Using Haar Feature-Based Cascade Classifiers for Face Detection on a Raspberry Pi for IoT Applications

Ethan Leyden
Department of Computer Science
Virginia Commonwealth University
Richmond, VA, USA
leydene@vcu.edu

Alan Dorn
Department of Computer Science
Virginia Commonwealth University
Richmond, VA, USA
dornn@vcu.edu

Cameron Clyde
Department of Computer Science
Virginia Commonwealth University
Richmond, VA, USA
clydecp@vcu.edu

***Abstract*—Internet of Things (IoT) devices are frequently implemented and popular solutions for many applications, including facial detection and recognition for home security systems. [1] Raspberry Pi's (Rpi) are small, efficient single-board computers that provide an affordable computing platform for numerous applications requiring a small physical footprint. [2] Facial detection is the process of identifying human faces in an image. This technology has become increasingly popular as it allows for people to be efficiently tracked without any human intervention. [3] Haar feature-based cascade classifiers are a popular tool in the field of computer vision used for object detection. In this paper, we will explore the utility of Haar feature-based classifiers for facial recognition on the Raspberry Pi by analyzing the performance of the algorithm, and investigating applications for face detection in this context. The researchers of this paper pose questions like "What are viable ways that Rpi's can be implemented by non-experts in home security solutions?" and "How well does an Rpi perform as a platform for a facial recognition system?"**

***Index Terms*—Internet Of Things; Haar feature-based cascade classifier; Facial Recognition; Raspberry Pi;**

I. Introduction

Face detection (the ability to recognize faces in an image) and face recognition (the ability to identify faces in an image) are well-studied computer vision tasks. The face detection problem is commonly solved using a Haar feature-based cascade classifier [4]. The method is usually chosen for its speed and conceptual simplicity. The classifier works by moving a sliding window across an image, in a raster scan fashion. As the window slides across the image, Haar features (depicted in Figure 1) are used to perform calculations on adjacent rectangles within the window. In the training step, AdaBoost is typically used [5], [6] to build the cascade structure in Figure 2 by evaluating the best Haar features in the set. When using the model after the training step, the sliding window and set of Haar features is used again to evaluate whether the features discovered in training exist within the window, if so the computation moves down the cascade model, and the image within the face is classified as a face when most or all of the features are found in the window. The method was first used by Viola and Jones in their groundbreaking paper on rapid object detection [7], demonstrating the approach to be computationally efficient. This is backed up by the 2021 literature review done by Hirzi et al [8]. Who finds that the Viola and Jones algorithm is still widely used. The computational efficiency of this object detection method is the most appealing characteristic to us.

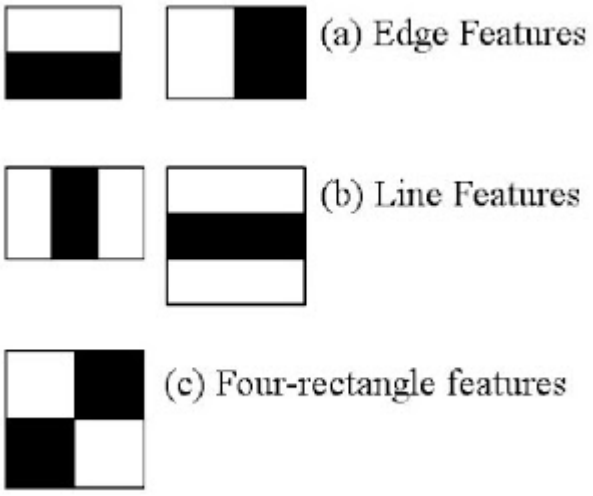

Figure 1: Haar features used by OpenCV, a highly popular open-source computer vision library

Haar features work by taking the difference of the sum of the pixels inside two to four adjacent rectangles (depicted in Figure 1). The values can indicate characteristics about the image within the bounds of the rectangles, and these characteristics can be assessed or evaluated using machine learning algorithms (popularly AdaBoost [4], [8], but some use neural networks [1], [9]). In Gupta et al. [10], researchers actually use Haar feature-based cascade classifiers on a Rpi as a precursor to face recognition, first detecting the face, then attempting to recognize it. The experiment was run on a Raspberry Pi 3B+ development kit with a processor that runs

