Fast Face Detection in One Line of Code

Michael Zucchi, B.E. (Comp. Sys. Eng.)

Unaffiliated, unfunded, personal research.

Keywords: local binary pattern, lbp, simd, efficient, gpu, opencl, classifier, impossible

Abstract

A memory-space and CPU-time efficient image classification algorithm is presented together with a face-oriented LBP transform. The algorithm can be efficiently implemented on modern computer hardware in as little as a single line of code[1] and scales embarrassingly well to currently available parallel architectures. The classifier is applied to the problem of face detection.

1 Introduction

Dissatisfied following significant efforts[11] at improving the performance the Viola & Jones cascade detector (VJ) [9] on parallel hardware, the author set about to create an object classifier that could be implemented efficiently using SIMD processor instructions or on GPUs.

The commonly used VJ detector works by creating a deep cascade of weak classifiers which together form a strong classifier. "Computational efficiency" is obtained by the use of a summed area table (SAT) to create averages of rectangular regions of pixels, and through early termination of the cascade in the case of a true negative. But because of the weak nature of the haar-like features employed it requires very large cascades that pollute the L1 cache and cannot fit in multi-core local data store (LDS). A modern variation called MB-LBP[6] can reduce the size of the classifiers due to the increased richness of the MB-LBP features compared to haar-like features but it still suffers from the similar problems. Neither the cascades nor the sparse SAT look-ups can be implemented directly using SIMD instructions due to diverging code and data paths. Another motivating factor for the author was his inability to get the cascade training process to produce any usable classifier with the training data at his disposal.

[1] Because One is better than Ten[7].

The goal of the author was simply to fit the classification problem onto SIMD and GPU hardware but this paper also presents a surprisingly capable face detector which is defined completely in only 512 bytes of constant storage and can be processed via a single line of C code.

2 Background

The initial starting point is the final stage of the classifier in [4].

In this classifier each image pixel location is assigned a probability based on the modified census transform (MCT) code and they are accumulated to form the output score.

$$H_j(\Gamma) = \sum h_x(\Gamma_x) \qquad (1)$$

Where Γ_x represents MCT code at a given location, and h_x is a local specific mapping of the probability of that code occurring there.

Because all pixel locations are processed this parallelises[sic] trivially but because h_x is implemented via a look-up table it cannot be written using SIMD instructions. The look-up table is also excessively large, a 20×20 image classifier requires 819 200 bytes of storage when using the 9-bit MCT.

By interpreting each test in (1) as individual classifiers rather than a system probability each need only resolve the probable class membership. If one assumes each test is approximately independent then it simply becomes a Naive Bayes classifier.

$$H_j(\Gamma) = \prod h_x(\Gamma_x) \tag{2}$$

However this still requires the same sized look-up table to store the likely hood values.

A solution is derived by observing that if the assumption used in [9] that many weak classifiers together make a strong classifier is correct then it still must hold to true even if the classifiers are further weakened. Based on this assumption the classifiers are further weakened by approximating the probability of class membership from each classifier is weak in either direction by some unknown but equal amount.

$$h_x(\Gamma) = 0.5 + \rho \hat{h_x}(\Gamma_x) \tag{3}$$

Where ρ is assumed weakness and $\hat{h_x}$ is a binary classifier that returns {-1,1}. The use of logarithms allows (2) together with (3) to be implemented mostly by basic integer arithmetic.

$$H_j(\Gamma) = e^{\left(A\left[\sum \check{h_x}(\Gamma_x)\right]+B\right)} \tag{4}$$

Where $\check{h_x}$ is a binary classifier which returns {0,1}, and A and B are constants. Because the classification task only requires the order be preserved this can be further simplified to just the inner equation.

$$\check{H_j}(\Gamma) = \sum \check{h_x}(\Gamma_x) \tag{5}$$

The astute reader will observe that the sum from (5) is essentially just the same calculation as in (1) even though the meaning of the summands has changed from probability weights to classifier decisions.

3 Proposed Algorithm

The proposed algorithm is simply an implementation of (5) taking advantage of the fact that each classifier only returns a single binary digit.

