International Journal of Science and Research (IJSR)
ISSN: 2319-7064
SJIF (2022): 7.942

An Evaluation of a Haar Cascade Classifiers using Multi-Resolution Images and Multi-Threading Resources on a Raspberry Pi

Qussay Salih and Lab Research Team

School of Business, Department of CSIT, Kwantlen Polytechnic University, Department, Surrey, Canada

Abstract: ***Image processing plays a crucial role in vision-based IoT sensors, serving various applications to enhance productivity. Researchers have highlighted computational challenges in object detection on low-cost devices like the Raspberry Pi. In today's fast-paced technological landscape, the need for automated systems delivering accurate results is paramount to task completion. This study introduces an effective multithreading approach for the Support Vector Machine (SVM) method. We have implemented a multithreading algorithm for the SVM recognition processes, harnessing the power of multicore CPU utilization. Our evaluation incorporates Memory usage, CPU Temperature, FPS, Confidence levels, and Elapsed time on the Raspberry Pi platform, with the primary goal of addressing real-time computation challenges using the Pi camera. The experimental results demonstrate a notable enhancement in detection confidence, affirming that multithreading significantly bolsters detection performance on Raspberry Pi processors across various image resolutions.***

Keywords: Haar Cascade Classifier, Haar-like features, Supervised Vector Machine algorithm, Real time , Raspberry Pi

1. Introduction

Face detection in simple term is detect the human face and identify the face. Many algorithms had been successfully managed to find and recognized the human face with verity of success rate. Haar cascade classification algorithm was one of them. The detecting and recognizing process starting by capture image is analyzed to determine the portion of the image contains a human face.

High-speed Haar Cascade Classifiers (HCC) are competitively efficient identifiers, as a result, the Human-Machine Interface (HMI) structure showed the ability to be used in a stable real-time environment. For this research the, the objective was first to learn the effective HCC face detectors, after which they were integrated into low cost, credit-card sized computer that plugs into a computer monitor Raspberry Pi 3B+.

In this work, by using HCC, we decided to accept the challenge to implement HCC on Raspberry Pi to analysis regular vs multithreading calcification study encompasses Time elapsed, Average Confidence, Average FPS, CPU and Memory Usage and Temperature (Heat) generated during recognition vs image capturing distance and resolution.

The main objective of this research is to define and present Raspberry Pi and abilities of its usage in the development of the next generation of inexpensive, intelligent, agile machine. All in which to evaluate accuracy detection and recognition. Finally, our machine produces the outcomes and detection rates.

2. Related Work

It has been reported that numerous researchers are experiencing problems with computation limitation in order to detect objects in low cost devices such as Raspberry Pi [1][2]. In this instance, the performance of the Raspberry Pi camera detecting the object of interest is slowed down by only being able to use one core on the Raspberry Pi. There are four cores on the Raspberry Pi, however, only one core can be used, resulting in the performance of the Raspberry Pi slowing down. [3]. It is also important to note that images that can be detected using supervised learning algorithms such as the selected algorithm for this study Haar Classifier may also only be able to detect images quickly if only one core is used, resulting in poor system performance when a single core is used.

A larger number of images stored as SVM models makes the detection using the Pi camera slower [3] [4]. In general, a modest type of multithreading occurs when a thread Processed until it is jammed by an event that usually creates a long latency [5].

A halt in processing may occur when the cache needs to access external chip memory, a task that can take hundreds of CPU cycles to retrieve the required data. Rather than waiting for this halt to resolve, the threading processor switches to another thread that is ready to run. Only when the data for the previous thread has been retrieved does it allow the previous data to be placed on the standby thread list. The primary objective of multithreading is to eliminate interruptions caused by data dependencies in the execution pipeline [6][7].

Since each thread is independent of the others, there is a possibility that a single instruction in the pipeline may require output from a previous stage of processing. In concept, this resembles the primitive multitasking seen in operating systems, where each active thread is allocated a CPU cycle. The most advanced form of multithreading is found in superscalar processors. While typical superscalar processors issue multiple instructions from one thread per CPU cycle, simultaneous multithreading (SMT) allows the superscalar processor to issue instructions from multiple threads in each CPU cycle. Recognizing that any single

thread has limited directive parallelism, this form of multithreading seeks to harness the parallelism found across various threads to minimize the inefficiencies associated with unused execution slots. The objectives of this study are focused on further evaluation involve Raspberry Pi 3B+ with a multicore using multithreading, aiming to analyze CPU and memory usages, CPU temperature, FPS, Confidence levels, and Elapsed time.

