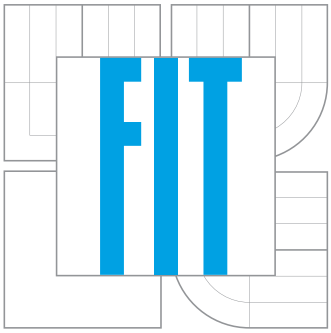
2D Classification and Real-Time Object Detection

AMIDA Technology Package Description

Roman Juránek, Pavel Svoboda

Graph@FIT
Brno University of Technology
Faculty of Information Technology
Božetěchova 2
612 66 Brno, Czech Republic



Developers:
Roman Juránek, Michal Hradiš, Pavel Zemčík, Adam Herout, Jiří Havel, Radovan Jošth, Lukáš Polok, Martin Žádník, Vítězslav Beran

Zemčík Pavel, `zemcik@fit.vutbr.cz`, responsible for AMIDA project
Herout Adam, `herout@fit.vutbr.cz`, responsible for Graph@FIT group

Last modified: December 25, 2009

Contents

1 Purpose of the Technology

This package contains a system that provides real time object detection. By the objects, in this case, we mean anything with stable pattern (e.g. face, car rear, etc.). The detection is based on a classifier trained by AdaBoost (or its modification) algorithm. Example of an output of face detection is shown in the Figure 1.

Figure 1: Detection of faces in a photography (source: Internet).

This package could be used in various applications. Most prominent of them are detection of faces and facial features. Other applications include detection of persons, licence plates, cars (sides and rears) and even low level detection tasks like interesting point detection. The advantage of this system is that all the applications share the same framework and only classifier determines what object is detected. Another advantage is low computational cost of the detection. The object detection is typically a part of larger systems like driving assistants, security systems or industrial quality control systems where it serves as a start for high level processing – e.g. object recognition, tracking etc.

1.1 Contents of the Package

The package contains:

- *Runtime framework* – The detection system and its source codes. The system includes SSE, GPU and CUDA detection engines.
- *Classifiers* – Example data files with classifier descriptions. Examples are: face detection and traafic sign detection.
- *Videos* – Face detection in news video and video of a system working in real-time.

2 Technical Description

This section describes technological background of the object detection implemented in this package. The section 2.1 deals with AdaBoost machine learning algorithm and its extensions, the section 2.2 describes feature extraction algorithms which are used to construct classifiers. The main parts of the package – training and runtime framework – are described in the sections 2.3 and 2.4.

2.1 General Object Detection with Boosting

The AdaBoost [1, 2, 3] is an algorithm that incrementally builds a *strong classifier* from simple *weak hypotheses*. The hypothesis can be for example convolution response extracted form an input image. The input for the training is a labeled training set. The output is a classifier, i.e. selected descriptive features. The classifier response is based on a linear combination of responses of the selected features. Although the elementary weak classifiers are very simple, the strong classifier exhibit very good performance on variety of detection and classification tasks.

The evaluation of the AdaBoost classifier can be computationaly demanding because for every sample all features need to be evaluated before the decision is reached. The Cascade [3] is a method that builds a chain of progressively more complex classifiers that can very quickly reject background samples. This decreases computational cost to the acceptable level for real time object detection. Another method similar to the Cascade is WaldBoost [4] which, based on Walds's Sequential Probability Ratio Test [5], builds classifier with optimal decision strategy (time to decision).

The object detection is performed by evaluating the classifier on all positions and scales. The positions with positive responses of classifiers can

be clustered to remove possible multiple detections of objects. The detected areas can be subjected to further processing like for example tracking, object recognition etc.

2.2 Features for Object Detection

The pattern recognition by statistical classifiers is based on low level features which extract information from image. As the rapid object detection often uses exhaustive search over image, large number of features have to be evaluated fast, that's why simple features are often used. Typical example of features are Haar wavelets [3, 6] or Local Binary Patterns [7].

This section describes image feature extractors used in the training framework and real-time object detection runtime framework.

Local Binary Patterns. The LBPs are widely used in texture processing. They were introduced by [8, 9]. LBPs in their basic form capture information about local textural structures by thresholding samples from a local neighborhood by its central value and forming the pattern code where each sample represents a single bit.

Typically, the circular neighborhood with 8 samples is used (8 bit pattern), but other variants are also possible. LBP are most frequently used in combination with local histograms to describe a local image area and segment the image. The LBP is not rotationally invariant, it is dependent on which sample is considered first when forming the code. Rotational invariance can be achieved by normalization of the pattern by shifting the bits – the lowest value is selected as the LBP result. The LBPs exhibit very good performance when used as features in object detection [7].

