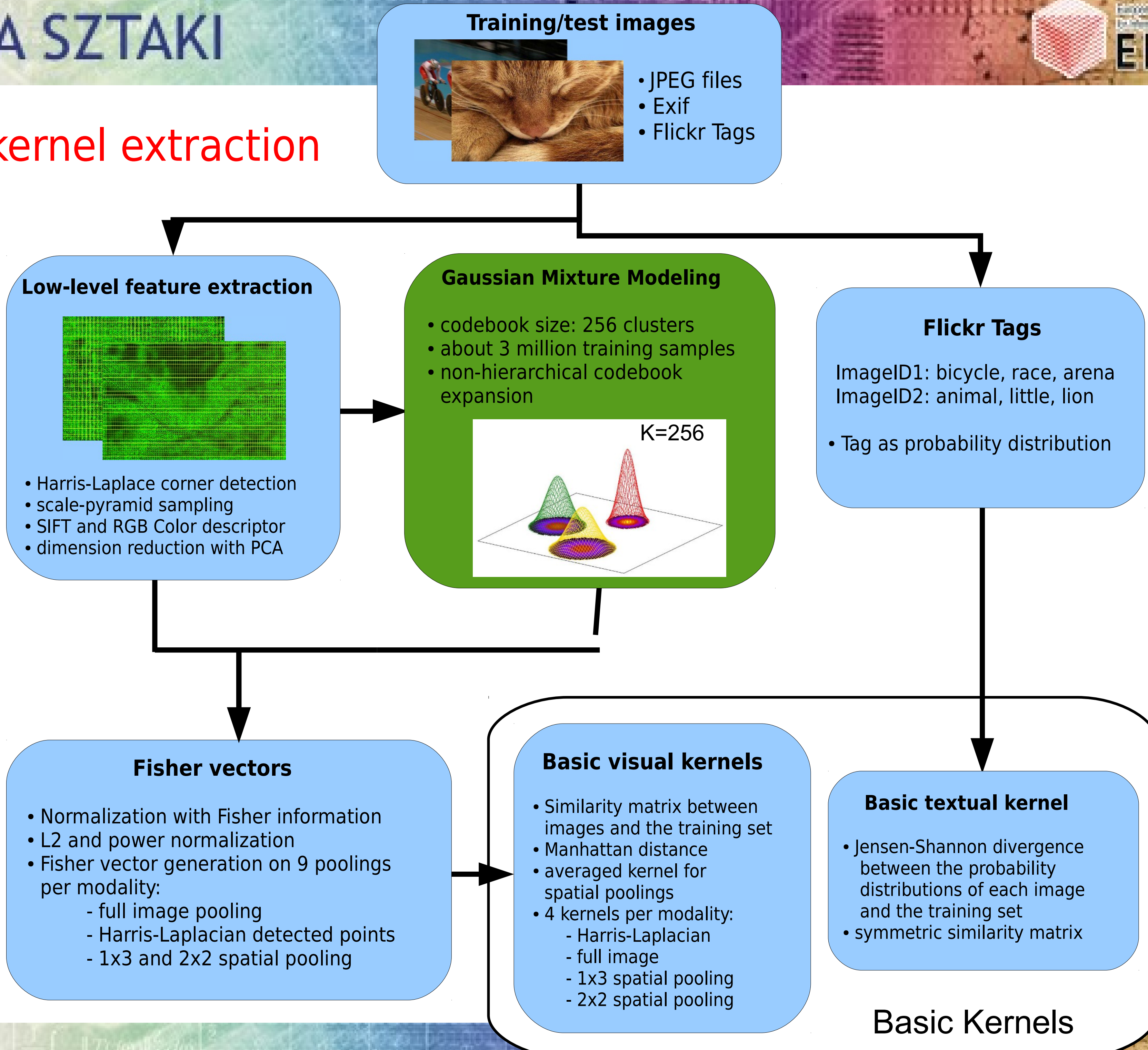
