National College of Ireland

Evaluating Object Detection Models for Small Object Recognition

MSc Research Project
Artificial Intelligence

Gonzalo Bugueño
Student ID: x23408731

School of Computing
National College of Ireland

Supervisor: Kislay Raj

National College of Ireland
Project Submission Sheet
School of Computing

Student Name:	Gonzalo Bugueño
Student ID:	x23408731
Programme:	Artificial Intelligence
Year:	2025
Module:	MSc Research Project
Supervisor:	Kislay Raj
Submission Due Date:	11/08/2025
Project Title:	Evaluating Object Detection Models for Small Object Recognition
Word Count:	3130
Page Count:	19

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	
Date:	15th September 2025

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	☐
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	☐
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	☐

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Evaluating Object Detection Models for Small Object Recognition

Gonzalo Bugueño
x23408731

Abstract

Performance in object detection is what makes a detector useful. This study aims to explore different techniques to improve object detection performance for challenging tasks, including dataset composition, augmentation techniques and synthetic data generation. For this purpose, a multitude of You Only Look Once (YOLO) models were trained with different datasets (ranging from annotated photographs to fully synthetic, augmented frames) and hyperparameters. The study focuses on the specific challenge of detecting coins in different settings (indoors, street) - a task inspired by the visual similarity between coins and typical urban litter like chewing gum. This scenario amplifies the challenge due to subtle textural differences, ambiguous object boundaries, and warping shape. The study demonstrates that it is possible to build a performant, small object detector with very limited resources by leveraging Blender's scripting API, applying augmentations, and choosing the right set of parameters which constitute the model. The final results highlight the importance of dataset composition and model architecture.

1 Introduction

Object detection remains a critical task in computer vision with applications in domains such as autonomous navigation and urban scene understanding. Architectures in the YOLO lineage, particularly YOLOv8 and the experimental YOLOv11 by Ultralytics, excel in real-time detection, striking an attractive balance between accuracy and inference speed Terven and Cordova-Esparza (2023); Jocher and Qiu (n.d.).

Synthetic datasets are artificially generated data that mimics the characteristics of real-world data but is created using simulation software. In contrast to traditional dataset creation, synthetic dataset creation holds the potential to generate various labelled samples (images, videos, depth data) only while incurring in an initial investment - the man-hours put into 3D modelling and scripting, - whereas real-world dataset creation is a continuous effort which requires continuous labour.

This study makes extensive use of Blender and its scripting abilities for dataset generation. Blender is a free, open-source 3D creation suite for modeling, animation, rendering, simulation, and video editing. It supports the entire 3D pipeline and is widely used by artists and professionals. Via scripting it is possible to control different aspects of the scene: lighting, lens distortion, object positioning, among others.

This study is contextualised within the broader domain of synthetic-to-real domain adaptation-recognising that models trained strictly on synthetic data often fail to gen-

eralise well in real-world scenarios Tremblay et al. (2018). Instead, combining synthetic with a limited set of real-world samples can significantly reduce domain gaps.

1.1 Motivation and Background

Collecting real-world training data for niche applications is impractical and expensive, since it is a continuous manual labour. Object size, non-optimal camera settings, photographic artifacts, and vendor-specific camera enhancing software can decide whether an object of interest is detected or not.

If successful, this approach could provide an easy base for future work with similar constraints. Also, it could help to reduce environmental impact by utilising e-waste smartphones with sub-par cameras deploying a robust model and dataset tailored towards to its hardware.

1.2 Research Questions

The aim of this study is to answer the following questions:

- How do different object detection models compare in small object recognition?
- Can object detection models generalise well with synthetic data?

1.3 Contributions

- A Blender-based synthetic dataset pipeline for generating an annotated dataset. (Fig. 1)
- Comparison YOLOv8 and YOLOv11 in a unified Ultralytics-driven framework across different training domain compositions.
- Analysis on how synthetic-only, real-only, and mixed datasets influence on detection performance and robustness to domain shift.

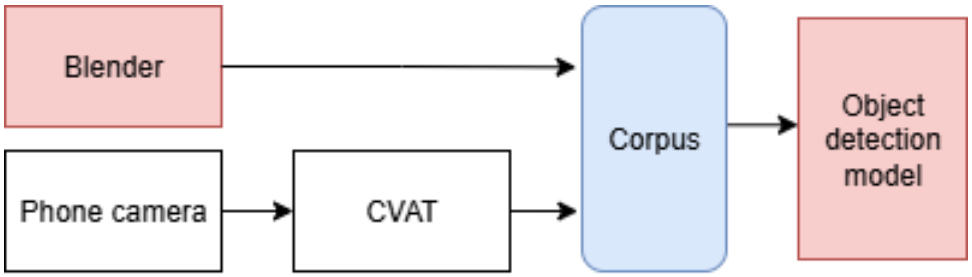


Figure 1: Experimental high-level pipeline: **Red boxes** indicate modules with tunable hyperparameters. The synthetic data is generated via Blender and annotated through a phone-camera + CVAT pipeline; both contribute to a unified corpus used to fine-tune the object detection model.

2 Related Work

2.1 Object Detection Architectures

Modern object detectors divide into one-stage and two-stage pipelines: for real-time applications (e.g., autonomous vehicles, video analysis) one-stage detectors are often preferred due to their inference speed. For tasks where accuracy is critical (e.g., medical imaging, security), two-stage may be preferred. While studies have demonstrated the superiority in accuracy of two-stage object detection models Carranza-García et al. (2020), improved and modern single-stage detectors are closing the gap and excel in speed Diwan et al. (2023). Among one-stage detectors, the YOLO family from versions v1 through v11 have matured significantly - almost surpassing single stage object detectors in accuracy while remaining easy on resource utilisation.

