Development of WaldoNet: A Novel Approach to Solving ”Where’s Waldo”

Barney H. Miao
barneym@stanford.edu

Andrew Lesh
aclesh@stanford.edu

Abstract

In this paper we introduce the development of WaldoNet, which involves the use of small object detection methods for finding Waldo (1-class small object detection) and other characters (4-class small object detection) within the ”Where’s Waldo” series of picture books. Specifically, this study conducts small object detection through a YOLOv9 (You Only Look Once) model, which is the latest iteration of a popular real time object detection algorithm [14]. For our baseline, we utilized existing publicly available pre-labeled images to train the YOLOv9 model, which had test accuracies of 43.7% and 40.6% for 1-class and 4-class small-object detection respectively. For our implementation, we focused on improving the dataset that is used to train the YOLOv9 model, leads to improved test accuracies of 54.1% and 82.8% for both 1-class and 4-class scenarios respectively.

1. Introduction

Distinguishing between objects in our surroundings based on their semantic importance is critical to survival and all aspects of human life, whether it means locating food, avoiding predators, or looking for your car in a crowded parking lot. Due to this paramount importance, object detection is one of the main problems that need to be solved in computer vision. Prominent object detection applications include: determining proper mask usage, being able to discern between various objects, and locating objects from images and videos[17]. For this study, we aim to develop WaldoNet, a convolutional neural network trained to aid humans in locating the cartoon character Waldo in a given static image. This project aims to serve as an amusing and child-friendly illustration of the capabilities of small-object detection models. Each book in the *Where’s Waldo* series consists of a series of large, complex drawing where the player is tasked with locating Waldo despite the many present distractions.

Unlike some other object detection problems, finding Waldo in a given image involves solving a small-object detection problem. Small object detection is a more difficult problem than detecting objects that are proportionally larger compared to their background. Three key issues are: 1) small objects tend to have less information to distinguish them from the background or similar categories than larger objects, 2) the precision needed to locate small objects is higher, and 3) there are more established models for detecting medium and large objects (models for small-object detection are relatively scarce) [18]. Figure 1 shows an example of the difficult problem posed by *Where’s Waldo*.

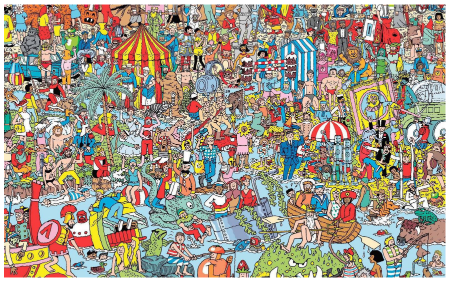

Figure 1. Example of the *Where’s Waldo* challenge. Can you find Waldo without the use of WaldoNet? If you want unleash the power of WaldoNet, please refer to Figure 8.

2. Related work

Overview of Object Detection: Variations of object detection that have been previously implemented fall into two general groups: (1) Two-stage detectors (utilizing proposal based regions), and (2) one-stage detectors (free of proposal-based regions).

Two stage detectors:

R-CNN detectors: R-CNN uses a two-stage approach of separate region proposal and object classification. A 4096 dimensional vector is extracted, by feeding each region proposal into a deep CNN [12, 5]. R-CNN and its variants all use region proposals instead of sliding windows to find objects in images [19]. In the case for R-CNN, selective search is used to generate potential bounding boxes, with multiples stages following. This results in a system that is very slow that takes up to 40 seconds per image at test time [14].

Improvements to R-CNN detectors: Improvements have been made to R-CNN in the form of Fast R-CNN [4] and Faster R-CNN [16], but they all still divide the detection problem into two stages: the region proposal stage and the detection stage [12]. Fast R-CNN and Faster R-CNN offer speed and accuracy improvements over R-CNN by using neural networks to generate proposed regions instead of the selective search algorithm[4, 16] and are much faster at test time (only about 200 milliseconds per image). R-CNN series (R-CNN, Fast R-CNN, and Faster R-CNN) has been successfully implemented in object detection problems ranging from: identifying traffic signs, PASCAL VOC 2012, and vehicle detection [2, 26, 8].

Feature pyramid network (FPN): FPN utilized the hierarchy of a CNN network that first passed an input image through a CNN before using a pooling layer to shrink the size of the feature maps. The shrunk feature maps are then up-sampled to the same size as the input image [9]. The generated FPN feature maps were shown to have positive impacts on detection accuracy, specifically with a more dramatic improvement for detecting small objects.

One-stage detectors:

Single shot detector (SSD): The SSD is a single shot detector, that does not include the use of a regional proposal network. This method is similar to the faster R-CNN method, with the primary difference being that SSD performed multiple feature layers to perform detection. However, SSD performs poorly on small object detection problems due to using shallow layers without deep semantic relevance [11, 12].

RetinaNet: The development of RetinaNet was inspired by the need to have a one-stage object detector with fast detection time while also reducing the accuracy gap with two-stage detectors [10]. The accuracy gap between one-stage and two-stage detectors was primarily due to the high imbalance in the numbers of positive and negative examples, as well as an imbalance in the number of "easy" and "hard" examples used in training. By having too many easy examples in training, the loss function will become dominated by these examples, resulting in a degenerated model. To help tackle this issue, RetinaNet utilizes the focal loss function, which adaptively reduces the weights of easy examples.

YOLO: the YOLO (You Only Look Once) model, has superior performance to other convolutional networks [21, 13]. This is because YOLO performs real-time object detection through a single pass in a neural network, which contributes to its greater speed and efficiency than traditional CNNs. The original YOLO model is a series of 24 convolutional layers (with a3 few maxpool layers) before two fully connected layers (see appendix Fig 11) [15]. Additionally, a precedent exists for using YOLO models in solving the *Where's Waldo* problem, making it a good starting point for our project [23].

3. Methods

In both 1-class and 4-class small object detection, our objective is to detect Waldo (and other characters) on an image and draw a bounding box around them using the YOLO model, specifically the newest available version YOLOv9. YOLOv9 introduces some upgrades to the system including programmable gradient information (PGI) and a generalized efficient layer aggregation network (GELAN) with convolutional layers as its architecture [22].

PGI is introduced as a method of auxiliary supervision during training that is intended to aid convergence for both large, deep networks and lightweight, shallower models [22]. In addition to a main branch, PGI has an "auxiliary reversible branch" to provide gradient information to the main branch when deep features of the data would have been otherwise lost due to an information bottleneck as well as "multi-level auxiliary information", which places an "integration network between the feature pyramid hierarchy layers of auxiliary supervision and the main branch, and then uses it to combine returned gradients from different prediction heads" to update the parameters without running into the broken information problem that often occurs during deep supervision (see appendix Fig 12) [22].

GELAN, which combines ELAN (efficient layer aggregation network) and CSPNet (cross stage partial network) [22]. ELAN can only use stacking of convolutional layers, but GELAN is generalized to work with any type of layer by including the additional transition stage of CSPNet. By employing a GELAN architecture that solely uses standard convolutional layers, the designers of YOLOv9 were able to achieve higher parameter usage, speed, and accuracy than models using depthwise convolution. In contrast with the original YOLO architecture which is a simpler series of convolutional layers, the GELAN architecture both feeds the output of each convolutional layer forward to the next while also combining each layer's output (see appendix Fig 13) [22].