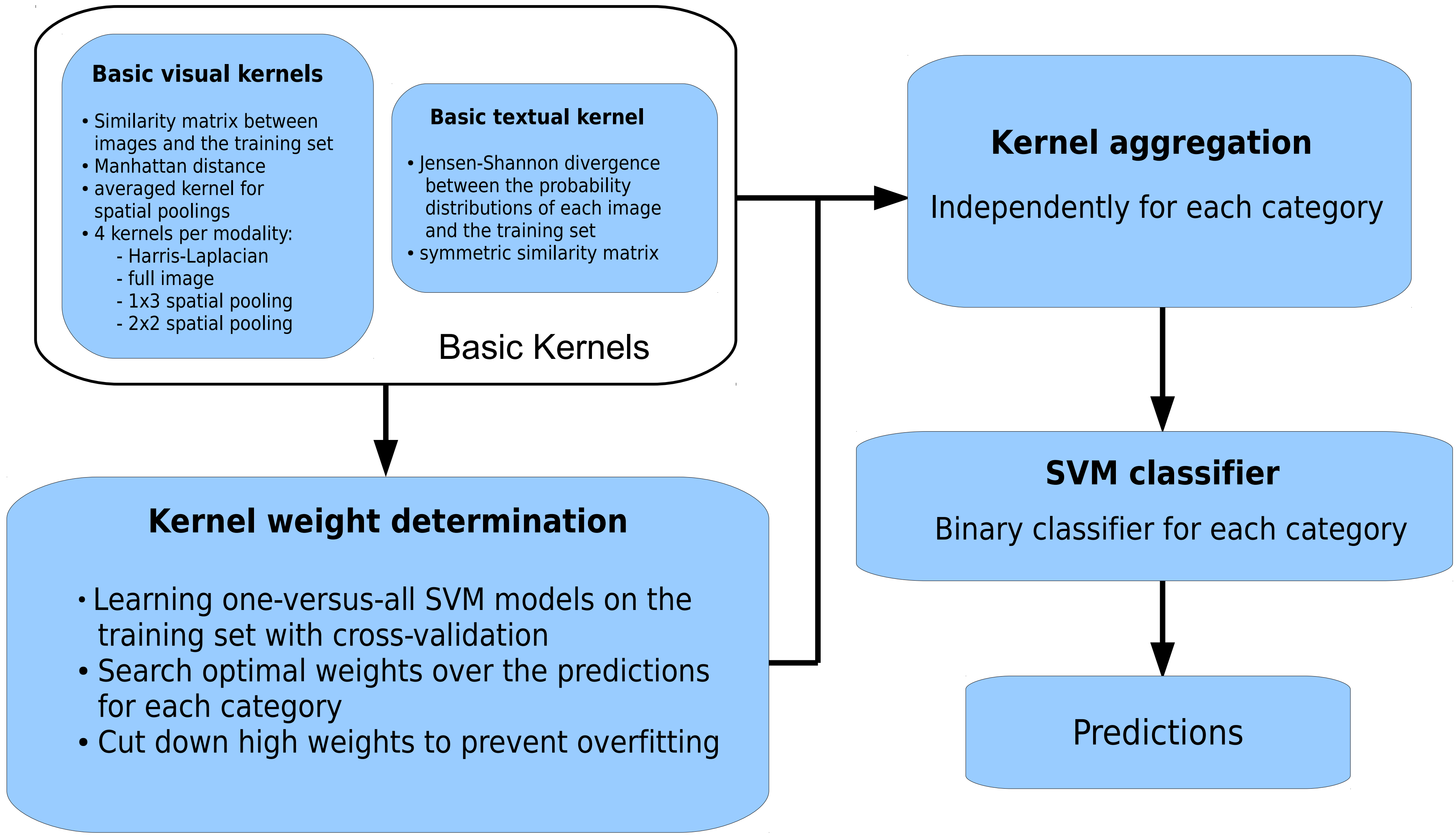
SZTAKI @ ImageCLEF 2011

