OPTIMIZING CNN MODELS FOR REAL-TIME HARDWARE CLASSIFICATION ON RESOURCE CONSTRAINED HARDWARE: A RASPBERRY PI 5 POWERED HARDWARE SORTING SYSTEM

Jeremy Wang [1], Jonathan Sahagun [2]

[1] Lexington High School, Lexington, MA, United States
[2] California State Polytechnic University, Pomona, CA, 91768

ABSTRACT

Within a work environment, the cleanliness and organization of tools contribute greatly to the workers' mental state and psychological well-being, allowing for more creativity and productivity in the workplace [1]. This is largely an issue within the industrial industry, where machines need to be built, maintained, and repaired. While being automated in larger factories, the tedious task of sorting hardware, such as screws, nuts, and bolts, is often performed manually in the context of smaller-scale factories and repair centers. However, the tediousness of this task often undermines the importance of the task and is often neglected or performed poorly. Existing industrial solutions to this issue are expensive and inflexible. This paper presents a low-cost, autonomous hardware sorting system that uses a custom-built Convolutional Neural Network (CNN) object detection model trained using TensorFlow and Keras. The system runs entirely on a Raspberry Pi 5 and uses a Microsoft Lifecam Studio webcam together with an H-bot gantry for mechanical sorting. The primary focus of research is on the optimization of the CNN for real-time deployment on resource-constrained hardware. Multiple lightweight architectures such as YOLOv8-Nano and MobileNetV2-SSD are proposed for examination and evaluated. A custom dataset was created and labeled using Roboflow, with images consisting of three hardware classes: screws, nuts, and standoffs. The trained model reached a mean average precision (mAP) of 91.5%, with ~125 ms for each inference while on the Raspberry Pi. When integrated with the mechanical system, the full pipeline sorted hardware at an average rate of 18.6 parts per minute with an accuracy of 90.0%. As the project is built with a budget of $300, this project demonstrates the feasibility of deploying lightweight deep learning models for automation tasks on embedded systems.

KEYWORDS

Convolutional Neural Network, Keras, Tensorflow, Edge AI, Model Optimization

1. INTRODUCTION

In industrial settings, productivity depends not only on efficient workflows, but also on the organization of tools and hardware. While organization in traditional companies with office jobs is simpler and cheaper to perform, organization in industrial facilities can be much more costly. Within these companies, the cost can be largely attributed to the organization of various tools and hardware used to manufacture and repair. After usage, tools and hardware are often scattered

David C. Wyld et al. (Eds): SIGI, CSTY, AI, NMOCT, BIOS, AIMLNET, MaVaS, BINLP – 2025
pp. 327-342, 2025. CS & IT - CSCP 2025 DOI: 10.5121/csit.2025.151926

about, and misplaced tools cangreatly reduce the efficiency of workers who have subsequent shifts. One of the reasons that this issue is commonly neglected is its tediousness. When tasked to sort hardware, a worker is given a pile of mixed-up pieces of hardware of all different shapes and sizes and is required to examine the pieces of hardware one-by-one and place them into respective containers. Depending on the size of the company and the scale of the product that is being worked with, the number of different varieties of fasteners and sizes of hardware can range on a large scale. McMasterCarr's catalog, commonly used by both hobbyists and industries, consists of over 200,000 items [2]. This complexity combined with tediousness is why the task is often neglected or performed poorly. In addition, this is also one of the main reasons why the sorting is difficult to automate. Existing solutions are either extremely expensive or inflexible. While larger companies may be able to afford a custom-built sorting system or hire specialized manual sorters, the problem is exacerbated in smaller businesses. Many smaller businesses end up with inefficient manual sorting processes, leading to a cycle of decreased productivity, lower worker morale, and reduced operation output.

Recent advancements in computer vision and machine learning have made it increasingly possible to automate complex visual detection tasks using affordable systems. However, doing so on resource-constrained hardware has significant challenges such as limited processing power and memory. Most popular object classification and detection models such as YOLO are not directly suited for such environments without significant optimization or third-party hardware acceleration. To address these issues, this paper presents a low-cost hardware sorting system built around an optimized custom-trained CNN using TensorFlow and Keras. The model is designed to perform on a Raspberry Pi 5 without external hardware support and uses live video input from a Microsoft Lifecam Studio webcam. Detected items are then sorted with a two-axis H-bot gantry system to bins corresponding to each hardware class. The key contribution of this research is in the development and optimization of the CNN model. Several lightweight architectures including YOLOv8-Nano, Ultralytics' YOLOv11-Nano, EfficientDet-D0, and MobileNetV2-SSD were evaluated based on inference latency, power usage, and detection accuracy. With the focus on model architecture and deployment-specific constraints, this project demonstrates the feasibility of building low-cost and scalable vision systems for automation in constrained environments. The complete system was built for under $300 USD, offering a scalable solution where industrial alternatives are either unaffordable or impractical.

