Comparative analysis of Yolo based algorithms in solar panel detection using resource-constrained hardware

Napious Mukandagumbo[1*], *Theo* van Niekerk[1], *John* Fernandes[2] and *Kyla* Burden[1]

[1] Department of Mechatronics, Nelson Mandela University, South Africa,
[2] Department of Marine Engineering, Nelson Mandela University, South Africa,

Abstract. This study presents a comparative analysis of YOLO models for solar panel detection with the implementation on edge devices. With the rapid adoption of renewable energy, the need for monitoring and inspecting solar panel structures is also increasing. This study dives into different YOLO models for solar panel detection. It also compares the use of CLAHE and Histogram Equalization as preprocessing techniques and investigates dataset augmentation to increase its robustness. During training, YOLOv8 emerged as the strongest model with a higher F1 Score of above 96% and accuracy, also proving to have the better inference rate on both the Raspberry Pi and Google Coral Accelerator.

1 Introduction

Over the last twenty years, renewable energy use has been on the rise due to the global transition to clean energy. This development has been a catalyst for the solar energy industry, putting it at the forefront of this transition and resulting in the development of sustainable infrastructure. It has also led to the development of solar farms with thousands of photovoltaic (PV) panels, which require frequent inspection for maintenance, fault detection, and ensuring operational safety. Traditional inspection methods, such as manual inspection methods, are labor-intensive, error-prone, and impractical in vast installations. [1]. This has led to the rise of automated inspections and the use of deep learning algorithms such as YOLO (You Only Look Once). However, deploying these models on resource-constrained edge devices, which are critical for portable inspection systems and cost effectiveness, poses a significant challenge. Concerning the application of georeferencing the correct solar panel whilst moving in a solar farm, a real-time approach is required. This work focuses on the detection of solar panels. Classifications into fault types are suggested as future work.

2 Literature review

This literature review examines past research on solar panel monitoring, identifying the gap in utilizing Raspberry Pi with Google Coral accelerators. Moving away from power-intensive GPUs, the review focuses on YOLO models as efficient alternatives that balance detection

*Corresponding author: s229701698@mandela.ac.za

accuracy with the practical power constraints required for effective edge deployment in solar farm environments.

Research has highlighted that even a small accumulation of dust on a solar panel can impact on the efficiency of the solar panel and, therefore, the efficiency of the whole solar farm. It has been estimated that output in solar power plants is lost due to dirt on solar panels, with an accumulation of $1g/cm^2$ of dust potentially reducing the yield of energy, resulting in a loss of R800Kw (peak) per year [2]. Considering that renewable energy is expected to increase to 50% of global energy production by the year 2040, with solar PVs amounting to 40% of that share, there is a critical need for scalable, efficient, and cost-effective methods to monitor and ensure the reliability of the solar establishments over time. [3, 4].

Historically, smaller installations have used manual inspections where an operator walks in and around the farm, checking on the solar panels occasionally, with larger companies outsourcing the inspection and cleaning to third parties. In most cases, these approaches are often reactive, where a deviation in output power triggers the response, which, however, can be both time-consuming and inefficient for the rapid maintenance required in large installations. [5]. Recent advances, however, have seen the deployment of camera-based systems integrated with robotics, where high-definition imagery is processed by convolutional neural networks (CNNs) to identify faults. Research by [8] combined image stitching via Speeded Up Robust Features (SURF) with Random Sample Consensus (RANSAC) and homography transformations, alongside post-processing implemented on Raspberry Pi platforms. While high-end hardware, such as GPUs (Graphic Processing Units) and FPGAs (Field Programmable Gate Arrays), has enabled real-time fault localization [3], [6], [7] these solutions overlook the requirements of low-power edge devices. [8].

Many existing studies focus on deploying deep learning models on high-end platforms such as NVIDIA GPUs or Jetson systems. [9] deployed a state-of-the-art system, achieving mAP of 92.7 % on the YOLOV11 model using the high-end platforms. Consequently, recent works have begun to address the development of lightweight models. For example, [10] used a Jetson Nano coupled with a UV light source for ultraviolet fluorescence inspections of solar panels. Their system achieved 89% precision in detecting defective PV cells, highlighting YOLO's adaptability to computationally limited edge devices. However, their research did not assess the use of other low-end and low-cost devices, such as a Raspberry Pi coupled with a Google accelerator for this application. Similarly, [11] evaluated soling detection using surveillance cameras and provided benchmarks across multiple edge platforms. They demonstrated that while FPGA implementations offer low latency, Vision Processing Units (VPUs), despite being cost-effective, exhibit slower inference times. Notably, Google Coral Accelerators (Edge TensorFlow Processing Units (TPUs)) remain underexplored for such applications, emphasizing the need for further hardware-specific optimizations in real-world solar farm settings.