at 1.4Ghz, which slightly better performance compared to the Intel Pentium III used by Viola and Jones [7], which was a generic desktop CPU at the time, which shows promise for the research done in this very paper. Image size is also an important factor to consider when assessing performance. In Viola and Jones's paper, they used 384 by 288 pixel images at 15 frames per second, and in Gupta et al. [10], the input image size and frames per second were notably not stated, however a webcam was used, so it would be reasonable to infer that an average webcam in 2016 was what they used in that paper. While this paper focuses on face detection, facial recognition may also be achievable, as shown by Radzi et al. [1] and Rouhsedaghat et al. [11], where researchers assess the ability of a Rpi or other low-resource computing device to recognize faces. Typically this is achieved using some combination of neural networks and Principal Component Analysis, but in Rouhsedaghat et al. [11] they used PixelHop++ and successive subspace learning.

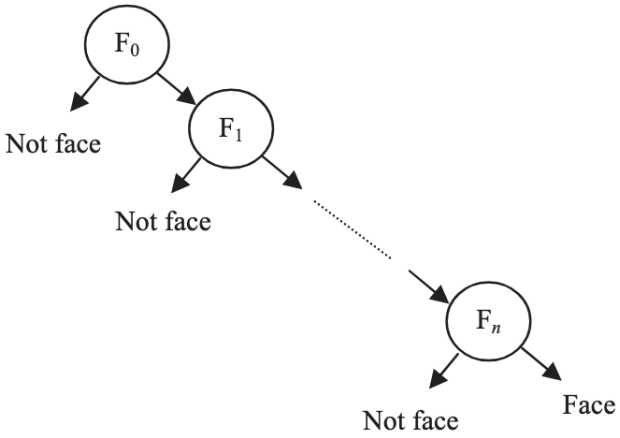

Figure 2: Cascade model used in Haar feature-based cascade classification [4]

II. Problem Statement

Facial detection plays a key role in applications such as home security systems, smartphone cameras, and other IoT technologies. However, implementing reliable and efficient facial detection systems on devices with limited processing power like Raspberry Pi (Rpi) poses significant challenges. Other methods of face detection, such as neural networks, may be too computationally expensive for real-time processing required of IoT applications. In order to reach these speed requirements, a more efficient method must be used. Therefore, utilizing Haar feature-based cascade classifiers satisfies this requirement with a lightweight solution for facial detection. In this project, we will investigate the use of Haar feature-based cascade classifiers for facial detection on a Raspberry Pi, focusing on finding a balance between facial detection accuracy and computational efficiency. The goal is to provide a practical and affordable facial detection system that can be implemented in IoT-related solutions. Specifically, we aim to address the following questions:

- *How effective is the Raspberry Pi as a platform for facial detection using Haar classifiers, considering its hardware limitations?*
- *Can the facial detection system meet the real-time processing requirements necessary for IoT applications, such as home security?*

By looking into these questions, we aim to assess the practicality of facial detection applications on Raspberry Pi using Haar feature-based cascade classifiers.

Figure 3: Two bounding boxes, prediction by the model in blue, annotation in white.

III. Method

We will assess the performance of face detection using a Haar feature-based cascade classifier by measuring the model's ability to detect faces, as well as the computational performance of a basic Python program that uses the model to detect faces in each frame of a live video feed. Model training is computationally expensive, and out-of-scope for the average non-expert looking to implement a face detection system for a home security system. However, while this paper is more focused on the hardware aspect of face detection, model performance is an important consideration when designing a face detection system. As such we decided to measure the performance of the particular model we used: OpenCV's pre-trained `haarcascade_frontalface_default.xml` model. After assessing the model's performance, we measured CPU and memory utilization of a basic Python program, as well as measure the amount of time it takes to perform face detection on a single live video frame in the same program.

A. Model Performance

Intersection over Union (IoU) is a popular evaluation metric used in object detection [12]. It is computed by dividing the area of the intersection of two bounding boxes, by the area of the union. If the bounding boxes are entirely disjoint and do not overlap, the IoU is 0, and if the bounding boxes overlap perfectly, the IoU is 1. This can then be translated into more generalized performance measures like true positives (TP), false negatives (FN), and false positive (FP) by using a threshold value to determine the minimum IoU for a predicted object to be classified as TP. A FP is considered any prediction that does not exceed the IoU threshold, and a FN is any annotation that doesn't have a corresponding prediction.

