UCL DEPARTMENT OF
COMPUTER SCIENCE

Research Note

RN/17/07

Deep Parameter Optimisation for Face Detection Using the Viola-Jones Algorithm in OpenCV : A Correction

7th June 2017

Bobby R. Bruce

Abstract

In our 2016 paper 'Deep Parameter Optimisation for Face Detection Using the Viola-Jones Algorithm in OpenCV' [2], we reported on an evolutionary, multi-objective approach to *deep parameter optimisation* that we reported could reduce execution time of a face detection algorithm by 48% if a 1.90% classification inaccuracy were permitted (compared to the 1.04% classification inaccuracy of the original, unmodified algorithm) and that further execution time savings were possible depending on the degree of inaccuracy permitted by the user. However, after publication we found an error in our experimental setup; instead of running the deep parameter optimisation framework using an evolutionary search-based approach we had been using a systematic one. We therefore re-ran the experiments using the intended evolutionary implementation alongside the systematic implementation for 1,000 evaluations and again for 10,000 evaluations. We found that the systematic setup is superior to the intended evolutionary setup in that it produces solutions which, when run on the test set, produce a richer Pareto frontier. The evolutionary approach, in the 10,000 evaluation setup, produced a better Pareto frontier on the training set. However, the majority of these solutions were infeasible or not Pareto optimal when run on the test set. We suspect this may be due to the evolutionary approach over-fitting to the training set.

```
for(int i=0; i < solution.getNumberOfVariables(); i++)
{
        int val=genotype[i];
        if(i == ((numEvaluations % solution.getNumberOfVariables()) - 1))
        {
                val +=((numEvaluations -1)/solution.getNumberOfVariables() + 1);
        }
        solution.getVariable(i).setValue(val);
}
```

Figure 1: Faulty Implementation

```
if(numEvaluations < populationSize)
{
        for(int i=0; i<solution.getNumberOfVariables(); i++)
        {
                int val = genotype[i];
                if(i == ((numEvaluations % solution.getNumberOfVariables()) - 1))
                {
                        val += ((numEvaluations - 1) / solution.getNumberOfVariables() + 1);
                }
                solution.getVariable(i).setValue(val);
        }
}
```

Figure 2: Fixed Implementation

1 Introduction

In 2016 'Deep Parameter Optimisation for Face Detection Using the Viola-Jones Algorithm in OpenCV' was presented at the International Symposium on Search Based Software Engineering [2]. This short investigation used *deep parameter optimisation* [11] to optimise a simple face detection algorithm implemented using OpenCV's Viola-Jones algorithm, permitting a reduction in face detection accuracy for decreased execution time.

The investigation, as part of the symposium's challenge track, was intended to show deep parameter optimisation's suitability at solving real-world multi-objective problems in software engineering. Results showed that the face-detection algorithm's execution time could be decreased by 48% with a 1.90% classification inaccuracy, and that considerably greater savings in execution time was possible if more inaccuracy were permitted.

Deep parameter optimisation is an instance of genetic improvement [8] (GI) — a sub-domain of search-based software engineering [5] where search techniques are utilised to aid software development. Typically GI modifies source-code through a set of operators such as deleting a line of source-code [3, 6, 9], or swapping API implementations [7]. In contrast, deep parameter optimisation targets specific parameters within source-code known to influence a target property, or properties, such as execution time [2, 11], memory consumption [11], or energy consumption [1]. We refer to these as 'deep parameters'. What constitutes a deep parameter is any element in a program's source-code that may exist in multiple known forms while maintaining an acceptable level of functional correctness. These parameters are exposed to a level in which they may be altered. Typically they are represented as a n-tuple of n exposed parameters. Within the work presented here, and in the 2016 paper, the exposed parameters are integer constants that can be altered to influence execution time while still classifying images (with varying levels of accuracy dependent on the parameters chosen).

