# Deep Parameter Optimisation for Face Detection Using the Viola-Jones Algorithm in OpenCV

Bobby R. Bruce[1(✉)], Jonathan M. Aitken[2(✉)], and Justyna Petke[1(✉)]

[1] SSE Group, Department of Computer Science, CREST Centre, UCL, London, UK
r.bruce@cs.ucl.ac.uk , j.petke@ucl.ac.uk
[2] Department of Automatic Control and Systems Engineering,
University of Sheffield, Sheffield, UK
jonathan.aitken@sheffield.ac.uk

**Abstract.** OpenCV is a commonly used computer vision library containing a wide variety of algorithms for the AI community. This paper uses *deep parameter optimisation* to investigate improvements to face detection using the Viola-Jones algorithm in OpenCV, allowing a trade-off between execution time and classification accuracy. Our results show that execution time can be decreased by 48 % if a 1.80 % classification inaccuracy is permitted (compared to 1.04 % classification inaccuracy of the original, unmodified algorithm). Further execution time savings are possible depending on the degree of inaccuracy deemed acceptable by the user.

**Keywords:** Deep parameter optimisation · Automated parameter tuning · Multi-objective optimisation · Genetic improvement · GI · SBSE · OpenCV · Viola-Jones Algorithm

## 1 Introduction

Traditional small mobile robotics applications have limited power and computing capacity. This is further complicated for Unmanned Aerial Vehicles (UAVs), which have limited battery-life and thus any excess weight is detrimental to the time that can be spent in the air. The efficiency and accuracy of the processing is thus essential in an Unmanned Aerial Vehicle (UAV) performing tasks. Typically these tasks can use visual servoing in order to direct flights to locate objects of interest [5], or to provide a larger field of view for ground-based vehicles [7]. This is especially important in areas in which the Global Positioning System (GPS) is unavailable, where an aerial vehicle can be used to localise a ground vehicle and provide extra information about routing [2].

For example, if the UAV is performing a visual survey of a region, any repetition of a route is wasteful. The optimisation of any visual processing is essential so that areas do not need to be re-covered and thus battery capacity is not wasted. Speeding up visual processing leads to images being processed at faster rates which allows the capture of more data [11].


F. Sarro and K. Deb (Eds.): SSBSE 2016, LNCS 9962, pp. 238–243, 2016.
DOI: 10.1007/978-3-319-47106-8_18

Therefore, we propose to optimise OpenCV, a very popular computer vision library within the robotics community, using the recently introduced technique of *deep parameter optimisation* [14]. Our results show that we can achieve significant efficiency gains when we trade-off runtime and image classification accuracy. The following sections present the details of our approach.

## 2 OpenCV

OpenCV[1] is a library for computer vision [3]. It was developed by Intel, then Willow Garage, leading to its integration in the popular robotic development architecture – (ROS) [8] and wide uptake within the robotics as well as the computer vision community. It is now maintained by Itseez, a software company that specialises in optimisation of real-world applications in computer vision, pattern recognition and machine learning[2].

Face detection in OpenCV is commonly implemented using the Viola-Jones algorithm [12,13]. The Viola-Jones algorithm searches an image, at multiple scales, shifting through the image one pixel at a time, for a collection of haar features, which are shapes of binary values defining areas of light and darkness, components of the object separate from the background. The selected set of haar features defines the detected objects. There is a common set of haar features implemented in OpenCV that detects human faces, and a cascade classifier can be trained with an appropriate set.

## 3 Deep Parameter Optimisation

Deep parameter optimisation [14] is a technique that delves deeper into parameters that can affect non-functional program properties than traditional approaches (e.g., used in the machine learning community [6]). This forms a larger search-space opening new routes over which optimisation can be performed. There is a three-step process for performing deep parameter optimisation: (1) Discovery of the locations for deep parameters; (2) Exposing deep parameters to be available for tuning; (3) Search-based tuning of the exposed parameters.