To determine the performance of these models we will be producing loss curves to track the overall loss during training and validation (per epoch). Equation 1 shows the breakdown of the YOLO loss function, with the main loss being broken down into localization loss, confidence loss (objectness), confidence loss for non-objects, and classification loss [14].

General YOLO loss function:

$$\mathcal{L}_{total} = \mathcal{L}_{localization} + \mathcal{L}_{confidence,obj} + \mathcal{L}_{confidence,non-obj} + \mathcal{L}_{classification}$$

$$\mathcal{L}_{total} = \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{ij}^{\text{obj}} \left[(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2 \right] .$$

$$+\left[\lambda_{\text{coord}}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{\text{obj}}\left(\sqrt{w_i}-\sqrt{\hat{w}_i}\right)^2+\left(\sqrt{h_i}-\sqrt{\hat{h}_i}\right)^2\right]$$
$$+\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{\text{obj}}\left(C_i-\hat{C}_i\right)^2$$
$$+\lambda_{\text{noobj}}\sum_{i=0}^{S^2}\sum_{j=0}^{B}\mathbb{1}_{ij}^{\text{noobj}}\left(C_i-\hat{C}_i\right)^2$$
$$+\sum_{i=0}^{S^2}\mathbb{1}_{i}^{\text{obj}}\sum_{c\in\text{classes}}\left(p_i(c)-\hat{p}_i(c)\right)^2 \quad (1)$$

Where S represents the number of grid cells in the image; B represents the number of bounding boxes predicted by each grid cell; λ_{coord} and λ_{noobj} are hyparameters controlling the importance of classification and confidence terms in the loss equation; x_i and y_i are the center coordinates of the predicted bounding box; w_i and h_i are the width and height of the predicted bounding box; C_i represents the confidence score of the presence of an object in the box; p_i represents the predicted probability of each class for the object within the predicted bounding box; and $\mathbb{1}^{\text{obj}}$ and $\mathbb{1}^{\text{noobj}}$ are indicator functions that equal 1 if an object appears in cell i and if the cell doesn't contain an object, respectively, and 0 otherwise.

Since the game of *Where's Waldo* is centered upon finding the titular character, Waldo, within the scene of each two-page spread, we began by training the YOLOv9 model with a 1-class object detection task (finding Waldo against the chaotic background). Because the books have additional characters to find in order to give the player a longer, more challenging experience, we also trained YOLOv9 for a 4-class version of the task in order to detect not only Waldo but the three additional main characters. We expected that a dedicated 1-class version would have higher accuracy for finding Waldo that might make it more useful for players that are focused on finding Waldo alone, which is what most players do. The 4-class version was intended to be more broadly useful while anticipating that the accuracy for finding any of the individual characters in the final version may be lower than that of the 1-class model due to us training on fewer images per character for the latter method (we used a separate dataset of the same size (2,075 images) for the 1-class and 4-class tasks).

3.1. Baseline Model: Using pre-existing Waldo data

Before preparing our own custom datasets for the 1-class and 4-class tasks, we used pre-existing datasets posted to the Roboflow imageset repository. The imageset we used for each was fairly small (only a few hundred total images, see Table 1). During our initial attempts for 1-class and 4-class small-object detection of Waldo characters, we noticed that the accuracy of the Waldo models were highly dependent on the manner of how the bounding boxes were drawn. For instance, if bounding boxes for input data were always drawn over the entirety of the body of the Waldo characters, it resulted in the YOLOv9 model having almost 0% accuracy. Therefore, for the baseline model we trained our YOLOv9 model on existing data that had bounding boxes primarily drawn over the face of each character. Later to further improve upon the performance of the baseline model, we focused on drawing bounding boxes exclusively on the face of the character.

3.2. Our Model: Using processed Waldo data

In order to achieve higher detection accuracy, we sought to increase the size of the datasets for both the 1-class and 4-class object detection tasks by at least an order of magnitude while decreasing the fraction of images that were background (not featuring a labeled character). To do this, we took images of the original book pages and cropped them into grids of 128x128 pixel images and superimposed the face of one character onto each of the images (except for a small portion of images that we left as background). The documentation for YOLOv9 suggests having a background image fraction between 1 and 10%, so we chose to have this fraction as 2% [20].

The baseline image sets (for both 1-class and 4-class classification tasks) we found and labeled with Roboflow had a very small number of total images, and this appears to be the norm for past attempts at developing image classifiers for solving the *Where's Waldo* game. The primary issue leading to the dearth of available images is the fact that within each large two-page spread, Waldo and each of the other characters only occur once. With each book having around a dozen spreads, that's only twelve images of each of the characters per book! Not wanting to spend money on purchasing additional books, we found photographic scans of the original four books, which still only amounts to 48 instances of each class.

It is recommended that for YOLO models in their official documentation that between 1 and 10% of images in the dataset should be background images in order to minimize the number of false positives [20]. Cropping each two-page spread into 128x128 pixel squares resulted in around 140 images per spread. For the 4-class problem, this means 97% of images are background and for the 1-class problem, over 99% of images are background. The only ways to decrease the fraction of background images in the dataset are to remove background images (which would result in a very small overall dataset) or to add more images containing Waldo and the other characters.

Our chosen solution was to produce synthetic images of the characters by randomly superimposing their faces on background images. First, for each original two-page spread, we located the four characters and manually cropped their faces out. Second, we manually erased the background of each face image using an image editing application (see Fig 3). Third, we ran a script that iterated through all of the automatically cropped 128 by 128 images from each two-page spread and superimposed one of the faces on each at a random location to create a new image. Finally, our script would stop superimposing faces when it reached the number of images determined for the labeled fraction (in our case 98%), leaving the remaining fraction as unmodified background images (in our case 2%).To stay consistent with color and contrast, all of the images originating from the same two-page spread would only have faces superimposed that were originally taken from that same two-page spread.

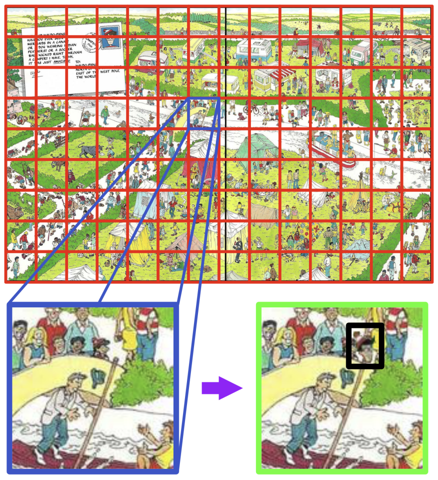

Figure 2. Top: 128 by 128 pixel cropping grid for original two-page spread. Left: Individual cropped image. Right: Cropped image with Waldo face superimposed at random location (boxed in black outline)

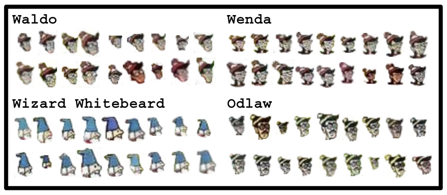

Figure 3. The isolated faces of each character used to create the superimposed images. One face was used from each original two-page spread from the books.

We chose 2% as the fraction of background images to use, close to the recommended lower bound for YOLO models. Accordingly, once our script had iterated through 98% of the cropped images for a particular two-page spread, it left the remaining portion unmodified. To make the dataset for training and evaluating the 1-class version of our model, we only superimposed faces of Waldo on the 98% of modified images. For the 4-class dataset, we superimposed an equal portion of the 98% with each of the four characters, one of each per image (25.4% of the images for each character). On the occasion that an image contained the original location of a character on the two-page spread, we labeled that as well, so a few images out of the 2,075 images in each dataset have two labelled faces (Table 1).