YOLOv1 introduced a single-pass grid-based predictor for bounding boxes Redmon (2018). YOLOv2 added anchor boxes, while YOLOv3 advanced further with a deeper Darknet-53 backbone and multi-scale outputs Terven and Cordova-Esparza (2023). YOLOv5 adopted CSPDarknet and mosaic augmentation Hussain (2024). The most recent versions (YOLOv8 and YOLOv11) embrace an anchor-free design with decoupled heads and modern backbone/neck architectures Jegham et al. (2025). In benchmarks on COCO at IoU=0.5, YOLOv8 achieves approximately 53.9% mAP at high speed, surpassing YOLOv5's 50.7% under similar settings Hussain (2024). YOLOv11 (medium variant) reportedly attains higher mAP with 22 % fewer parameters than YOLOv8m Jegham et al. (2025). Independent evaluations confirm that YOLOv11 offers the best accuracy-speed tradeoff; earlier versions lagged behind in **small-object recall**, while YOLOv11 significantly improves on that front while maintaining efficiency Jegham et al. (2025).

Another architecture, RetinaNet, is a one-stage detector that uses a ResNet+FPN backbone and twin heads for classification and regression, along with the focal loss function to address class imbalance. In domains like pill recognition, RetinaNet reaches slightly higher precision (mAP = 82.9 %) compared to YOLOv3 (80.7 %), albeit with much lower throughput (one third the FPS) Tan et al. (2021). Generally, RetinaNet excels in precision and dense or cluttered scenes, while YOLO emphasises speed.

2.2 Transfer learning

Modern object detectors are often pre-trained on large datasets and then fine-tuned on the target data. Using pre-trained weights from a large benchmark (COCO, ImageNet, etc.) and then fine-tuning on the custom dataset can greatly reduce training time and data requirements. Studies show how fine-tuning pre-trained weights may improve - or at the very least, stay on par with - a blank neural network Tajbakhsh et al. (2016). In practice one would choose a model (e.g. YOLOv8 or RetinaNet) and adjust hyperparameters (learning rate, number of layers to freeze, augmentation) to improve performance on the new data.

2.3 Synthetic Data Generation

Tools enabling synthetic dataset creation are central to modern object detection research. Unity's Perception package, for instance, allows scripted generation of images with randomised object placement, textures, lighting, and backgrounds, and automatically gen-

erates labels [1]. Similarly, BlenderLine provides an end-to-end pipeline using Blender for rendering annotated images from CAD models van den Hoven (2023). These platforms often leverage domain randomisation, introducing variability in visuals so real-world scenes become just another variation, thereby narrowing the render to real gap Gavriel et al. (2024).

Simulation environments, 3D or game engines make for trivial automated annotation.

2.4 Data Annotation

Annotation tools such as CVAT offer open-source interfaces for manual or semi-automated bounding-box labeling [2]. Roboflow Annotate is a cloud-based platform incorporating AI-assisted autolabeling to accelerate annotation workflows [3]. Such tools allow to export to common formats (e.g. COCO, YOLO)

2.5 Domain Adaptation

Bridging the synthetic-to-real domain gap can be tackled via several strategies. Domain randomisation works by varying rendering parameters such as textures, lighting, backgrounds, and noise works on the premise that real-world scenes are just another variation of the scene. Models trained on such data show robust performance Gavriel et al. (2024); Tobin et al. (2017); Zhu et al. (2025). Other mechanisms include feature or pixel-level alignment via generative adversarial networks, and self-training: training a model on synthetic data, generating pseudo-labels for real data, and re-training on those labels.

2.6 Ultralytics YOLO

Ultralytics offers a streamlined interface for training and evaluating YOLO models versions v5 to v11[4]. With a simple CLI and Python API users can launch training with minimal boilerplate. [5]. The package handles dataset loading, augmentation, model instantiation, checkpointing, and others.

Hyperparameters can be tweaked via YAML files, Python API, and CLI flags, simplifying experimentation. However, this convenience comes at the cost of lower low-level configurability.

The YOLO family of object detectors offers a balance of accuracy, speed, and deployment flexibility, making it one of the most widely adopted models for real-time object detection tasks. Its single-stage, end-to-end architecture enables direct prediction of object locations and classes in a single forward pass, giving significant latency advantages over two-stage methods while maintaining competitive precision and recall. Over successive iterations, the YOLO models have incorporated innovations such as multi-scale prediction, advanced augmentation strategies, anchor-free detection, and efficient backbone-neck implementations, all of which improve robustness to small objects and complex backgrounds. The family is also notable for its ease of use: modern implementations, particularly from Ultralytics, provide high-level APIs, automatic preprocessing,

[1] Using Unity Perception to train an object detection model with synthetically generated images: `https://blog.roboflow.com/unity-perception-synthetic-dataset-tutorial/`

[2] What is CVAT (Computer Vision Annotation Tool)?: `https://github.com/opencv/cvat`

[3] Roboflow Annotate: Label Images Faster Than Ever `https://roboflow.com/annotate`

[4] Models Supported by Ultralytics: `https://docs.ultralytics.com/models/`

[5] Ultralytics YOLO: Train Mode Documentation `https://docs.ultralytics.com/modes/train/`

pretrained weights, and integrated training and evaluation pipelines, allowing researchers to use state-of-the-art object detection models with minimal configuration.

2.7 Synthetic vs. Real Data

Advantages of synthetic data are their cost-effectiveness, precise labelling, and the ability to model edge cases. Studies have shown their effectiveness to improve the generalisation of different models (2D, and even 3D tasks) Nikolenko (2019).

In the study made by Özeren and Bhowmick (2025), the team mixes synthetic with real data of pedestrian and car objects for autonomous driving models which span 2D and 3D recognition, enhancing model robustness.

3 Methodology

This study evaluates the performance of YOLOv8 and YOLOv11 object detection models for the task of detecting a single class (coin) in different environments -carpets, sidewalks, etc.,- including motion blur and noisy backgrounds. The evaluation methodology was designed to systematically explore the effect of training data composition, model architecture, and image resolution on detection performance, with a particular focus on recall, as false positives incur high costs in the target experiment "moving coin detector".