Local Rank Functions. The LRFs [10] are based on the idea that the intensity information in the image can be well represented by the order of the values (intensities) of the pixels or small pixel regions (e.g. summed 2×2 pixel rectangular areas). The feature input is a set of samples from image. The output is calculated from ranks of two (or more in general case) selected items from the set. The ranks are subtracted (Local Rank Difference – LRD) or treated as a vector (Local Rank Pattern – LRP).

The features have been designed so that they have equal descriptive and generalization power as their state-of-the-art alternatives, but at the same

time to be efficiently implementable in hardware. These features prove to be efficient, not only in the hardware implementations (tested in FPGA chips), but also when implemented using the SSE instruction set of the contemporary CPU's and implemented in the graphics processors (GPU's).

2.3 Training Framework

The base component of the package is training framework [11] which produces classifiers. Based on training set of images and configuration settings, the software generates a classifier with desired properties. The framework implements variety of training algorithms (AdaBoost, WaldBoost, Gentle AdaBoost) and supports a number of weak hypotheses types and low level features.

The framework is written in C/C++ language in a modular fashion and thus it is easy to extend with new features. It uses some freeware components like libxml2 for xml processing, OpenCV for image processing, OpenMP for paralellization of training and Gnuplot for visual output of training statistics.

Training Data. Basic input is a set of image files. The images can be read in two ways: *a)* each subarea is taken as a sample (suitable for *background* class) or *b)* using image annotation, the framework can automatically extract anotated areas (optinally with some random transformation) and use the images as training samples (suitable for *object* class). The training data and their division to classes is described in simple dataset XML file which defines data *subsets*.

Configuration. The training process is set up by a configuration file. It is simple XML file which defines several parameters: algorithm settings (number of features, error rates, etc.), data input settings (dataset, number of samples, etc.) and weak hypotheses settings (type of hypotheses and features used in training). Finally it defines testing parameters, i.e. what data to use and what tests to perform.

Outputs. Once the training is finished, the framework produces a classifier file which is a XML structure with selected hypotheses and their parameters. Further it provides log file with transcription of training process and finally

files with training and testing statistics – ROC and PRC curves,speed statistic and sample rejection statistic.

2.4 Object Detection Runtime Framework

The classifiers produced by the training framework presented in previous section can be used in the detection framework. Either directly loaded from XML file, or they can be transformed to form of header file `.h` and statically compiled to a program. The framework supports LRD and LRP (subsets of Local Rank Functions – Section 2.2) and LBP features.

The detection engine searches for objects in image performing exhaustive search over all positions and scales (and even rotations if desired). Detected hypotheses are optionally processed by a Non-maxima suppression algorithm which removes possible multiple detections of objects. The coordinates, sizes and orientations of detected objects are then passed to user to perform high level processing (tracking, recognition, etc.).

There are different versions of optimized engines. The SSE engine exploits the SIMD instruction set on contemporary CPUs, and particulary SSE2 which supports simultaneous processing of sixteen 8 bit integers in single register. The GPU and GP-GPU implementations uses properties of contemporary graphic cards – shaders and multiprocessor CUDA architecture. The last engine version is implemented on programmable hardware.

SSE [10, 12, 13] The imahe in this implementation is represented as a pyramid created by downscaling. The scaling is done by highly optimized SSE 8-to-7 bilinear scaler and 2-to-1 averaging scaler. For each pyramid level, convolution images are created and rearranged in the memory to layout suitable for SSE evaluation. The features are then evaluated using paralel processing by SSE instructions. This implementation is roughly six times faster than the one with no optimizations. It also uses OpenMP for further performance improvement.

GPU [10, 14]. The classification in the GPU is realized as a sequence of fragment shaders that is executed on a texture loaded from system memory to GPU memory. The output (texture with responses of the classifier) is loaded back to the system memory. Classifier parameters are supplied to the

GPU as textures. The classifier parameters are represented as RGB texture, the response table as grayscale texture.

CUDA [10, 15]. This implementation uses the parallel processing by CUDA architecture implemented on modern graphic cards. The detection is based on a processing kernel that is executed on each position of image performing classification task. The architecture contains hundreds of cores, so hundreds of image subwindows can be processed in parallel.

Both, the GPU and CUDA implementations are roughly two times faster than the SSE implementation which makes them suitable for HD video processing with minimal usage of CPU. The CPU can thus be used for complex high level processing of the detection results.

FPGA [16, 10] The LRD image features and WaldBoost based classification was implemented in programmable hadrware. Designed architecture is similar to specialized processor. The classifier and its parameters are supplied as a sequence of instructions which are avaluated on input image. A processing module with FPGA can work on 21 fps with $640 \times 480px$ image or 6 fps with $320 \times 240px$ image.

2.5 Technical Data

The input is grayscale image either static of from camera, the output is a set of coordinates in image where an objects are detected.