4. Dataset

All images in the data sets used originally derive from the *Where's Waldo* books, which each consist of series of two-page spreads. Each two-page spread features a large, complex cartoon drawing of hundreds of people, animals, and other objects with Waldo located in a different location within each spread, along with additional characters like Wenda, Odlaw, and Wizard Whitebeard. For reference, Wenda is a female counterpart to Waldo who dresses very similarly, Odlaw is an evil counterpart to Waldo with black and yellow striped clothing instead of red and white stripes, and Wizard Whitebeard is a wizard that helps Waldo on his adventures. See Figure 2 in the appendix for a visual comparison of the characters.

Images of Waldo are available in two relevant forms online: (1) Original two-page spreads from the *Where's Waldo* book series, and (2) smaller cropped images of these books in public datasets (256x256, 128x128, and 64x64)[1]. Cropping the large, two-page spreads into smaller square images is helpful for increasing the number of total images, but also for limiting each image to a more manageable size for training. For the YOLOv9 models, preliminary runs were made on both publicly available sources of Waldo images with pre-drawn bounding boxes and on images that we manually labelled. For the pre-labelled data, they were obtained on Roboflow (76 for 1-class and 313 for 4-class) for 1-class classification scenarios (Waldo vs. background) and 4-class scenarios using the expanded list of characters (Waldo, Wenda, Odlaw, Wizard Whitebeard, and background)[24, 3, 7].

4.1. Labeling Waldo

While in some images Waldo's body and entire outfit are clearly visible, in some images only Waldo's head is visible with other people and objects occluding him. This is a choice by the illustrator to increase the challenge of finding him. The only part of Waldo guaranteed to be visible in every image from the books is his face. Accordingly, bounding boxes around Waldo's entire body will frequently include non-Waldo occluding objects. If a model is trained with these (often occluded) full-body bounding boxes, the model will be searching for features of Waldo that are often absent or features that don't even have a semantic connection to the proper Waldo class. Hence, for the final set of custom images, we ran the model of our study on images with bounding boxes only around the face of the characters.

To prepare our cropped images for object detection, the images were manually labelled using Roboflow. Initial attempts (not shown here) involved two types of bounded boxes: (1) surrounding Waldo's entire body and (2) surrounding only Waldo's face (Figure 14). The YOLOv9 model was unable to identify Waldo in any instance (0% validation accuracy) when trained with strategy 1, while the model trained on just Waldo's face was able to achieve our initial reported accuracy in section 5 (87% test accuracy). This difference is likely caused by occlusion, as explained in the previous paragraph.

For the final implementation of the model, we cropped individual images from photographs of the original two-page spreads from the first two *Where's Waldo* books (the original *Where's Waldo?* (1987) and its sequel *Where's Waldo Now?* (1988)), with twelve spreads from the first book and seven spreads from the second book. A python script automatically cropped each of the spreads into a grid of 128x128 pixel images (with zero padding to fill in any overhangs of the 128x128 pixel cropping window, causing a few of the images to have a black border on one or two sides). This resulted in 2,075 total cropped images, with an average of approximately 109 images per spread.

Baseline Model	
Scenario:	train/ val/ detect
1-class	159, 15, 8 $\rightarrow$ obtained from [7]
4-class	396, 56, 32 $\rightarrow$ obtained from [24]

Our Implementation	
Scenario:	train/ val/ detect & num per class [train]
1-class	1453, 415, 207 2022, 42
4-class	1453, 415, 207, 526, 520, 504, 513, 45

Table 1. Breakdown of image data used to train baseline and our model of WaldoNet for 1-class (Waldo, Background) and 4-class (Waldo, Wenda, Odlaw, Wizard, Background) detection. Note the table numbers are in the order shown in this caption.

We found images of two-page spreads from the first four *Where's Waldo* books and initially cropped all of them, but we only manually labeled the first 2,075 images for both the 1-class and 4-class datasets due to the significant labor time required for labeling. For the 1-class dataset, it only took about five hours to label all of the images because there was only one class to choose from. For the 4-class dataset, labelling took over twice as long (over ten hours) because of the need to alternate between labeling different classes prevented us from simply hitting the same combination of buttons in the Roboflow labeling application over and over. It was also harder to find multiple characters than the same character over and over due to the increased variety each time. While labeling both datasets, it would often take several seconds to locate the character's face, as the face is still a fairly small object to find, even within a 128x128 pixel square, and some would require much longer when they stood out less against the background.

5. Experiments

For this study we developed a method to solve *Where's Waldo* using the latest iteration of YOLO models — YOLOv9 — for 1-class and 4-class small object detection problems. We leverage a pre-trained YOLOv9 model on the MS COCO object dataset (classes = 80), and retrained the model on 1-class and 4-class *Where's Waldo* images labeled using Roboflow. In these scenarios we used the weights and cfg file defined in yolov9-e.pt and yolov9-e.yaml due to their speed in training and performing inference [25]. For our initial attempts at hyperparameter tuning we changed the parameters defined in the "hyp.scratch-high.yaml" file, as discussed later.

5.1. Evaluation Benchmark

Several evaluation metrics were used to evaluate the performance of the baseline model and WaldoNet on 1-class and 4-class "Where's Waldo" detection. In this study, we put more emphasis on metrics related to the classification ability of our models, as the accuracy of the bounding box is of lesser importance compared to the correct identification of a character. Confusion matrices, showing the classification performance were developed for the baseline and WaldoNet models. From the confusion matrices, we calculated the precision, recall, and F_1 score of the models. Typically, models with high recall and high precision will have a large area under the precision-recall curve. High precision is indicative of low false positive rates and high recall means the model has a low false negative rate. In terms of evaluating the test accuracy of the models we make use of Equation 2.

$$Accuracy = \frac{Count\ of\ images\ with\ correct\ classification}{Total\ images} \tag{2}$$

5.2. Hyperparameter Tuning

To help determine the optimal set of hyper-parameters to use for our baseline and final model, we changed the initial learning rate, initial batch size, and number of training epochs. Table 2 shows selected hyper-parameters that deviate from the ones used in a typical YOLOv9 model. The rationale for these choices is summarized below.

Parameter	Baseline	WaldoNet
Initial learning rate	0.01	0.01
optimizer	SGD	SGD
Momentum	0.973	0.973
Batch size 1-class	32	25
Batch size 4-class	32	16
Epochs	50	100

Table 2. Select hyper-parameters used for YOLOv9 in Baseline and WaldoNet models

Learning rate: In the YOLOv9 model there are two learning rate parameters: (1) initial learning rate and (2) final learning rate. For the purposes of this study we only adjusted the initial learning rate, and kept the final learning rate at the default of 0.01. Models that had a lower learning rate (0.001 or the default 0.01) typically had lower final training and validation errors, while models that used a higher initial learning rate had greater final losses. We can see that the YOLO model is sensitive to changes in the initial learning rate (LR) on the training and validation loss, as a LR of 0.1 results in overstepping as shown by the drastic increase in the training loss after a few epochs (other losses spiked, so they are not shown there). However, a smaller learning rate seems to help improve the generalizability of the YOLO model, as it has a slightly lower validation loss in comparison to the other models.