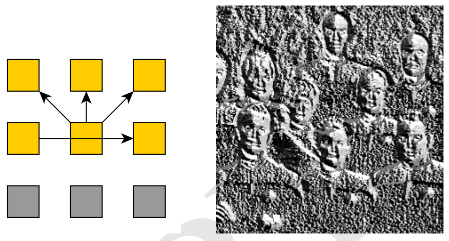
$$\check{H_j}(\Gamma) = \sum \Phi\left(\tilde{h_x}, \Gamma_x\right) \tag{6}$$

Where $\Phi(v, b)$ returns the value of bit position b of integral value v. $\tilde{h_x}$ is the classifier for location x encoded as a binary-bit stream.

Assuming the classifier look-up table can be represented in a single data type, this expression can directly translate into a single line of C code as shown in figure 1.

```
score = 0; \
   for (int i = 0; i < size; i++) \
      score += (h[i] >> X[i]) & 1;
```

Fig. 1: Taking some liberties with formatting this is the Fast Face Detector in One Line of Code. In C.



(a) Comparisons. (b) Example image.

Fig. 2: Custom Compact LBP code, C-LBP$_4$.

At first glance[2] this appears to include insufficient arithmetic and signal to create a useful image classifier.

4 Compact LBP

In order to fit the classification table into a single processor register the MCT was replaced with a local binary pattern[2]. Initially a Centre-Symmetric LBP[5] was used but provided insufficient high-frequency response at the scale used so a custom 4–bit LBP code was created, dubbed C–LBP (Compact LBP).

This pattern was derived via experimentation and unpublished research into the classification strength of individual LBP bits on faces. Its pixel comparison pattern and an example is shown in figure 2. As is desirable it shows a strong response in facial regions.

5 Experiment

The algorithm was applied to the challenging task of detecting faces in images.

A single 16 × 16 pixel classifier was created using the C-LBP as the signal operator. Although

[2] And all subsequent ones thereafter.

the author experimented with larger windows and higher bit counts – which improve the results – a smaller window and 4–bit code was chosen as it more dramatically demonstrates the outcome. In this case the entire classifier fits in just 512 bytes of memory.

5.1 Training

The classifier was trained only using images from the Color FERET[8] dataset. All images came from the FA partition with faces marked as wearing eye-classes and a few poorly registered faces discarded, resulting in 881 source images.

These were eye-aligned and normalised and the face image extracted. From this same base image 10 negative training examples were selected by random, excluding regions within a radius of half the window width of the face centre. A fixed number of each image was synthesized with up to ±10% variation in scale and the total truncated to 16384 images each.

Training was via a somewhat naive implementation[3] of a genetic algorithm[10] (GA) written in OpenCL executing on a GPU with a Java host. Training was executed several times and the best classifier chosen manually. Each training run took less than 5 minutes.

6 Results

The classifier produced was executed at multiple scales on a single image from the CMU+MIT[1] test set.

Figures 3a and 3b show the results of a multi-scale scan and the heat map of a single scale respectively.

The classifier shows high accuracy and surprisingly strong discrimination between face and non-face regions.

7 Performance and Scalability

To evaluate performance and scalability the classifier was executed for every window in a 512×512 image – 246 016 independent probes – using several implementations as shown in table 1. This scale of

[3] The author's first attempt at GA.

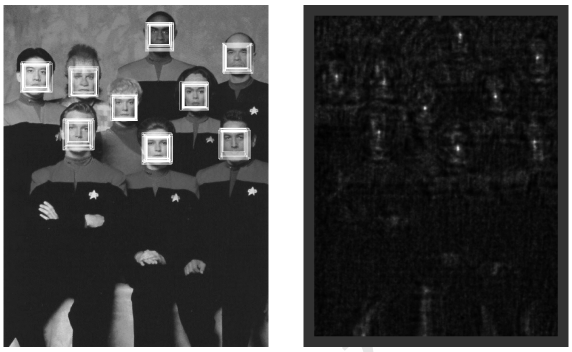

(a) Raw hits for multi-scale. (b) Heat map of single scale.

Fig. 3: Result of running 16x16 classifier.

image is representative of searching for a 64×64 pixel face in a 1080P signal.

