Comparative study of small objects detection using YOLO and DOTA dataset

Jonas Vilasboas Moreira

***Abstract*—Detecting small objects in aerial images presents a significant challenge due to low resolution and complex backgrounds. This study performs a comparative analysis of multiple YOLO models—YOLOv5n/m, YOLOv8n/m, YOLOv11n/m, and YOLOv12n/m—using the DOTA dataset to evaluate their performance in detecting small-scale features. Models are assessed using metrics like mAP, precision, recall, and latency across consistent training settings. The best-performing model, YOLOv11m, demonstrated notable improvements in detection accuracy and efficiency, especially when trained with preprocessed image slices. This model was further optimized for TinyML applications and successfully deployed on a Raspberry Pi device, validating its suitability for real-time, low-power embedded vision tasks.**

***Index Terms*—Computer Vision, Small Object Detection, YOLO, DOTA dataset, TinyML, Edge AI**

I. Introduction

COMPUTER vision is a field of artificial intelligence that develops models capable of interpreting images and videos, simulating human visual perception. Its goal is to extract relevant information for tasks such as object recognition and detection. Detecting small objects in images is a significant challenge in computer vision research due to the low resolution of these objects and the high risk of them blending into the background. Applications such as recognizing vehicles, people, and other objects in aerial, urban, or satellite images often involve small objects, which require models capable of capturing details at small scales. Evaluating and comparing detection models specifically designed for small objects is essential for progress in areas such as surveillance, environmental monitoring, and public safety.

The objective of this research is to train and evaluate the performance of different versions of object detection algorithms applicable to computer vision tasks. After identifying the model with the best performance, additional training iterations will be conducted to further refine its parameters. Subsequently, the selected model will be optimized for TinyML applications and deployed on an embedded IoT platform, specifically a Raspberry Pi device. To accomplish this, the performance of the YOLOv5n, YOLOv5m, YOLOv8n, YOLOv8m, YOLOv11n, YOLOv11m, YOLOv12n, and YOLOv12m models will be systematically evaluated under identical experimental configurations and using the same dataset. The DOTA dataset, which consists of annotated aerial images, will be used for training and validation purposes.

The remainder of this paper is organized as follows: Section II reviews the relevant literature. The comparison proposal is described in Section III, while the comparison methodology is detailed in Section IV. Label conversion and image conversion are discussed in Sections V and VI, respectively. Dataset verification is covered in Section VII, and the test environment is described in Section VIII. The results are presented in Section IX, followed by the Tiny-ML experiment in Section X. Finally, Section XI provides concluding remarks and future directions.

II. Literature Review

Ultralytics' YOLOv5 [1] is a deep learning model for computer vision tasks, implemented using the PyTorch framework. It builds upon advances from previous object detection models, incorporating best practices from recent research and development. YOLOv5 has been widely adopted for tasks such as object detection, segmentation, and image classification due to its balance between computational efficiency and detection performance.

YOLOv8 [2] introduces architectural modifications that include modern backbone and neck components aimed at enhancing feature extraction and object detection performance. The model adopts an anchor-free split head, eliminating the need for predefined anchors, which simplifies the detection process and can improve overall efficiency. YOLOv8 was designed to balance accuracy and inference speed, making it suitable for real-time computer vision applications. Furthermore, it provides a set of pre-trained models with varying sizes and capacities, allowing for adaptation to different computational constraints and performance requirements.

YOLO11 [3] incorporates architectural improvements in both the backbone and neck components, with the objective of enhancing object detection performance and enabling more efficient processing of complex tasks. The model is designed to maintain a balance between detection accuracy and computational efficiency, making it suitable for deployment on edge devices, cloud platforms, and systems equipped with graphics processing units. In addition to object detection, YOLO11 extends support to tasks such as instance segmentation, image classification, pose estimation, and oriented object detection (OBB).

YOLO12 [4] [5] incorporates an Area Attention Mechanism designed to handle large receptive fields by dividing feature maps into equal-sized horizontal or vertical regions, aiming to reduce computational cost while maintaining detection performance. The model also integrates Residual Efficient Layer Aggregation Networks (R-ELAN), a feature aggregation module proposed to address optimization challenges commonly found in large attention-based models. The attention architecture is designed to integrate efficiently with the YOLO framework, streamlining operations to improve computational efficiency. YOLO12 extends its applicability beyond object

detection to include tasks such as instance segmentation, image classification, pose estimation, and oriented object detection (OBB). The model architecture focuses on achieving a trade-off between accuracy and computational complexity, with support for deployment on a variety of platforms, including edge devices and cloud environments.