Batch Size: We used varying batch sizes of 8, 16, and 32 to train the baseline model. The models were trained locally using an NVIDIA RTX 3060 GPU. For the final model, we used batch sizes of 8, 16, and 25, due to memory limits with the local GPU. Figure 4 shows some of the experiments using different batch sizes, revealing that models with larger batch sizes (either 16 or 25/32) had lower final training and validation errors.

Number of Epochs: From Figure 4 Initially, the default YOLOv9 model starts with 25 epochs. However, after plotting the loss curves, it was evident that the losses were still decreasing. Hence, we increased the number of epochs to 50 to help further reduce the validation loss of the model. When we were initially training the baseline models we used 50 epochs, and looked at the loss curves to see if our model was potentially over-fitting. Although our training loss for box and class is still decreasing, we see that the validation loss has mostly leveled off at the end of the 50 epochs. If we were to continue to train the model in this scenario, we might see further reductions in the training loss, but we would potentially be over-fitting our data. We accordingly used 50 epochs to train the models. In the case of the final model, the validation loss was still noticeably decreasing at 50 epochs, so we continued running those models for a total of 100 epochs.

5.3. Baseline model

For the baseline model we used the publicly sourced data from Roboflow as summarized in Table 1. The images obtained from Roboflow were split into separate folders for training, validation, and testing and a .yaml file which is needed for YOLOv9. Upon obtaining the data, the YOLOv9 model is retrained on the *Where's Waldo* images.

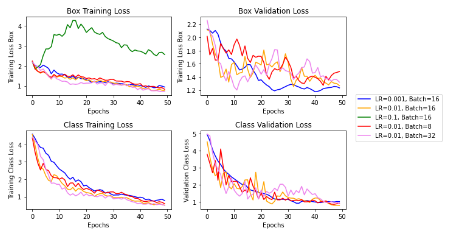

Figure 4. Training and validation loss curves for baseline model.

After performing hyperparameter tuning, we ran the model on the tuned parameters to get the final accuracies for the 1-class and 4-class tasks for the baseline model using pre-existing data. Table 3 shows the obtained accuracies and F_1 score for both detection scenarios. Figure 5 shows the confusion matrix of the model predictions.

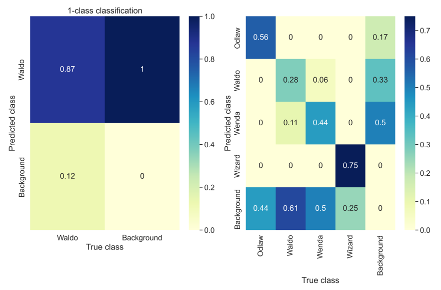

Figure 5. Confusion matrix of Baseline Model on 1-class and 4-class detection of Where's Waldo characters using YOLOv9.

5.4. Our Implementation: WaldoNet

After training the YOLOv9 model on our custom dataset and the tuned parameters, we obtained substantially lower losses (Figure 6) than those of the baseline model (Figure 4).

From the loss curves depicted in Figure 6, the training loss generally decreases for the box during the entire duration of 100 epochs. However, the training classification errors and the validation errors (both classification and box) level off after 20 epochs. In this case the model is not

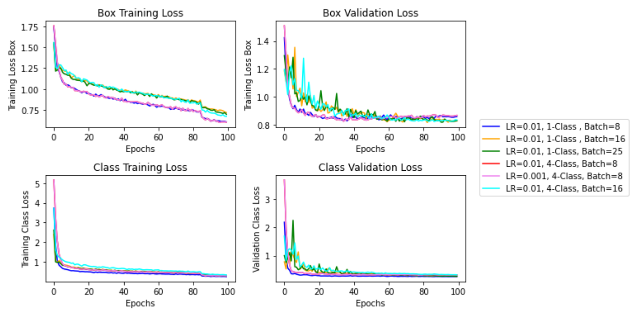

Figure 6. Training and validation loss curves for drawing bounded box around characters and classes for 50 epochs.

overfitting too much, as the training and validation errors level off at around 0.3-0.4 loss for both box and classification losses. If we wanted to further reduce the loss of the model, it would be more prudent to implement the methods described later in section 6. The confusion matrix from WaldoNet validation is shown in Figure 7.

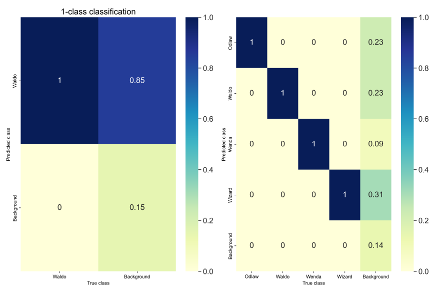

Figure 7. Confusion matrix of WaldoNet Model on 1-class and 4-class detection of Where's Waldo characters using YOLOv9.

	Baseline		WaldoNet	
Classes	Accuracy	F_1 Score	Accuracy	F_1 Score
1-Class detection				
Waldo	43.7%	0.304	54.1%	0.481
4-Class detection				
Overall	40.6%	0.454	82.8%	0.772
Waldo	76.7%	0.647	81.3%	0.897
Wenda	41.8%	0.335	81.3%	0.897
Odlaw	41.9%	0.429	91.7%	0.957
Wizard	100%	0.857	76.3%	0.865

Table 3. Accuracy and F_1 score of baseline and WaldoNet model.

Table 3 shows the obtained accuracies and F_1 score for both detection scenarios. While Figure 7 shows the confusion matrix of the model predictions. Although it appears that the test accuracy of WaldoNet on 4-class detection problem fares better than 1-class detection, this is a little misleading. This is because if we looked at the counts for 1-class detection we will have 201 TP, 6 TN, 0 FN, and 34 FP, which results in a test accuracy of 85.9% for 1-class detection.

5.5. Prediction Results

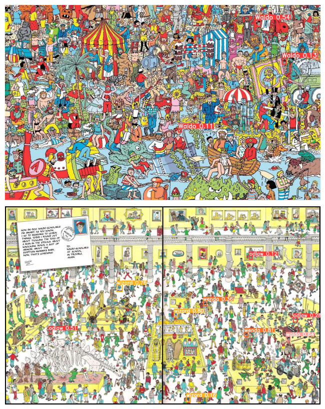

Figure 8. Mini image of detection results. For the full-sized image please refer to Figure 9.

Figure 8 shows the inferences made by WaldoNet for 1-class and 4-class detection problems. The model is able to identify the correct location of all characters, but makes some false identifications. For example, in the top image (1-Class identification), Waldo can be reasonably identified (relatively high enough confidence), while the model makes an incorrect detection by mistaking Wenda as Waldo. It additionally mistakes someone with a face with red objects in their hair as being Waldo. These mistakes make sense, as one can reasonably think that these characters are Waldo at first glance based on the limited information provided by the pixels enclosed within the bounding box.

Similarly, for the bottom image in Figure 8, WaldoNet is able to identify correctly all characters from the *Where's Waldo* series correctly, but it makes similar erroneous classifications. However, most of these errors are made with a much lower confidence than the correct identifications of the characters. In the example image shown, the highest confidence prediction made for each class is consistent with the ground truth.

6. Future Work

- **Limit number of Waldo boxes drawn per image**: From Figure 15 we can see instances where more than one box for Waldo is drawn in a given image. To help reduce these cases, we can potentially modify the model such that the number of bounded boxes that it can draw per image is equal to the number of classes. Given that all true background images were correctly identified as such and that true positive rate for each model is close to 100%, this simple change should ideally lower the false positive rate from approximately 85% to zero.