3. Comparing multithread vs non-multithread Haar Cascade using Raspberry Pi research method

In contrast to automated face recognition, human face recognition involves the identification of facts or confirmation of personality. In essence, the technique is based on two stages, the first being the identification of faces, and the second being the recognition of those faces.

Raspberry Pi is a mini computer (the size of a credit card) that can perform varied tasks, however, because it does not have a powerful processor, it can perform these tasks with a lower level of computation power. Despite the Raspberry Pi being slower than a normal laptop or desktop computer, it is still a fully-fledged Linux machine with all the capabilities that imply, at the same time which uses a very low amount of power.

This study is divided into four phases which contains collection of data, Detection, training, classification. The research study mainly focuses on evaluating Raspberry Pi limitation in terms of resource concern when a heavy processing power were required. The selected method was Haar Cascade classifier that represents SVM training algorithms. The main evaluation factors are implementing Haar cascade classifier to evaluate resource constrain over for none multithreaded vs multithreaded processes that to determine the benefit of implementing Multithreading vs none multithreaded processing in respect to CPU and Memory usage, CPU Temperature, FPS, Confidence, Elapse time.

a) Data Collection

Camera connected to Raspberry Pi 3B+ capture images for the study. Camera model and specification: Raspberry Pi Camera, Key studio 5MP 1080p Camera Module with OV5647 Sensor Video Webcam. The sensor has a native resolution of 5 megapixel, and has a fixed focus lens on board. Each description contains distinct information. The images were collected at varying resolutions, encompassing five frame sizes: 1200x900, 900x675, 600x450, 300x225, and 100x75. This progressive increase in resolution was employed to assess the system's efficiency. Performance data, obtained during benchmark detection tests, were compiled and analyzed, considering attributes such as resolution and image quantity. Moreover, we have incorporated a feature involving a rotated face image with a 30-degree visible perspective

b) Haar Cascade Classifiers

The application of the HCC effectively incorporates three variables. This framework functions by processing a comprehensive array of features, which can be readily visualized within a fixed timeframe, rather than focusing on precise frame processing values. Through the utilization of a feature-based approach, the system reduces class variability to maximize divisional variability that to speed up the process.

Furthermore, the integration of an enhancement algorithm enables the compilation of a select subset of relevant features and guidance, facilitating the identification of new data. Within the cascade architecture, each layer of classifiers progressively assimilates a diminishing percentage of training data, rendering the model highly efficient.

c) Haar-like features and Adaboost Training

Haar features on the image makes it easy to find out the edges or the lines in the image, or to pick areas where there is a sudden change in the intensities of the pixels. Below see Fig 1, represent A sample of Haar features used in the Original Research Paper published by Viola and Jones.

The Haar features traversal on an image would involve a lot of mathematical calculations. As we can see for a single rectangle on either side, it involves 16-pixel value additions (for a rectangle enclosing 16 pixels). Imagine doing this for the whole image with all sizes of the Haar features. Fig 2. shows the making of an Integral Image. Each pixel in an Integral image is the sum of all the pixels in its left and above.

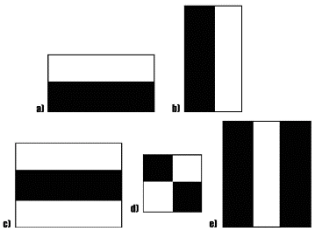
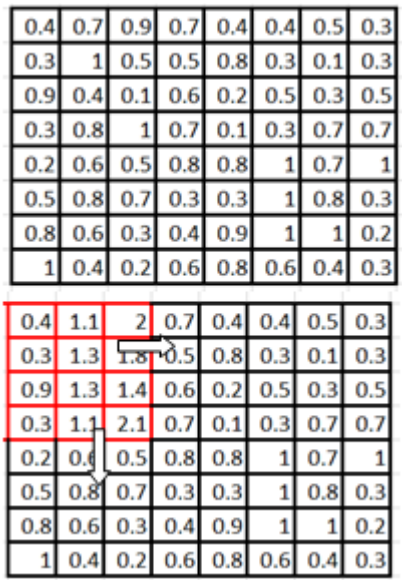


Figure 1: A sample of Haar features used in the Original Research Paper published by Viola and Jones.