3.1 Data Collection

3.1.1 Real-World Datasets

While the experiment concentrates on synthetic data, a few real-life images and videos under different conditions were taken for dataset composition and evaluation purposes. These photographs and frame videos are annotated using CVAT. Each sub-dataset shares a common context: same floor, same physical environment. Contrary to the synthetic renders, bounding box annotations in the real-world datasets were categorised into four qualitative types:

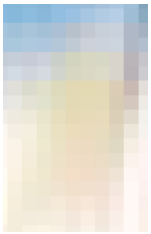
- **Clear**: unambiguous coin, defined shape, no motion blur
- **Elongated**: distorted shape due to lens distortion, still defined and no significant motion blur
- **Distinguishable**: blurred. original colour still present and still visually separable from background
- **Inferred**: object presence implied by context, otherwise unrecognizable

Figure 2 shows the appearance of subjects with different a "visibility" attribute. These categories were used in exploratory analysis to better interpret model performance across varying visibilities.

(a) Clear subject: colour and shape are defined.

(b) Elongated subject: colour, shape is defined, albeit elongated.

(c) Distinguishable subject: Blurred heavily. Colour and shape ambiguous.



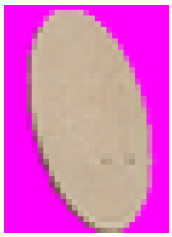
(d) Inferred via context: Shape and colour are mostly lost.

Figure 2: Examples of visibility types

3.1.2 Synthetic Datasets

Leveraging Blender's scripting support, a Python script was created to render a coin object under different circumstances, for making the neural network gather more information about the object. Figure 3 demonstrates the render of a coin object as seen by a boundary box.

Synthetic data is automatically annotated via Blender's scene metadata export.



(a) Close object

(b) Far away object

(c) Object at a medium distance

Figure 3: Samples of synthetic corpus at different positions. Note: Pink is meant to denote transparency. Note: these samples have had their aspect ratio changed from 16:9 to 1:1, which may make them appear squished.

Table 1 shows the parameters in the parameter grid for the synthetic dataset generation. The Cartesian product of this grid is unravelled at runtime by the Blender script runtime and rendered.

3.1.3 Cold Synthetic Dataset Augmentation

Referring to modifications done to the image and immediately saved to the disk, some augmentations were devised for the renders described in 3.1.2, including: directional blurring, background shifting and zooming, and background replacement.

The intention behind augmentation of the synthetic dataset is to test the gap between simulation and real.

3.2 Model Tuning

Different pre-trained models were tested against different hyperparameters to find an optimum corpus and set of hyperparameters. By exploring the parameter grid described in 6, it was possible to train a multitude of models with different settings and compare their performance.

3.3 Dataset Composition

Different composite datasets were made with the intention of evaluating different distributions, validated with real data:

1. All synthetic
2. All real
3. Non-optimal all real
4. Considerably synthetic

4 Design Specification

4.1 Synthetic Pipeline

As shown by Figure 4, the pipeline for the synthetic data generation goes as follows:

1. Blender produces an annotated dataset - CVAT 1.1 due to its ease of use and compatibility with the real-world annotated dataset.
2. Different models require different formats of dataset, so converters are used to adapt or convert the original dataset into a compatible dataset for the model.
3. If desired, augmentations can be applied to the dataset, cold[6] and in-memory if supported by the kind of dataset needed by the model.
4. The model gets to be trained on the desired, augmented dataset.

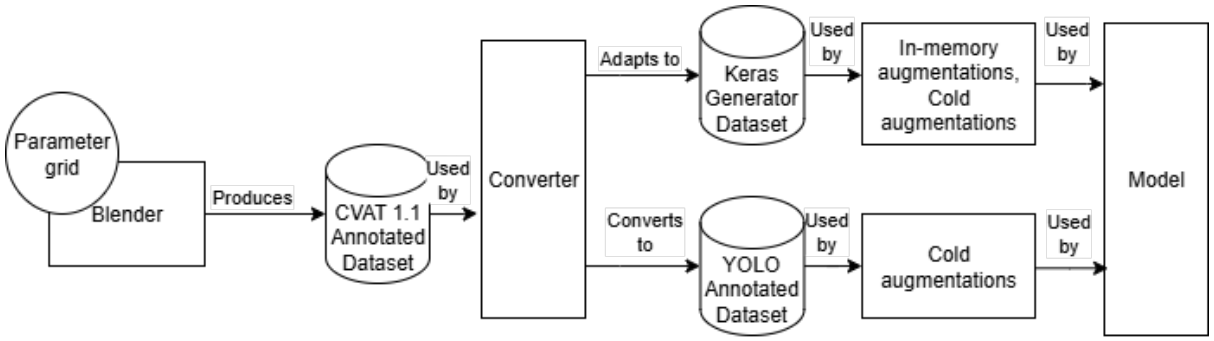


Figure 4: Start-to-finish high-level view of the synthetic data generation

[6]This means the result will be stored as a file

4.2 Composite Datasets

An example of such dataset composition is depicted by the diagram shown in Figure 5. This functionality implemented in Python allows for arbitrary corpus composition.

