Pedestrian Traffic Assistant

Gonçalo Silva
DETI
Universidade de Aveiro
Aveiro, Portugal
goncalolsilva@ua.pt

Samuel Teixeira
DETI
Universidade de Aveiro
Aveiro, Portugal
steixeira@ua.pt

***Abstract*—Pedestrians are among the most vulnerable road users, facing heightened risks due to their lack of protection in traffic. These risks are further exacerbated by impairments that affect their visual, motor, or auditory abilities, making navigation in traffic more challenging. This highlights the need for innovative approaches to enhance pedestrian safety. In this paper, we present a Computer Vision (CV) solution for the detection of crosswalks and traffic light signals (red and green) that could be applied to improve the condition of visually impaired people. To accomplish the target, we selected a You Only Look Once (YOLO)11s model, that achieved an F1 score of 97 at 32.2% confidence, precision of 97.4% and recall of 96%. This work is meant to be a framework to apply in edge computing for practical purposes.**

***Index Terms*—YOLO, Object Detection, Computer Vision, Machine Learning, Deep Neural Networks, Deep Learning, Neural Networks,**

I. Introduction

A pedestrian can be described as a person who travels on foot or using any mobility aid device, specifically on a road or pavement close to where other vehicles (with or without motor) travel as well [1]. Pedestrians by themselves can be vulnerable to accidents with vehicles and there are various factors for this. For starters, people are unpredictable. A paper of the National Library of Medicine accounted for one third of its 10543 observed pedestrians to be distracted while crossing roadways, mainly due to technologies like headphones or phone texting [2]. A distracted pedestrian is less likely to look at traffic when going into a roadway, which imposes the risk of accident.

But other issues can make Pedestrians prone to collisions. Specifically, visual impairments, such as blindness. In Australia, a study reveals that 1 in 12 pedestrians living with blindness were hit by a motor vehicle or a cyclist [3]. According to the Insurance Institute for Highway Safety, 7522 pedestrians died on traffic accidents in 2022, accounting for 18% of all motor vehicle crash deaths [4]. Those are considerable measurement if we take into account, the amount of blind people across the World. The International Agency for the Prevention of Blindness counts 43 million people to be blind worldwide, with approximately 260 million having moderate to severe sight deficiency [5].

The aftermath of pedestrian accidents can range widely from minor injuries to spinal cord injuries, head injuries or internal injuries, all of which can pose life threatening situations, disability, psychological stress and elevated financial bills [6].

With all of this data, we can conclude that pedestrians are prone to traffic collisions due to their nature or due to health conditions, such as blindness, and that the consequences of this collisions can mark the life or even lead to the death of a pedestrian. Considering this, a system which could aid on the correct identification of a safe or potentially unsafe road crossing situation, would be a great step towards reducing accidents and improve road safety.

This paper is aimed at the development of a Machine Learning (ML) model for real-time detection and classification of crosswalks and pedestrian traffic lights (red or green lights). This work is a foundation, so that other people can use our model to implement practical solutions for this problem.

II. State of the Art/Related Work

Object detection is a very important field in Computer Vision, allowing the detection of all kinds of objects, animals and people (faces, cars, animals, health conditions, etc.). In the present, object detection is divided into two main categories, Traditional Machine Learning Methods and Deep Leaning. There exists roughly three types of Machine Learning Methods, which are Generalized Hough Transform, Harris Corner Detection and SIFT (Scale-Invariant Feature Transform), defined by geometric feature extraction, corner detection and local feature description. The three of them have some big disadvantages, being the first two sensitive to geometric features in the image or changes, like size, rotation and gray values, whilst the last one addresses rotation and scale problems, it still suffers from the remaining drawbacks [7].

Moving from traditional Machine Leaning methods, Deep learning came as Holy Grail to object detection, since it didn't require a highly effective feature engineering and was more efficient for training large datasets. With the rapid advancements in the field, specially in the last ten-fifteen years, vast techniques have been suggested and proposed for Object Detection with Deep Learning. With the most common being based in Region Based Convolutional Neural Networks (R-CNN). Quickly newer and faster method were invented, like Fast R-CNN, Faster R-CNN, YOLO and Mask R-CNN[7].

A. Feature Engineering

Traditional feature engineering for object detection are based on analyzing specific traits in the image, by reducing

unnecessary details to focus on key features such as edges, corners, and textures. Techniques like the Generalized Hough Transform simplified complex shapes into mathematical models, enabling robust pattern recognition under partial occlusion. Harris Corner Detection identified points of interest where intensity varied significantly, making them reliable markers for object localization and motion tracking. The Scale-Invariant Feature Transform (SIFT) further advanced this approach by introducing descriptors invariant to scale and rotation, enhancing robustness against transformations and background clutter [8]. These methods relied heavily on predefined heuristics, making their use extremely situational, since each use case needs to be reviewed, in order to identify the correct features, making their implementation extremely costly. For instance, if aspects like image angle, coloration, among others change, the output can change dramatically.