0.4	0.7	0.9	0.7	0.4	0.4	0.5	0.3
0.3	1	0.5	0.5	0.8	0.3	0.1	0.3
0.9	0.4	0.1	0.6	0.2	0.5	0.3	0.5
0.3	0.8	1	0.7	0.1	0.3	0.7	0.7
0.2	0.6	0.5	0.8	0.8	1	0.7	1
0.5	0.8	0.7	0.3	0.3	1	0.8	0.3
0.8	0.6	0.3	0.4	0.9	1	1	0.2
1	0.4	0.2	0.6	0.8	0.6	0.4	0.3

0.4	1.1	2	0.7	0.4	0.4	0.5	0.3
0.3	1.3	1.8	0.5	0.8	0.3	0.1	0.3
0.9	1.3	1.4	0.6	0.2	0.5	0.3	0.5
0.3	1.1	2.1	0.7	0.1	0.3	0.7	0.7
0.2	0.6	0.5	0.8	0.8	1	0.7	1
0.5	0.8	0.7	0.3	0.3	1	0.8	0.3
0.8	0.6	0.3	0.4	0.9	1	1	0.2
1	0.4	0.2	0.6	0.8	0.6	0.4	0.3

Figure 2. shows the making of an Integral Image

This would be a hectic operation even for a high-performance machine. Haar value calculation formula can be applied to the determination of hair-like characteristics.

Pixel value=(sum of the Dark pixels/Number of Dark pixels)–(Sum of the Light pixels/Number of Light pixels) (1)

Next is Adaboost Training combines a set of "weak classifiers" to generate a powerful "strong classifier" that the object detection method can effectively employ. This process involves the identification of valuable features and the instruction of classifiers on how to utilize them.

To create the weak learners, a sliding window traverses the input image, calculating Haar characteristics for each image segment. This approach differs from a straightforward threshold that distinguishes between objects and non-objects. These created classifiers are considered "weak," but to construct an accurate strong classifier, a multitude of Haar features is required. In the final stage, the weak learners may be combined with the strong learners.

A. Classifiers cascade

After each classifier is trained, the classifier's weight is calculated based on its accuracy. More accurate classifiers are given more weight. A classifier with 50% accuracy is given a weight of zero, and a classifier with less than 50% accuracy is given negative weight.

Let's look first at the equation for the final classifier.

$$H(x) = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right) \quad (2)$$

The final classifier consists of 'T' weak classifiers. $h_t(x)$ is the output of weak classifier 't' (in this paper, the outputs are limited to-1 or +1). $Alpha_t$ is the weight applied to classifier 't' as determined by AdaBoost. So, the final output is just a linear combination of all of the weak classifiers, and then we make our final decision simply by looking at the sign of this sum.

The classifiers are trained one at a time. After each classifier is trained, we update the probabilities of each of the training examples appearing in the training set for the next classifier.

The first classifier (t=1) is trained with equal probability given to all training examples. After it's trained, we compute the output weight (alpha) for that classifier.

$$\alpha_t = \frac{1}{2} ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right) \quad (3)$$

The output weight, $Alpha_t$, is fairly straightforward. It's based on the classifier's error rate, 'e_t'. e_t is just the number of misclassifications over the training set divided by the training set size.

B. Multithreading enactment on Raspberry Pi

This study places a stronger emphasis on internal performance, particularly focusing on the Raspberry Pi's capabilities in executing face detection and recognition tasks involving extensive image storage and various image resolutions. Pre-evaluation procedures were conducted for Haar cascade classifier processes to monitor and log the resource-intensive computations both before and after the implementation of multithreading on the Raspberry Pi system.

The study involves the detection of trained faces using the Haar cascade classifier. When the Pi camera is employed for face detection, and a recognized image of a person's face is detected, a visual indicator in the form of green frames surrounds the detected image, serving as a marker for the identified image.

In computer architecture, multithreading refers to the ability of a Central Processing Unit (CPU) (or a single core within a multi-core processor) to concurrently execute multiple processes or threads, with support from the operating system. This approach distinguishes itself from multiprocessing. In multithreaded applications, processes and threads share single or multiple core resources, which encompass computing units, CPU caches, and look aside translation buffers (TLB) [9]. In contrast, a multiprocessing system consists of multiple complete processing units across one or more cores. Multithreading is specifically designed to enhance the utilization of a single core by leveraging thread-level parallelism, as well as command-level parallelism. Given their complementary nature, these two techniques are sometimes combined in a multithreading CPU system, alongside a multi-core CPU.

The integration of multithreading code into the existing codebase is implemented during the model detection process. During the preliminary assessment, when streaming images from the Pi camera, we encountered issues related to lagging. Notably, this lag was observed during image annotation and training for the Haar cascade classifier model. The root cause of this lag can be attributed to the presence of our extensive SVM model files, featuring a large number of images. These files impose a considerable load on our computer's performance and lead to high CPU memory usage.