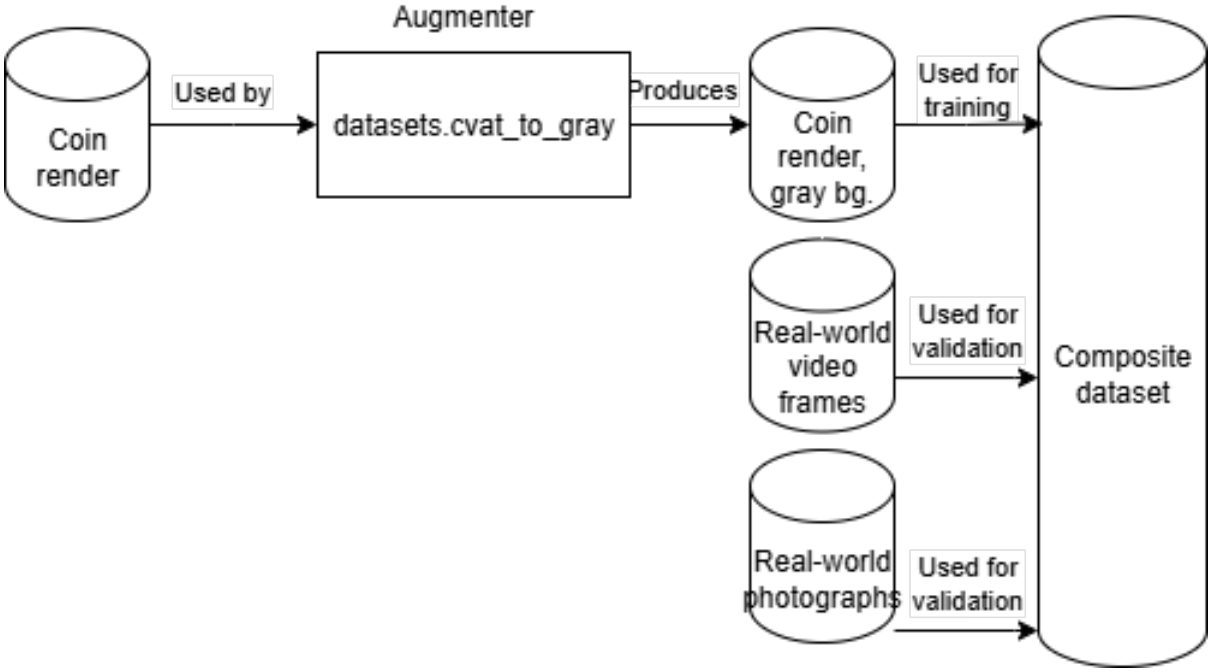


Figure 5: Composite of datasets which play different roles in the final dataset.

4.3 Parameter Grids

4.3.1 Synthetic Parameter Grid

The script responsible for the rendering of the dataset allows for the tuning of different hyperparameters described in Table1.

Table 1: Parameters for the generation of a varied dataset.

Parameter	Explanation
coin_x_metres	Coin's X coordinate (m)
coin_y_metres	Coin's Y coordinate (m)
coin_yaw	Rotation (degrees)
sun_angle	Sun angle (Blender)
sun_energy	Sun strength
sun_rotation	Elevation, azimuth
res_factor	Render resolution factor

4.3.2 Model Parameter Grid

Dataset composition was prioritised over extensive hyperparameter tuning - Ultralytic's implementation of their YOLO models handles these effectively by default. Table 2 shows the parameters available for exploration.

Table 2: Parameters for object detection model.

Parameter	Explanation
Image size	Input resolution. Increasing yields more object detail
Pretrained model	Defines the YOLO model to be used and its pretrained weights
Dataset	The desired composite dataset with its splits

5 Experiment Implementation

5.1 Generated Synthetic Datasets

Two datasets were generated for the evaluation with the hyperparameters shown in Table 3.

As defined by the parameter grid, Figure 6 shows the possible positions of the coin object rendered by Blender for both datasets.

Synthetic datasets do not have a visibility attribute, because by default they do not have any sort of distortion - apart from lens distortion. Figure 7 shows all the possible augmentations for a rendered sample.

Table 3: Parameters for the `synthetic` and `simple` datasets, the latter having less than the former. Explanations defined in Table 1.

Parameter	`synthetic`	`simple`
coin_x_metres	-0.5, -0.25, 0, 0.25, 0.5	-0.5, -0.25, 0, 0.25, 0.5
coin_y_metres	-0.1, 0, 0.25, 0.5	-0.1, 0, 0.25, 0.5
coin_yaw	0, 45, 90, 135, 180, 225, 270, 315	0
sun_angle	0.53	0.53
sun_energy	5, 10	10
sun_rotation	(15,90), (75,180), (15,270)	(75,180)
res_factor	1	1
Total	960	20

Augmentative scripts were devised to further the use of the synthetic samples **before training, into disk "cold"**; for example, the "_gray" variants of the primary datasets described in 5.2.1 are the results of one of these scripts. Figure 7 shows the augmentations applied to the initial synthetic dataset. Table 4 describes the different types of augmentations implemented.

5.2 Dataset Composition

5.2.1 Primary datasets

The following datasets could be considered "sub-datasets" as they aren't used on their own but are constituent members of a composite dataset.