DOTA [6] [7] [8] is a dataset specifically developed for object detection in aerial images, containing high-resolution images acquired from a variety of sensors and platforms. The dataset includes objects with varying scales, orientations, and shapes, and provides annotations using bounding boxes, enabling detailed object localization. DOTA has evolved over three major versions. The first version, DOTA-v1.0, comprises approximately 2,800 images with annotations for 15 object categories. DOTA-v1.5 retains the same number of images and categories but introduces a new class and increases the total number of annotated objects to approximately 403,000, including objects with dimensions smaller than 10 pixels. The latest version, DOTA-v2.0, is the most extensive, consisting of around 11,000 images and approximately 1.79 million annotated objects distributed across 18 categories.

III. Comparison Proposal

The primary objective of this research is to evaluate and compare the performance of different YOLO models trained on the same dataset, with the aim of identifying the model that offers the best trade-off between accuracy and computational efficiency for deployment on low-power IoT devices in the context of TinyML applications.

To achieve this objective, the first phase of the research involves evaluating the performance of the following YOLO models: YOLOv5n, YOLOv5m, YOLOv8n, YOLOv8m, YOLOv11n, YOLOv11m, YOLOv12n, and YOLOv12m, specifically in the task of small object detection in images. The second phase focuses on refining the model with the best performance from the initial evaluation. Finally, the third phase consists of deploying the optimized model on an IoT device within a TinyML framework.

Unlike most YOLO research that employs the COCO (Common Objects in Context) dataset [9], this study utilizes the DOTA (Dataset for Object Detection in Aerial Images) dataset [6]. The COCO dataset focuses on detecting and segmenting everyday objects such as people, vehicles, animals, and household items in natural images. In contrast, DOTA is specialized for high-resolution aerial imagery acquired from satellites, drones, or airplanes, with image sizes ranging from thousands to tens of thousands of pixels.

To evaluate the models, the following metrics will be employed: mean Average Precision at IoU threshold 0.50 (mAP @ 0.50), mean Average Precision averaged over IoU thresholds from 0.50 to 0.95 (mAP @ 0.50:0.95), Precision, Recall, Latency, Loss, and Model Size. The mAP @ 0.50 metric considers a prediction correct if the Intersection over Union (IoU) between the predicted and ground truth bounding boxes is at least 50%. The mAP @ 0.50:0.95 metric provides a more comprehensive evaluation by averaging precision across multiple IoU thresholds ranging from 0.50 to 0.95. Precision measures the proportion of true positive detections among all positive predictions, whereas Recall quantifies the proportion of actual objects correctly detected by the model. Latency, measured on the CPU, represents the average inference time per image, indicating the model's processing speed. The final training and validation loss values indicate the degree of fit of the model to the training and validation datasets, respectively. Finally, Model Size, measured in megabytes, is a critical factor for deployment on memory-constrained devices.

IV. Comparison Methodology

To ensure a fair comparison among the selected YOLO models, all will be trained for an identical number of epochs using images resized to the same dimensions, with consistent batch sizes, optimizers, and learning rates. Upon completion of training, a best.pt checkpoint file will be generated for each model, from which performance metrics will be extracted.

All models will be trained using the DOTA v1 dataset, which contains 15 object categories, approximately 2,800 images, and around 188,000 annotated instances. The dataset is divided into training, validation, and testing subsets with proportions of 50%, 16%, and 33%, respectively. Moreover, all training will be conducted within the same runtime environment to maintain consistent computational conditions.

V. Label Conversion

YOLO traditionally employs Horizontal Bounding Boxes (HBB) for object detection, which are rectangular boxes aligned with the image's X and Y axes, as illustrated in Figure 1.

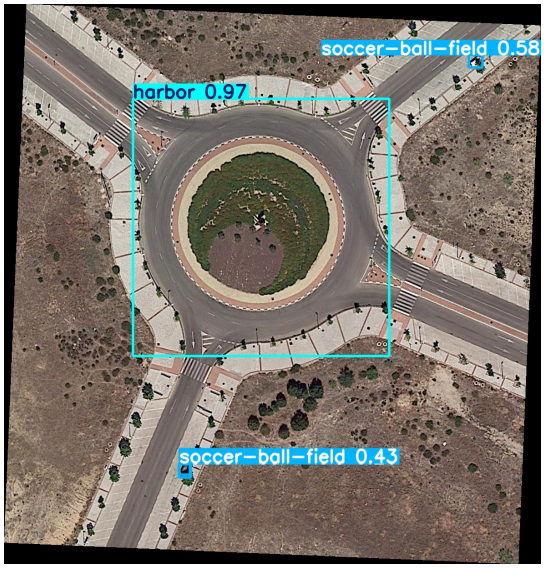

Fig. 1. Example of an image with a horizontal bounding box.