To give a complete picture of the model's performance, we measured the IoU for each predicted image and plotted a histogram, as well as plotted the precision and recall each threshold value between 0.00 and 1.00, in increments of 0.01. When considering images with multiple annotated and predicted faces, we had to consider how to assign predictions to annotations. Consider the scenario where there are two annotations and one prediction, with that prediction overlapping both annotations equally. We resolved this by computing an IoU value for all prediction-annotation, and selecting the pairs with the highest IoU, excluding pairs whose annotation or prediction had already been selected. The code we wrote to compute IoU for each prediction in an image can be found in our repository on GitHub.[1]

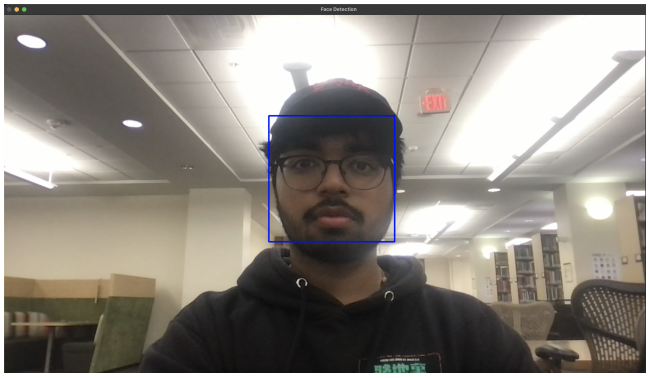

Figure 4: A VCU student testing out face_detection.py

B. CPU and Memory Performance

CPU and memory utilization are common metrics when considering the computational performance of a program. We devised a basic Python script (face_detection.py) that displays a live video feed of a webcam or other connected camera (such as the Rpi Camera Module 2), and runs each frame though the face detection model. When a face is detected, a blue bounding box is added to the image where the model detected a face. We measured the CPU and memory utilization of face_detection.py using a separate script (test_utilization.py) that spawns face_detection.py as a subprocess, and reports the CPU and memory utilization using the `psutil` package.

In this paper, CPU utilization is given as a sum of the percent utilization for each core. For example, if one core is being used 75%, and another 80%, then the total utilization is given as 155%. Memory utilization is given in MB. While face_detection.py is runnning, test_utilization.py prints the CPU and memory utilization to the console once per second. When face_detection.py is terminated, test_utilization outputs the minimum, maximum, and average CPU and memory utilization to stdout.

When listing the results for each computer used in this study, they will be listed by the names given in Figure 5. The relevant hardware specifications for each computer are given in Figure 5 as well.

Computer Name	CPU	Memory Capacity
Raspberry Pi 3B+	Broadcom BCM2837B0, Cortex-A53 (ARMv8) 64-bit SoC	1GB LPDDR2 SDRAM
2019 Macbook Pro	Intel i7-9750H	16GB DDR4
2021 Macbook Pro	M1 Pro (Apple Silicon)	16GB LPDDR5
2021 Acer Swift	Intel i7-1165G7	16GB LPDDR4X
2022 Macbook Air	M2 (Apple Silicon)	8GB LPDDR5

Figure 5: Hardware specifications for each computer used in this study

C. Frame-By-Frame Performance

In addition to CPU and memory utilization, we also measured the elapsed time taken to detect faces within a frame. This code is written directly in face_detection.py, and is run when the `-t` argument is passed to the program. When the argument is given, Python's built-in `time` module is used to measure the elapsed time taken to a) receive the image from the camera, and b) detect the faces. Any computation done after the face detection, between frames, is irrelevant to the face detection aspect itself and, as such, is not measured. The built-in timer *does not* measure the elapsed time to draw the returned bounding box on the image, print to stdout that a face was detected, and display the image on the screen. The results for each frame are then aggregated into minimum, maximum, and average milliseconds elapsed to perform face detection – grouped by the number of faces detected in a frame. In other words, the program will report the minimum, maximum, and average milliseconds elapsed to detect faces for all frames with 0 faces, 1 face, 2 faces, etc. up to the maximum number of faces detected in a frame throughout the runtime of the program.

[1] Click this link to view the GitHub repo.