From a programming perspective, the use of a single thread for each process in the fourth code iteration results in the overload of a single CPU core's memory, causing a decline in tracking accuracy. It's worth noting that the Raspberry Pi 3B+ has four available cores, and distributing memory usage and CPU load more evenly across these cores could lead to improved tracking accuracy.

Given the limitations of using only one CPU core due to memory constraints, the implementation of multithreading alternatives is expected to enhance real-time tracking performance.

4. Multipoint Recognition Procedure

The chosen face detection and recognition procedure was notably distinctive. Its purpose was to emulate real-world face detection scenarios. Ideally, a security face detection camera is installed to capture human faces. However, individuals do not always look directly at the camera with 100% precision. The true challenge in image capture is selecting the optimal image. Meanwhile, the subject being captured can adjust their face at various angles while the camera attempts to capture and verify their identity for

access. Hence, a multipoint procedure was implemented to enhance the authenticity of image capture and recognition. This procedure employed a Raspberry Pi device to assist in the process while the individual adjusted their face in all four directions (lookup, lookdown, look left, look right) Nevertheless, the captured image was exclusively considered for individuals whose faces were at least partially visible to the camera.

1) Application was started
2) When camera first boots displaying on screen user waits 1 second directly looking at the camera
3) User looks right for one second at a 30 degrees angle
4) User looking left for one second at a 30 degrees angle
5) User then looks up at 30 degrees angle
6) User the looks down at a 30 degrees angle
7) Finally, user looks at screen and when directly looking at camera
8) When the camera delay on screen catches up to the user to where the user is in the final position looking back at the screen the application is exited
9) Total elapsed time of each test is timed at 6 seconds

The illustrated procedure above demonstrates the image recognition confidence percentage. The purpose of the time displays waits 1 second is to inform the user capturing started and ready for the first move. After image capturing the HCC started for face recognition and classification. The study focus mainly to study multithreaded vs non-multithreaded face recognition accurate confidence, total number captured FPS, CPU usage, Memory usage, CPU temperature.

5. Results

Table 1: Show Avg. FPS, CPU and Memory usage results for Multithread vs none for the listed resolution

Average FPS

Resolution	None-Multithread	Multithread
100 x 75	26.28	31.44
300 x 225	8.63	9.83
600 x 450	3.60	6.76
900 x 675	1.80	6.47
1200 x 900	1.11	4.63

Usage CPU

Resolution	None-Multithread	Multithread
100 x 75	73.80	86.95
300 x 225	154.50	123.50
600 x 450	235.50	171.50
900 x 675	260.50	160.50
1200 x 900	267.50	162.50

Usage MEM.

Resolution	None-Multithread	Multithread
100 x 75	8.85	9.00
300 x 225	8.95	9.10
600 x 450	9.20	9.25
900 x 675	11.75	9.75
1200 x 900	14.10	10.10

A Comparison of CPU Memory usage in Frame Per Second

Table 1. shows the collected results CPU Temperature, CPU and Memory usage in Frame Per Second. Currently only two processes are running – Haar Cascade Classification model and performance monitoring code. To interpret Average FPS vs resolution while implementing multithread vs none multithread results. We can recognize as the resolution increases; the FPS generally decreases for both Multithread and Non-Multithread processing. This is expected because higher resolutions require more computational power to process. For lower resolutions (100x75 and 300x225), Multithread processing provides a slight FPS advantage over Non-Multithread processing. This indicates that multithreading is beneficial in these cases. At a resolution of 600x450, the Multithread approach exhibits a substantial performance improvement over Non-Multithread. The difference in FPS is quite significant. At a resolution of 900x675, the performance improvement withMultithread is even more pronounced. It results in a notable boost in FPS.

At the highest resolution of 1200x900, the Multithread approach still outperforms Non-Multithread, although the gap is not as large as at lower resolutions.

The results suggest that Multithread processing generally provides better FPS compared to Non-Multithread processing as well as Multithread processing is more efficient in terms of CPU usage across various resolutions. The advantage becomes more significant as the resolution increases.

In regards to Memory (MEM) usage vs resolution we recognized as the resolution increases, generally tends to increase for both Multithread and Non-Multithread processing. This is expected because higher resolutions require more memory to store and process the data.

At all resolutions, Multithread generally has slightly higher memory usage compared to Non-Multithread. However, the differences are relatively small. Therefore, we recognize there isn't a significant advantage for either approach in terms of memory efficiency.