The system can operate on standard PC with Intel CPU. Parts of the system requires CPU with SSE2 instruction set (which is standard on todays PCs) or CUDA capable graphic card. The detection typically works on $5 - 50$ fps (depending on size of input image, detection settings, classifier and CPU). There is no upper speed limit so in some cases (small input, constrained conditions), the detection can process even hundreds or frames per second. On the other hand, when large input image is supplied it can take even seconds to process it.

3 Limitations and Possible Extensions

Although the training framework support different algorithms and feature types, the detection framework supports only WaldBoost classifiers that are

homogenous (i.e. only one feature type is used). Length of the classifiers is not limited for PC but the FPGA implementation supports classifiers with lenght up to 256 weak hypotheses (which is caused by limited amount of memory on the chip). This limitation is not very serious as the 256 hypotheses is enough for most applications. Features used in classifiers are restricted to be LRD (SSE, GPU, CUDA, FPGA), LRP (SSE, CUDA) or LBP (SSE, CUDA) and size of feature block is restricted to maximum $2 \times 2px$. As our experiments proved, this limitation does not influence classification precision by any measurable amount.

Future work on the runtime framework will focus on implementing LRP and LBP features on GPU and update of CUDA engine to OpenCL. Also we are currently experimenting with new type of image feature which will be implemented on both PC (CPU and GPU) and FPGA.

References

[1] Yoav Freund and Robert E. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. In *EuroCOLT '95: Proceedings of the Second European Conference on Computational Learning Theory*, pages 23–37, London, UK, 1995. Springer-Verlag.

[2] Robert E. Schapire and Yoram Singer. Machine learning, 37(3):297-336, 1999. improved boosting algorithms using confidence-rated predictions, 1999.

[3] Paul Viola and Michael Jones. Rapid object detection using a boosted cascade of simple features. pages 511–518, 2001.

[4] Jan Sochman and Jiri Matas. Waldboost – learning for time constrained sequential detection. In *CVPR '05: Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 2*, pages 150–156, Washington, DC, USA, 2005. IEEE Computer Society.

[5] A. Wald. *Sequential Analysis*. John Wiley and Sons, Inc., 1947.

[6] Constantine P. Papageorgiou, Michael Oren, and Tomaso Poggio. A general framework for object detection. *Computer Vision, IEEE International Conference on*, 0:555, 1998.

[7] Lun Zhang, Rufeng Chu, Shiming Xiang, ShengCai Liao, and Stan Z. Li. Face detection based on multi-block lbp representation. In *ICB*, pages 11–18, 2007.

[8] Timo Ojala, Matti Pietikäinen, and David Harwood. A comparative study of texture measures with classification based on featured distributions. *Pattern Recognition*, 29(1):51 – 59, 1996.

[9] Timo Ojala, Matti Pietikäinen, and Topi Mäenpää. Multiresolution gray-scale and rotation invariant texture classification with local binary patterns. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(7):971–987, 2002.

[10] Adam Herout, Pavel Zemčík, Michal Hradiš, Roman Juránek, Jiří Havel, Radovan Jošth, and Martin Žádník. *Low-Level Image Features for Real-Time Object Detection*, page 25. IN-TECH Education and Publishing, 2009.

[11] Michal Hradiš. Framework for research on detection classifiers. In *Proceedings of Spring Conference on Computer Graphics*, pages 171–177. Comenius University in Bratislava, 2008.

[12] Adam Herout, Michal Hradiš, Roman Juránek, and Pavel Zemčík. Implementation of the "local rank differences" image feature using simd instructions of cpu. In *Proceedings of Sixth Indian Conference on Computer Vision, Graphics and Image Processing*, page 9, 2008.

[13] Roman Juránek, Adam Herout, and Pavel Zemčík. Impelementing local binary patterns with simd instructions of cpu. In *Proceedings of Winter Seminar on Computer Graphics*, page 5. West Bohemian University, 2010, (submited).

[14] Lukáš Polok, Adam Herout, Pavel Zemčík, Michal Hradiš, Roman Juránek, and Radovan Jošth. "local rank differences" image feature implemented on gpu. In *Proceedings of the 10th International Conference on Advanced Concepts for Intelligent Vision Systems*, Lecture Notes In Computer Science; Vol. 5259, pages 170–181. Springer Verlag, 2008.

[15] Adam Herout, Radovan Jošth, Pavel Zemčík, and Michal Hradiš. Gp-gpu implementation of the "local rank differences" image feature. In *Proceedings of International Conference on Computer Vision and Graphics*

2008, Lecture Notes in Computer Science, pages 1–11. Springer Verlag, 2008.

[16] Pavel Zemčík and Martin Žádník. Adaboost engine. In *Proceedings of FPL 2007*, page 5. IEEE Computer Society, 2007.