Timing only includes the execution of the classifier at every window location and no data transformation costs which are similar across different algorithms.

For the VJ detectors the same image was used but because of the ability to short-circuit processing this test is only a rough indication of processing time. Also, the image does not contain a face at the correct scale so it likely underestimates the run time. The VJ cascades all use a window size of 20×20.

The C and Java versions of the proposed algorithm are a straightforward implementation of figure 1. The NEON version is optimised to the point of classifying more than 1 pixel per clock cycle. The OpenCL implementations are basically unoptimised but with the second using vector types. OpenCL provides further facilities that are expected to provide additional gains.

All implementations of the VJ algorithm are the work of the author. The Java implementation is similar to the one in [3]. The OpenCL implementation is from [11] and represents significant effort to fit the algorithm onto GPUs. The ARM C version represents the most effort at optimising the algorithm and involves deep data structure reorganisation and constant removal.

Hardware is Intel i7 X980 with AMD Radeon HD 7970, and Parallella-16/rev0 (Cortex-A9 @ 733 MHz). gcc version is 4.6.3, Oracle JDK is version 7.21, os is GNU.

Implementation	Time (ms)
Java proposed	111.
C proposed	35.
OpenCL proposed	1.45
OpenCL SIMD8 proposed	0.638
Java VJ (default)	107.
Java VJ (alt)	183.
OpenCL VJ (default)	21.5
OpenCL VJ (alt)	12.6
ARM C proposed (-O2 -marm)	550.
ARM C proposed (-O3 -marm)	287.
ARM NEON proposed	73.5
ARM C VJ (alt)	1000.

Tab. 1: Execution time on 512x512 image.

As expected, such a simple algorithm scales very well on parallel hardware. Even a scalar implementation is competitive with VJ on the only platform VJ works well on despite the brute-force nature of the algorithm.

8 Conclusions

The goal of creating a SIMD and GPU efficient classifier was clearly reached.

Somewhat surprisingly an easily trained classifier also appears to work as a modestly robust and highly accurate classifier for detecting faces. This result is surprising given the apparent information-sparseness of the 4–bit C–LBP code and the rather poor negative training examples used in the experiment.

The key is the training, which effectively discards all the maths and performs a randomised walk to find a solution.

Apart from exploratory work on the basic classifier, LBP codes, and training process, investigation is ongoing into ways to improve the efficiency of the complete multi-scale detection process (at least, when I feel like it beats TV).

Michael Zucchi ⟨*notzed@gmail.com*⟩

(Please do not contact me if you are a student or a paid programmer looking for help with implementation.)

References

[1] CMU + MIT face database.

[2] T. Ahonen, A. Hadid, et al. Face description with local binary patterns: Application to face recognition. *IEEE Trans. Pattern Anal. Mach. Intell.*, 28(12):2037–2041, Dec. 2006. ISSN 0162-8828.

[3] G. Bradski. Opencv library. *Dr. Dobb's Journal of Software Tools*, 2000.

[4] B. Fröba and A. Ernst. Face detection with the modified census transform. In *FGR*, pp. 91–96. IEEE Computer Society, 2004.

[5] M. Heikkilä, M. Pietikäinen, et al. Description of interest regions with center-symmetric local binary patterns.

[6] S. Liao, X. Zhu, et al. Learning multi-scale block local binary patterns for face recognition.

[7] M. Özuysal, P. Fua, et al. Fast keypoint recognition in ten lines of code. In *In Proc. IEEE Conference on Computing Vision and Pattern Recognition*. 2007.

[8] P. J. Phillips, H. Wechsler, et al. The FERET database and evaluation procedure for face recognition algorithms. *Image and Vision Computing*, 16(5):295–306, 1998.

[9] P. Viola and M. Jones. Rapid object detection using a boosted cascade of simple features, 2001.

[10] D. Whitley. A genetic algorithm tutorial. *Statistics and Computing*, 4:65–85, 1993.

[11] M. Zucchi. socles image processing library. 2012.

Portions of the research in this paper use the FERET database of facial images collected under the FERET program, sponsored by the DOD Counterdrug Technology Development Program Office (USA).

This paper was made possible by

Free Software.