Shot indoors

- **090**: Short video, exposure time of 1/90s, video stabilisation off

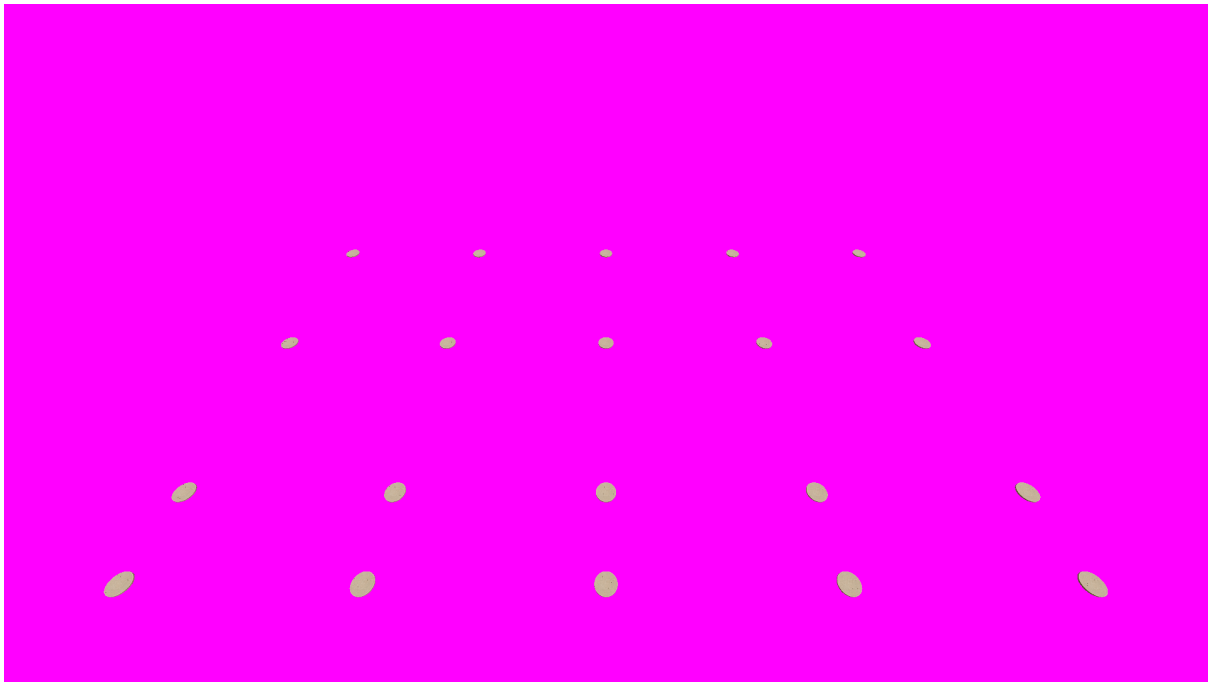


Figure 6: Composite of all the possible object positions specified in the Blender script, at its original.

Setting	Effect
Background colour	Makes the background a plain colour.
Background image	Makes the background a randomly chosen image.
Background zoom	Zooms in or out the background image.
Background horizontal offset	Moves the background horizontally.
Background vertical offset	Moves the background vertically
Background rotation	Rotates the background at an angle.
Angled blur	Applies a directional blur to the whole image.

Table 4: Implemented cold augmentations

- **190**: Short video, exposure time of 1/90s, video stabilisation on, generally blurry
- **30**: Short video, exposure time of 1/60s, video stabilisation off
- **60**: Short video, exposure time of 1/60s, video stabilisation off
- **90**: Short video, exposure time of 1/90s, video stabilisation off
- **clear1**: 4 still pictures, exposure time 1/90s
- **clear2**: 6 still pictures, exposure time 1/90s

Shot outdoors

- **clear3**: 55 still pictures, exposure time 1/1000s

Synthetic

- **synthetic**: 960 images, different positioning and lighting

(a) Raw PNG render

(b) Augmented by plain background

(c) Augmented by sidewalk bg.

(d) Various augmentations

Figure 7: Augmentations applied to sample

- **synthetic_gray**: same as `synthetic` but with gray background
- **simple**: 20 images, reduced hyperparameter count
- **simple_gray**, same as `simple` but with gray background
- **simple_bg**, same as `simple` but with all augmentations*

* = Settings: blur_prob=0.5, blur_kernel_range=(5, 15), horizontal_prob=0.5, vertical_prob=0.25, rotation_prob=0.75, zoom_prob=0.25

5.2.2 Composite datasets

Table 5 shows the datasets as seen by the models. Note that the "Filter" column makes reference to the visibility attribute explained in Section 3.1.1.

Composite dataset	Constituent	Split	Filter
ds0	synthetic_gray	train	*
	clear1	val	*
	clear2	val	*
	clear3	val	*
ds1	30	train	*
	60	train	*
	90	train	*
	090	train	*
	190	train	*
	clear1	val	*
	clear2	val	*
	clear3	val	*
ds2	30	train	elongated, distinguishable, inferred
	60	train	elongated, distinguishable, inferred
	90	train	elongated, distinguishable, inferred
	090	train	elongated, distinguishable, inferred
	190	train	elongated, distinguishable, inferred
	30	val	clear
	60	val	clear
	90	val	clear
	090	val	clear
	190	val	clear
ds3	30	train	*
	60	train	*
	90	train	*
	090	train	*
	190	train	*
	simple	train	*
	clear3	train	*
	clear1	val	*
	clear2	val	*
ds4	simple_bg	train	*
	clear3	val	*
	clear1	val	*
	clear2	val	*

Table 5: Dataset configurations. Each dataset seen by the final model is the result of the composition of various underlying datasets.

5.3 Model Training

Training was conducted using the Ultralytics Python package, which abstracts complex configuration steps while reducing direct control over certain hyperparameters. All models were trained with identical hyperparameter settings apart from image size and dataset composition. Training was halted based on validation loss stagnation for 20 consecutive epochs, as per the Ultralytics default early stopping criterion.

Table 6: First run, values of parameters for object detection model grid-search.

Parameter	Values
Image size	1024px, 512px
Pretrained model	yolov8n.pt, yolov8s.pt, yolo11n.pt, yolo11s.pt
Dataset	ds0, ds1, ds2, ds3

Table 7: Second run, values of parameters for object detection model grid-search.

Parameter	Values
Image size	1024px, 512px
Pretrained model	yolov8n.pt, yolov8s.pt, yolo11n.pt, yolo11s.pt
Dataset	ds4

5.4 Software

The following list is the software/libraries and their versions used to carry out this study.

Tool / Library	Purpose
Blender 4.4	Synthetic dataset generation
Python 3.10.12	Scripting
Ubuntu 22.04.5	Operating system platform
CUDA 12.8	GPU libraries
ultralytics 8.3.176	YOLO implementation on Python
notebook 7.4.4	Python interactive notebooks

Table 8: Tools and their purposes

5.5 Hardware

Training of neural networks can be an extremely long process if run on standard consumer-grade hardware. For this implementation, a compute instance with a NVIDIA H100 SXM with 80 GB of VRAM (53.5 TFLOPS, 363.8 DLPerf) was rented[7] due to its high performance.

6 Evaluation

6.1 Evaluation Protocol

Evaluation was performed on the validation dataset described by Table 5. The model never gets trained on the validation split, making it a fit candidate to evaluate the model's generalisation capabilities.

[7]Rent GPUs: `https://vast.ai/`

The main metric for experiment is recall. Recall -or sensitivity- measures the proportion of actual positives correctly identified by a model, used for applications where **an erroneous prediction is costly**. Recall is calculated as:

$$\text{Recall} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Negatives (FN)}}$$

A detection is counted as a true positive if the predicted class matches the ground truth, and the bounding box overlaps with a ground truth box beyond a specified IoU threshold (0.5 by default).