Table 2: Show Avg. recognition confidence, CPU Temperature and Time elapsed outcome for Multithread vs none for all listed resolution

Average Confidence

Resolution	None-Multithread	Multithread
100 x 75	-85.52	59.57
300 x 225	4.83	11.63
600 x 450	44.44	49.14
900 x 675	57.79	63.94
1200 x 900	59.41	68.01

Temp C

Resolution	None-Multithread	Multithread
100 x 75	54.50	55.04
300 x 225	57.73	58.00
600 x 450	60.15	59.88
900 x 675	61.22	60.15
1200 x 900	61.76	60.69

Time Elapsed

Resolution	None-Multithread	Multithread
100 x 75	6.64	8.47
300 x 225	7.77	8.34
600 x 450	9.86	9.57
900 x 675	13.05	9.52
1200 x 900	16.62	10.45

For all resolutions, Multithread consistently yields higher average confidence values compared to Non-Multithread. This indicates that the Multithread approach tends to provide better confidence in image recognition.

Both Multithread and Non-Multithread exhibit very similar CPU temperatures across different resolutions. There isn't a significant difference in CPU temperature between the two approaches.

In most cases, the Non-Multithread approach has a shorter time elapsed compared to Multithread. However, the difference is not uniform across all resolutions, and at the highest resolution (1200 x 900), the time elapsed is significantly longer for both approaches.

If time elapsed is critical, Non-Multithread may be preferable for some resolutions. While Multithread tends to provide better average confidence in image recognition.

6. Conclusions

The development of the intelligent system aims to improve the computing performance by optimizing the resources in a Raspberry Pi. Experimental results show a decent improvement achieved using multithreading in Haar-based face detection for real-time applications processes recognized confidence. Based on the research conducted, there are several suggestions for further improvements, such as less computation demand SVM algorithms or implement deep learning should be a study candidate for future work. In conclusion, the study intended to benefit future IoT research utilizing Raspberry Pi as selected hardware so that they can accommodate the computation power limitation which they can encompass working on camera as an image sensor IoT device for a work that sensitive to high performance.

References

[1] V. Patchava, H. B. Kandala, and P. R. Babu, “A Smart Home Automation technique with Raspberry Pi using IoT,” in 2015 International Conference on Smart Sensors and Systems (IC-SSS), 2015, pp. 1–4.
[2] E. Bilgin and S. Robila, “Road sign recognition system on Raspberry Pi,” in 2016 IEEE Long Island Systems, Applications and Technology Conference (LISAT), 2016, pp. 1–5.
[3] N.F.A Zainal, R. Din, M.F. Nasrudin, S. Abdullah, A.H.A Rahman, S.N.S Abdullah, K.A.Z. Ariffin, S.M. Jaafar, N.A.A Majid. (2018). Robotic Prototype And Module Specification For Increasing The Interest Of Malaysian Students In Stem Education.-International Journal Of Engineering And Technology (Uae).
[4] M. R. Rizqullah, A. R. Anom Besari, I. Kurnianto Wibowo, R. Setiawan, and D. Agata, “Design and Implementation of Middleware System for IoT Devices based on Raspberry Pi,” in 2018 International Electronics Symposium on Knowledge Creation and Intelligent Computing (IES-KCIC), 2018, pp. 229–234.
[5] R. Rinku and M. Asha Rani, “Analysis of multi-threading time metric on single and multi-core CPUs with Matrix Multiplication,” in 2017 Third International Conference on Advances in Electrical, Electronics, Information, Communication and Bio-Informatics (AEEICB), 2017, pp. 152–155.
[6] W. F. Abaya, J. Basa, M. Sy, A. C. Abad, and E. P. Dadios, “Low cost smart security camera with night vision capability using Raspberry Pi and OpenCV,” in 2014 International Conference on Humanoid, Nanotechnology, Information Technology, Communication and Control, Environment and Management (HNICEM), 2014, pp. 1–6.
[7] Azmi, I., Shafei, M. S., Nasrudin, M. F., Sani, N. S., & Abd Rahman , A. H. . ArUcoRSV: Robot localisation using artificial marker. In J-H. Kim, H. Myung, & S-M. Lee (Eds.), Robot Intelligence Technology and Applications-6th International Conference, RiTA 2018, Springer Verlag, 2019, pp. 189-198.
[8] Resource Optimisation using Multithreading in Support Vector Machine
[9] Wong Soon Fook, Abdul Hadi Abd Rahman, Nor Samsiah Sani, Afzan Adam, “International Journal of Advanced Computer Science and Applications(IJACSA)”, Volume 11 Issue 4, 2020