While prior work has already demonstrated CNN deployment on embedded platforms such as the Raspberry Pi, most studies rely on external accelerators or present isolated inference performance without system-level integration. As a result, there remains limited understanding of how different lightweight detection architectures and optimization strategies behave under the strict latency and resource constraints of unassisted Raspberry Pi hardware. This work addresses that gap by systematically evaluating four representative lightweight object detection modelstrained on a custom dataset of industrial hardware components. Each model was benchmarked under multiple quantization strategies and evaluated not only for detection accuracy but also for inference latency, memory usage, and real-time viability when integrated into a robotic sorting pipeline. By quantifying these trade-offs, this study contributes new empirical evidence for deploying CNN-based object detection systems on resource-constrained devices, while demonstrating their feasibility for low-cost industrial automation tasks.

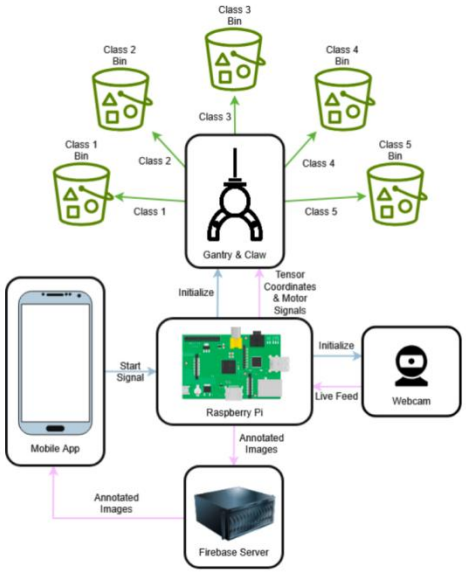

Figure 1. Overall architecture of the hardware sorting system, including camera input, Raspberry Pi inference, and Firebase Database. Pi Image [3].

2. LITERATURE REVIEW

2.1. Sorting Systems

Mechanical sorting has long been employed in industrial settings, particularly for homogeneous or size-based classification tasks. Well-made mechanical systems are extremely efficient and do their designated task exceptionally well with little to no errors. Superior mechanical systems must be created rigorously and tuned for tolerance toward issues such as jams and processing of foreign materials. These systems often use vibratory feeders, gravity chutes, and mechanical sieves to sort parts by diameter, weight, or shape. An example of this is BubsBuilds' sieve design, which relies on the different sizes of screw heads (Fig. 3).

However, due to increasingly large amounts of specialization needed for each new classification of hardware added to a factory's catalog, the systems can get extremely expensive and bulky at large scales. As such, mechanical sorting systems that appear in factories are considerably more complex and expensive. An example is Feiyu Machinery's industrial level hardware sorting line (Fig. 4).

Larger-scale sorting may rely on techniques such as industrial optical inspection combined with robotic pick-and-place, such as those made by Key Technology or Buhler. These systems are effective as they use computer vision and mechanical movement to sort the hardware efficiently, yet they generally cost tens of thousands of dollars and are designed for fixed, repetitive tasks. An example of a successful pick-and-place style system is that of Apera AI's robotic arm showcased at Automate 2023 [7]. Due to the large costs, these systems typically cannot be used by small workshops or educational institutions. Several hobbyist projects have explored sorting as well, using Arduino-based mechanical sorters and OpenCV-based color thresholding, but these lack the consistency and flexibility required for deployment in dynamic environments [8]. These approaches often perform poorly in cases where objects overlap or with when there is a larger variety of hardware [9].

Figure 2. The completed system, including the Raspberry Pi 5, webcam, and H-bot gantry used for sorting

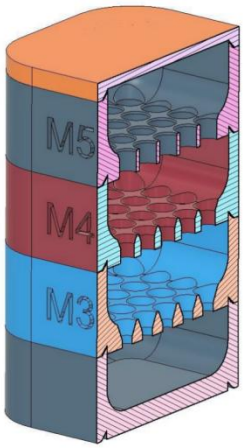

Figure 3. A section analysis of BubsBuilds' 3D printable sieve CAD model [4].

Figure 4. A photo looking across Feiyu Machinery's hardware sorting and packaging line [5].

2.2. Deep Learning for Object Classification

Traditional computer vision methods, such as edge detection, Hough transforms, and template matching, used to be standard for image processing. However, these methods are not generalizable and perform poorly in real-world scenarios where lighting varies or objects are rotated [9, 10, 11, 12].

Since the introduction of AlexNet, Convolutional Neural Networks (CNNs) have revolutionized object image processing tasks. Other architectures that built offAlexNet such as VGGNet, ResNet, and DenseNet improved accuracy by increasing depth [13, 14, 15, 16]. These models, however, require significant processing resources, making them difficult to work with on devices with limited processing power.

In object detection specifically, two-stage models like Faster R-CNN have higher accuracy but slower inferences, making them unsuitable for real-time applications on devices with limitations [17]. Single-shot detectors like YOLO, SSD, and RetinaNet have faster performance by combining region proposal and classification in one pass, however running stock models in real time still require dedicated GPUs or TPUs [18, 19, 20]. The YOLO models have improved greatly over time, and as such, YOLOv5 and YOLOv8 are quite widely used. However, they still require optimization for use on hardware like the Raspberry Pi.

Other methods such as feature extraction algorithms (SIFT, SURF, or HOG) are rarely used in real-time sorting applications due to slow processing times and lighting variations [22, 23, 24].