In addition to recall, **Average Precision at IoU threshold 0.5 (AP@0.5)** is also reported for the only class - *coin* class. AP@0.5 is a standard object detection metric that evaluates how well the model detects and localises coins. It is calculated by integrating the precision-recall curve, considering a detection correct if the Intersection over Union (IoU) between the predicted bounding box and the ground truth is at least 0.5. The IoU is calculated as:

$$\text{IoU} = \frac{\text{Area of Overlap}}{\text{Area of Union}}$$

This metric provides a more comprehensive view of model performance for the coin class, as it jointly considers both detection accuracy and localisation quality.

AP@0.5 is computed as:

$$\text{AP} = \int_0^1 p(r)\, dr$$

where $p(r)$ is the precision-recall curve at IoU threshold 0.5.

Other metrics (AP@[.50:.95]) were computed but are presented only in the Appendix.

The choice of recall as the primary metric reflects the practical requirement of maximising detection of true objects, even at the expense of allowing some false positives. This is consistent with work in critical and anomaly detection and akin to the experiment.

The F1 score is a harmonic mean of precision and recall, providing a balanced measure of a model's performance that accounts for both false positives and false negatives. It is particularly useful when dealing with imbalanced datasets, as it equally weights precision (the proportion of correct positive predictions) and recall (the proportion of actual positives correctly identified). The F1 score is calculated as:

$$F1 = 2 \times \frac{\text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}} = \frac{2TP}{2TP + FP + FN} \tag{1}$$

6.2 Evaluation results

Table 9 and 10 display the metrics returned by the model's validation evaluation. [8]

6.2.1 Curves

Curves in Figure 8 show their respective curves for each model.

[8] Model Validation with Ultralytics YOLO : `https://docs.ultralytics.com/modes/val/`

Table 9: Best performing models, top 3, first run

Size	Model	Dataset	Precision%	Recall%	F1 Score%	AP@0.50%
1024	yolov8s.pt	ds3	100.00	99.08	99.54	99.50
1024	yolov8n.pt	ds3	93.11	90.91	91.99	93.59
1024	yolo11s.pt	ds3	85.05	72.73	78.41	82.47

Table 10: Best performing models, top 3, second run

Size	Model	Dataset	Precision%	Recall%	F1 Score%	AP@0.50%
1024	yolo11s.pt	ds4	72.03	66.67	69.24	72.22
1024	yolov8s.pt	ds4	84.81	59.09	69.65	62.75
512	yolo11s.pt	ds4	63.63	34.85	45.03	36.22

6.3 Discussion

The results shown in Table 9 give a clear picture in regards to dataset performance: ds3, whose composition consists majorly of real-world data, and is the common factor among the top constituents of the list, towers over the others in performance.

The difference on performance of the best model using `ds1` -a fully real-world dataset, featuring exclusively frames of videos, not stills- with that of `ds3` may be due to the addition of the `clear3` sub-dataset to the training split, and not the addition of `simple`.

As expected, the more resolution, the more performance: size 1024 is a top performer since it allows the neural network to see more detail.

However, for the second run which only features one synthetic shown in Table 10 -this time heavily augmented,- the metrics are promising. With an F1 Score of 69.24% and AP@0.50 of 72.22%, this proves a render of a small object, followed by heavy augmentations, does yield a performing model, albeit not as performant as if trained with real data.

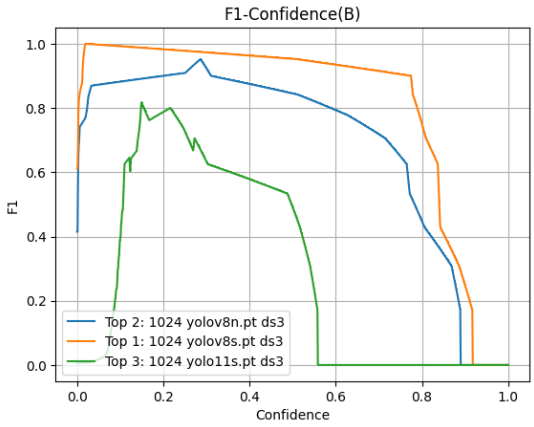
The visibility categories had a direct effect on convergence and overall detection quality. Models trained exclusively on "inferred" samples performed poorly, showing how the lack of visual features restrict learning.

Synthetic data underperformed compared to real data, likely because real-world images contain subtle cues such as compression artifacts, complex lighting, and cues that get produced with cohesive scenes, among others that were absent in the synthetic pipeline.

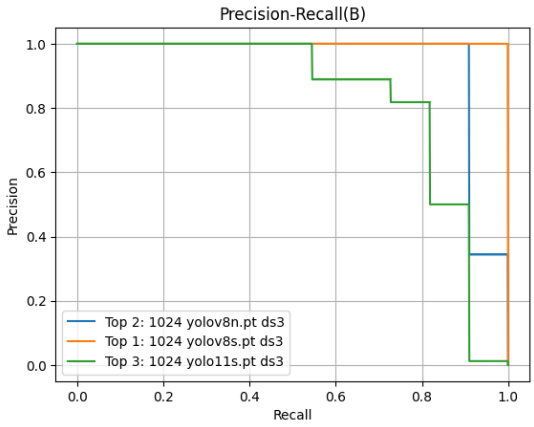
A key factor influencing model performance was the balance between dataset quality and hyperparameter tuning. In this study, dataset quality proved more decisive. Small object detection was particularly sensitive to image preprocessing, since reductive

Table 11: First run's best performing models, grouped by dataset.