## 4 Related Work

Previously studies have investigated the potential of optimising the Viola-Jones algorithm or adjusting it to perform more favourably under differing conditions [1,9,10].

Aby et al. [1] explored optimisation of the Viola-Jones algorithm on an embedded, single-board, computing platform – the Beagle Board. Rather than

[1] OpenCV's source code is available at: https://github.com/Itseez/opencv/.
[2] Itseez software company website: http://itseez.com/.

directly optimising the algorithm, they scheduled the heavy computational tasks on the Digital Signal Processor, freeing up the main ARM processor to complete ancillary computational tasks. Whilst this technique provides an improvement of processing time of the Viola-Jones algorithm it does not attempt to optimise the algorithm itself.

Rahmen et al. [9] developed an algorithm that uses skin colour and shape processing to detect faces. Initially this segments the images using typical skin colours before looking for smaller shapes that characterise faces. They achieved good performance, and indicate a favourable improvements in processing speed.

Ren et al. [10] applied a series of optimisation techniques in order to improve performance. They focused on removing the need to use dedicated extra processing power, rather than looking for software-based solutions. They tested three different optimisation approaches:

- Data Reduction – reducing the resolution of images used for face identification, increasing the shift between images from the standard one pixel, increasing the sizes used at each scale step and defining a larger minimum face size terminating the algorithm more quickly.
- Search Reduction – using key frames to limit the number of frames that need processing for a given video sequence.
- Numerical Reduction – using fixed-point formatted numbers rather than floating point to save on computation.

This paper provides an extension of this optimisation work as it applies deep parameter optimisation to the Viola-Jones algorithm itself. Rather than shifting processing, or attempting pre-filtering, we adjust the parameters themselves. Unlike the work of [10] the adjustments made throughout the optimisation are not limited to different areas, but operate across the complete algorithm. By using deep parameter optimisation, we can expose hidden options for optimisation.

## 5 Experimental Setup

Given OpenCV is a library, we developed a small command-line level program to utilise the OpenCV's functionality we wished to be optimised. This program, `classify_images`, took a directory of images as a lone argument. When executed `classify_images` produces output identifying which images contained faces and which did not. `classify_images` utilises the `CascadeClassifier::detectMultiScale` method with `CascadeClassifier` initialised using `haarcascade_frontalface_alt.xml` (included by default in OpenCV).

We created a dataset of 10,000 images which contain faces[3] and 10,000 images which do not[4]. This was then split into a training set containing 1,500 images with faces and 1,500 without, and a test set containing 8,500 images with faces

[3] Obtained from the University of Massachusetts 'Labelled Faces In The wild' dataset - http://vis-www.cs.umass.edu/lfw/lfw.tgz.

[4] Obtained from the Caltech-256 dataset – http://www.vision.caltech.edu/Image_Datasets/Caltech256/256_ObjectCategories.tar.

and 8,500 without. Prior to any form of optimisation `classify_images` incorrectly classified 0.90 % of the training set and 1.04 % of the test set.

We then profiled the software to find which files were the most heavily utilised in OpenCV when classifying images. We found that the top two files were `cascadedetect.cpp` and `cascadedetect.hpp`. We then proceeded to extract all integer constants from these files. This process involved using a regular expression to highlight all instances of integer constants. Before doing so we carried out a replacement of all occurrences of `[variable]++` to `[variable]+=1,` increasing the number of constants available for extraction.

We then replaced all instances of integer constants found with unique C Define Compilation Macros. These were extracted to a file called `defines.hpp` which was then included in both `cascadedetect.cpp` and `cascadedetect.hpp`. In total, `defines.hpp` contained 537 integer constants. `defines.hpp` can be seen as a source-code level configuration file which we altered.

While it would have been possible to proceed at this point with the parameter tuning process, considerable savings can be made by carrying out sensitivity analysis – the process of selecting a subset of parameters to optimise the desired non-functional properties.