- **Optimize the proportion of background images**: It is recommended that 0-10% of images in the dataset should be background images help reduce the occurrence of false positive predictions. Since our dataset has 2% background images currently, we could see if raising this amount closer to 10% is beneficial. [6].
- **Find Waldo in real life**: An expansion we may pursue would be to take Waldo's head/body and randomly insert him into real life backgrounds. In this case we can expand the model to finding him when he is beside real life objects (real life background).

7. Discussion

Training the YOLOv9 model on our dataset of synthetic, superimposed images allowed us to substantially improve the true positive detection rate for finding Waldo in the 1-class task and finding all four characters in the 4-class detection task. There was also substantial improvement in the false positive detection rates when compared to the baseline versions. The official documentation for YOLOv9 recommends having a fairly small fraction of unlabeled background images in the dataset in order to minimize the false positive rate, so we followed the recommendation by employing a background fraction of 2%. This went against our own intuition that it would be better to have a similar number of background and character-containing images, but we acquiesced to the expert advice and indeed lowered the false positive rate.

We suspect that our false positive rate is still fairly high (85% for 1-class) with our datasets despite having a very small background image fraction in large part because of how chaotic the images are. If a classifier is being trained to detect a fairly distinct looking object that a person can easily detect against its background, then fewer pure background images are likely necessary. However, *Where's Waldo* is different from the usual object detection tasks, both because it is an example of small object detection where the object to be found is much smaller than the overall image and because the desired objects (Waldo and the other characters) are intentionally challenging to find, as they are scattered within extremely complex backgrounds with lots of similar looking drawn features with similar colors and line weights). Accordingly, we expect that *Where's Waldo* subverts the usual rules of thumb for YOLO and would require a larger than usual fraction of pure background images in order to decrease the false positive rate.

We argue that having a fairly high false positive rate is not actually important for the intended application of assisting a player with the *Where's Waldo* game. In a hypothetical version of *Where's Waldo* with some pages that lacked Waldo and the other characters entirely in order to waste the player's time, it would be important to have a very low false positive rate. However, in real life, every single two-page spread in each book is guaranteed to have Waldo and the other characters present. Because of this, it doesn't matter if the model will detect Waldo when he isn't present, so long as it has a very high true positive rate. Fortunately, this is indeed the case, and we were able to increase the true positive rate of our model substantially by moving from baseline available imagesets to our custom, much larger datasets. For example, and as previously mentioned, our baseline 1-class true positive rate for finding Waldo was 87% and this increased to over 99% with our custom dataset.

While being trained on small, cropped images, our model still works when fed large images of the entire book pages. At this scale, it will tend to detect two or three instances of each class with high enough predicted probability to draw bounding boxes for them. When testing in this way, we found that the highest predicted probability always corresponded with the ground truth, even with book pages that the model was never trained on. In its current state, this acts as a means of greatly narrowing down the search space for a *Where's Waldo* player: rather than looking over the entire complex scene, they only need to look at a few selected spots and have a guarantee of finding the target character at one of those spots, so the goal is still accomplished of making the *Where's Waldo* game much easier and faster. In the future, we could modify how YOLOv9 draws bounding boxes by having it only draw a bounding box around the prediction with the highest estimated probability, as this would ensure that only the correct bounding boxes are drawn and solve the game directly.

It would still be nice, in abstract, to lower the false positive rate, and we suspect we could do this by increasing the resolution of our book images, which are fuzzy, resulting in many faces of Waldo and other characters being pixelated and difficult for a human to recognize the features of. We were surprised how high the true positive rate was despite this, as it was close to 100% for each character anyway.

8. Conclusion

Our implementation using YOLOv9 and our large, custom dataset achieved a substantial improvement in true positive validation and test accuracy over the baseline, publicly available *Where's Waldo* datasets. Where for the implementation of WaldoNet leads to 54.1% and 82.8% in 1-class and 4-class detection problems.

References

[1] A. Bilogur. Where's waldo, Oct 2017.

[2] C. Cao, B. Wang, W. Zhang, X. Zeng, X. Yan, Z. Feng, Y. Liu, and Z. Wu. An improved faster r-cnn for small object detection. *Ieee Access*, 7:106838–106846, 2019.

[3] H. ENCE428. Wally solver 2 dataset. `https://universe.roboflow.com/henry-ence428/wally-solver-2`, apr 2023. visited on 2024-05-16.

[4] R. Girshick. Fast r-cnn. In *Proceedings of the IEEE international conference on computer vision*, pages 1440–1448, 2015.

[5] R. Girshick, J. Donahue, T. Darrell, and J. Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 580–587, 2014.

[6] S. Goswami. False detection (positives and negatives) in object detection. *arXiv preprint arXiv:2008.06986*, 2020.

[7] L. Harde. Whereswaldo-yolov5 dataset. `https://universe.roboflow.com/lasse-harde/whereswaldo-yolov5`, nov 2021. visited on 2024-05-16.

[8] S.-C. Hsu, C.-L. Huang, and C.-H. Chuang. Vehicle detection using simplified fast r-cnn. In *2018 International Workshop on Advanced Image Technology (IWAIT)*, pages 1–3. IEEE, 2018.

[9] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie. Feature pyramid networks for object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2117–2125, 2017.

[10] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár. Focal loss for dense object detection. In *Proceedings of the IEEE international conference on computer vision*, pages 2980–2988, 2017.

[11] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg. Ssd: Single shot multibox detector. In *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part I 14*, pages 21–37. Springer, 2016.

[12] Y. Liu, P. Sun, N. Wergeles, and Y. Shang. A survey and performance evaluation of deep learning methods for small object detection. *Expert Systems with Applications*, 172:114602, 2021.

[13] N. Phuong. Find waldo with yolov2. *https://medium.com/@nataliephuong/about*, 2020.

[14] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 779–788, 2016.

[15] J. Redmond and A. F. Santosh Divvala, Ross Girschick. You only look once: Unified, real-time object detection. *29th IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016.

[16] S. Ren, K. He, R. Girshick, and J. Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *Advances in neural information processing systems*, 28, 2015.

[17] G. Saranya, D. Sarkar, S. Ghosh, L. Basu, K. Kumaran, and N. Ananthi. Face mask detection using cnn. In *2021 10th IEEE International Conference on Communication Systems and Network Technologies (CSNT)*, pages 426–431. IEEE, 2021.

[18] K. Tong, Y. Wu, and F. Zhou. Recent advances in small object detection based on deep learning: A review. *Image and Vision Computing*, 97:103910, 2020.

[19] J. R. Uijlings, K. E. Van De Sande, T. Gevers, and A. W. Smeulders. Selective search for object recognition. *International journal of computer vision*, 104:154–171, 2013.

[20] Ultralytics. Tips for best training results, 2024.

[21] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao. Yolov7: Trainable bag-of-freebies sets new state-of-the-art for real-time object detectors. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 7464–7475, 2023.

[22] C.-Y. Wang and H.-Y. M. L. I-Hau Yeh. Yolov9: Learning what you want to learn using programmable gradient information. *arXiv*, 2024.

[23] C.-Y. Wang, I.-H. Yeh, and H.-Y. M. Liao. Yolov9: Learning what you want to learn using programmable gradient information. *arXiv preprint arXiv:2402.13616*, 2024.