This approach is straightforward and widely adopted in conventional datasets such as COCO. In contrast, the DOTA dataset employs Oriented Bounding Boxes (OBB), which are rotated rectangles aligned with the orientation of the objects, providing a more accurate representation for objects that are not horizontally aligned, as illustrated in the Figure 2.

Fig. 2. Example of an image with a oriented bounding box.

For this study, all dataset annotations were converted from Oriented Bounding Boxes (OBB) to Horizontal Bounding Boxes (HBB) using an algorithm that follows the steps outlined below:

1) Defines the directories for the original annotations, corresponding images, and the output folder for the new label files.
2) Specifies the object categories to be retained during the conversion.
3) For each annotation file:
   a) Reads the object annotations present in the file.
   b) Filters out invalid entries and those not belonging to the selected categories.
   c) Converts the oriented bounding boxes (OBB) into horizontal bounding boxes (HBB) aligned with the image axes.
   d) Transforms the horizontal boxes into YOLO format, with normalized center coordinates and dimensions relative to the image size.
   e) Saves the converted annotations into new label files compatible with YOLO training.
4) Repeats the process for all annotation files and their corresponding images, skipping any missing images.

VI. Image Conversion

Training YOLO models requires specifying a standard input image size or accepting the default automatic size, typically 640 pixels. However, resizing high-resolution images—originally averaging approximately 2000 pixels—to such smaller dimensions results in a significant loss of object detail, which complicates detection during training, validation, and testing.

One approach to mitigate this problem is to increase the input image size to values such as 1024 or 1280 pixels. Nevertheless, larger images demand greater computational resources, necessitating smaller batch sizes to avoid memory limitations during training.

To address these challenges, this study proposes an alternative approach that complements training with the original dataset. A preprocessed version of the dataset is generated by reducing the image dimensions while preserving sufficient object resolution, thereby enabling more efficient training without compromising detection accuracy.

The image preprocessing algorithm employed in this study proceeds according to the following steps:

1) Defines the directories for the original images, original annotation files, and the output folders for the sliced images and their corresponding labels.
2) Specifies the tile size used to split large images into smaller patches.
3) For each dataset subset (training and validation):
   a) Lists all images in the subset directory.
   b) For each image:
      i) Reads the image and its corresponding YOLO-format annotation file.
      ii) If the image dimensions are smaller than or equal to the tile size, copies the image and annotation directly to the output directory without slicing.
      iii) Otherwise, calculates how many horizontal and vertical tiles are needed to cover the entire image.
      iv) Iterates over each tile region, cropping the image and adjusting the bounding boxes:
         A) Converts YOLO bounding boxes to pixel coordinates.
         B) Computes the intersection between each bounding box and the current tile.
         C) Discards boxes with no intersection or that become too small after cropping.
         D) Converts the cropped bounding boxes back to YOLO format relative to the tile size.
         E) Saves the new tile image and its adjusted annotations.
4) Repeats the slicing and label adjustment process for all images in the specified subsets.

After defining and validating the image slicing algorithm, the final step before processing is to select an appropriate tile size. Determining this size requires an analysis of the distribution of image dimensions within the dataset. To facilitate this, a dedicated algorithm was developed.

The objective of this algorithm is to iterate over all images in the training and validation folders of the dataset and collect statistics regarding their dimensions (width and height). For each image, the algorithm records its dimensions and classifies it into one of four categories: (i) images with both width and height less than or equal to 1024 pixels; (ii) images with width less than or equal to 1024 pixels; (iii) images with height less than or equal to 1024 pixels; and (iv) images with both width and height greater than 1024 pixels. Based on this classification, the algorithm computes statistics including the minimum, mean, and maximum widths and heights; the total number of images per category; the mean width of images with width less than or equal to 1024 pixels; the mean height of images with height less than or equal to 1024 pixels; and the mean width and height of images with both dimensions exceeding 1024 pixels.