While YOLO remains a dominant force in the real-time object detection arena, alternative architectures are also emerging. Yang et al (2024) proposed a lightweight decoder-only transformer model for defect detection in electroluminescence images of solar panels. The approach achieved competitive accuracy with 87.4 mAP latency performance on the GPU at 6.1ms [12]. However, its slowness in CPU performance underscores the need for a relevant, continued single-stage deployment of edge devices.

Contrast-Limited Adaptive Histogram Equalization (CLAHE), originally developed for medical imaging and face recognition [13], has been adapted for solar inspections to address illumination variability. And [14] (2023) applied CLAHE to thermal images, improving hotspot detection by redistributing pixel intensities while limiting noise amplification. Similarly, [13] demonstrated CLAHE's robustness in low-light conditions, a finding directly applicable to PV inspections under variable environmental lighting. These studies highlight

CLAHE's potential to enhance model generalizability without requiring large datasets critical for deployment on edge systems. Preprocessing and data set augmentation have been used to enhance datasets for robustness and efficiency. Restack identified that for deep learning applications in variable lighting conditions and environmental conditions, benefit from dataset augmentations [15].

Energy consumption is also a critical metric in the deployment of automated solar inspections on edge devices. While high-end GPUs and FPGAs offer excellent processing speeds, they are often impractical for portable systems due to high power requirements. Mira et al. (2024) provided valuable benchmarks using traditional machine learning techniques, which achieved 99% accuracy with drastically lower energy costs compared to conventional CNNs, albeit at significantly reduced frame rates [16]. However, for real-time detection tasks required in fault localization, YOLO's capacity to balance efficiency, accuracy, and FPS makes it a more viable choice, especially when hardware-specific optimizations are applied.

3 Methodology

This section dives into the architecture used and looks at the dataset preparation and augmentation done on the data. It compares different pre-processing techniques, namely CLAHE and Histogram equalization. CLAHE and Histogram Equalization were chosen because they directly address the uneven illumination, glare, and shadows common in outdoor solar farms. Compared to heavier methods like Retinex or deep-learning-based enhancement, these techniques offer a lightweight, real-time solution suitable for deployment on resource-constrained platforms such as the Raspberry Pi. Augmentation techniques employed to increase robustness are also addressed, particularly focusing on the challenges associated with variable lighting conditions.

3.1 Dataset description

The dataset for this project consists of approximately 2,000 images of solar panels taken from a solar farm, captured at various angles to increase diversity in the dataset.

3.1.1 Annotation process

The dataset was created by capturing a wide range of images of solar panels using a phone camera in the solar farm at Nelson Mandela University. This includes both still images and videos of solar panels as the camera moved through the rows of the farm. The resulting dataset contains over 2,000 images in JPG format, with annotations done using RoboFlow's annotation tools. A hybrid annotation approach was employed, combining polygon and bounding box methods for precise labelling. This hybrid approach ensures accurate annotations, particularly for images captured from various angles, enabling efficient iteration and model training.

3.2 Deep learning architecture

YOLO detectors are single-shot detectors (SSD) that use a single convolutional neural network (CNN) to directly predict the class and bounding boxes within an image. SSD is faster and more efficient compared to multiple-shot detectors since they eliminate the need for multiple stages and can be run in real time. The CNN structure is composed of three main components:

- **Backbone:** extracts features from input images
- **Neck:** Aggregates and transforms features across different scales
- **Head:** Performs final detection and classification

Fig. *1* Shows the adapted architecture followed in this paper, building upon the CNN architecture.