2.3. Lightweight CNN Architectures

Deploying deep learning models on constrained devices requires both architectural efficiency and post-training optimization. Lightweight models such as MobileNet and EfficientNet-Lite are designed for mobile use, using methods like depthwise separable convolutions and inverted residual blocks [25, 26, 27, 28].

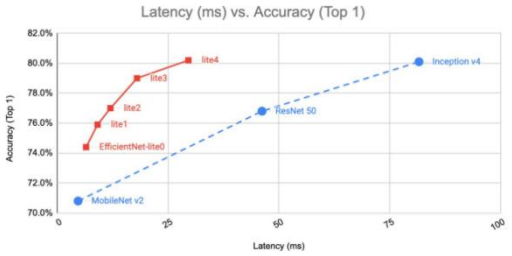

Figure 5. Latency vs Accuracy comparisons of MobileNet v2, ResNet 50, Inception v4, and EfficientNet-lite models (integer-only quantized models running on Pixel 4 CPU with 4 threads) [29].

Model compression techniques are often used in reducing size and inference time of models, optimizing them for limited hardware. Some commonly used are pruning, quantization, and knowledge distillation. Network pruning works by removing redundant connections specific weights from the model post-training [30, 31, 32]. Quantization converts model weights to lower precision, reducing memory usage [33]. Knowledge distillation trains a smaller model to mimic a larger network's outputs [34].

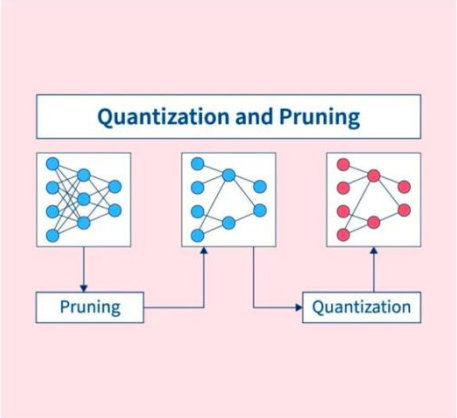

Figure 6. Illustration of pruning and quantization optimization methods [35].

Frameworks like TensorFlow Lite and ONNX Runtime have pipelines using such optimizations, but tradeoffs in performance and accuracy must be considered.

2.4. Vision-Based Sorting on Raspberry Pi

The Raspberry Pi has been extremely popular in medium size hobby projects due to its affordability and user-friendliness. The newer Raspberry Pi 5 model with 8GB RAM has a price-point of $80 and may be lower when bought second-hand or when buying an older version such as the 4B or Pico [36]. The newer model is beginning to be used more for AI due to its new additions of increased CPU performance and GPU acceleration, allowing for real-time inferences using smaller CNNs without external hardware accelerators like the Google Coral TPU or Nvidia Jetson Nano (Fig. 7).

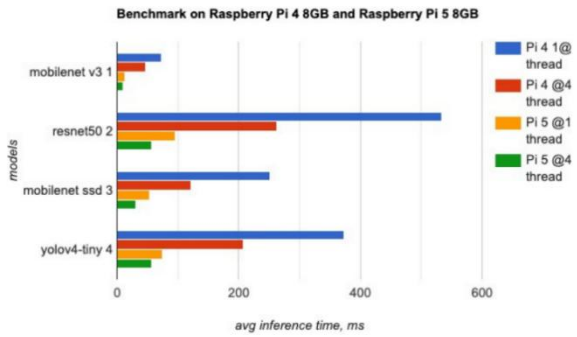

Figure 7. Comparison of MobileNetv3, ResNet50, MobileNet SSD, and YOLOv4-Tiny AI models on Raspberry Pi 4 vs 5 [37].

Previous successful work includes real-time traffic sign recognition and face detection/recognition [38, 39]. For this project, latency is a clear bottleneck, especially with larger models, as camera input, image preprocessing, model inference, and mechanical actuation must all occur in sequence within time limits. This means that model optimization is crucial.

3. SYSTEM DESIGN AND METHODOLOGY

The complete system consists of several main subsystems: the vision module, classification module, actuation module, and mobile application. All used codes can be found in appendix A, CAD files in appendix B, and the bill of materials in appendix C.

3.1. Vision Module

The vision module captures and preprocesses images for classification. The system is implemented with a Microsoft Lifecam Studio webcam. The camera runs with a max of 30 fps at 1080p (1920x1080 pixels) without any other loads on the Raspberry Pi. It is mounted vertically, facing the sorting area attached to the main gantry frame with 2 1/2”OD PVC pipes and 3D printed adapters (Figure 8). The camera connects to the Raspberry Pi5 via USB 2.0. For consistency in lighting reflections and to help with edge detection, 2 12’x 24’’ black ABS sheets are placed underneath the gantry. Using the Python OpenCV library, several optimizations can be applied to the images captured from the webcam before processing. First, each image is cropped to a set region of interest (ROI). The images are in RGB captured with a resolution of 1080p, which OpenCV converts to a 3D NumPy array of size: 1920 ×1080×3. To reduce the amount of unnecessary information processed, it is cropped to size, centered around the work area (Figure. 9): 420×420×3. This reduces the number of values being fed into the AI model each frame from 6220800 to 529200, a nearly 12 times reduction in data processed per frame.