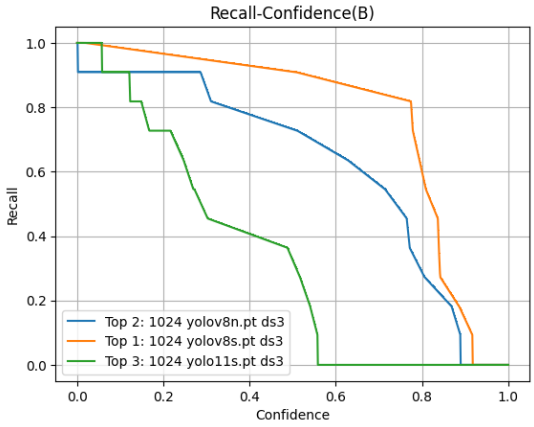
Size	Model	Dataset	Precision%	Recall%	F1 Score%	AP@0.50%
1024	yolov8s.pt	ds3	100.00	99.08	99.54	99.50
1024	yolo11s.pt	ds0	60.62	60.63	60.62	56.87
1024	yolo11s.pt	ds1	44.63	41.52	43.02	32.81
512	yolov8n.pt	ds2	36.34	33.99	35.12	20.31



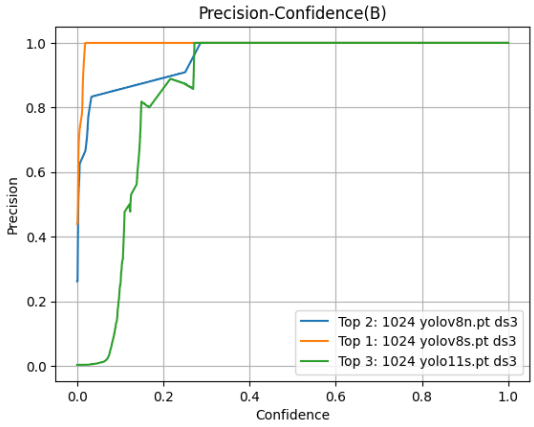
(a) F1-Confidence



(b) Precision-Recall



(c) Recall-Confidence



(d) Precision-Confidence

Figure 8: Boundary box curves for top 3 best performing models for first run

operations (downscaling) caused a significant loss in detail, especially considering the dimensions in the canvas of these small objects. While hyperparameter tuning can be important, the abstraction of them by the model provider made their impact secondary for this study.

A granular inspection of `ds3` reveals that the inclusion of the `clear3` (55 stills) and `simple` (20 synthetic samples, totaling 75 samples) had a decisive impact, complementing the noisier video frames of `30, 60, 90, 090, 190`. This suggests the sharp samples acted as anchors during training.

In contrast, the results showed by, `ds4`, only used `simple_bg` for training (synthetic with real world-backgrounds and augmentations), validated against `clear1, clear2, clear3` (stills) demostrate that heavily augmented synthetic data can replace substitute some real-world data with decent results.

In cluttered scenes the models occasionally misclassified look-alike objects like flattened gum or bottle caps. This behaviour is critical in real-world deployments, where excessive false positives reduce trust in the system.

7 Conclusion

This study tried to answer how different object detection models perform with small object recognitions and whether these models can generalise well with synthetic data. For this purpose, augmentations and varied model parameters and dataset compositions were brought up. The results show that with heavy augmentation (custom background with zoom, rotation, blur) **it is possible to make a well performing object detector**. For this purpose, the models **yolov8s.pt, yolo11s.pt** (YOLOv8, YOLOv11; both the "small" variant and pretrained on COCO) may be used due to their performance for the task in hand.

7.0.1 Future Work

Future work may involve a composite dataset study (quantifying the influence of a sub-dataset); advanced synthetic dataset generation, (fully 3d scene, shaders, realistic shadow casting, scene randomisation, etc.); evaluating at a bigger scale (more hyperparameters for models and dataset synthesis); pipeline agnosticity (one pipeline for all models); random search; and optimising the augmentative pipeline to work in memory and just-in-time. These adjustments would help narrow the synthetic-to-real gap, especially for cases where sample irregularities affect detection.

References

Carranza-García, M., Torres-Mateo, J., Lara-Benítez, P. and García-Gutiérrez, J. (2020). On the Performance of One-Stage and Two-Stage Object Detectors in Autonomous Vehicles Using Camera Data, *Remote Sensing* **13**(1): 89.

Diwan, T., Anirudh, G. and Tembhurne, J. V. (2023). Object detection using YOLO: challenges, architectural successors, datasets and applications, *Multimedia Tools and Applications* **82**(6): 9243–9275.

Gavriel, P., Norton, A., Kimble, K. and Zimmerman, M. (2024). Towards an efficient synthetic image data pipeline for training vision-based robot systems, *arXiv preprint arXiv:2411.06166* .

Hussain, M. (2024). Yolov5, yolov8 and yolov10: The go-to detectors for real-time vision.

Jegham, N., Koh, C. Y., Abdelatti, M. and Hendawi, A. (2025). Yolo evolution: A comprehensive benchmark and architectural review of yolov12, yolo11, and their previous versions.

Jocher, G. and Qiu, J. (n.d.). Ultralytics yolo.
URL: *https://github.com/ultralytics/ultralytics*

Nikolenko, S. I. (2019). Synthetic Data for Deep Learning. arXiv:1909.11512 [cs].

Redmon, J. (2018). Yolo: Real-time object detection (website), `https://pjreddie.com/darknet/yolo/`.

Tajbakhsh, N., Shin, J. Y., Gurudu, S. R., Hurst, R. T., Kendall, C. B., Gotway, M. B. and Liang, J. (2016). Convolutional Neural Networks for Medical Image Analysis: Full Training or Fine Tuning?, *IEEE Transactions on Medical Imaging* **35**(5): 1299–1312.

Tan, L., Huangfu, T., Wu, L. and Chen, W. (2021). Comparison of retinanet, ssd, and yolo v3 for real-time pill identification, *BMC Medical Informatics and Decision Making* **21**: 324.

Terven, J. and Cordova-Esparza, D. (2023). A comprehensive review of yolo architectures in computer vision: From yolov1 to yolov8 and yolo-nas, *arXiv preprint arXiv:2304.00501* .

Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W. and Abbeel, P. (2017). Domain Randomization for Transferring Deep Neural Networks from Simulation to the Real World. arXiv:1703.06907 [cs].