B. CNN

With the introduction of R-CNN in 2014 [9], object detection transitioned from a fragmented, target-specific detection paradigm to a unified framework where Convolutional Neural Networks (CNN)s directly learned features for object localization and classification. R-CNN employs a selective search algorithm to propose regions of interest, which are then processed independently through CNNs for feature extraction and classification, significantly improving detection accuracy but at a high computational cost.

The evolution continued with Fast R-CNN, which addressed R-CNN's inefficiencies by integrating region-of-interest pooling into the CNN pipeline, allowing the entire image to be processed in a single forward pass. Faster R-CNN further optimized this by introducing (Region Proposal Networks (RPN)s) to generate proposals directly within the network, establishing a truly end-to-end architecture. This innovation not only reduced the computational load but also allowed real-time performance to become feasible in object detection tasks [10].

C. Transformers

Transformer models have revolutionized machine learning, providing a versatile and powerful architecture for a wide range of tasks. Introduced by Vaswani et al. [11], the Transformer architecture is based on the concept of self-attention, which enables the model to weigh the importance of different parts of the input sequence when making predictions. Unlike traditional CNNs, Transformers excel in capturing long-range dependencies, making them particularly effective for sequential and spatial data.

Transformers have been widely adopted in Natural Language Processing (NLP), where they form the backbone of state-of-the-art models like BERT [12] and GPT [13]. In recent years, their application has expanded to computer vision tasks, with models such as Vision Transformer (ViT) demonstrating competitive performance in image classification by treating images as sequences of patches [14].

Our interest in transformer networks came by NVIDIA's transition from a CNN-based solution in its acDLSS 3 Frame Generation system to a transformer-based architecture in acDLSS 4 [15]. Early benchmarks and testing have highlighted significant perceived improvements in both visual quality and performance. These enhancements are particularly evident in sharper image details and smoother frame generation. The success of this transition can be attributed to transformers' superior ability to handle context information, allowing them to have an advantage in spatial and temporal correlations in sequential data, which is critical for Frame Generation [16].

D. YOLO

YOLO was proposed in 2016 [17] as a unified, real-time object detection algorithm that simplified the traditional multi-stage pipeline of object detection into a single-shot process [18]. Unlike R-CNN, which breaks down detection into a sequence of steps involving region selection, feature extraction, and classification, YOLO directly predicts bounding boxes and class probabilities from image pixels using a single CNN. This regression-based approach contrasts with the proposal-classification paradigm of R-CNN, making YOLO much faster and more suitable for real-time applications.

YOLO divides the input image into a grid, where each grid cell predicts four values for the bounding box (center coordinates, width, and height), a confidence score indicating how likely it is that an object is present, and the probabilities for each class. This unified framework allows YOLO to incorporate global information from the entire image when making predictions, reducing errors caused by background clutter and making it faster than traditional detection systems that rely on region proposals [7] [18].

The loss function used in YOLO treats small and large bounding boxes equally, which can lead to inaccurate localization, especially for smaller objects. Additionally, when multiple objects appear in a single grid cell, the algorithm struggles with recall, often missing objects or misclassifying them because it assumes that all objects in the same cell belong to the same class (it's worth mentioning that more recent YOLO versions, such as v11n, are adapting solutions for problems like overlapping and small objects). Despite these drawbacks, YOLO remains a top choice for real-time detection, where speed is crucial, even though its accuracy still lags behind methods that use region proposal networks, such as Faster R-CNN [10]. In Table I, there are present all the usable YOLOv11 pre-trained models, with their respective parameters and benchmark scores.

E. DEtection TRansformer (DETR) and Real-Time Detection Transformer (RT-DETR)

DETR, introduced by Carion et al. [20], was a groundbreaking advancement in the field of object detection. Unlike traditional approaches, such as Faster R-CNN or YOLO, DETR eliminates the need for region proposal networks by employing a transformer-based architecture that directly predicts object bounding boxes and class labels. Its end-to-end

Model	mAPval 50-95	Speed CPU ONNX (ms)	Speed T4 TensorRT10 (ms)	Params (M)	FLOPs (B)
YOLO11n	39.5	56.1 ± 0.8	1.5 ± 0.0	2.6	6.5
YOLO11s	47.0	90.0 ± 1.2	2.5 ± 0.0	9.4	21.5
YOLO11m	51.5	183.2 ± 2.0	4.7 ± 0.1	20.1	68.0
YOLO11l	53.4	238.6 ± 1.4	6.2 ± 0.1	25.3	86.9
YOLO11x	54.7	462.8 ± 6.7	11.3 ± 0.2	56.9	194.9

TABLE I: Performance comparison of YOLO models, for images with size of 640x640 pixels [19]

design simplifies the detection pipeline and achieves high accuracy on benchmark datasets. However, DETR's reliance on global attention and iterative refinement leads to slower convergence and higher computational demands compared to its CNN-based counterparts.