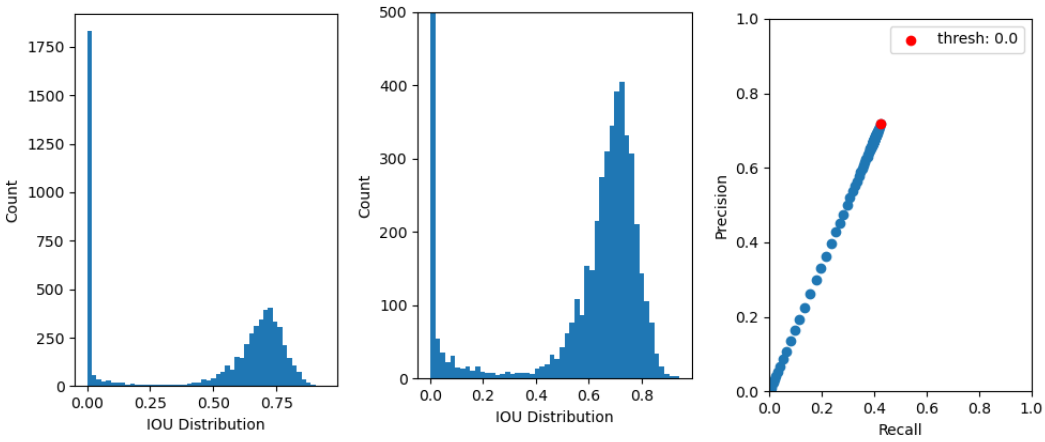

Figure 6: Results on the validation set.

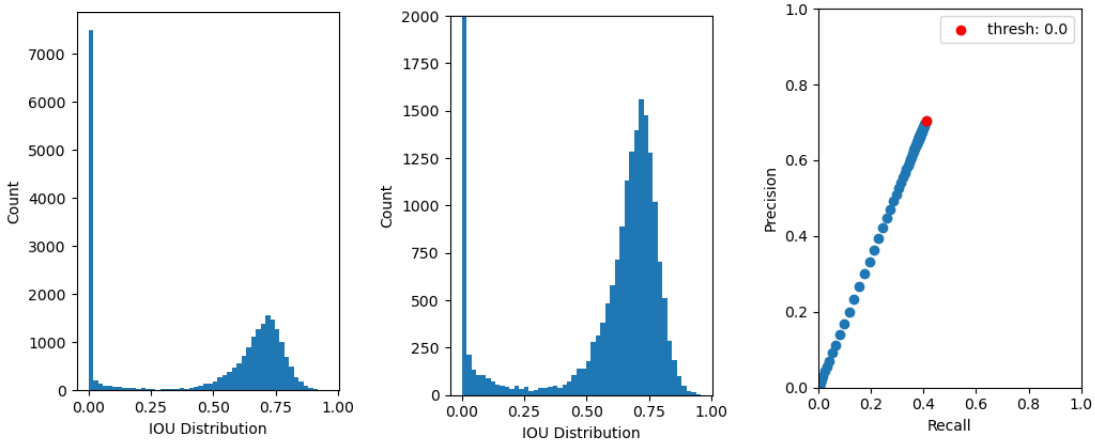

Figure 7: Results on the training set.

IV. Results

A. Model Performance

Shown in Figure 6 are the results of classification with `haarcascade_frontalface_default.xml` on the validation images, and in Figure 7 are the results on the training images. On the left and center graphs, the IoUs of each predicted face are plotted on a histogram on the interval [0, 1] with a bucket size of 0.05. The left graph has an unaltered y-axis to give a complete picture of the predicted faces, and the center graph has an adjusted y-axis to more clearly show the distribution of valid detections.

A ”valid detection” is any detection that had a calculated IoU greater than 0. It is possible to adjust our definition on position of a false positive by choosing a value within the IoU interval as a threshold, and classifying anything greater than or equal to the threshold as a true positive, and anything below as a false positive. The rightmost graph is a plot of recall and precision for possible IoU thresholds on [0.01, 1.00] at an interval of 0.01. Highlighted in red is the precision and recall for the threshold with the highest F1 score (0.00). Note that a threshold of 0.00 inevitably includes false positives, which is discussed in more detail in the Discussion section.