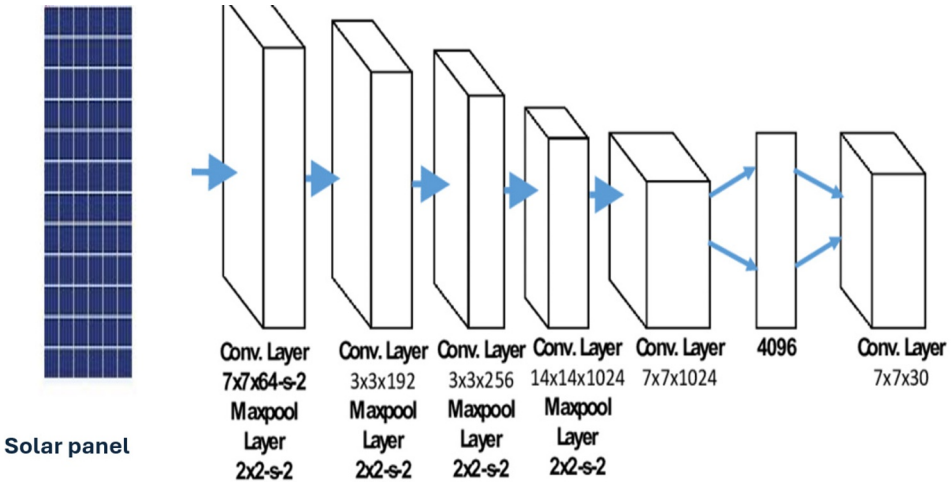

Fig. 1. Adapted YOLO-based CNN architecture used for solar panel detection.

3.2.1 YOLO working principle.

The YOLO algorithm divides an image into S×S grids, where each grid cell is responsible for localizing and predicting the presence of objects within its region. It also classifies the detected objects and assigns a confidence score that indicates the availability of an object and whether it has been correctly identified. In addition, YOLO treats object detection as a regression problem rather than a classification problem.

3.2.2 Bounding box regression

YOLO uses bounding box regression to predict the rectangular regions (bounding boxes) that show objects in an image. The algorithm determines the attributes of these bounding boxes, such as their coordinates, dimensions, and confidence scores, through a single regression step.

3.3 Dataset Preparation

The images are resized to 640 x 640 to be able to divide into the S x S grid. Images are resized to match YOLO's input requirements, which ensures compatibility with the models' architecture.

Capturing images on a solar farm presents unique challenges, primarily due to the varying lighting conditions throughout the day. These changes can cause significant color distortions in the images, making it necessary to apply preprocessing techniques.

3.3.1 Data augmentation

Data augmentation is done to improve model generalization and robustness under different lighting conditions and different camera angles. The following augmentation strategies were implemented:

- **Horizontal Flip:** Creating several versions of an image in various orientations gives the learning model more information to learn from. The augmentation creates an illusion that the panels are being captured from various angles.
- **Clockwise and Counter-clockwise Rotation ($\pm 90°$):** This simulates various camera angles in which the panels could be orientated.
- **Saturation Adjustment (±25%):** By simulating changes in illumination, time of day, or camera settings, this augmentation replicates how color saturation may change. It helps enhance detection in a range of environmental settings and enables the model to manage different illumination conditions.
- **Exposure Adjustment (±11%)**: The model can detect panels in both bright and dimly lit environments because exposure fluctuations replicate real-world lighting situations, such as fluctuating sunlight or cloud cover.
- **Blur (up to 2.5 px):** A blur simulates an out-of-focus camera, which is beneficial in the farm environment and for the application where there is vibration.

3.3.2 CLAHE

CLAHE is a tile-based approach that works by dividing the image into non-overlapping regions. Each tile is processed independently, and then the results are combined through interpolation to ensure a smooth transition. This method normalizes intensities across the image and ensures that noise is minimized, making it particularly useful for handling images with inconsistent lighting in solar farm environments.

The dataset is pre-processed using CLAHE and Histogram equalization, which is then augmented using the augmentation techniques. Pre-processed results can be seen in Table 1 in Section 4.2.

3.4 Evaluation metrics

The following metrics were used to determine the feasibility of the YOLO model using True Positives, False Positives, True Negatives, and False Negatives. Accuracy is the overall correctness of the model, as shown in (1).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \quad (1)$$

Precision is how often the model guesses correctly, which is the accuracy of positive predictions. Ratio of true positive predictions as shown in (2).

$$Precision = \frac{TP}{TF + TP} \quad (2)$$

'Recall' is the measure of how well the model identifies actual positive instances and is calculated as shown in (3).

$$Recall = \frac{TP}{TF + FN} \quad (3)$$

The F1 score is a statistical measure of the accuracy of the models. This assesses the performance of the model more reliably, as it is a balanced metric, as shown in (4).

$$F1_{Score} = 2 \times \frac{Precision\ x\ Recall}{Precision + Recall} \quad (4)$$

An epoch is the amount of time a dataset passes through an algorithm as the machine learning model is being trained.

3.5 Training process

To evaluate the model's performance, a comparative analysis was conducted, focusing on metrics such as mean Average Precision (mAP), precision, and F1 score. The F1 score will be the key performance metric of interest for this study.