[24] N. Workspace. Wally dataset. `https://universe.roboflow.com/new-workspace-qplfj/wally-rbk6w`, mar 2022. visited on 2024-05-16.

[25] W. K. Yiu. Yolov9. `https://github.com/WongKinYiu/yolov9`, Mar 2024. visited on 2024-05-16.

[26] T. Zhou, Z. Li, and C. Zhang. Enhance the recognition ability to occlusions and small objects with robust faster r-cnn. *International Journal of Machine Learning and Cybernetics*, 10:3155–3166, 2019.

9. Acknowledgments

We would like to acknowledge the teaching team of CS 231N and our project mentor Tiange Xiang for his help and advice during the development of WaldoNet. Additionally, we used the publicly available YOLOv9 model that is obtainable through `https://github.com/WongKinYiu/yolov9`.

10. Contributions

Author Information: Barney Miao and Andrew Lesh. Barney did the literature review, trained the model, evaluated the performance, and wrote the paper. Andrew implemented the described methods to generate the data for 1-class and 4-class detection and wrote the paper,

11. Appendix

This section contains additional information relating to the development and implementation of WaldoNet. Please see the captions of each image for more information.

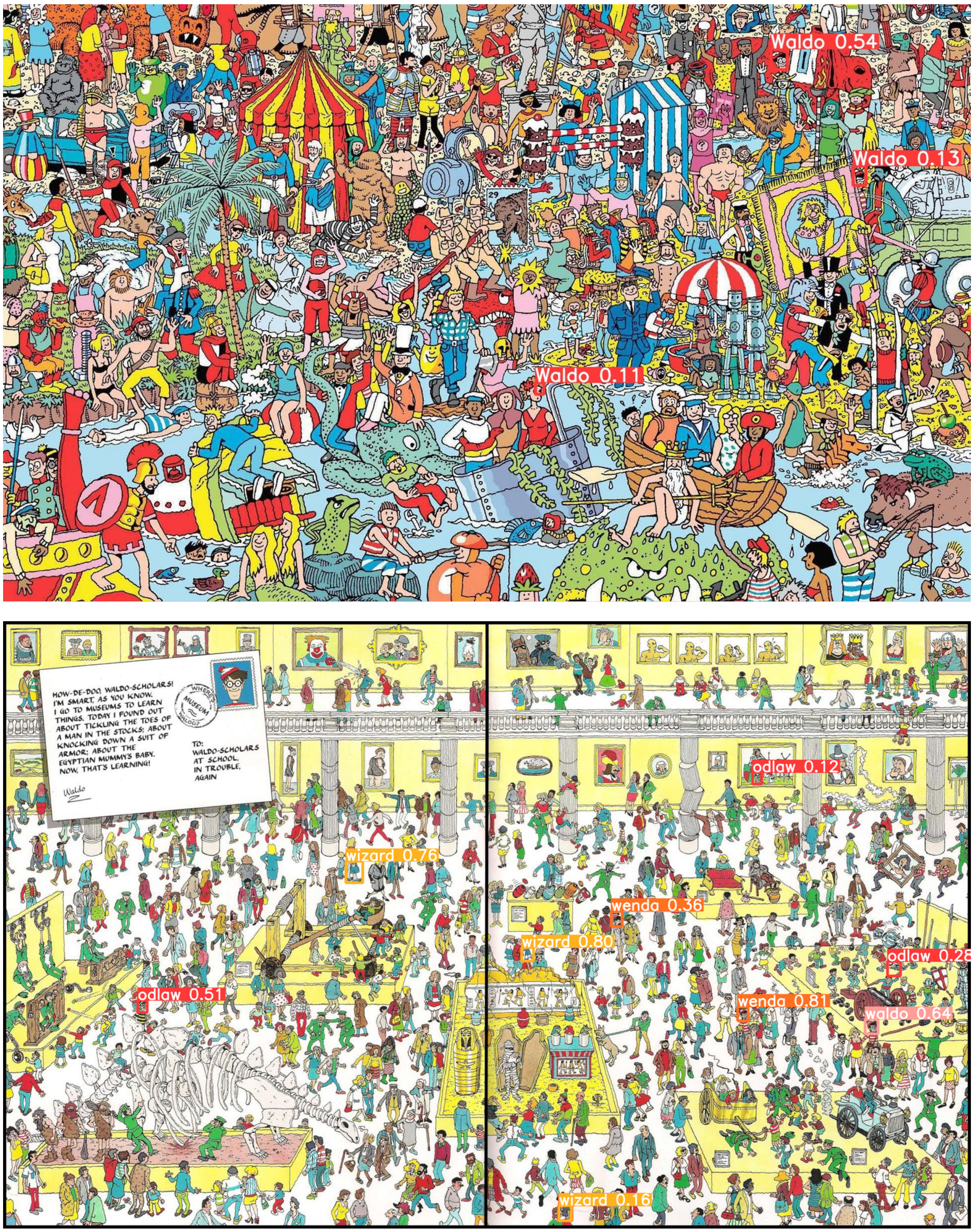

Figure 9. Full-sized image of WaldoNet detection on 1-class and 4-class scenarios

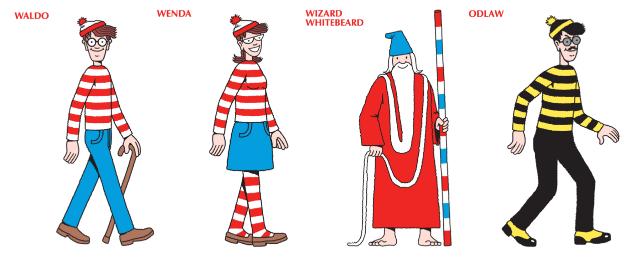

Figure 10. The primary characters for players to find in the *Where's Waldo* books.

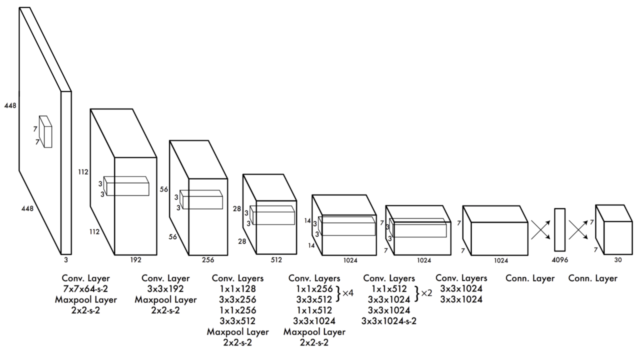

Figure 11. Architecture of the original YOLO model.[15]

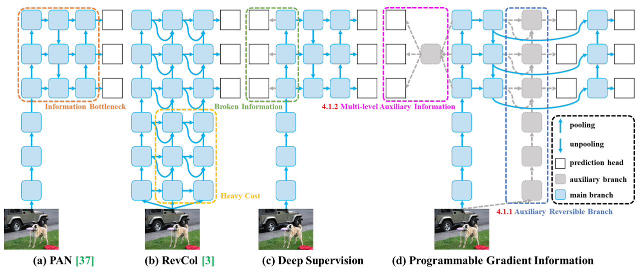

Figure 12. PGI (auxiliary supervision method used in YOLOv9) and related methods[22].