Given this solution representation it is intuitive to view the tuple as a genotype which can then be modified using a genetic algorithm [10]. If one wishes to adopt a multi-objective approach, such as permitting degradation in output quality in exchange for faster execution, they may utilise a multi-objective genetic algorithm such as NSGA-II [4]. This is setup we used during our 2016 investigation into optimising OpenCV but, due to a bug explained in Section 2, a systematic search was used instead of an evolutionary one meaning values in the tuple were incremented one by one. The results reported here show that, surprisingly, this bug helped produce a better Pareto frontier than the 'correct' evolutionary approach would have. We suspect this is due to the evolutionary approach overfitting to the training set.

2 Fault And Fix

Within the 2016 paper we seeded the initial generation with a single instance of the original solution (that is, all the deep parameters as they appear in the original source-code). The remainder of the initial population was generated by iterating through the parameters of the original solution and generating a variant equal to the original but with that parameter being incremented by 1 or 2. There was, however, a fault that resulted in each generation being populated by this initialisation algorithm. The original source is shown in Figure 1. As can be seen, there is no code ensuring this is only applied to the initial generation. It was, therefore, applied to every generation. Though there was selection, mutation, and crossover, once a new population was created it was immediately overwritten.

The MOEA framework[1] (the framework we use to run NSGA-II [4]), maintains the current best Pareto frontier. This bug thereby nullifies evolution and *systematically* goes through the search space looking for Pareto optimal solutions starting from the original, unmodified solution.

Within this investigation we implement the fix shown in Figure 2, allowing the evolutionary code to run as was originally intended.

3 Experiment Rerun

Using the code outlined in Figure 2 we re-ran the experiments, running for 1,000 evaluations (as in the original paper) and again for 10,000 evaluations on the training set. We then took the Pareto frontier produced in both the 1,000 and 10,000 evaluation runs and executed each Pareto optimal point on the test set and re-plotted the Pareto frontier, removing dominated solutions and solutions that crashed or timed out. We then did the same using the original, faulty implementation (i.e. with the faulty code shown in Figure 1). All other parameters were identical to the 2016 paper, we strongly advise reading the 2016 paper prior to this correction in order to understand the experimental setup in greater detail.

4 Results

In Figure 3 we see the results of running the two techniques for 1,000 evaluations and for 10,000 on the training set (the red, left-most point in each graph shows the original, unmodified solution). It should be noted that in the case of the evolutionary approach running for 10,000 evaluations, the MOEA framework finished execution after only 9,000 evaluations as MOEA stops after several generations of stagnation despite the evaluation limit set.

When looking exclusively at these results, both the evolutionary and systematic approaches produce 'healthy' Pareto frontiers. While there is not much difference between running the systematic approach for 1,000 or 10,000 evaluations, the evolutionary approach does improve its Pareto frontier in the 10,000 instance to the extent it is better than the systematic approach (Figure 5b shows an overlay of these two Pareto frontiers).

In Figure 4 we see the results of the Pareto frontiers shown in Figure 3 run on the test set. That is, each point on the Pareto front produced by MOEA (running on the training set) run on the test set. The results were then re-plotted with dominated, or invalid solutions (those which cause the program to crash or timeout) removed.

Comparing Figure 4 to Figure 3 we can see that the Pareto front of the systematic approach largely holds when run on the test set in both the 1,000 and 10,000 evaluation instances. However, the Pareto frontier of the evolutionary approach does not. In fact, the Pareto frontier of the 10,000 evaluation instance only maintains 2 points out of the original 9. The 1,000 evaluation run is a little better, maintaining 4 of 10.