The dataset was preprocessed using several techniques, including contrast adjustment through methods like Histogram Equalization, to address the challenges posed by varying lighting conditions in the solar farm. These preprocessing steps are essential for enhancing image quality, which is critical when dealing with natural environments where lighting and color distortion can affect image clarity.

The model will be trained using a version of YOLO with and without the preprocessing techniques. The primary objective is to identify which model, and preprocessing combination provides the best F1 score performance.

3.5.1 Early stopping implementation

An early stopping mechanism was implemented to prevent overfitting during training. The process monitored validation mAP50 with a patience parameter of 50 epochs. Training automatically terminated when validation performance showed no improvement for fifty consecutive epochs, with the best-performing model weights retained. Implementation details: - Monitoring metric: Validation mAP50 - Patience: 50 epochs - Save strategy: Best weights only - Example: YOLOv5m stopped at epoch 181 (best: epoch 82)

3.6 Preprocessing technique comparison

The comparison between CLAHE and Histogram Equalization for preprocessing in YOLOv5 training revealed minor but notable differences, as depicted in Table 1.

Table 1. CLAHE vs histogram equalization

Metric	**CLAHE**	**Histogram equalization**	**Difference**
Precision	0.94	0.9435	-0.005
Recall	0.965	0.968	+0.003
Map50	0.982	0.979	-0.003
Map 50-95	0.948	0.944	-0.004

Histogram Equalization demonstrated superior precision and recall, indicating fewer false positives/negatives, while CLAHE outperforms in mAP50 and mAP50-95, showing better detection consistency across varying object sizes and overlap scenarios, after which the models are trained using CLAHE and on different architectures.

3.6.1 Real world impact of CLAHE pre-processing

Contrast-Limited Adaptive Histogram Equalization (CLAHE) enhances local contrast while limiting noise, enabling robust object detection under extreme or variable lighting. In solar farm environments, where moving shadows and glare create bright–shadow gradients, CLAHE can reveal features in shaded or low-contrast regions without degrading performance in well-lit areas. The authors in [16] also demonstrated that CLAHE-based preprocessing improved the detection of objects hidden in shadows and increased confidence scores in challenging scenes, supporting more reliable and efficient inspection operations.

3.7 Training models

During training, performance was evaluated using precision (P), recall (R), mean Average Precision (mAP50), and mean Average Precision across IoU thresholds from 0.5 to 0.95 (mAP50-95). After the proper preprocessing and augmentations, models were trained to determine the best model that works best with the available edge-enabled hardware. As discussed earlier, the key factor looked at is the F1 score. The setup on Fig. ***2*** Shows the setup for performing the tests using the Raspberry Pi CPU only for the YOLO models and the Coral USB accelerator. The setup allowed the conduction of tests for power consumption while running inference and while idle.

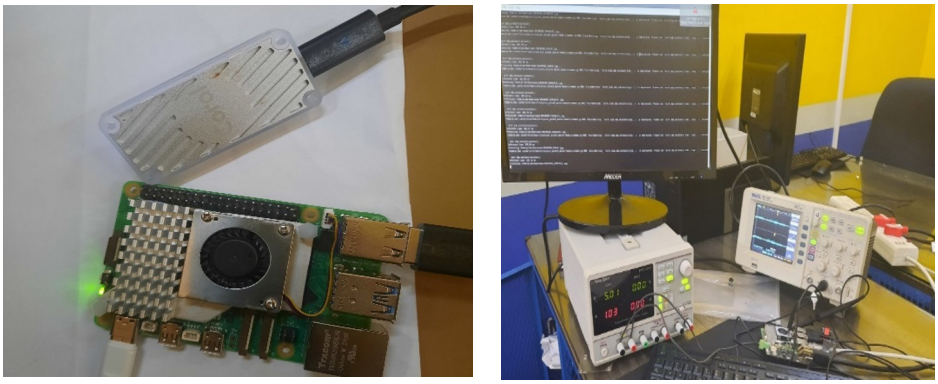

Fig. 2. Experimental setup for model testing. Left: Raspberry Pi with Coral USB accelerator for TPU-based inference. Right: Raspberry Pi CPU-only setup.

3.8 Coral accelerator

Running YOLO on the Coral USB accelerator includes converting the PyTorch models into an 8-bit quantized TensorFlow Lite format and compiling with the Edge TPU compiler. This means post-training INT8 quantization of the model weights: for YOLO v5, this is done by using export.py, while YOLO v8 uses the Ultralytics Python API or CLI with format = “edgetpu”. The edge TPU provides significant inference speed improvements over CPU-only processing, making it well-suited for real-time applications using resource-constrained hardware. The quantization is achieved as depicted in Fig. ***3***.