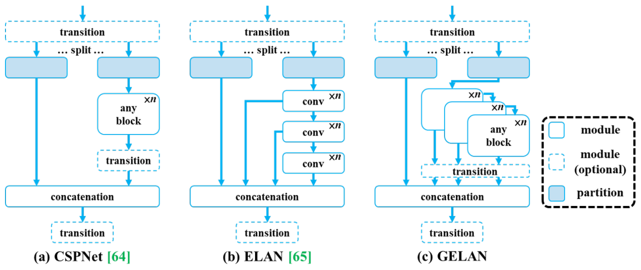

Figure 13. GELAN architecture and its inspirations, CSPNet and ELAN[22].

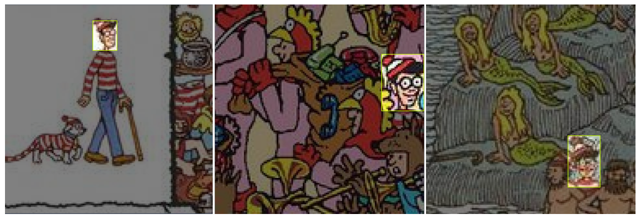

Figure 14. Examples of Waldo images labelled on Roboflow.

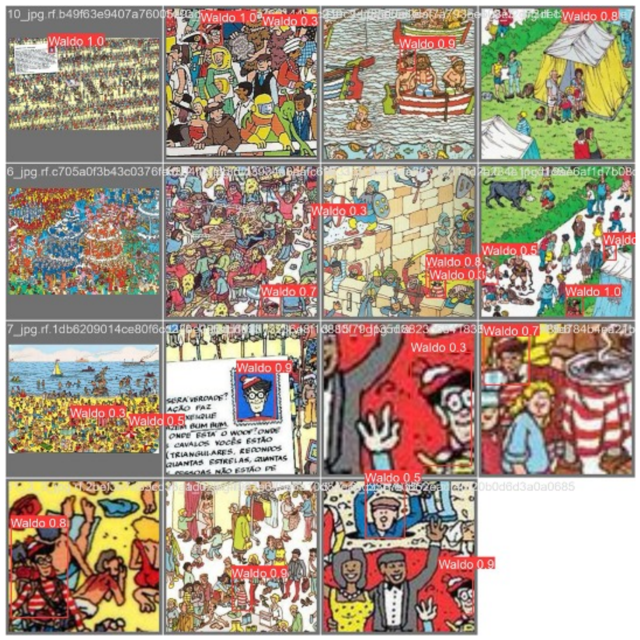

Figure 15. Additional examples of 1-class Waldo detection from Baseline YOLOv9 model.

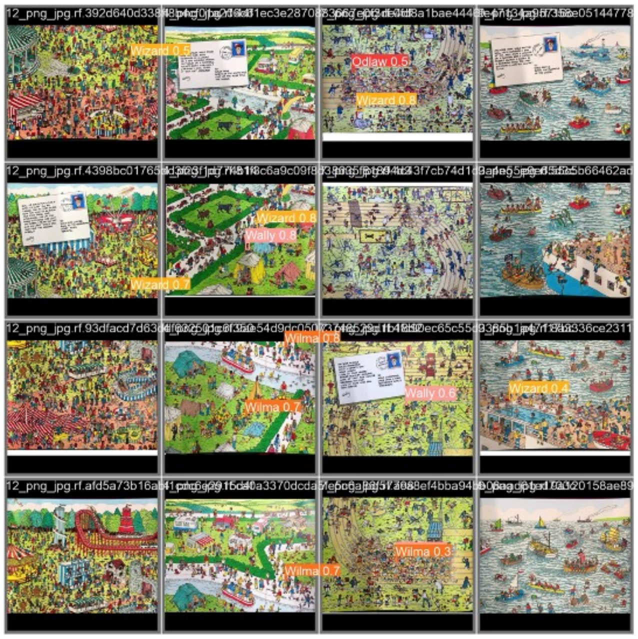
Figure 16. Example of 4-class Waldo detection from Baseline YOLOv9 model.

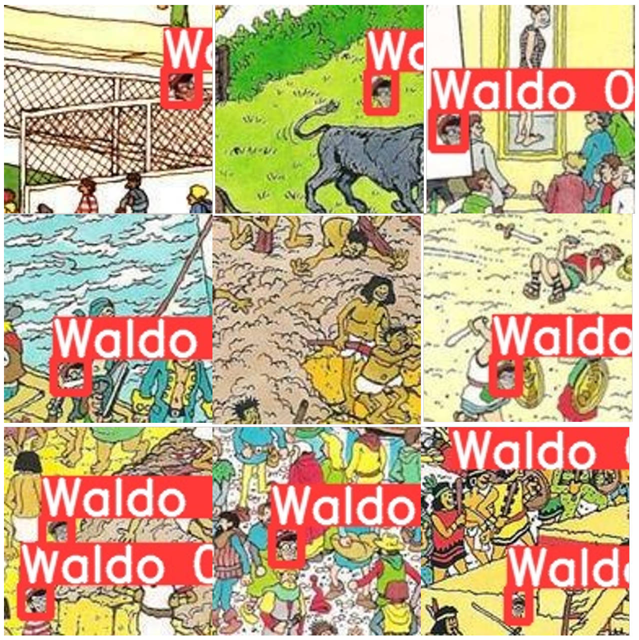

Figure 17. Additional examples of 1-class Waldo detection from WaldoNet YOLOv9 model.

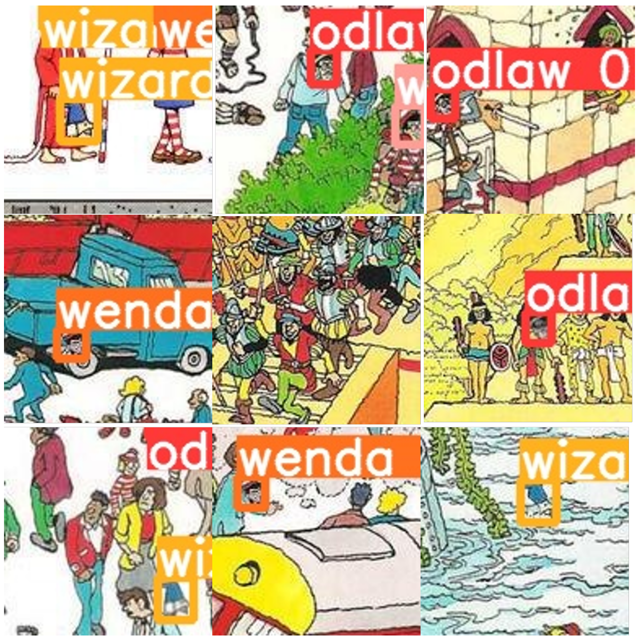

Figure 18. Example of 4-class Waldo detection from WaldoNet YOLOv9 model.