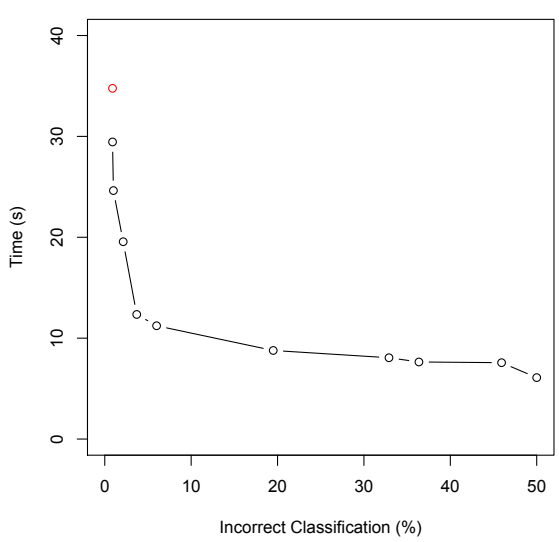
Somehow, as the the number of evaluations increased the Pareto frontier of the evolutionary technique worsened when run on the test set. Our suspicion is this is likely due to the Pareto Frontier becoming overfitted to the training set (and thereby worsening its performance when run on the test set).

Figure 5 has been included to more clearly show the comparisons between the evolutionary and systematic approaches for the 1,000 and 10,000 runs on both the training and test sets.

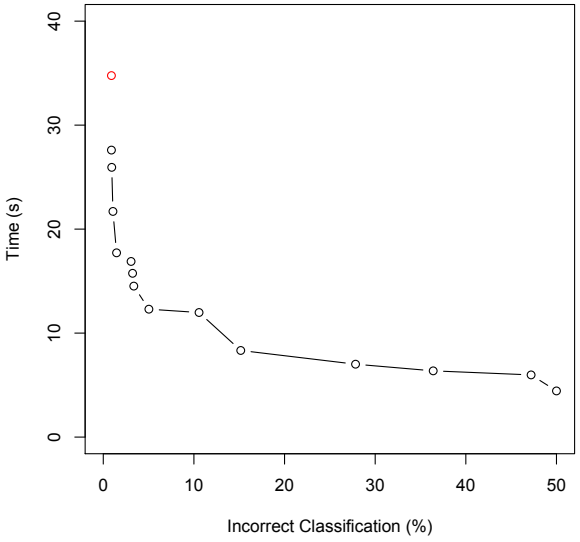
5 Conclusion

In this short paper we highlight a mistake in a previously published work and correct it. This mistake resulted in our deep parameter tuning setup optimising using a systematic search instead of an evolutionary one.

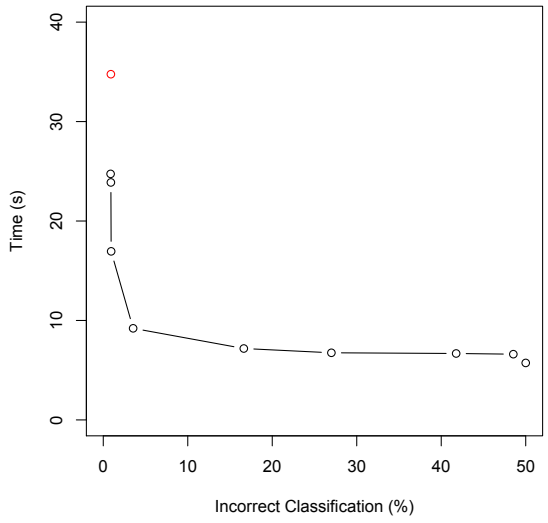
[1] MOEA Framework version 2.9 available at `http://moeaframework.org`