	Validation	Training
Image Count	3347	13386
Pred. Faces	6096	24155
False Positives	27.12%	28.47%
Mean IoU (w/ FP)	0.4582	0.4503
Mean IoU (w/o FP)	0.6287	0.6295
Std. Dev (w/o FP)	0.2008	0.2012
Recall (T:0.00)	0.4206	0.4063
Precision (T:0.00)	0.7106	0.6989
F1 (T:0.00)	0.5285	0.5139
Recall (T:0.01)	0.4170	0.4038
Precision (T:0.01)	0.7046	0.6946
F1 (T:0.01)	0.5239	0.5107

Figure 8: Statistics on model performance

The histograms reveal a notable distribution pattern, with a significant portion of IoU values being concentrated at 0, indicating the presence of false positives. 27.12% of the 6096 predictions on the validation set were false positives. Because the proportion of false positives to the rest of the dataset were so high, the distribution metrics we considered such as mean and standard deviation were skewed down, so in our

distribution analysis, we filtered out all 0 IoU values from our calculations to focus on predictions that shared some overlap on the ground truth (the annotated data). Excluding false positive IoU's raise the mean IoU from 0.4582 to 0.6287, which more closely indicates the quality of bounding boxes on valid detections.

Figure 8 presents a comprehensive summary of the experimental results evaluating model performance. The table includes key metrics such as the number of images processed, the total number of predicted faces, and the proportion of predictions classified as definite false positives (where IoU equals 0). Additionally, it provides the mean IoU values calculated both with and without false positives (denoted as "w/ FP" and "w/o FP," respectively) and the standard deviation of IoU values excluding false positives. Finally, recall, precision, and F1 scores are reported for two specific thresholds: 0.00, which yielded the highest F1 score, and 0.01, which more closely aligns with predictions matching the ground-truth annotations.

B. CPU and Memory Performance

In Figure 9, the results from our experiment of running a face detection program are listed. For each computer, the program `face_detection.py` was run for approximately 5 minutes, and intermittently presented anywhere from 0 to 5 faces. The maximum and average CPU and memory utilization are presented, and the minimum is excluded from this paper since the value is more representative of the state of the machine at the beginning of the runtime of the program, and does not provide meaningful insight into system performance during runtime. The results from the Raspberry Pi are listed first and emphasized, since that platform is the focus of the paper.

	CPU (%)		Memory (MB)	
	Max	Mean	Max	Mean
Raspberry Pi 3B+	***367.7***	***305.3***	***147.1***	***141.5***
2019 Macbook Pro	767.0	609.5	136.5	126.5
2021 Macbook Pro	408.1	356.6	434.9	354.7
2021 Acer Swift	539.1	398.8	162.8	161.0
2022 Macbook Air	388.0	332.9	335.6	331.6

Figure 9: CPU and memory utilization for various computers

C. Frame-By-Frame Performance

Finally, in our analysis of frame-by-frame performance, we have plotted the maximum, average, and minimum values in Figures 10, 11, and 12 respectively. To save space on the plots, we've abbreviated the names of each device in the legend. "Macbook Pro" was abbreviated to "MBP", "Macbook Air" to "MBA", "Raspberry Pi 3B+" to "Rpi", and "Acer Swift" to "AS". These are the same computers used in the CPU and memory utilization experiments. Note that y-axis of the minimum plot, Figure 12 has been adjusted to more clearly show the differences between devices, and is different than the y-axis in Figures 10 and 11. Since the Raspberry Pi is the primary platform of our study, results from the device are in bright green, intentionally chosen for its visual prominence.