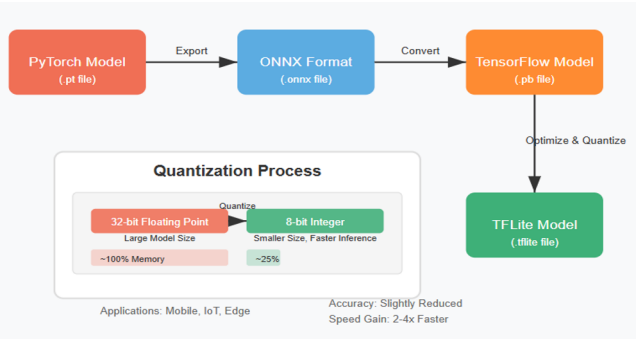

Fig. 3: YOLO model conversion to TensorFlow Lite model.

3.9 Metrics

The following metrics are used to determine the best model for solar panel detection on an edge device:

- **Inference speed:** This shows how fast the model can manage an input (measured in frames per second).
- **Latency:** This is the time the model takes between receiving the input and producing an output.
- **Model size:** This is the storage footprint of the model.
- **Power consumption:** This is the energy drawn by the system during inference. Power consumption is calculated by subtracting the idle power from the average power consumption, as the model is inferring. Idle current is at 0.76A with voltage stable at 5.1V.

4 Results and analysis

4.1 Baseline results

The baseline results without pre-processing or augmentation provide a reference point for training the model. Fig. *4* Indicates that both training and validation are steadily decreasing, which highlights that the model is learning effectively. Precision, recall, and mAP are all increasing with epochs, and there is no sign of overfitting, but they collapse on the validation dataset.

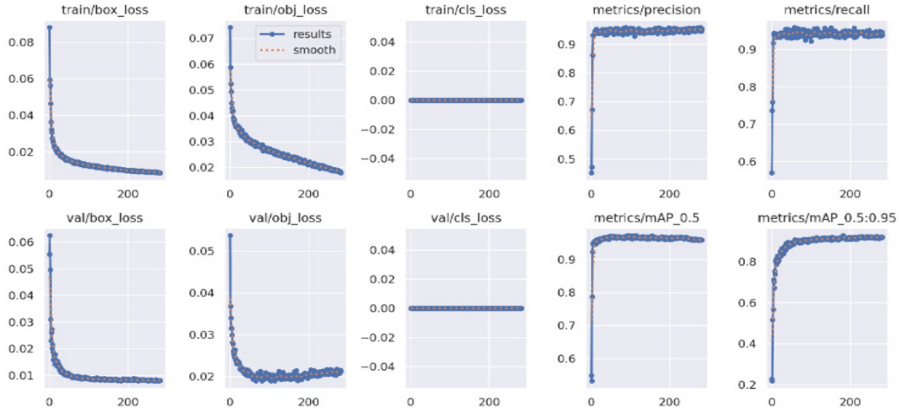

Fig. 4. Training and validation loss curves for YOLO without preprocessing or augmentation.

This contradiction reveals severe overfitting as the model memorizes simplistic training patterns but struggles in complex real-world scenarios. For instance, it fails to detect entire rows of solar panels under environmental changes, as shown in Fig. ***5***. The declining metrics highlight poor generalization, emphasizing the need for preprocessing and augmentations to enhance robustness to unseen data variations.

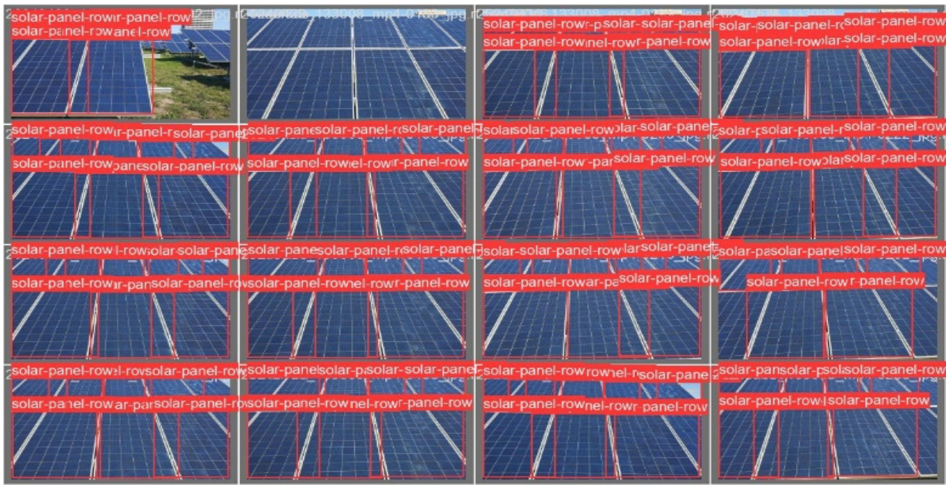

Fig. 5. Sample test results from a model trained without preprocessing.