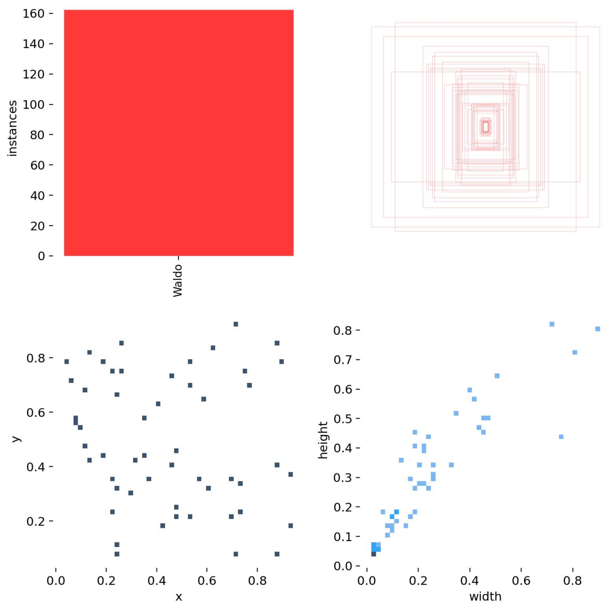

Figure 19. Distribution of bounded boxes for baseline model. In this case, the data obtained from Roboflow will typically vary widely and are inconsistent. For example, the boxes in some images may cover just the face of the characters, while the boxes in some other images may cover the entire body, resulting in the wide distribution in the width and heights of the drawn boxes.

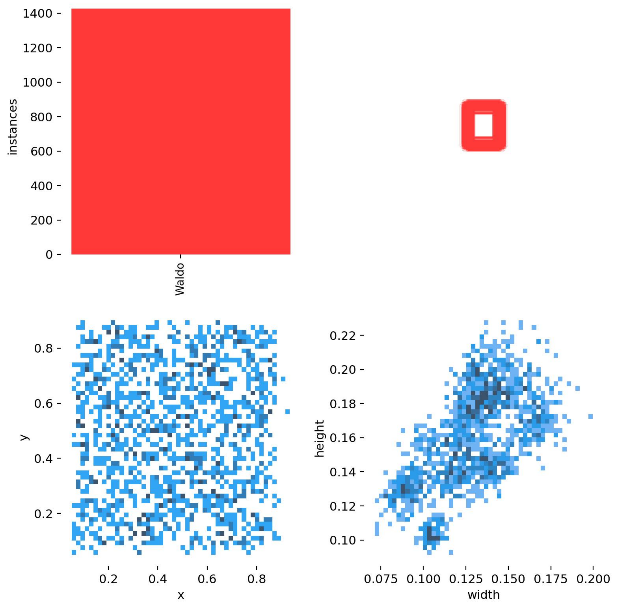

Figure 20. Distribution of bounded boxes for WaldoNet model with 1-class. In this case, the generated data will be much more consistent in comparison to the data obtained from Roboflow, as the boxes now only cover the faces of the characters.

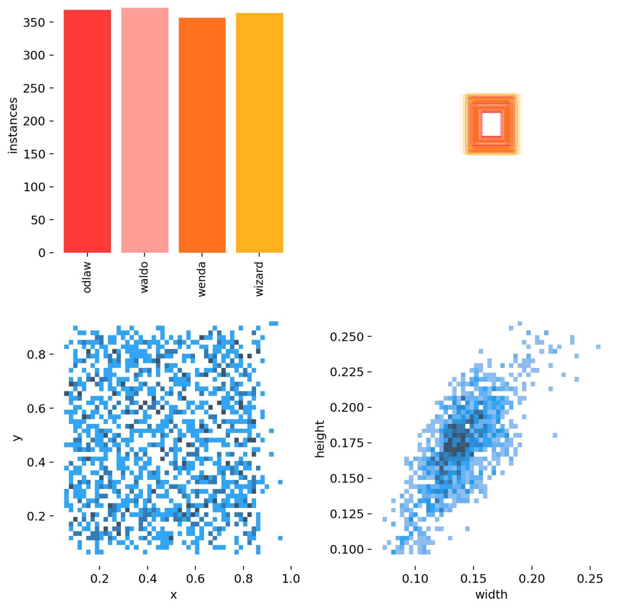

Figure 21. Distribution of bounded boxes for WaldoNet model with 4-class. In this case, the generated data will be much more consistent in comparison to the data obtained from Roboflow, as the boxes now only cover the faces of the characters.

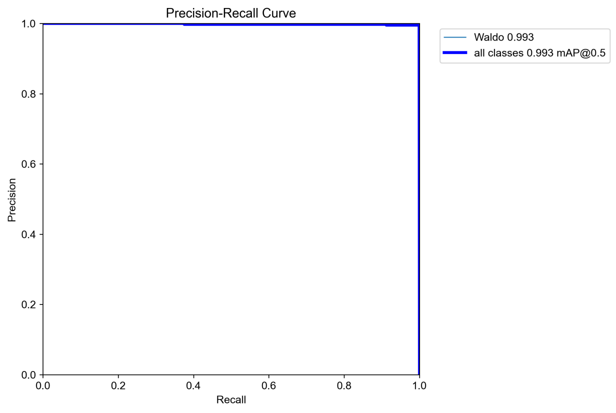

Figure 22. Precision-Recall curve for 1-class detection using WaldoNet.

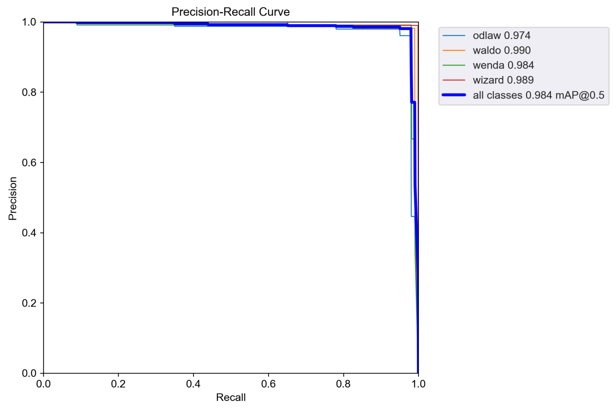

Figure 23. Precision-Recall curve for 4-class detection using WaldoNet.

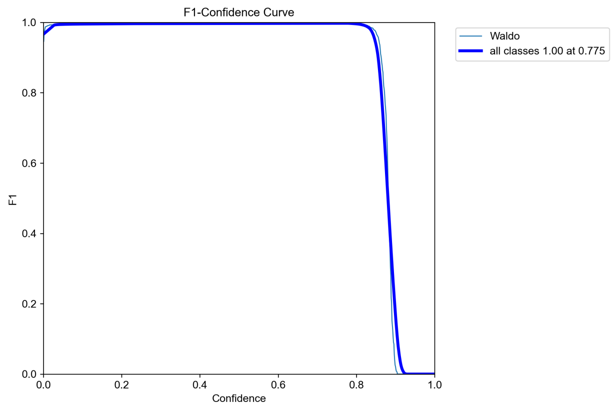

Figure 24. F1 curve for 1-class detection using WaldoNet.

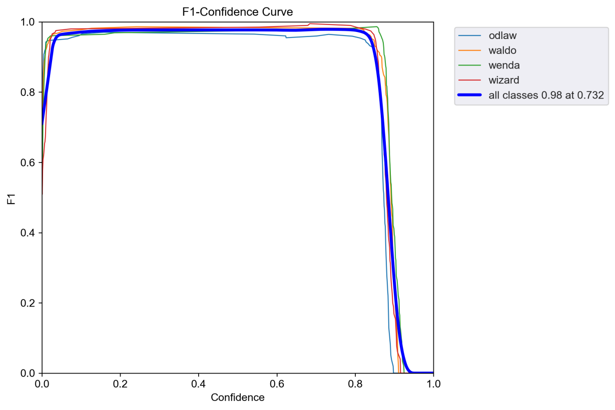

Figure 25. F1 curve for 4-class detection using WaldoNet.