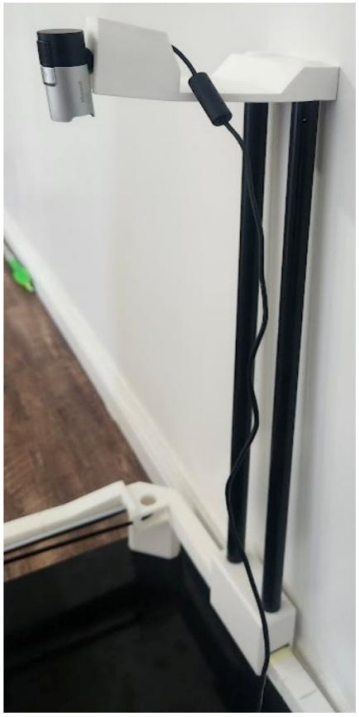

Figure 8. Microsoft Lifecam Studio webcam setup

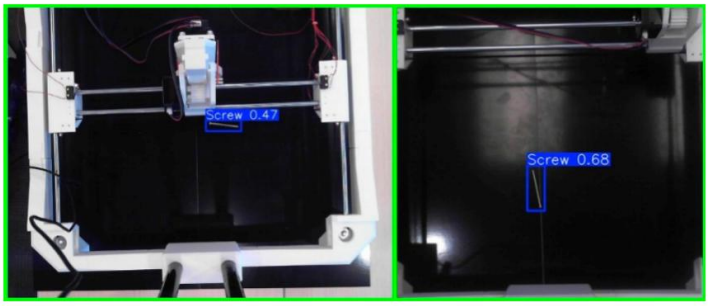

Figure 9. Comparison of before and after cropping images (Left: Uncropped [1920 × 1080]; Right: Cropped [420 × 420]).

Gaussian blurring is then applied to the image with a 3 × 3 kernel size to reduce sensor noise and increase consistency of the model [40, 41]. The kernel size used is 3 × 3 with k = 1, and standard deviation σ = 1. The cv.filter2D() functionfrom OpenCV does this [42].

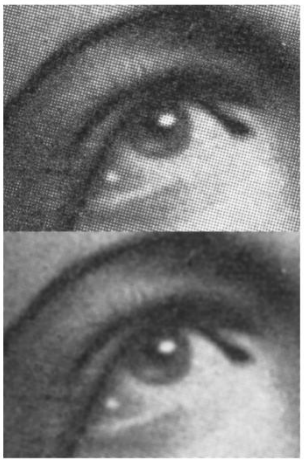

Figure 10.Image before and after gaussian blurring [43].

The image is then processed through Contrast-Limited Adaptive Histogram Equalization (CLAHE) to enhance local contrasts [44]. Unlike global histogram equalization, which enhances contrast uniformly accross the entire image, CLAHE enhances on smaller regions of the image while avoiding overamplification of noise. This helps especially when lighting scenarios vary. Images are then resized to 256 x 256 for compatibility with the neural network and then normalized by scaling pixel intensities of each channel from 0 to 1.

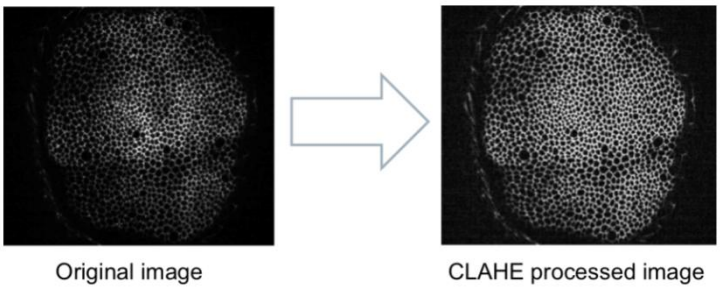

Figure 11. Image before and after CLAHE [45].

As the camera is mounted at a fixed position above the sorting bed, every image has a consistent perspective distortion. Instead of detecting a planar reference marker such as an ArUco tag, the camera's pose is assumed to be constant. So, a one-time calibration is needed to compute the homography matrix HϵR^{3x3} that maps points from the perspective view to that of an orthographic view [46]. OpenCV's "warpPerspective" function converts the images once the homography matrix is found.

3.1. Actuation Module

The actuation module is the subsystem that performs the physical sorting of the hardware. It is based on an H-bot gantry mechanism controlled through the Raspberry Pi with the "DigitalOutputDevice" class in the "gpiozero" Python library. Two pancake NEMA 17 stepper motors drive the X-Y planar motion using GT2 belts. 8mm OD steel rods act as rails for the end effector. The frame is 3D printed out of Tough PLA, being relatively inexpensive yet reliable. The end effector is mounted on a rack-and-pinion (Z-axis), also controlled by a NEMA 17 stepper motor. The end effector itself is a 3D printed claw shaped specifically to pick up different types of hardware. The actuation module also includes five limit switches, to prevent unexpected breakages.