Bálint Daróczy
joint work with
András Benczúr, Róbert Pethes
Data Mining and Web Search Group
Computer and Automation Research Institute
Hungarian Academy of Sciences

Basic kernel extraction



Learning method

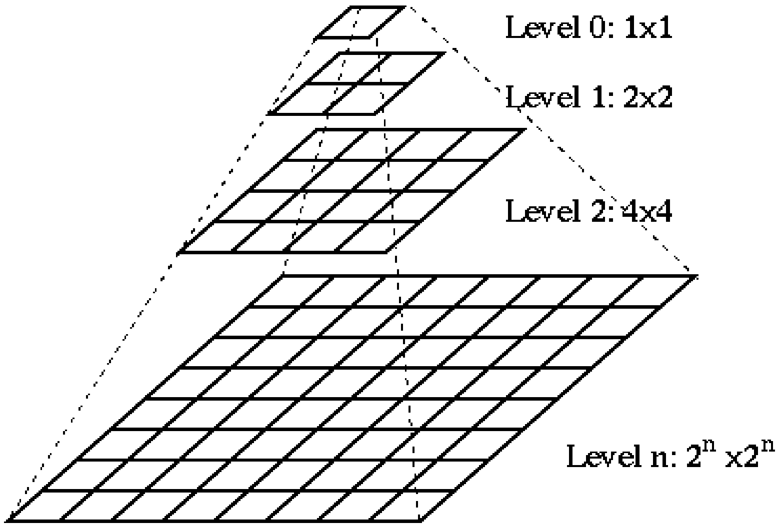


Low level image descriptors

Sampling strategies:

Dense pyramid

Harris-Laplacian



Patch descriptors:

Scale Invariant Feature Transform (SIFT) [1]

RGB Colour moments (why not HSV?)

~~Histogram of Oriented Gradients (HOG)~~

~~Locale Binary Pattern (LBP)~~

Sampling strategies

Dense pyramid:

3 scales

(scale factor=1.6)

„global” type problems

sampling at 4 pixels on every scale

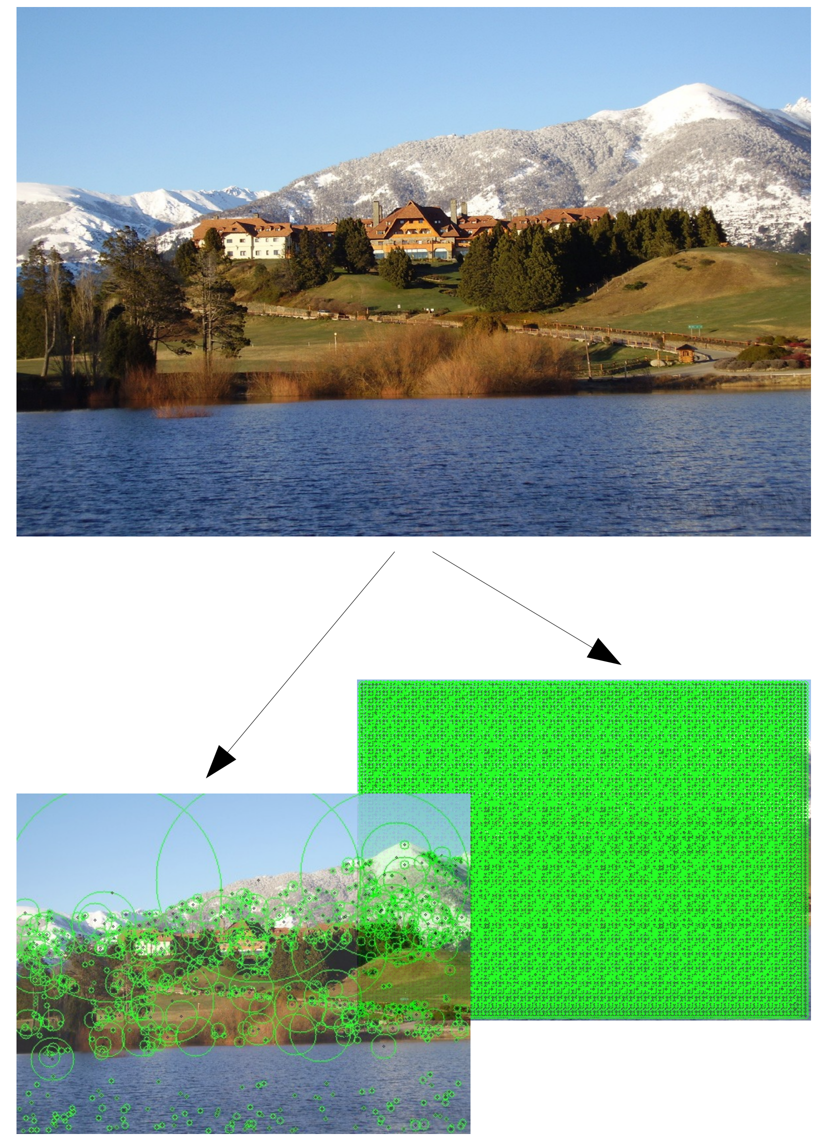
→ ~12-13k descriptors per image

Harris-Laplacian:

prevent homogeneous areas

„object” type problems

→ ~1.5-2k descriptors per image



Gaussian Mixture Modeling

Based on standard expectation maximization (EM)

- not hierarchical
- transformation of feature space → depends on the original space
- how to prevent zero variance?

Algorithm 1 The GPU GMM algorithm

Input: data points $\{x_i\}_{i=1}^{N}$, dimension D, mixture number K.

Output: $\{P_i\}_{i=1}^{K}$, $\{\mu_i\}_{i=1}^{K}$, $\{\sigma_i\}_{i=1}^{K}$, where $\mathcal{N}_i$ is normally distributed with parameters (μ_i, σ_i) and σ_i is assumed to be diagonal.

Initialize the Gaussian distributions with random parameters

repeat

  for all n and k **do**

    Expectation: Compute likelihood $p_{nk} = \frac{f_k(x_n)P_k}{\sum_{i=1}^{K} f_i(x_n)P_i}$ where f_i is the density of $\mathcal{N}_i$

  for all k and d **do**

    Maximization: compute $P_k = \frac{\sum_{n=1}^{N} p_{nk}}{N}$, $\mu_{kd} = \frac{\sum_{n=1}^{N} p_{nk} x_{nd}}{\sum_{n=1}^{N} p_{nk}}$ and $\sigma_{kd}^2 = \frac{\sum_{n=1}^{N} p_{nk}(x_{nd} - \mu_{kd})^2}{\sum_{n=1}^{N} p_{nk}}$

until until converge

Implementation of GMM on GPGPU

Numerical bottleneck:

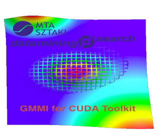
- Naive GMM implementation frequently underflows when computing density σ^2_{kd}
- Results in division by zero in next iteration BUT can be fully solved by using logarithms and “buckets”:

$$\log(p_{nk}) = \log(f_k(x_n)) + \log(P_k) - \log \sum_{i=1}^{K} \exp(\log(f_i(x_n)) + \log(P_i))$$

$$\log(P_k) = \log \sum_{n=1}^{N} \exp \log(p_{nk}) - \log N$$

$$\mu_{kd} = \frac{\sum_{n=1}^{N} \exp(\log(p_{nk}) + \log(x_{nd}))}{\sum_{n=1}^{N} \exp \log(p_{nk})}$$

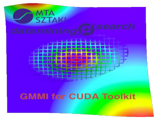
$$\sigma^2_{kd} = \frac{\sum_{n=1}^{N} \exp(\log(p_{nk}) + 2\log(x_{nd} - \mu_{kd}))}{\sum_{n=1}^{N} \exp \log(p_{nk})}$$



For details see our freely downloadable code for CUDA and x86 CPU-s with test examples: http://datamining.sztaki.hu/?q=en/GPU-GMM

Fisher calculation on GPGPU

- Highly parallel algorithm [2]
- With K=256 and the original dimension is D=96 the number of independent calculations is D_{fisher}=K*D*2=49152
- Under reasonable conditions, time depends only on #low level features
- Sparsity? We applied the same methods to calculate p_{nk} as in our GMM algorithm → Fisher vectors are not sparse!
- Fisher, L2 and power normalization can be integrated efficiently into the Fisher calculation algorithm
- Our CUDA implementation is 4x faster as a good implementation on a single CPU core and 16x faster when the CPU is computing the same algorithm (K=256, N=10k , D=96)
  Note: if we apply the faster, approximated CPU implementation, the Fisher vectors are sparser
- the advantage is higher with more Gaussians, more type of poolings or higher number of feature vectors



For details see our freely downloadable code for CUDA and x86 CPU-s with test examples: http://datamining.sztaki.hu/?q=en/GPU-GMM

Poolings and visual kernel calculation

We extracted Fisher vectors on four different poolings:

- all of the extracted features (full)
- only the features detected by the Harris-Laplacian
- spatial pyramids: 1x3 and 2x2 [3]

Dimensional problems:

- even the Fisher vector calculated over the lowest number of feature vectors (Harris-Laplacian) is not sparse
- the final dimension of extracted Fisher vectors is 9xK*2*D per image
- how to weight differently the higher dimensional spatial pyramid vectors and the full and the Harris-Laplacian vectors ?