Tremblay, J., Prakash, A. and et al. (2018). Training deep networks with synthetic data: Bridging the reality gap by domain randomization, *arXiv preprint arXiv:1804.06516* .

van den Hoven, M. (2023). Blenderline: A blender pipeline for generating synthetic images of production lines, `https://github.com/maxvandenhoven/blenderline`.

Zhu, X., Henningsson, J., Li, D., Mårtensson, P., Hanson, L., Björkman, M. and Maki, A. (2025). Domain Randomization for Object Detection in Manufacturing Applications using Synthetic Data: A Comprehensive Study. arXiv:2506.07539 [cs].

Özeren, E. and Bhowmick, A. (2025). Evaluating the Impact of Synthetic Data on Object Detection Tasks in Autonomous Driving. arXiv:2503.09803 [cs].

A Full Results

Size	Model	Dataset	Precision	Recall	F1Score	Speed	AP@0.50	AP@0.55	AP@0.60	AP@0.65	AP@0.70	AP@0.75	AP@0.80	AP@0.85
1024	yolov8n.pt	ds0	0.28	83.33	0.55	2.47	32.67	29.81	24.28	19.58	9.58	5.14	1.41	0.24
1024	yolov8n.pt	ds1	0.13	37.88	0.25	1.74	0.10	0.09	0.05	0.02	0.01	0.01	0.00	0.00
1024	yolov8n.pt	ds2	0.05	49.62	0.10	9.07	0.04	0.03	0.03	0.02	0.02	0.01	0.00	0.00
1024	yolov8n.pt	ds3	93.11	90.91	91.99	12.16	93.59	93.59	93.59	91.68	91.68	81.72	71.54	47.24
1024	yolov8s.pt	ds0	50.24	53.03	51.60	1.31	42.19	42.19	38.81	38.70	31.92	22.98	10.44	1.02
1024	yolov8s.pt	ds1	47.82	25.76	33.48	1.77	30.15	26.95	24.45	21.53	14.03	3.00	1.02	0.25
1024	yolov8s.pt	ds2	20.54	19.74	20.13	6.09	10.03	10.03	10.03	9.77	9.18	7.63	7.11	3.30
1024	yolov8s.pt	ds3	100.00	99.08	99.54	5.65	99.50	99.50	99.50	99.50	99.50	99.50	66.14	20.62
1024	yolo11n.pt	ds0	63.26	44.37	52.16	2.45	45.73	42.50	38.85	34.42	31.96	24.00	11.29	0.50
1024	yolo11n.pt	ds1	0.02	4.55	0.03	1.75	0.01	0.01	0.01	0.00	0.00	0.00	0.00	0.00
1024	yolo11n.pt	ds2	0.03	29.01	0.06	5.84	0.02	0.02	0.02	0.01	0.01	0.00	0.00	0.00
1024	yolo11n.pt	ds3	0.18	54.55	0.36	6.14	0.16	0.16	0.12	0.09	0.09	0.04	0.02	0.00
1024	yolo11s.pt	ds0	60.62	60.63	60.62	2.48	56.87	56.87	56.87	45.39	26.78	11.70	2.67	0.26
1024	yolo11s.pt	ds1	44.63	41.52	43.02	7.02	32.81	32.81	32.40	29.34	22.71	10.08	1.97	1.16
1024	yolo11s.pt	ds2	8.90	9.16	9.03	10.89	2.37	2.37	2.31	1.98	1.72	1.70	0.59	0.23
1024	yolo11s.pt	ds3	85.05	72.73	78.41	9.62	82.47	82.47	82.47	82.38	82.38	73.00	10.46	0.00
512	yolov8n.pt	ds0	21.84	3.03	5.32	2.84	4.08	2.71	2.40	1.51	1.50	0.00	0.00	0.00
512	yolov8n.pt	ds1	47.10	1.43	2.77	1.59	10.68	3.66	1.79	0.42	0.42	0.42	0.03	0.02
512	yolov8n.pt	ds2	36.34	33.99	35.12	2.69	20.31	16.55	12.25	8.84	2.84	0.79	0.17	0.03
512	yolov8n.pt	ds3	0.75	54.55	1.48	8.80	37.42	37.42	23.78	16.12	15.77	0.00	0.00	0.00
512	yolov8s.pt	ds0	17.66	13.64	15.39	1.29	6.46	6.44	3.60	1.34	0.37	0.17	0.00	0.00
512	yolov8s.pt	ds1	46.15	10.61	17.25	0.84	8.47	6.69	6.68	6.29	3.04	1.67	0.00	0.00
512	yolov8s.pt	ds2	17.21	7.63	10.58	2.47	3.55	3.51	3.51	3.46	2.51	1.80	0.54	0.04
512	yolov8s.pt	ds3	100.00	45.39	62.44	5.68	46.12	46.12	46.12	46.12	31.68	31.68	9.70	0.00
512	yolo11n.pt	ds0	0.04	10.61	0.07	1.29	0.02	0.02	0.01	0.01	0.00	0.00	0.00	0.00
512	yolo11n.pt	ds1	14.29	1.52	2.74	0.37	7.54	7.54	7.54	7.54	7.54	7.54	7.54	7.54
512	yolo11n.pt	ds2	0.01	8.40	0.02	2.58	0.01	0.00	0.00	0.00	0.00	0.00	0.00	0.00
512	yolo11n.pt	ds3	0.06	18.18	0.12	4.39	0.07	0.03	0.00	0.00	0.00	0.00	0.00	0.00
512	yolo11s.pt	ds0	30.31	21.21	24.96	0.93	17.41	17.26	16.27	13.77	10.50	3.75	3.25	0.92
512	yolo11s.pt	ds1	11.50	15.15	13.08	1.59	4.90	4.86	4.57	3.45	3.34	3.28	2.73	2.18
512	yolo11s.pt	ds2	3.97	2.29	2.90	3.59	0.25	0.25	0.24	0.23	0.21	0.19	0.07	0.01
512	yolo11s.pt	ds3	76.75	9.09	16.26	4.26	11.17	11.17	10.24	10.24	10.24	10.21	9.88	9.51

Table 12: Results of grid search