A threshold value of 1024 pixels was adopted as a reference, based on its superior performance observed during preliminary model training. Nevertheless, this threshold can be adjusted according to the specific requirements of the application or the characteristics of the dataset.

The exploratory analysis of the training dataset revealed significant variation in image dimensions. A total of 1,411 images were analyzed, with widths ranging from 387 to 12,029 pixels and an average width of 2,304.67 pixels. Heights varied from 278 to 8,115 pixels, with an average height of 2,181.29 pixels. It was observed that 201 images have both width and height less than or equal to 1,024 pixels, representing a notable portion of the dataset, while 990 images have both dimensions greater than 1,024 pixels. Additionally, 116 images have a width less than or equal to 1,024 pixels, and 104 images have a height less than or equal to 1,024 pixels. Regarding specific averages, images with width less than or equal to 1,024 pixels present an average width of 766.98 pixels, whereas images with height less than or equal to 1,024 pixels present an average height of 783.57 pixels. Among images with both width and height greater than 1,024 pixels, the average width and height are 2,848.18 and 2,710.67 pixels, respectively.

The exploratory analysis of the validation dataset also revealed significant variation in image dimensions. A total of 458 images were analyzed, with widths ranging from 353 to 13,383 pixels and an average width of 2398.01 pixels. The heights ranged from 511 to 6759 pixels, with an average of 2210.98 pixels. It was observed that 58 images have both width and height less than or equal to 1024 pixels, while 318 images have both dimensions greater than 1024 pixels. Additionally, 48 images have a width less than or equal to 1024 pixels, while 34 images have a height less than or equal to 1024 pixels. The specific averages indicate that images with width less than or equal to 1024 pixels present an average width of 765.44 pixels, while those with height less than or equal to 1024 pixels present an average height of 799.49 pixels. Among the images with both width and height greater than 1024 pixels, the average width and height are 3045.28 and 2731.42 pixels, respectively.

The exploratory analysis of the training and validation datasets demonstrated consistency, revealing substantial variation in image dimensions across both sets. This variability highlights the necessity of preprocessing to enhance model training performance. A tile size of 1024 pixels was adopted as the standard, considering that the average dimensions of the larger images approximate 2000 pixels. However, due to the diversity in image sizes, the slicing algorithm partitions images into 1024-pixel tiles only when their dimensions exceed this threshold. For images smaller than 1024 pixels, no slicing is applied; instead, these cases are handled directly by the training algorithm, as expected in a training procedure without prior preprocessing.

VII. Dataset Verification

After converting the annotations of the DOTA v1 dataset from oriented bounding boxes (OBB) to horizontal bounding boxes (HBB), a sample-based verification of the images was conducted to assess the correctness of the conversion. For this purpose, the Roboflow online platform was utilized, which provides tools for dataset visualization and preliminary computer vision model testing. Figure 3 illustrates an example of the verification process performed using Roboflow.

Fig. 3. Example of image verification using Roboflow.

This tool allows for the verification of the correct positioning of the bounding boxes as well as the inspection of the label assigned to each object.

VIII. Test Environment

Two types of tests were conducted in this study. Initially, all models were trained in a local environment using Anaconda, a platform that facilitates the development of Python projects through package management and virtual environments. The computer used for training is equipped with an 11th Generation Intel Core i7-11800H processor and a GeForce GTX graphics card with 4 GB of dedicated video memory.

Subsequently, the same tests were conducted on Google Colab Pro, utilizing runtimes with T4 GPUs. The T4 runtime employs an NVIDIA T4 GPU with 16 GB of dedicated memory, suitable for moderate workloads. Additionally, the Colab Pro environment offers extended memory capacity (High RAM), which supports the efficient processing of larger datasets and contributes to more stable training sessions.

The tests conducted in the local environment revealed significant limitations related to image size and batch size, particularly when compared to the tests performed on Google Colab Pro. In some cases, the execution time in the local environment was more than twice as long as that observed in Colab Pro. Further details regarding the test results are presented in the following section.

IX. Comparison Results

In the initial test scenario, the models YOLOv5n, YOLOv5m, YOLOv8n, YOLOv8m, YOLOv11n, YOLOv11m, YOLOv12n, and YOLOv12m were evaluated in a local environment. Although the machine was equipped with a 4 GB GPU, it was insufficient to run the models, and all training was therefore performed exclusively on the CPU, with an average training time of approximately one day per model. The models were trained for 50 epochs using images resized to 640 pixels, with a batch size of 1, due to hardware limitations that prevented the use of larger batches. The results of this initial test scenario are summarized in Table I.