→ use pre-computed normalized kernels!

Calculate for each pooling p $K_p(F_{i\_p},F_{j\_p})$ where F_{i_p},F_{j_p} is the Fisher vectors extracted from the i-th image of the full dataset and the j-th image of the training set

- if we adopt kernels instead of Fisher vectors the dimension of the high level descriptor of an image will be lower (8k vs. 40K/49k.)

Pre-computed visual kernel

We can combine Fisher vectors with different modalities and visual methods using distance matrices as kernels.

Our pre-liminary results on the Pascal VOC 2007 dataset[4] indicated to use Manhattan distance for normalized Fisher vectors:

$$K_{k_{visual}}(X, I_t) = \frac{dist_{Manhattan}(F_k(X), F_k(I_t))}{\max *_X \arg\max_t K_k(X, I_t)}$$

Pre-liminary results: The L1-kernel outperforms both the Fisher kernel and the Super-Vector[5]

RGB color descriptors, dense sampling	Normalized Fisher vector #Gaussians=256		Super-Vector, K-means, s=0.1	
	Dot product	Pre-computed kernel	K=256	K=2048
Dimension	49152	8000	20480	163840
AvgMAP	0.4244	0.4501	0.4081	0.4279

Curse of sampling

(pre-liminary results)

To select the optimal number of Gaussians we calculated Gaussian Micture models with K=128/256/512/1024. We tested the resulted GMMs with Harris-Laplacian detected and dense sampled features on the Pascal VOC 2007 dataset:

RGB color descriptor	K=128	K=256	K=512	K=1024
Harris-Laplacian	0.4019	0.4057	0.4070	
Dense sampling	0.4377	0.4501	0.45803	0.4677

The results of Harris-Laplacian pooling are more disappointing if we calculate Fisher vectors for SIFT features:

Grayscale SIFT	K=128	K=256	K=512	K=1024
Harris-Laplacian	0.4327	0.443	0.4295	0.4157
Dense sampling	0.4584	0.4669	0.4579	

Textual kernel and combination

Textual kernel

To use the provided Flickr tags we thought of the tags of an image as probability distributions. To select the optimal similarity measure (kernel) for tag based methods we splitted the training set into two parts and trained linear SVM models with the libSVM package[6]. The best performing textual kernel:

$$K_{k_{textual}}(X, I_t) = dist_{Jensen-Shannon}(X, I_t)$$

Combination

This splitting gave us the opportunity to establish a group of visual and textual kernels and pre-define weights over them. To extract weights we calculated the output scores of the basic kernel based svm classifiers → to prevent overfitting the resulted weights should be in [-11,11].

To treat all the kernels equally we calculated pre-computed kernels as distance matrices and transformed them to meet the following requirements:

- symmetric if possible
- normalized with the maximum
- K(X,X)=0

The final combined kernel:

$$K(X,Y)_c = \frac{1}{|K|} \sum_{k=1}^{K} \alpha_{ck} \sum_{t=1}^{T} K_k(X, I_t) * K_k(Y, I_t)$$

Performance of the pre-computed kernels

We splitted the training set into two equal sized sets (2x4k images)

- the performance of the basic visual kernels are poor
- the Jensen-Shannon divergence overperforms the KL kernels
- the textual and visual kernels complement each other