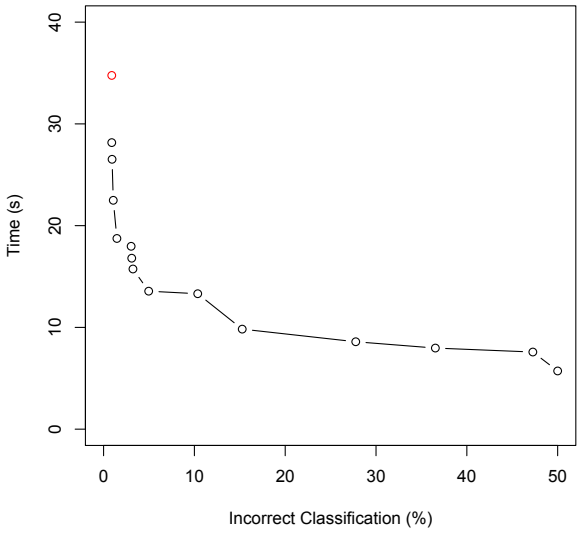
(a) 1k Evaluations — Evolutionary



(b) 1k Evaluations — Systematic

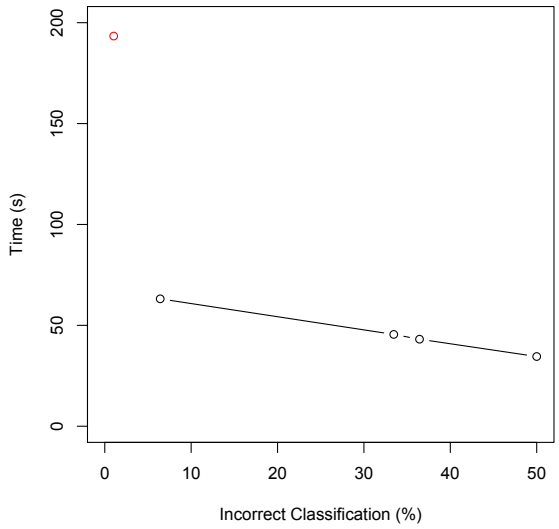


(c) 10k Evaluations — Evolutionary

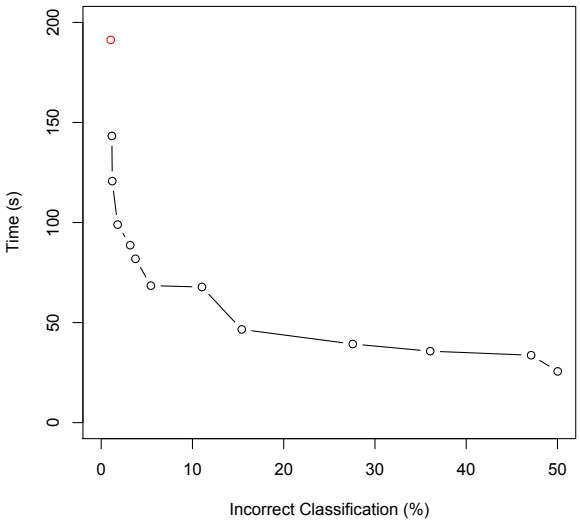


(d) 10k Evaluations — Systematic

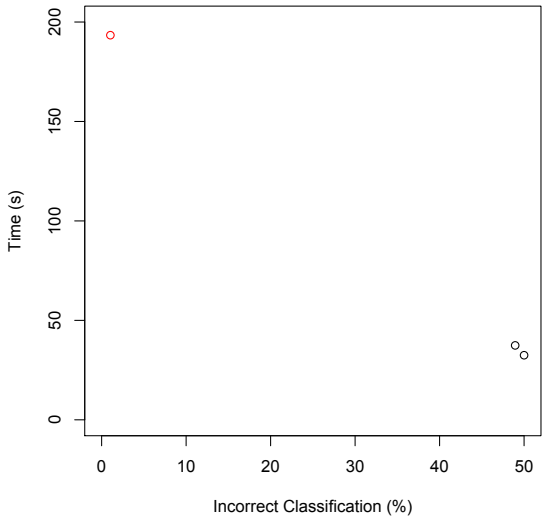
Figure 3: Training Set



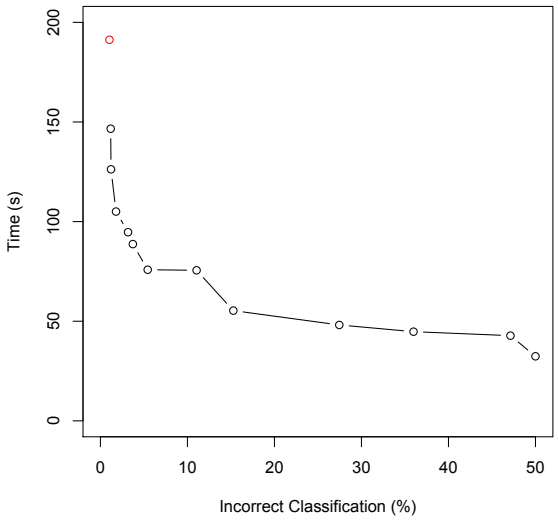
(a) 1k Evaluation Evolutionary



(b) 1k Evaluation Systematic

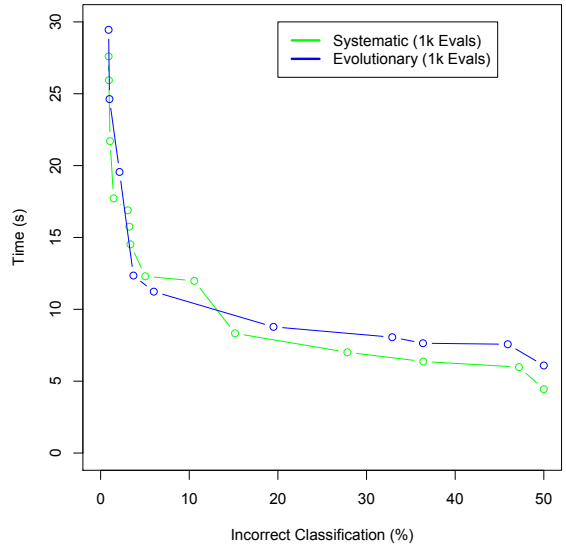


(c) 10k Evaluation Evolutionary

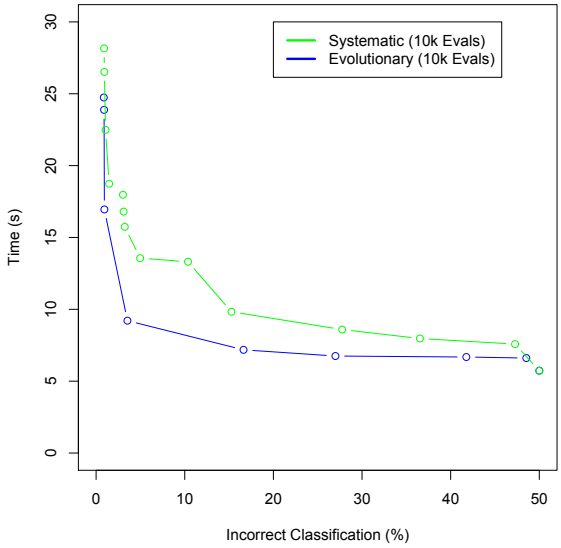


(d) 10k Evaluation Systematic

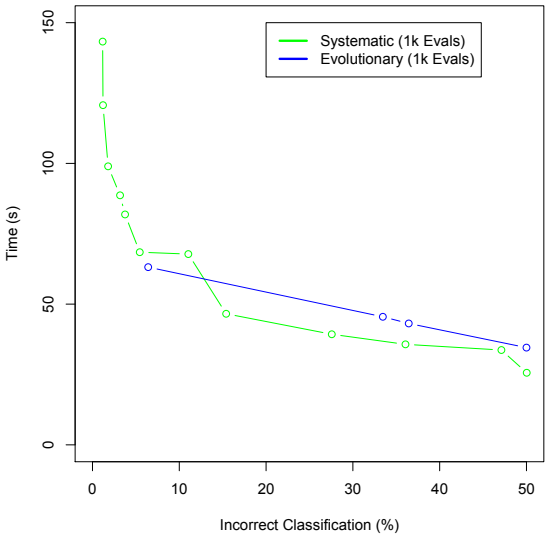
Figure 4: Test Set



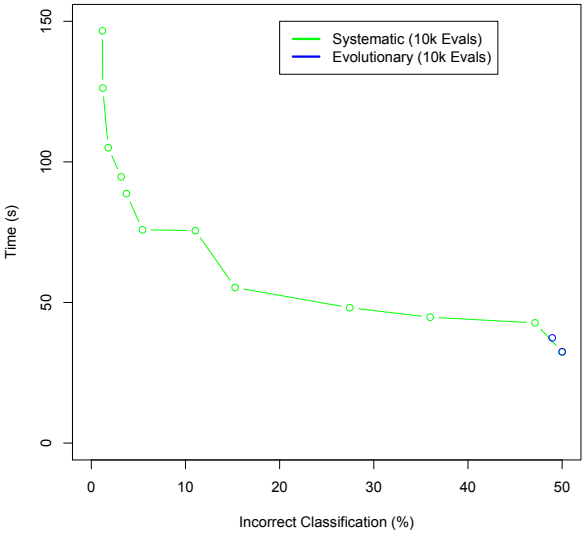
(a) 1k Evaluation (Training Set)



(b) 10k Evaluation (Training Set)



(c) 1k Evaluation (Test Set)



(d) 10k Evaluation (Test Set)

Figure 5: Pareto frontiers compared

To our surprise, this mistake resulted in better solutions than the fixed evolutionary approach. Our results shown that while, in some cases, the evolutionary approach was capable of producing a superior Pareto frontier on the training set, this was not translated to a superior Pareto frontier when run on the test test. We suspect this is due to the evolutionary approach overfitting to the training set.