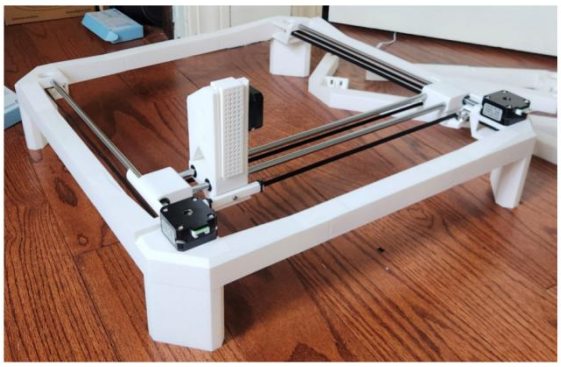

Figure 12. The H-Bot gantry of the system, with the end effector removed.

To move the end effector accurately, a virtual grid of size 64x64 is used. The center of a tensor taken after model inference is taken by:

Table 1: Gantry Performance Metrics

Metric	Value
Max XY deviation	4.4 mm
Standard deviation	2.5 mm
Maximum travel speed	150 mm/s
Avg object placement time	3.2s

to that position based on the coordinates of the tensor's center, lowers the end effector, and brings the piece of hardware to the bin it corresponds to.

3.2. User Mobile Application

To provide user interaction and accessibility, a mobile application was developed using Flutter and Android Studio. The application communicates to the Pi through Firebase's realtime database, allowing communication over long-distance. The application has an account system to

link the raspberry pi to the user's mobile device. The app has several features, the first being the z axis calibration. As the actuation module does not have a limit switch on the z-axis, the user needs to manually calibrate the z axis by setting the zero with the app when the end effector is touching the sorting bed. Another feature is that the user starts the sorting by sending a start signal from the app. When sorting, the app retrieves images from the real-time firebase database, producing a live feed of the sorting for the user. Errors are also shown to the user through the application.

Figure 13. Screenshot of the Application

3.3. Training and Methodology

To evaluate detection across different CNN models, various models with different settings were trained on the same dataset. The core objective was to find a model + settings that allowed for both accurate segmentation and computational efficiency. All training was done on a workstation using an AMD Radeon Pro W7500 GPU and all models were evaluated on a Raspberry Pi 5 with 8 GB RAM.

A custom dataset was made by manually labeling 505 frames collected from the vision module described in section A. Each frame was annotated in Roboflow with 200 frames of screws, 100 of nuts, and 100 standoffs. The dataset was split 70% training, 20% testing, and 10% validation. The following data augmentations were applied to reduce

$$(x_{\text{center}}, y_{\text{center}}) = \frac{x_1 + x_2}{2}, \frac{y_1 + y_2}{2}$$

where x1, x2, y1, y2 are the corners of the bounding box. The Raspberry Pi then sends the signals to move the end effect or overfitting: 15◦ rotations and contrast adjustment, with the total dataset consisting of 656 images (which is sufficient due to controlled lighting, camera position, and webcam conditions). All images were sized to 256 × 256 before training.

Several CNN-based object detection models were trained using Tensorflow and Keras: YOLOv8-Nano, MobileNetV2-SSD, EfficientDet-D0, Yolov11-Nano with Ultralytics [47]. All models predict bounding box coordinates with class probabilities.

Models were trained with the Adaptive Moment Estimation (ADAM) optimizer [48]. The models were trained for 80 epochs each and batch size 16. After training, models were converted to the PyTorch .pt format and quantized (float16 and int8) using the TensorFlow Lite Post-Training Quantization toolkit.The models are then downloaded onto the Raspberry Pi, and each tested with 20 images each where mAP, precision, and inference times were recorded.

4. RESULTS

Performance was averaged over 20 randomized test images, each containing multiple hardware components under controlled lighting and over the same background. Models trained with the Adam optimizer consistently outperformed those using RMSProp in accuracy across training epochs.The quantization reduced the sizes of the models by ~60% and run ~15 ms faster per inference with <1% accuracy losses.

Table 2: Quantitative evaluation of the tested models.

Model	mAP 0.5	Inference Time (ms)	Model Size (MB)	Quant.	Power Draw (W)
YOLOv8-Nano	0.887	145	5.4	int8	5.1
YOLOv11-Nano	0.915	125	5.0	int8	5.0
Mobile-NetV2-SSD	0.829	190	7.3	float16	5.4
Efficient-Det-D0	0.902	325	15.8	float16	6.0

YOLOv11-Nano is the most favorable for embedded real-time use, as it performed the best across all categories. YOLOv8-Nano performed slightly worse than YOLOv11 mainly in mAP, power draw, and inference time. EfficientDetD0 had the second highest mAP but had significantly higher latency. MobileNetV2-SSD lacked in both inference time and accuracy compared to the other three. All models were evaluated on live feed images. All models also had trouble and occasionally had false negatives when pieces of hardware were overlapped. This is likely due to lighting conditions and the reflectivity of some pieces of hardware causing stacked hardware to no longer appear to be a type of hardware the CNN recognizes.

5. DISCUSSION

This evaluation of various lightweight models on the Raspberry Pi indicates the need of balance between accuracy and speed of models. YOLOv11-Nano was chosen as the final model due to its fast inference times (125ms) and detection accuracy. EfficientDet-D0, while being the second most accurate of the models tested (mAP@0.5 0.902), had a significantly higher inference time (325ms), almost double that of YOLOv8-Nano. In larger or more constrained systems, this small difference in inference time decides which model is used in a specific application. This shows a big challenge in AI models on resource-constrained devices: larger models with higher accuracies designed for desktop or cloud environments often fail to meet the latency or processor requirements of that of embedded systems. Also, MobileNetV2-SSD although lightweight and with acceptable inference times, was less reliable than YOLO models.

An additional consideration is robustness under deployment variability. Although the dataset and testing pipeline controlled for lighting, camera pose, and perspective distortion, reflections on metallic hardware or shadows introduced by environmental changes may impair detection. This points to a broader question of how embedded vision systems generalize when removed from

laboratory settings. Domain adaptation, active learning pipelines, or real-time recalibration could strengthen resilience in uncontrolled factories or workshops.

From a systems integration perspective, the results illustrate the critical role of inference speed as a bottleneck in robotic manipulation loops. A detector operating at 8 fps versus 3 fps both alters output speeds but also shifts the design requirements of the actuation system. As a result, model selection for embedded deployments must be evaluated within an end-to-end framework rather than in isolation.

Beyond the hardware-sorting task, the methodology has implications for scalable automation in small- to medium-sized enterprises (SMEs). Many SMEs lack the capital to invest in industrial-grade vision systems costing tens of thousands of dollars. Showing that reliable detection and sorting can be achieved for under $300 suggests automation for workshops, maker spaces, and prototyping environments. Additionally, the modular design of the H-bot gantry and mobile application interface makes the system extensible: the same platform could be adapted to sort electronic components, agricultural products, or recyclables with minimal dataset retraining.

Future research should explore hybrid approaches. Techniques such as knowledge distillation could compress accurate but heavy detectors into lightweight student networks, while neural architecture search (NAS) could automate the exploration of architectures tuned to Raspberry Pi-class devices. Leveraging temporal information across video frames, rather than treating each frame independently, could reduce single-frame errors and improve reliability in cluttered scenes [49, 50].

6. CONCLUSIONS

This paper presented the design, training, optimization, and deployment of a low-cost hardware sorting system powered by a Raspberry Pi 5. By evaluating and comparing multiple lightweight CNN-based object detection architectures, this work identified the optimal trade-offs between detection accuracy, inference latency, and resource consumption necessary for practical embedded deployment

The YOLOv11-Nano model achieved a mAP of 91.5% using inferences of around 125 ms on the Raspberry Pi. This speed and precision allow real-time robotic sorting of screws, nuts, and standoffs with high reliability. The system has a low hardware cost (under $300 USD) and a 3D printable modular design to emphasize its accessibility for small workshops and prototyping scenarios.

The broader implication is that affordable and modularautomation can extend beyond large enterprises into small workshops, prototyping labs, and educational environments. With minor dataset retraining, the system could scale to diverse fields, from electronic component sorting to agricultural packaging. As embedded AI hardware continues to evolve, integrating accelerators such as Google Coral TPU or NVIDIA Jetson modules could further expand system capacity while staying affordable.

This research demonstrates that embedded AI is not limited to isolated inferences but can support complete systems with real industrial relevance. The project provides a blueprint for future research into low-cost, scalable, and resilient automation systems, addressing the gap between AI models and practical deployment in resource-constrained environments.

REFERENCES

[1] Horgan, Terrence G., Noelle K. Herzog, and Sarah M. Dyszlewski. "Does your messy office make your mind look cluttered? Office appearance and perceivers' judgments about the owner's personality." Personality and Individual Differences 138 (2019): 370-379.

[2] Ely, Robert, and Anne E. Adams. "Unknown, placeholder, or variable: what is x?." Mathematics Education Research Journal 24.1 (2012): 19-38.

[3] Jolles, Jolle W. "Broad-scale applications of the Raspberry Pi: A review and guide for biologists." Methods in Ecology and Evolution 12.9 (2021): 1562-1579.

[4] Hoelz, Kevin, Lukas Kleinhans, and Sven Matthiesen. "Wood screw design: influence of thread parameters on the withdrawal capacity." European Journal of Wood and Wood Products 79.4 (2021): 773-784.

[5] Jia, Feiyu, Yongsheng Ma, and Rafiq Ahmad. "Review of current vision-based robotic machine-tending applications." The International Journal of Advanced Manufacturing Technology 131.3 (2024): 1039-1057.

[6] Ibrahim, Isa Ali, and Muhammad Ahmad Baballe. "The Ups and Downs of Robotic Arms: Navigating the Challenges." Journal homepage: https://gjrpublication. com/gjrecs 4.05 (2024).

[7] Snelson, Chareen. "YouTube across the disciplines: A review of the literature." MERLOT Journal of Online learning and teaching (2011).

[8] Sebell, Rasmus, and George Wahlberg. "Skittles and M&M's sorting machine: Design and development of a color sorting machine." (2024).

[9] Lin, Chao-wen, et al. "The effects of reflected glare and visual field lighting on computer vision syndrome." Clinical and Experimental Optometry 102.5 (2019): 513-520.

[10] Zangana, Hewa Majeed, Ayaz Khalid Mohammed, and Firas Mahmood Mustafa. "Advancements in edge detection techniques for image enhancement: A comprehensive review." International Journal of Artificial Intelligence & Robotics (IJAIR) 6.1 (2024): 29-39.

[11] Hassanein, Allam Shehata, et al. "A survey on Hough transform, theory, techniques and applications." arXiv preprint arXiv:1502.02160 (2015).

[12] Hashemi, Nazanin Sadat, et al. "Template matching advances and applications in image analysis." arXiv preprint arXiv:1610.07231 (2016).

[13] Krizhevsky, Alex, Ilya Sutskever, and Geoffrey E. Hinton. "Imagenet classification with deep convolutional neural networks." Advances in neural information processing systems 25 (2012).

[14] Simonyan, Karen, and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." arXiv preprint arXiv:1409.1556 (2014).

[15] He, Kaiming, et al. "Deep residual learning for image recognition." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

[16] Huang, Gao, et al. "Densely connected convolutional networks." Proceedings of the IEEE conference on computer vision and pattern recognition. 2017.

[17] Ren, Shaoqing, et al. "Faster r-cnn: Towards real-time object detection with region proposal networks." Advances in neural information processing systems 28 (2015).

[18] Redmon, Joseph, et al. "You only look once: Unified, real-time object detection." Proceedings of the IEEE conference on computer vision and pattern recognition. 2016.

[19] Liu, Wei, et al. "Ssd: Single shot multibox detector." European conference on computer vision. Cham: Springer International Publishing, 2016.

[20] Lin, Tsung-Yi, et al. "Focal loss for dense object detection." Proceedings of the IEEE international conference on computer vision. 2017.

[21] Redmon, Joseph, and Ali Farhadi. "YOLO9000: better, faster, stronger." Proceedings of the IEEE conference on computer vision and pattern recognition. 2017.

[22] Cruz-Mota, Javier, et al. "Scale invariant feature transform on the sphere: Theory and applications." International journal of computer vision 98.2 (2012): 217-241.

[23] Bay, Herbert, et al. "Speeded-up robust features (SURF)." Computer vision and image understanding 110.3 (2008): 346-359.

[24] Kitayama, Masaki, and Hitoshi Kiya. "HOG feature extraction from encrypted images for privacy-preserving machine learning." 2019 IEEE international conference on consumer electronics-Asia (ICCE-Asia). IEEE, 2019.

[25] Howard, Andrew G., et al. "Mobilenets: Efficient convolutional neural networks for mobile vision applications." arXiv preprint arXiv:1704.04861 (2017).

[26] Tan, Mingxing, and Q. V. Le. "EfficientNet: rethinking model scaling for convolutional neural networks. CoRRabs/1905.11946 (2019)." arXiv preprint arXiv:1905.11946 (1905).
[27] Chollet, François. "Xception: Deep learning with depthwise separable convolutions." Proceedings of the IEEE conference on computer vision and pattern recognition. 2017.
[28] Sandler, Mark, et al. "Mobilenetv2: Inverted residuals and linear bottlenecks." Proceedings of the IEEE conference on computer vision and pattern recognition. 2018.
[29] Sangar, Gopal, and Velswamy Rajasekar. "Optimized classification of potato leaf disease using EfficientNet-LITE and KE-SVM in diverse environments." Frontiers in Plant Science 16 (2025): 1499909.
[30] LeCun, Yann, et al. "Backpropagation applied to handwritten zip code recognition." Neural computation 1.4 (1989): 541-551.
[31] Hassibi, Babak, David G. Stork, and Gregory J. Wolff. "Optimal brain surgeon and general network pruning." IEEE international conference on neural networks. IEEE, 1993.
[32] Han, Song, Huizi Mao, and William J. Dally. "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding." arXiv preprint arXiv:1510.00149 (2015).
[33] Banner, Ron, Yury Nahshan, and Daniel Soudry. "Post training 4-bit quantization of convolutional networks for rapid-deployment." Advances in neural information processing systems 32 (2019).
[34] Hinton, Geoffrey, Oriol Vinyals, and Jeff Dean. "Distilling the knowledge in a neural network." arXiv preprint arXiv:1503.02531 (2015).
[35] Gupta, Manish, and Puneet Agrawal. "Compression of deep learning models for text: A survey." ACM Transactions on Knowledge Discovery from Data (TKDD) 16.4 (2022): 1-55.
[36] Johnston, Steven J., and Simon J. Cox. "The raspberry Pi: A technology disrupter, and the enabler of dreams." Electronics 6.3 (2017): 51.
[37] Minott, David, Salman Siddiqui, and Rami J. Haddad. "Benchmarking Edge AI Platforms: Performance Analysis of NVIDIA Jetson and Raspberry Pi 5 with Coral TPU." SoutheastCon 2025. IEEE, 2025.
[38] Isa, Ida Syafiza Binti Md, Ja Yeong Choy, and Nur Latif Azyze Bin Mohd Shaari. "Real-time traffic sign detection and recognition using Raspberry Pi." International Journal of Electrical and Computer Engineering 12.1 (2022): 331.
[39] Zamir, Muhammad, et al. "Face detection & recognition from images & videos based on CNN & Raspberry Pi." Computation 10.9 (2022): 148.
[40] Gedraite, Estevão S., and Murielle Hadad. "Investigation on the effect of a Gaussian Blur in image filtering and segmentation." Proceedings ELMAR-2011. IEEE, 2011.
[41] Haddad, Richard A., and Ali N. Akansu. "A class of fast Gaussian binomial filters for speech and image processing." IEEE Transactions on Signal Processing 39.3 (1991): 723-727.
[42] Gangal, Ayushe, Peeyush Kumar, and Sunita Kumari. "Complete scanning application using OpenCv." arXiv preprint arXiv:2107.03700 (2021).
[43] Dobusch, Leonhard, and Gordon Mueller-Seitz. "Strategy as a practice of thousands: the case of Wikimedia." Academy of Management Proceedings. Vol. 1. Briarcliff Manor, NY 10510: Academy of Management, 2012.
[44] Pizer, Stephen M., R. Eugene Johnston, James P. Ericksen, Bonnie C. Yankaskas, Keith E. Muller Medical Image Display Research Group. "Contrast-limited adaptive histogram equalization: Speed and effectiveness."Proceedings of the first conference on visualization in biomedical computing, Atlanta, Georgia. Vol. 337. 1990.
[45] Reza, Ali M. "Realization of the contrast limited adaptive histogram equalization (CLAHE) for real-time image enhancement." Journal of VLSI signal processing systems for signal, image and video technology 38.1 (2004): 35-44.
[46] Hamey, Leonard GC, Jon A. Webb, and I-Chen Wu. "Low-level vision on Warp and the Apply programming model." Parallel computation and computers for artificial intelligence. Boston, MA: Springer US, 1988. 185-199.
[47] Glenn Jocher et al., "UltralyticsGithub Repository". https://github.com/ultralytics/ultralytics
[48] Kingma, Diederik P., and Jimmy Ba. "Adam: A method for stochastic optimization." arXiv preprint arXiv:1412.6980 (2014).
[49] Scheffel, Jan, and Kristoffer Lindvall. "Temporal smoothing-A step forward for time-spectral methods." Computer Physics Communications 270 (2022): 108173.