MAP/basic kernel	Avg	85-Baby	71-dog	10-Summer	6-Landscape	24-Mountains
Color full	0.2519	0.0465	0.1487	0.3108	0.6203	0.2940
Color HL	0.2324	0.0509	0.1344	0.3145	0.5794	0.2103
Color sp1x3	0.2528	0.0365	0.1839	0.2853	0.5912	0.2712
Color sp2x2	0.2516	0.0301	0.1473	0.2779	0.6026	0.2560
SIFT full	0.2741	0.0294	0.1718	0.2597	0.7195	0.4003
SIFT HL	0.2414	0.0215	0.1916	0.2449	0.6737	0.2428
SIFT sp1x3	0.2827	0.0345	0.1817	0.2749	0.7226	0.3478
SIFT sp2x2	0.2778	0.0222	0.1927	0.2532	0.7228	0.3717
KL symm	0.2554	0.2975	0.6328	0.1722	0.4607	0.1571
KL nonsymm	0.2498	0.3227	0.6262	0.1653	0.4479	0.1648
JS stammed	0.3110	0.3380	0.7460	0.1719	0.5884	0.2410
JS nost	0.3140	0.3227	0.7743	0.1694	0.5550	0.2274

Results

	Kernel aggregation	MAP	EER	AUC
1. visual + textual run3	weighted	0.438744	0.243574	0.827621
1. visual + textual run3	weighted	0.436294	0.241691	0.827747
1. visual + textual run3	averaged	0.420406	0.243885	0.828322
4. visual run2	averaged	0.371795	0.261183	0.809399
5. visual run1	averaged	0.367054	0.264328	0.805142
6. textual run	only one	0.345616	0.338127	0.717966

- the kernel weightning resulted 0.0183 gain in MAP over the averaged kernel
- despite the poor results of the basic visual runs the averaged visual run outperformed the textual run

Some examples for basic kernel weights → similarity to the basic splitted results?

	Color full	Color HL	Color 1x3	Color 2x2	SIFT full	SIFT HL	SIFT 1x3	SIFT 2x2	Jensen-Shannon
Dog	-0.25	0.3	0.1	0.3	0.35	0.3	0.0	0.05	10.45
Baby	1	1.1	-0.1	-0.1	-0.2	-0.3	0.9	-0.1	10.0
Mountains	0.7	0.6	0.0	0.0	0.9	-0.15	-0.05	0.2	3.6
Landscape	0.05	0.7	1.1	0.0	0.2	1.3	0.9	2.1	4

Confidence score and binary annotation

Since the output of our classifier was a summarized values of the weighted dot-products of the support vectors and the test instances, we calculated the confidence scores with the sigmoid function:

$$Prediction_{float} = \frac{1}{1 + exp^{-1 * svm_{output}}}$$

For the example-based evaluation we needed to define a mapping from the floating point predictions into a binary annotation. Threshold selection based on:

1. backsubsititution over training set
2. achieving the same +/- ratio as in training set

The previous had much higher F-score (0.593088 vs. 0.545341) and higher Semantic R-Precision (0.71928 vs. 0.70853).
Note: from our submissions the averaged visual kernel had the highest Semantic R-Precision (0.72902450)

Conclusions

1. We used Fisher vector as high-level image descriptor

- two low level features: SIFT and RGB Color
- Gaussian Mixture Model with 256 Gaussians

2. We adopted Jensen-Shannon divergence for Flickr tags

3. We calculated pre-computed kernels and combined them before the SVM based classification procedure

4. Future plans: segmentation? 100K+ categories? Combination with search engines?

Thank you!

MTA SZTAKI ERCIM

References

1. D.G. Lowe: „Object recognition from local scale-invariant features. In International", Conference on Computer Vision, volume 2, pages 1150–1157, 1999
2. F. Perronnin and C. Dance: „Fisher kernels on visual vocabularies for image categorization", In IEEE Conference on Computer Vision and Pattern Recognition, 2007. CVPR'07, pages 1–8, 200
3. C. Schmid S. Lazebnik and J. Ponce: „Beyond Bags of Features: Spatial Pyramid Matching for Recognizing Natural Scene Categories", In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, New York, June 2006, 2006
4. Mark Everingham: "Overview and results of classification challenge", In ICCV,Pascal VOC 2007, Workshop, 2007, 200
5. Kai Yu. Tong Zhang Xi Zhou and Thomas Huang: "Image Classification using Super-Vector Coding of Local Image Descriptors", In 11th ECCV,2010, 201
6. Chih-Chung Chang and Chih-Jen Lin: "LIBSVM: a library for support vector machines", 2001. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm