | $N_d^{5/2}(N_{fg}-N_d)^{1/2}(\log_2 N_{bg})Score_{base}$ | 28.2000 |
| 15 | $N_d^2(N_{fg}-N_d)^2(N_{bg})Score_{base}$ | 27.7790 |
| 16 | $N_d^2(N_{fg}-N_d)^{3/2}(\log_2 N_{bg})Score_{base}$ | 28.6295 |
| 17 | $N_d^3(N_{fg}-N_d)^{3/2}(\log_2 N_{bg})Score_{base}$ | 26.9798 |
| 18 | $N_d^2(N_{fg}-N_d)(\log_e N_{bg})Score_{base}$ | 29.2378 |
| 19 | $N_d^2(N_{fg}-N_d)(\log_2 N_{bg})Score_{base}$ | 29.2378 |
| 20 | $N_d^{5/2}(N_{fg}-N_d)(\log_2 N_{bg})Score_{base}$ | 29.2267 |
| 21 | $N_d^3(N_{fg}-N_d)(\log_2 N_{bg})Score_{base}$ | 29.1554 |
| 22 | $N_d^2(N_{fg}-N_d)\sqrt{N_{bg}}Score_{base}$ | 28.9850 |

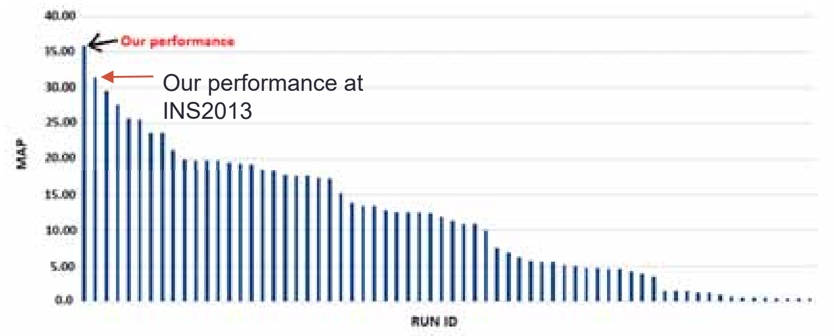
# DPM reranking with average fusion

- Experiment on TRECVID INS2013 and INS2014

| Config | Description | INS2013 | INS2014 |
|---|---|---|---|
| BOW+DPM | Simple average fusion of BOW and DPM | 32.89 | 28.21 |
| Fusion BOW+DPM | Final fusion of BOW and DPM | 35.42 | 32.49 |

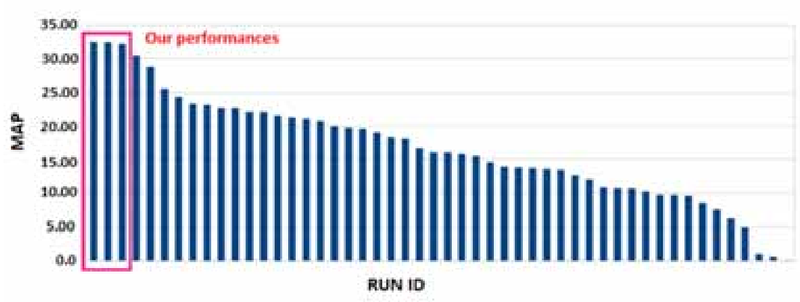
⇒ The proposed method significantly improves the average fusion

## Our performance compare to other teams




TRECVID INS 2013

## Our performance compare to other teams




TRECVID INS 2014

## Conclusion

- Brief explanation of TRECVID Instance Search
- Wild instance search benchmark
- BoW-based NII baseline system is explained
- Good for instance search of objects and landmarks (scene)
- Asymmetric dissimilarity is explained (included in the baseline)
- DPM-based reranking (not included.  yet...)