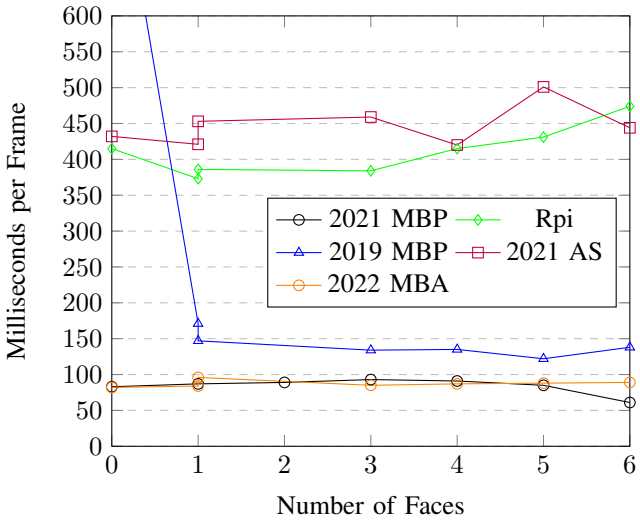

Figure 10: Maximum frame by frame performance

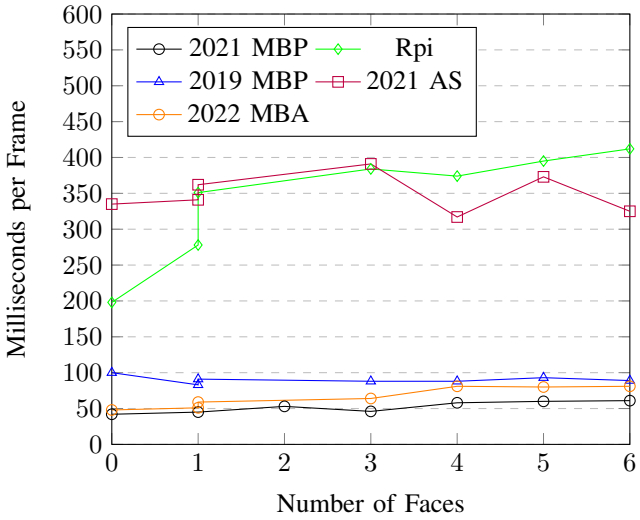

Figure 11: Average frame by frame performance

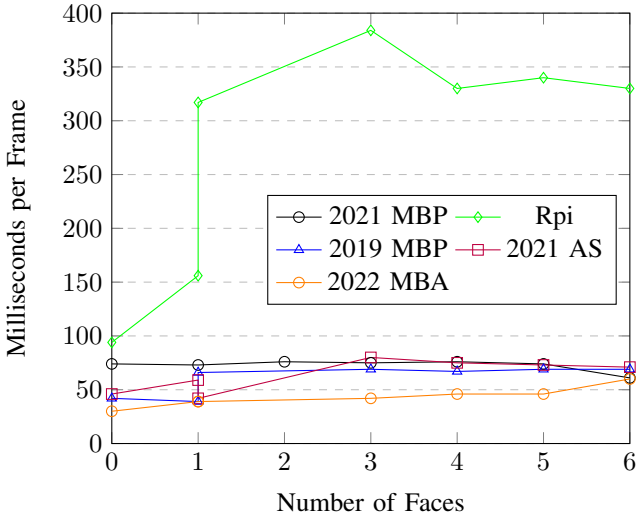

Figure 12: Minimum frame by frame performance

V. Discussion

A. Model Performance

In our evaluation of model performance, we predicted faces on two annotated datasets by evaluating a larger dataset's validation and training set separately. Note that the training set contains more than four times the number of images in the validation set, which has the potential to indicate patterns on the model's predictions as the volume of data increases. First we evaluated the model performance independent of input volume by analyzing the results on the training set – since larger datasets are generally more representative of real-world distributions. Then, in the section "Performance Trends with Increased Data Volume" we compared the two datasets, and analyzed how the performance results changed as the input volume increased.

1) General Performance Trends: The false positive rate on the training set was high at 28.47%. This is over a fourth of the predicted faces within each image set. High false positive rates are consistent across haar cascade classifiers [4], [7], [9], and this could potentially be improved if we chose a different model [13]. This study is focused on the hardware aspects of face detection, but an analysis of multiple models could provide valuable insights, and could be an avenue for further research.

A "valid detection" is not strictly limited to any prediction with an IoU greater than 0.00. The threshold can be adjusted, with higher IoU thresholds imposing stricter requirements for the overlap between predicted and ground truth bounding boxes. We analyze the accuracy as the threshold increases from 0.00 to 1.00 by plotting the precision and recall for every threshold in that range on increments of 0.01, as well as plotting the IoU for all predictions on a histogram. Figure 7 depicts those plots for the training set, but the validation set is notably similar.

The mean IoU excluding 0.00 on the training set is 0.6295, which is noticeably lower than the peak of the curve on the histogram. This can be attributed to the slight skew of predictions. As previously stated, the false positive rate of the model was high (more than a fourth of predictions did not overlap with any annotations), but it is even higher than that if you increase the threshold to the local minimum (approximately 0.25). On the training set, the false positive rate increases almost 5%, from 28.47% to 34.77%.