For each of the 537 integer constants we first added one, compiled the OpenCV library, then run `classify_images` on a single face image randomly selected from the training set. If `classify_images` compiled, run, and produced a result without crashing, it passed what we refer to as 'stage 1'. If an integer passed 'stage 1' we then added 50 to the integer value. `classify_images` was then compiled with the modified OpenCV and run on the training set. To pass this stage ('stage 2') the modified version had to complete compilation and complete execution in a time different to the original (outside of the 95 % confidence interval for the original, unmodified, `classify_images` run 100 times on the training set). 'stage 1' can be viewed as a step to filter out parameters that are too sensitive, while 'stage 2' can be viewed as a step to filter out those that are not sensitive enough. After these two stages of sensitivity analysis we were left with 51 deep parameters for optimisation.

We tuned these parameters using the NSGA-II algorithm [4] implemented in the MOEA framework[5]. For the execution time objective we used UNIX's `time` utility on `classify_images` when classifying the training set. The second objective, classification inaccuracy, was calculated as a percentage of incorrect classifications by `classify_images` on the training set. NSGA-II attempts to minimize both of these objectives. We further reduced the search-space by only allowing parameters to be increased to a maximum of 64 and to be decreased to a minimum of 0.

We ran NSGA-II on 100 individuals over 10 generations in an Ubuntu 14.04.4 m4.large Amazon EC2 Instance ($2 \times 2.4$ GHz Intel Xeon E5-2676 v3 processor, 8 GiB of memory, SSD Storage). The initial generation was seeded with an individual containing the original parameter settings. The remainder of the initial population was generated by iterating through the parameters and generating a variant equal to the original but with the parameter being increased by 1 or 2.

[5] MOEA framework available at: http://moeaframework.org/.

Once complete, the MOEA framework returned the Pareto front of solutions. To ensure these results were not over-fitted to the training set, we ran each Pareto optimal solution on the test set, removing any which crash or were dominated by other solutions to produce the final Pareto optimal set.

## 6 Results

The NSGA-II algorithm produced a Pareto front that contained 14 solutions when run on the training set. When ran each of these Pareto optimal solutions on the test set, one failed to complete execution and another was dominated by other solutions in the set thus leaving 12 Pareto optimal solutions and the original, unaltered program which was also found to be Pareto optimal when run on the test set[6]. These are shown in Fig. 1 (the original program included as the left-most solution).

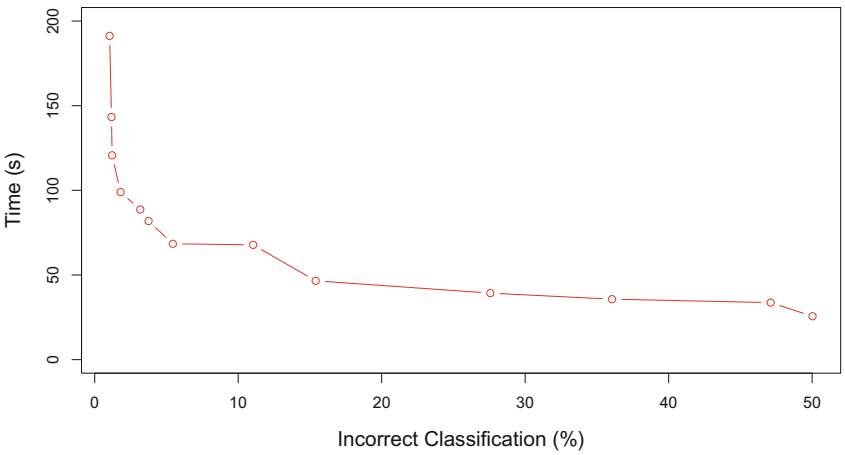


**Fig. 1.** The Pareto front of solutions when run on the test set of 17,000 images.

## 7 Conclusions

We used deep parameter optimisation to investigate improvements to face detection using the Viola-Jones algorithm in OpenCV, allowing for a trade-off between execution time and classification accuracy. In this study, a basic form of deep parameter optimisation decreased the execution time of the Viola-Jones algorithm by 48 % with a 1.80 % classification inaccuracy when evaluated on a test