4.2 Model comparison

The performance of various YOLO models was evaluated across different epochs, as shown in Table 2 . The table shows different models from YOLOv5 to v11, as the paper is trying to ascertain the best model.

Table 2. Model comparison table

Model	Epoch	Precision%	Recall%	F1 score%	mAP50%
Yolo v5s	50	94.8	96.2	95.4	97.9
	100	94.2	96.3	95.2	97.8
	300	95.2	94.7	94.9	96.9
Yolo v5m	50	93.6	97.3	95.4	98
	100	94.5	96.9	95.6	97.6
	300	94.5	96.9	95.6	97.6
Yolo v5l	50	94.3	96.8	95.5	97.7
	100	94.8	96.8	96.1	94.3
	300	95.4	95.2	95.3	95.2
Yolo v8s	50	94.7	96.7	95.7	97.5
	100	**95.1**	**96.7**	**95.8**	**97.9**
	300	95.2	96.6	95.9	97.9
Yolo v8m	50	94.2	96.7	95.6	96.8
	100	94.6	97.6	96.0	98.0
	300	94.6	97.6	96.0	98.0
Yolo-NAS	*	88.4	98.7	93.2	97.2
Yolov11	*	98.2	95.0	96.6	98.2
Roboflow 3	*	94.9	97.5	96.2	98.1

The following trends were observed from these results.

- Yolov5 models initially showed a strong precision and mean average precision at 50 epochs: however, the performance declines at three hundred epochs, showing signs of overfitting. The early stop mechanism was implemented to prevent the model from memorizing rather than generalizing well to unseen data. For instance, YOLOv5m training stopped at 181 epochs, with the best results reached at 82 epochs.
- The YOLOv8 variant consistently outperformed most other models. The smaller model achieved a much higher performance across all epochs with minimal signs of overfitting. At three hundred epochs, the v8s model achieved 95.2% precision and a 97,9% mAP, rivalling larger models despite its smaller computational footprint.
- YOLO-NAS, YOLOv11, and Roboflow 3: These models were evaluated without full retraining. YOLOv11 recorded the highest precision (98.2%) and competitive performance across other metrics. Roboflow 3 also showed strong results with an F1 score of 96.2% and mAP50 of 98.1%. YOLO-NAS stood out with the highest recall (98.7%) but had lower precision.

The early stopping and the saving of weights at a lower epoch number suggest that further training would not improve the model's generalization on the validation set. The decline indicates that the models started memorizing the training data rather than generalizing well to unseen data.

4.3 Scenario 1: Running YOLO on a Raspberry Pi (CPU-Only)

As a comparative problem, a few models were chosen based on compatibility and evaluated on the Raspberry Pi with just the CPU. The test results for the different models are shown in Table 3. Fig. 6 shows a few test results displaying confidence levels ranging from 0.38 to 0.92 generated by inferring some of the models on the Raspberry Pi test setup.

Fig. 6. Sample solar panel test results.

Table 3. Running on Raspberry PI (CPU only)

Model	Average Inference speed (FPS)	Model Size MB	Power consumption (W)
YOLOv5s	2.6	12	6.58
YOLOv5m	1.73	42	7.00
Yolov8s	1.28	22	7.09
Yolov8m	0.91	52	8.36

4.4 Scenario 2: Running YOLO on a Raspberry Pi with a Coral USB Accelerator

The Coral USB Accelerator (which contains an Edge TPU) is designed to handle quantized TensorFlow Lite models. Offloading inference computations to the TPU boosts inference speed and reduces latency, while also lightening the load on the Raspberry Pi's CPU. The performance is as shown in Table 4.

Table 4. Running inference on a Raspberry Pi with a Coral USB Accelerator.