All the source-code and data for this investigation (including source-code and data for the 2016 paper) can be found on GitHub[2].

References

[1] M. A. Bokhari, B. R. Bruce, B. Alexander, and M. Wagner. Deep parameter optimisation on Android smartphones for energy minimisation — A tail of woe and a proof-of-concept. In *Proceedings of the Companion Publication of the 2017 Genetic and Evolutionary Computation Conference – GECCO Companion 2017*, 2017.

[2] B. R. Bruce, J. M. Aitken, and J. Petke. Deep parameter optimisation for face detection using the viola-jones algorithm in opencv. In *International Symposium on Search Based Software Engineering – SSBSE 2016*, pages 238–243. Springer, 2016.

[3] B. R. Bruce, J. Petke, and M. Harman. Reducing energy consumption using genetic improvement. In *Proceedings of the 2015 Genetic and Evolutionary Computation Conference – GECCO 2015*. Association for Computing Machinery (ACM), 2015.

[4] K. Deb, A. Pratap, S. Agarwal, and T. A. M. T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182–197, 2002.

[5] M. Harman and B. F. Jones. Search-based software engineering. *Information and Software Technology*, 43(14):833–839, 2001.

[6] W. B. Langdon and M. Harman. Optimising existing software with Genetic Programming. *IEEE Transactions on Evolutionary Computation*, 99, 2015.

[7] I. Manotas, L. Pollock, and J. Clause. Seeds: a software engineer's energy-optimization decision support framework. In *Proceedings of the 36th International Conference on Software Engineering – ICSE 2014*, pages 503–514. ACM, 2014.

[8] J. Petke, S. O. Haraldsson, M. Harman, W. B. Langdon, D. R. White, and J. R. Woodward. Genetic Improvement of software: A comprehensive survey. *IEEE Transactions on Evolutionary Computation*, 2017.

[9] J. Petke, M. Harman, W. B. Langdon, and W. Weimer. Using genetic improvement & code transplants to specialise a C++ program to a problem class. In *Proceedings of the 17th European Conference on Genetic Programming – EuroGP 2014*, 2014.

[10] M. Srinivas and L. M. Patnaik. Genetic algorithms: A survey. *Computer*, 27(6):17–26, 1994.

[11] F. Wu, W. Weimer, M. Harman, Y. Jia, and J. Krinke. Deep parameter optimisation. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation – GECCO 2015*, pages 1375–1382. ACM, 2015.

[2] `https://github.com/BobbyBruce1990/DPT-OpenCV`