[6] The source for the deep parameter optimisation algorithm we used and data discussed here is available from: https://github.com/BobbyBruce1990/DPT-OpenCV.git.

set of 17,000 images (compared to a 1.04 % inaccuracy when using the original algorithm). This technique shows the capacity for improvement within a widely used implementation of the Viola-Jones algorithm and provides a sound basis for further exploitation of more complex Search Based Software Engineering-(SBSE)methods. The source to achieve this has been made openly available on GitHub (See footnote 6).

## References


1. Aby, P., Jose, A., Dinu, L., John, J., Sabarinath, G.: Implementation and optimization of embedded face detection system. In: International Conference on Signal Processing, Communication, Computing and Networking Technologies (ICSCCN), pp. 250–253 (2011)
2. Aitken, J.M., McAree, O., Veres, S.: Symbiotic relationship between robots - a ROS ARDrone/YouBot library. In: Proceedings of UKACC International Conference on Control (CONTROL) (2016)
3. Bradski, G., Kaehler, A.: Learning OpenCV: Computer Vision with the OpenCV Library. O'Reilly Media, Inc., Upper Saddle River (2008)
4. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. Evol. Comput. **6**(2), 182–197 (2002)
5. Goodrich, M.A., Morse, B.S., Gerhardt, D., Cooper, J.L., Quigley, M., Adams, J.A., Humphrey, C.: Supporting wilderness search and rescue using a camera-equipped mini UAV. J. Field Robot. **25**(1–2), 89–110 (2008)
6. Hoos, H.H.: Automated algorithm configuration and parameter tuning. In: Hamadi, Y., Monfroy, E., Saubion, F. (eds.) Autonomous Search, pp. 37–71. Springer, Heidelberg (2012). doi:10.1007/978-3-642-21434-9_3
7. Hsieh, M.A., Cowley, A., Keller, J.F., Chaimowicz, L., Grocholsky, B., Kumar, V., Taylor, C.J., Endo, Y., Arkin, R.C., Jung, B., Wolf, D.F., Sukhatme, G.S., MacKenzie, D.C.: Adaptive teams of autonomous aerial and ground robots for situational awareness. J. Field Robot. **24**(11–12), 991–1014 (2007)
8. Quigley, M., Conley, K., Gerkey, B., Faust, J., Foote, T., Leibs, J., Wheeler, R., Ng, A.Y.: ROS: an open-source robot operating system. In: ICRA Workshop on Open Source Software, vol. 3, p. 5 (2009)
9. Rahman, M., Ren, J., Kehtarnavaz, N.: Real-time implementation of robust face detection on mobile platforms. In: IEEE International Conference on Acoustics, Speech and Signal Processing, pp. 1353–1356 (2009)
10. Ren, J., Kehtarnavaz, N., Estevez, L.: Real-time optimization of Viola-Jones face detection for mobile platforms. In: IEEE Circuits and Systems Workshop: System-on-Chip-Design, Applications, Integration, and Software, pp. 1–4 (2008)
11. Shubina, K., Tsotsos, J.K.: Visual search for an object in a 3D environment using a mobile robot. Comput. Vis. Image Underst. **114**(5), 535–547 (2010)
12. Viola, P., Jones, M.: Rapid object detection using a boosted cascade of simple features. In: Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, IEEE (2001)
13. Viola, P., Jones, M.: Robust real-time face detection. Int. J. Comput. Vis. **57**(2), 137–154 (2004)
14. Wu, F., Weimer, W., Harman, M., Jia, Y., Krinke, J.: Deep parameter optimisation. In: Proceedings of the 2015 on Genetic and Evolutionary Computation Conference, pp. 1375–1382 (2015)