Model	Inference speed (FPS)	Model Size MB	Power consumption (W)
YOLOv5s – int8	18.7	3.8	3.2
YOLOv5m -int8	12.3	10.7	3.5
Yolov8s-int8	15.8	6.1	3.3
Yolov8m-int8	9.4	13.6	3.6

4.5 Key insights

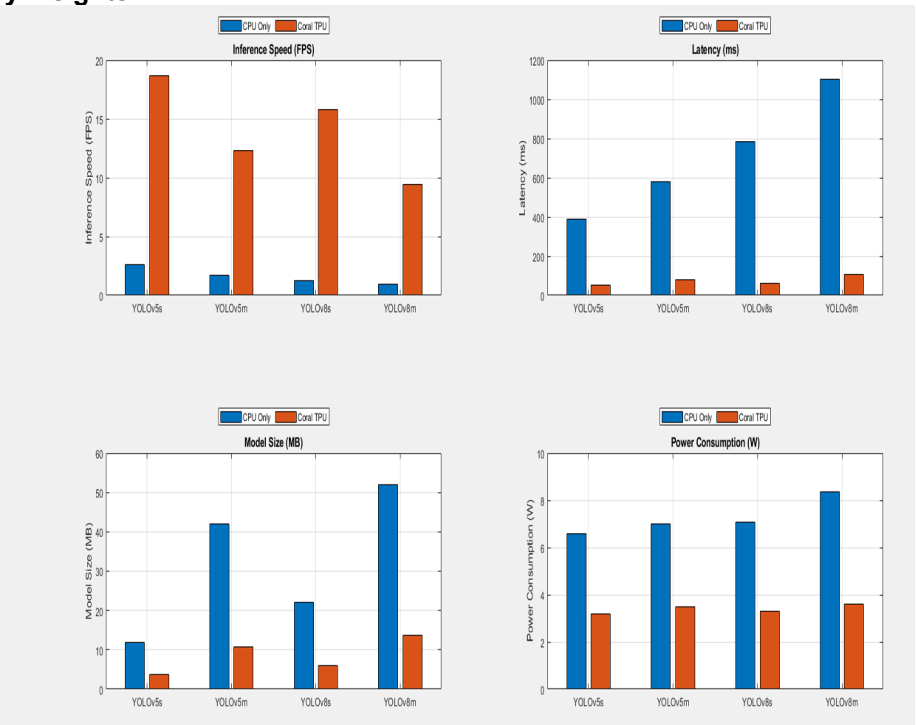

Fig. 6. Comparison of inference speed and latency when running YOLO models on Raspberry Pi (CPU-only) vs. Raspberry Pi with Coral USB Accelerator

Running the model on the TPU drastically reduces the latency time and dramatically increases the inference speed. Fig. Shows the comparison of the models.

5 Overall discussion

When comparing the performance metrics across YOLO variants, several patterns emerged. YOLOv5 models exhibited strong initial performance but showed signs of overfitting at higher epoch counts, necessitating early stopping techniques. Similarly, YOLOv8l training

ceased after one hundred epochs without improvement, with optimal results achieved at epoch 56. The early stopping mechanism effectively prevented overfitting. This approach optimizes computational efficiency while maintaining model performance. While YOLOv8m achieved the best balance of accuracy and computational efficiency, with an F1 score of 96.0% and mAP50 of 98.0%. YOLOv11 demonstrated slightly higher precision (98.2%), but YOLOv8s offered better overall performance and balance between accuracy and speed characteristics for edge deployment with an F1 Score of 95.9% and mAP OF 97.9%. YOLOv8s maintained strong detection capabilities whilst enabling superior inference performance even when changed to a quantized model. The implementation of CLAHE preprocessing provided marginal improvements in mAP metrics, particularly beneficial for challenging lighting conditions typical in outdoor solar farms. However, the small performance gap suggests that either preprocessing technique could be viable depending on specific deployment requirements. When deploying these models on resource-constrained devices, the performance differences became more pronounced. CPU-only deployment on Raspberry Pi resulted in low frame rates across all models, making real-time inspection challenging. However, integrating the Coral USB Accelerator dramatically improved performance, with inference speeds increasing by 8–10 times. This improvement makes real-time solar panel detection viable even on low-power edge devices. The 8-10x speed improvement (15.8 FPS) enables continuous movement with real-time detection and geo-tagging, reducing inspection time by 60-75%. Power consumption reduction will also enable the vehicle used for monitoring to travel longer distances.

6 Conclusion and future work

This study sought to compare various YOLO models for solar panel detection with the implementation on edge devices. The study showed that deep learning models can be used to effectively detect solar panels in real time with higher accuracy, and more feasible if a USB Google Coral accelerator is included. Although YOLOv8m achieved the highest F1 score overall, YOLOv8s emerged as the optimal model for edge deployment, balancing superior inference speed and accuracy, making automated solar farm inspections feasible on resource-constrained devices. The comparison of the preprocessing techniques also revealed that using CLAHE results in improvements in robustness, particularly in variable lighting conditions common in outdoor environments.