The predictive performance of face detection models can be difficult to quantify in comparison to simple classification models. Popular metrics include Intersection-over-Union, but also include more familiar metrics like precision, recall, and F1 score. In Figure 7, we've plotted the precision and recall for thresholds in the range [0.00, 1.00] on an interval of 0.01. As the threshold decreases, precision and recall share a linear non-decreasing relationship, peaking at a maximum precision of 0.6989 and recall of 0.4038 on the training set. The F1 score of the model, with a threshold of 0.00 is 0.5139. The F1 score excluding false positives is intuitively lower, albiet only slightly, with an F1 score of 0.5107 at a threshold of 0.01. The biggest change in F1 score are in fact when the threshold is increases through the range where the majority of the IoU values are concentrated. This trend is evident on the plot of precision and recall where the increased spacing between points along the line correspond to the primary curve of the IoU distribution.

2) Performance Trends with Increased Data Volume: Nearly every statistic that we measured during our predictive performance analysis showed that performance decreased as the volume of input to the model increased, however only slightly. From the validation to the training set, the input volume increased by 400% (quadrupled). The slight decrease in performance could possibly indicate a small discrepancy between our analysis and real-world results, especially as the variability and edge cases found in the real world are encountered using the model we analyzed.

The results on the predictive performance of `haarcascade_frontalface_default.xml` show that the model has a high tendency to predict false positives in general. For both the validation and training images, the proportion of IoU values equal to 0 for predicted faces were 27.12% and 28.47% respectively. The proportion of false positives increased by almost 1%, as the number of predictions quadrupled, which could indicate that as the number of predicted faces increase, so does the proportion of false positives by a small (perhaps negligible) amount.

With the exception of the mean excluding false positives and standard deviation, every metric decreased slightly, implying worse performance on a wider range of data. The increase in standard deviation implies this also, since a higher standard deviation means a higher variation in predictive performance.

B. CPU and Memory Performance

At first glance, the measured CPU and memory performance of `face_detection.py` across various hardware platforms seem unintuitive. The program had the lowest CPU utilization on the Raspberry Pi, with an average CPU utilization of 305.3%. The worst CPU utilization was on the 2019 Macbook Pro, where the mean CPU utilization was 609.5% (more than six cores being used!). However this device had the lowest mean memory usage at 126.5MB. The worst memory usage was the 2021 Macbook Pro at 354.7MB.

We hypothesize that the explanation of these results lies in the implementation of OpenCV and how it may optimize performance depending on the available hardware. Additionally, various operating systems will make different resource allocation choices depending on the creators of those operating systems. For example, the creators of Raspberry Pi OS likely optimize resource allocation for low-resource environments when MacOS developers instead optimize resources to use the full resources available to them. To illustrate this point, consider that 141.5MB is one tenth of the available RAM on the Raspberry Pi in our experiment, but 354.7MB is a mere 2%.

C. Frame-By-Frame Performance

Finally, we analyzed the frame-by-frame performance of each hardware platform by measuring the number of milliseconds it takes to process a frame and aggregating the results given the number of faces predicted within the frame. The first observation we had on the frame-by-frame results are that performance is largely consistent across the number of faces in frame among the Macbooks. The only exception to this is the maximum compute time for the 2019 Macbook Pro on 0 faces, which was unusally high. The unexpectedly high maximum processing time for a live video frame in the face detection program could be attributed to routine system-level factors. These include other system processes temporarily taking priority over the execution of `face_detection.py`, leading to resource contention. Additionally, anomalies occurring during the program's initialization, such as delays in allocating memory or initializing dependencies, may have contributed to the observed outliers in processing time. These factors highlight the influence of non-deterministic operating system behaviors on real-time performance, even when ample resources are available.

The second observation is that the compute time on the Raspberry Pi is significantly slower than almost any other platform in our study, which is consistent with expectations. Average compute time for a single frame was between a third and a half of a second, which is an important factor to consider when designing a home security system involving face detection. The only platform that occasionally exceeded the Raspberry Pi in compute time was the 2021 Acer Swift, which upon further investigation was running 100+ Google Chrome tabs. This finding highlights the impact of system resource contention and background processes on compute performance, even for hardware designed with more substantial capabilities.

VI. Conclusion

Overall the results of this study demonstrate that it is generally feasible to use a Raspberry Pi as tool for face detection in a home security environment. At the time of writing, a Raspberry Pi 3B+ is available from Adafruit for $35, and the Raspberry Pi Camera Module V2 is available from Amazon for $17.41, which means that the cost of a do-it-yourself (DIY) implementation is just over $50. While indoor/outdoor security cameras range from just under $20 to more than $100 per camera, most do not offer the same degree of freedom as a DIY solution, and many cameras are connected to the manufacturers' infrastructure on the basis of providing live video feeds accessible over the internet, and recordings that are saved to the cloud. In a DIY solution, an amateur home security system could provide services on par with current solutions on the market for the average consumer.