[50] Liu, Jiaming, et al. "M3SOT: Multi-frame, multi-field, multi-space 3D single object tracking." Proceedings of the AAAI Conference on Artificial Intelligence. Vol. 38. No. 4. 2024.

APPENDIX A: SUPPLEMENTARY MATERIAL

A. Actuation Module Code

```
directionLeft = DigitalOutputDevice(21)
stepLeft = DigitalOutputDevice(20)
directionRight = DigitalOutputDevice(8)
stepRight = DigitalOutputDevice(7)
directionClaw = DigitalOutputDevice(10)
stepClaw = DigitalOutputDevice(9)
directionZ = DigitalOutputDevice(3)
stepZ = DigitalOutputDevice(2)
global Xcoord
global Ycoord
Xcoord = 0 # temp
Ycoord = 0 # temp

def move_up():
        global Ycoord
        Ycoord += 1
        directionLeft.value = False
        directionRight.value = True
        move()

def move_left():
        global Xcoord
        Xcoord -= 1
        directionLeft.value = True
        directionRight.value = True
        move()

def move_down():
        global Ycoord
        Ycoord+=1
        directionLeft.value = True
        directionRight.value = False
        move()

def move_right():
        global Xcoord
        Xcoord+=1
        directionLeft.value = False
        directionRight.value = False
        move()

def move():
        for i in range(get_steps()):
                stepRight.on()
                stepLeft.on()
                sleep(0.00005)
                stepRight.off()
                stepLeft.off()
                sleep(0.00005)

def goto(X, Y):
        global Xcoord
        global Ycoord
        move_Zup()
        while Xcoord != X:
                if Xcoord > X:
                        move_left()
                elif Xcoord < X:
                        move_right()
        while Ycoord != Y:
                if Ycoord > Y:
                        move_down()
                elif Ycoord < Y:
                        move_up()
        move_Zdown()
```

B. Training Dataset

The dataset used for training the models of this project can be found with this link: https://universe.roboflow.com/ sortingpi/hardwaredetection-ag0gc

Appendix B: Project CAD Files
CAD files for the project can be downloaded with this link: https://www.dropbox.com/scl/fi/e5hl25m6b0duf7nwmouhi/SortingPiCADFiles.zip?rlkey=ju0mcuhwfijsk8vxekox0os0k&st=jox8arrj&dl=1

Appendix C: Bill of Materials

Table 3: Bill of Materials.

Part Name	Description	Quantity	Unit Cost ($)	Total Cost ($)
Raspberry Pi 5	8 GB RAM	1	80.00	80.00
Raspberry Pi 5 Power Supply	5.1V 27W	1	12.59	12.59
16 GB SD Card	Any size >8GB	1	12.99	12.99
Usongshine Nema 17 Stepper Motor	5 ct	1	30.99	30.99
BTT TMC2209 V1.3 Stepper Motor Driver	4 ct	1	22.99	22.99
Timing Belt and Pulley Kit	Amazon Kit, includes pulleys	1	16.99	16.99
Microsoft Lifecam Studio	Most webcams will work	1	60.00	60.00
Linear Motion Rod 8mm OD	2 ct	3	11.99	35.97
Total				272.52