This study forms part of an ongoing project using an Automated Guided Vehicle for solar farm monitoring. The YOLOv8s model identified will be integrated into the core vision system of the AGV, enabling it to navigate the solar farm simultaneously performing detection and inspection tasks. AGV integration enables predictive maintenance strategies across installation sizes.

Future work could include:

- Fault classification - extend the work beyond panel detection to fault classification, checking for cracks, hotspots, and soiling.
- Integration with autonomous system – as a background for autonomous work, the work can be combined with autonomous vehicles like drones and other mobile robots.

References

1. S. Jaybhaye, O. Thakur, R. Yardi, V. Raut, A. Raut, Solar panel damage detection and localization of thermal images. *J. Fail. Anal. Prev.* **23**, 1980–1990 (2023)

2. J.K. Kaldellis, A. Kokala, Quantifying the decrease of the photovoltaic panels' energy yield due to phenomena of natural air pollution disposal. *Energy* **35**, 4862–4869 (2010)
3. M. Meribout, V.K. Tiwari, J.P. Peña Herrera, A.N.M.A. Baobaid, Solar panel inspection techniques and prospects. *Measurement* **112466** (2023). https://doi.org/10.1016/j.measurement.2023.112466
4. L. Kruitwagen, K.T. Story, J. Friedrich, L. Byers, S. Skillman, C. Hepburn, A global inventory of photovoltaic solar energy generating units. *Nature* **598**, 604–610 (2021)
5. Y. Higuchi, T. Babasaki, Classification of causes of broken solar panels in solar power plants. in *Proc. IEEE INTELEC*, 127–132 (2017)
6. N.Venkatesh, V. Sugumaran, Fault diagnosis of visual faults in photovoltaic modules: A review. *J. Energy Eng.* (2021). https://doi.org/10.1080/15435075.2020.1825443
7. S.N. Venkatesh, V. Sugumaran, Machine vision-based fault diagnosis of photovoltaic modules using a lazy learning approach. *Measurement* **191** (2022). https://doi.org/10.1016/j.measurement.2022.110786
8. S. Kole, C. Agarwal, T. Gupta, S. Singh, SURF and RANSAC: A conglomerative approach to object recognition. *Int. J. Comput. Appl.* **109**, 7–9 (2015) https://doi.org/10.5120/19174-0645
9. A. Ghahremani, S.D. Adams, M. Norton, S.Y. Khoo, A.Z. Kouzani, Detecting defects in solar panels using the YOLO v10 and v11 algorithms. *Electronics* **14**, 2 (2025).
10. A.B. Di Renzo, C.R. Zamarreno, C. Martelli, J.C.C. Da Silva, Edge device for ultraviolet fluorescence inspection of photovoltaic panels. in *Proc. IEEE Sensors* (2023)
11. W. Zhang, et al., SoilingEdge: PV soiling power loss estimation at the edge using surveillance cameras. *IEEE Trans. Sustain. Energy* **15**, 556–566 (2024). https://doi.org/10.1109/tste.2023.3320690
12. Y. Yang, J. Zhang, X. Shu, L. Pan, M. Zhang, A lightweight Transformer model for defect detection in electroluminescence images of photovoltaic cells. *IEEE Access* (2024). https://doi.org/10.1109/access.2024.3520239
13. P. Musa, F. Al Rafi, M. Lamsani, A review: Contrast-limited adaptive histogram equalization (CLAHE) methods to help the application of face recognition. in *Proc. 3rd Int. Conf. Informatics and Computing (ICIC)* (2018)
14. Yolo training data augmentation techniques | Restackio, Accessed: May 01, 2025. [Online]. Available: https://www.restack.io/p/data-augmentation-answer-yolo-training-techniques-cat-ai
15. J.L. Mira, J. Barba, F.P. Romero, M.S. Escolar, J. Caba, J.C. López, Benchmarking of computer vision methods for energy-efficient high-accuracy olive fly detection on edge devices. *Multimed. Tools Appl.* (2024). https://doi.org/10.1007/s11042-024-18589-y
16. T-s. Wang, G.T. Kim, M. Kim, J. Jang, Contrast Enhancement-Based Preprocessing Process to Improve Deep Learning Object Task Performance and Results. *Applied Sciences*, *13*(19), 10760. (2023) https://doi.org/10.3390/app131910760