Concerning predictive performance, the model's ability to reliably identify faces is questionable. In the context of home security, a DIY home security system may be required to recognize faces at a distance, at an angle, or in another scenario where the live video feed may be distorted. If the homeowner expects a highly accurate face detection system, they should look elsewhere. With a high false positive rate, it would be inadvisable to expect that every detected face is a legitimate detection.

There exists a plethora of research topics that could potentially expand the results of this study. Namely, future research could focus on the accuracy or performance of different face detection models, other home security uses of a Raspberry Pi, or feasibility of expanding the rudimentary face detection program we have written for this paper. The findings of this study highlight the trade-offs between computational efficiency and predictive accuracy, emphasizing the need for optimized hardware-software integration to enhance the real-world applicability of face detection algorithms on resource-constrained devices like the Raspberry Pi.

VII. References

[1] S. A. Radzi, M. M. F. Alif, Y. N. Athirah, A. Jaafar, A. Norihan, and M. Saleha, "Iot based facial recognition door access control home security system using raspberry pi," *International Journal of Power Electronics and Drive Systems*, vol. 11, no. 1, p. 417, 2020.

[2] A. Nayyar and V. Puri, "Raspberry pi-a small, powerful, cost effective and efficient form factor computer: A review," *International Journal of Advanced Research in Computer Science and Software Engineering (IJARCSSE)*, vol. 5, pp. 720–737, 12 2015.

[3] S. H. Katsanis, P. Claes, M. Doerr, R. Cook-Deegan, J. D. Tenenbaum, B. J. Evans, M. K. Lee, J. Anderton, S. M. Weinberg, and J. K. Wagner, "A survey of us public perspectives on facial recognition technology and facial imaging data practices in health and research contexts," *PloS one*, vol. 16, no. 10, p. e0257923, 2021.

[4] L. Cuimei, Q. Zhiliang, J. Nan, and W. Jianhua, "Human face detection algorithm via haar cascade classifier combined with three additional classifiers," in *2017 13th IEEE international conference on electronic measurement & instruments (ICEMI)*. IEEE, 2017, pp. 483–487.

[5] R. A. Aras and H. Endang, "Implementation of haar cascade and adaboost algorithms in photo classification on social networks," *Inspiration: Jurnal Teknologi Informasi dan Komunikasi*, vol. 13, no. 1, pp. 59–68, 2023.

[6] S. Niranjan, T. J. Jebaseeli, S. A. Raj, and S. Marshal, "Drowsiness detection using adaboost method and haar cascade classifier to improve safety of drivers," in *International Conference on Soft Computing: Theories and Applications*. Springer, 2023, pp. 131–141.

[7] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition. CVPR 2001*, vol. 1. Ieee, 2001, pp. I–I.

[8] M. F. Hirzi, S. Efendi, and R. W. Sembiring, "Literature study of face recognition using the viola-jones algorithm," in *2021 International Conference on Artificial Intelligence and Mechatronics Systems (AIMS)*, 2021, pp. 1–6.

[9] A. N. Razzaq, R. Ghazali, and H. A. H. Al Naffakh, "Face detection techniques: A holistic overview," *Library Progress International*, vol. 44, no. 2s, pp. 1022–1032, 2024.

[10] I. Gupta, V. Patil, C. Kadam, and S. Dumbre, "Face detection and recognition using raspberry pi," in *2016 IEEE international WIE conference on electrical and computer engineering (WIECON-ECE)*. IEEE, 2016, pp. 83–86.

[11] M. Rouhsedaghat, Y. Wang, S. Hu, S. You, and C. C. J. Kuo, "Low-resolution face recognition in resource-constrained environments," 2020. [Online]. Available: https://arxiv.org/abs/2011.11674

[12] H. Rezatofighi, N. Tsoi, J. Gwak, A. Sadeghian, I. Reid, and S. Savarese, "Generalized intersection over union: A metric and a loss for bounding box regression," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[13] H. Rowley, S. Baluja, and T. Kanade, "Neural network-based face detection," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 20, no. 1, pp. 23–38, 1998.