TABLE I
PERFORMANCE COMPARISON OF YOLO MODELS TRAINED WITH CPU.

Model	mAP @ 0.50	mAP @ 0.50:0.95	Precision	Recall	Latency (CPU)	Loss Final (train)	Loss Final (val)	Size
yolov5n	0.394	0.207	0.651	0.378	90 ms	0.244	0.246	13.6 MB
yolov5m	0.496	0.319	0.723	0.455	268 ms	2.488	2.837	48.1 MB
yolov8n	0.334	0.199	0.629	0.314	37 ms	3.414	3.747	6.2 MB
yolov8m	0.437	0.270	0.612	0.411	273 ms	3.290	3.791	52 MB
yolov11n	0.392	0.240	0.704	0.359	55 ms	3.139	3.511	5.4 MB
yolov11m	0.445	0.276	0.637	0.412	285 ms	3.321	3.672	40.5 MB
yolov12n	0.176	0.091	0.564	0.183	64 ms	5.243	5.454	5.5 MB
yolov12m	0.441	0.272	0.645	0.401	332 ms	3.403	3.784	40.7 MB

Table I presents the results of the first test scenario conducted in a local environment, highlighting the performance differences among the YOLO model versions and sizes. The model that achieved the best overall performance was YOLOv5m, with the highest mAP@0.50 (0.496) and mAP@0.50:0.95 (0.319), as well as the highest precision (0.723) and recall (0.455), although it is one of the largest models (48.1 MB). YOLOv12m and YOLOv11m exhibited similar performance, with mAP@0.50 values of 0.441 and 0.445, respectively. The smaller n-series models, while more compact, demonstrated inferior performance, with YOLOv12n standing out negatively for obtaining the lowest mAP@0.50 (0.176) and recall (0.183), despite being one of the smallest models (5.23 MB). These results indicate that, under the conditions of this test scenario, the medium-sized models delivered superior performance, albeit at the cost of increased model size and higher latency.

An analysis of the results presented in Table I indicates that the models trained for only 50 epochs exhibited limited performance. Furthermore, due to hardware limitations in the local environment, it was not possible to adjust the batch size, which was restricted to a single image. Therefore, a new round of experiments was conducted using Google Colab. In this new setup, a runtime environment with GPU support was employed, allowing the models to be trained for 100 epochs with a batch size of 16 images and a resolution of 640 pixels. The results obtained from this configuration are summarized in Table II.

The results presented in Table II indicate that the m (medium) models achieved the best overall performance in terms of mAP and precision metrics. The YOLOv11m model demonstrated the highest performance, achieving an mAP@0.50 of 0.543 and an mAP@0.50:0.95 of 0.358, along with a favorable balance between precision (0.724) and recall (0.506), and relatively low final training and validation losses. Other models, such as YOLOv5m, YOLOv8m, and YOLOv12m, also delivered competitive results, all with mAP@0.50 values above 0.51 and mAP@0.50:0.95 around 0.34. In contrast, the n-series models, while more lightweight, exhibited lower performance, with mAP@0.50 ranging from 0.402 to 0.428 and mAP@0.50:0.95 between 0.248 and 0.268. It is also noteworthy that the latest model versions, YOLOv11 and YOLOv12, showed consistent improvements over previous versions, particularly in their m variants, which maintained high performance while preserving relatively moderate model sizes.

Following the initial two rounds of testing, a new experiment was conducted on Google Colab using only the models that exhibited the best performance, specifically all models from the m family. In this experiment, the models were trained for 100 epochs with images resized to 1024 pixels and a learning rate of 0.001. The default YOLO batch size of 16 images was unfeasible for this image resolution due to memory constraints; consequently, a reduced batch size of 8 images was employed. Additionally, the AdamW optimizer was adopted in this stage, replacing the default YOLO optimizer (SGD) used previously. AdamW was selected for its improved convergence and training stability in challenging scenarios, such as detecting very small objects, where precise weight fine-tuning is critical for model accuracy.

Table III presents the results of the third test scenario, in which only models from the m family were evaluated. These models were trained for 100 epochs with images resized to 1024 pixels, using the AdamW optimizer. The model achieving the best overall performance was YOLOv11m, attaining the highest mAP@0.50 (0.608), mAP@0.50:0.95 (0.415), and recall (0.564), while also exhibiting the smallest model size among the m variants (40.5 MB). The YOLOv8m and YOLOv5m models also demonstrated strong performance, with very similar results in terms of mAP and precision. Specifically, YOLOv8m achieved an mAP@0.50 of 0.596 and recall of 0.553, whereas YOLOv5m attained an mAP@0.50 of 0.593 and recall of 0.551. These results indicate that, under the conditions of this experiment, YOLOv11m offers the best trade-off between detection performance and model size, confirming its suitability for applications requiring both accuracy and efficiency.

An analysis of the results presented in Table III confirms that the best-performing model was YOLOv11m. Consequently, a final experiment was conducted using only this model, with the same parameters as in the previous test. The main difference in this experiment was the use of an alternative version of the DOTA dataset, in which the images were sliced according to the algorithm described in Section VI.

The YOLOv11m model was trained once using the preprocessed dataset with image slicing. Training was performed for 100 epochs with images resized to 1024 pixels, a learning rate of 0.001, and the AdamW optimizer. After training, the model was evaluated in two distinct scenarios. In the first evaluation (YOLOv11m OD), the original dataset (without image slicing) was used, yielding a mAP@0.50 of 0.543 and mAP@0.50:0.95 of 0.372, with precision and recall values of

TABLE II
PERFORMANCE COMPARISON OF YOLO MODELS TRAINED WITH GPU (GOOGLE COLAB).

Model	mAP @ 0.50	mAP @ 0.50:0.95	Precision	Recall	Latency (CPU)	Loss Final (train)	Loss Final (val)	Size
yolov5n	0.402	0.248	0.682	0.374	87 ms	3.024	3.395	5 MB
yolov5m	0.516	0.340	0.762	0.465	265 ms	2.230	2.870	48.1 MB
yolov8n	0.416	0.257	0.665	0.385	48 ms	2.887	3.333	6.2 MB
yolov8m	0.517	0.339	0.762	0.467	270 ms	2.146	2.872	52 MB
yolov11n	0.424	0.264	0.711	0.387	54 ms	2.865	3.294	5.5 MB
yolov11m	0.543	0.358	0.724	0.506	290 ms	2.161	2.786	40.5 MB
yolov12n	0.428	0.268	0.661	0.400	63 ms	2.794	3.243	5.5 MB
yolov12m	0.531	0.350	0.732	0.487	338 ms	2.193	2.790	40.8 MB

TABLE III
PERFORMANCE COMPARISON OF YOLO MODELS TRAINED WITH GPU (GOOGLE COLAB) — OPTIMIZED HYPERPARAMETERS.

Model	mAP @ 0.50	mAP @ 0.50:0.95	Precision	Recall	Latency (CPU)	Loss Final (train)	Loss Final (val)	Size
yolov5m	0.593	0.397	0.766	0.551	613 ms	2.238	2.760	50.5 MB
yolov8m	0.596	0.403	0.774	0.553	736 ms	2.134	2.755	52.1 MB
yolov11m	0.608	0.415	0.772	0.564	813 ms	2.204	2.704	40.5 MB

0.721 and 0.513, respectively. CPU latency was measured at 807 ms, with final training and validation losses of 2.322 and 3.087, respectively, and a final model size of 40.5 MB. In the second evaluation (YOLOv11m SD), using the alternative dataset (with image slicing), the same model achieved a mAP@0.50 of 0.738 and mAP@0.50:0.95 of 0.504, with precision and recall values of 0.768 and 0.700, respectively. CPU latency was 818 ms, with identical final training and validation losses and model size as in the first evaluation. Table IV summarizes these results. The discrepancy between the two evaluations indicates that applying image slicing solely to the training set is insufficient; consistent pre-processing must also be applied to the validation and test sets to ensure reliable model performance.

Among all YOLO models trained in this study, the YOLOv11m model trained using the alternative version of the DOTA dataset (with preprocessed images) demonstrated the best performance and was selected for use in the subsequent stages of this research.

X. TINY-ML

The deployment of the YOLO11 model on Raspberry Pi devices demonstrates the potential to execute advanced object detection algorithms on low-power, cost-effective embedded platforms. This capability is particularly relevant for IoT and TinyML applications, where the combination of high accuracy and computational efficiency is essential to enable intelligent systems in resource-constrained environments.

From a technical perspective, the setup involves installing a compatible Python environment and the Ultralytics package. The trained model is transferred to the device, enabling real-time inference on images captured by cameras connected to the Raspberry Pi.

The YOLO11 algorithm is based on a convolutional neural network architecture that performs object detection in a single stage by simultaneously localizing and classifying objects. Exporting the model to optimized formats, such as NCNN, facilitates efficient execution on embedded devices, reducing latency and computational resource consumption.

The NCNN format is a lightweight, high-performance neural network inference framework optimized for mobile and embedded devices. NCNN is designed to efficiently execute deep learning models with low latency and minimal resource consumption, making it particularly suitable for platforms with limited computational power, such as Raspberry Pi.

During inference on the Raspberry Pi, the model demonstrated the ability to accurately detect objects. However, some false positives were also observed, indicating occasional misclassifications that should be addressed in future improvements.

XI. CONCLUSION

This study conducted a detailed comparative analysis of various YOLO algorithm versions applied to the detection of small objects in aerial imagery using the DOTA dataset. By standardizing the test environment and implementing preprocessing techniques such as label conversion and image slicing, a significant improvement in model performance was achieved, particularly in the detection of low-scale targets.

Among the evaluated models, YOLOv11m emerged as the most effective, achieving superior results in terms of precision, recall, and mAP, while maintaining computational efficiency. Its successful deployment on a Raspberry Pi device further validated its suitability for IoT and TinyML scenarios, highlighting the feasibility of integrating advanced object detection algorithms into resource-constrained embedded systems.

The findings underscore the importance of fine-tuning and data engineering in training models tailored for specific applications. For future work, the exploration of additional hyperparameter configurations and the application of image processing techniques to the test datasets are proposed, aiming to enhance model performance and robustness in real-world inference conditions.

TABLE IV
PERFORMANCE COMPARISON OF THE YOLOV11M MODEL TRAINED WITH THE PRE-PROCESSED DATASET WITH IMAGE SLICING.

Model	mAP @ 0.50	mAP @ 0.50:0.95	Precision	Recall	Latency (CPU)	Loss Final (train)	Loss Final (val)	Size
yolov11m (OD)	0.543	0.372	0.721	0.513	807 ms	2.322	3.087	40.5 MB
yolov11m (SD)	0.738	0.504	0.768	0.700	818 ms	2.322	3.087	40.5 MB

REFERENCES

[1] G. Jocher, "Ultralytics yolov5," 2020. [Online]. Available: https://github.com/ultralytics/yolov5

[2] G. Jocher, A. Chaurasia, and J. Qiu, "Ultralytics yolov8," 2023. [Online]. Available: https://github.com/ultralytics/ultralytics

[3] G. Jocher and J. Qiu, "Ultralytics yolo11," 2024. [Online]. Available: https://github.com/ultralytics/ultralytics

[4] Y. Tian, Q. Ye, and D. Doermann, "Yolov12: Attention-centric real-time object detectors," *arXiv preprint arXiv:2502.12524*, 2025.

[5] ——, "Yolov12: Attention-centric real-time object detectors," 2025. [Online]. Available: https://github.com/sunsmarterjie/yolov12

[6] G.-S. Xia, X. Bai, J. Ding, Z. Zhu, S. Belongie, J. Luo, M. Datcu, M. Pelillo, and L. Zhang, "Dota: A large-scale dataset for object detection in aerial images," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[7] J. Ding, N. Xue, G.-S. Xia, X. Bai, W. Yang, M. Yang, S. Belongie, J. Luo, M. Datcu, M. Pelillo, and L. Zhang, "Object detection in aerial images: A large-scale benchmark and challenges," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, pp. 1–1, 2021.

[8] Y. L. G.-S. X. Q. L. Jian Ding, Nan Xue, "Learning roi transformer for detecting oriented objects in aerial images," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[9] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. Zitnick, "Microsoft coco: Common objects in context," vol. 8693, 04 2014.

Jonas Vilasbôas Moreira holds a master's degree in Telecommunications Engineering and specializations in Network Engineering and Telecommunications Systems, as well as in SOA Cloud Computing and Connectivity, all from the National Institute of Telecommunications (Instituto Nacional de Telecomunicações - Inatel), Brazil, completed in 2023, 2018, and 2015, respectively. He has worked as a system specialist since 2012. From 2019 to 2023, he was a collaborator at the Information and Communications Technologies (ICT) Laboratory at Inatel. His research interests focus on the exploration of artificial intelligence for object detection.