To address these limitations, RT-DETR was proposed by Zhao et al. [21] as an evolution, significantly improving its performance for real-time applications. RT-DETR, while maintaining or even exceeding the detection accuracy of YOLO models. This milestone establishes a new standard for object detection, demonstrating that transformer-based architectures can outperform CNN-based models not only in accuracy but also in computational efficiency, a domain traditionally dominated by YOLO [21].

Building upon this foundation, RT-DETRv2 introduces further enhancements to make DETR even more competitive for real-time applications. This version integrates a "bag-of-freebies" approach, which includes optimizations like dynamic query selection, multi-scale feature aggregation, and lightweight positional encodings. These improvements address the bottlenecks of earlier iterations, enhancing detection speed and precision without significantly increasing computational costs [22]. RT-DETRv2 establishes itself as a robust alternative for scenarios demanding low-latency object detection, setting a new benchmark in the transformer-based detection domain. Similar to YOLO, RT-DETRv2 also offers multiple models, with various parameters, size and performance. Their comparison is present in Table II.

Model	APval	AP50val	#Params (M)	FPS
RT-DETRv2-S	48.1 (+1.6)	65.1	20	217
RT-DETRv2-M*	49.9 (+1.0)	67.5	31	161
RT-DETRv2-M	51.9 (+0.6)	69.9	36	145
RT-DETRv2-L	53.4 (+0.3)	71.6	42	108
RT-DETRv2-X	54.3	72.8 (+0.1)	76	74

TABLE II: Performance comparison of RT-DETRv2 models for images with size of 640x640 pixels [23]

F. Our Choice

Based on the fact that we are pursuing a real-time, reliable system, only YOLO and RT-DETR or RT-DETRv2 meet the requirements of our application. We are mostly interested in obtaining a balance of speed, accuracy, and adaptability to real-time detection scenarios.

G. Related Work

In our research for related work, we could not find any model that was made for detecting both pedestrian traffic lights and crosswalks. Out of all the work, 2 works were selected:

- CDNet (Crosswalk Detection Network) [24]
- MASK R-CNN for Pedestrian Crosswalk Detection and Instance Segmentation [25]

The first work has the concept of creating a system where crosswalks are detected and decisions over vehicle crossing behavior are taken, so it is more focused on drivers than pedestrians (the system is implemented in Jetson Nano's that look at images from vehicle's cameras). The authors obtained an average F1 score of 94.83% with 33.1 FPS imaging, and allegedly 98% on optimal weather conditions.

The second work was a pedestrian crosswalk detection model using R-CNN. The purpose of the work is not well defined, though it is inferred that this work is more driver centered as well and is more of a proposal to be used by people on edge computing (which is similar to what we've done). The authors detected more than 97% of the labels of pedestrians walking on crosswalks.

III. Dataset

The dataset used, was uploaded by chanyoung to Roboflow [26], an Open Source Computer Vision platform. This dataset is composed of 1960 images, divided in three classes, *crosswalk*, representing a regular zebra crosswalk, *greenlight* and *redlight* which refer to regular pedestrian green and red lights, respectively. The distribution of this dataset is present in Table III

Use	nº images	%
Train	1374	70
Valid	389	20
Test	197	10

TABLE III: Dataset split: number of images and percentage for each subset [26]

The dataset has been reviewed and contains images in multiple angles and conditions, as well as quality (essentially, following YOLO recommendations for datasets [27] and preventing data leakage [28]). An example of 16 random images taken from the dataset, complete with their respective annotations, can be found in Figure 1

The training of the dataset, followed the already established values, of the previous work done, which include the following techniques [29]:

-

Fig. 1: Examples of dataset images, complete with their annotated labels [26]

1) **hsv_h: 0.015**
   - **Impact**: Adjusts the hue of the image by a factor of 0.015. Can help the model generalize better by making it invariant to slight color changes.
2) **hsv_s: 0.7**
   - **Impact**: Adjusts the saturation of the image by a factor of 0.7, helping the model to handle variations in color intensity.
3) **hsv_v: 0.4**
   - **Impact**: Adjusts the value (brightness) of the image by a factor of 0.4, which can help the model generalize to different lighting conditions.
4) **degrees: 0.1**
   - **Impact**: Translates the image by up to 10% of its dimensions, potentially making the model more invariant to small translations of objects within the image.
5) **scale: 0.5**
   - **Impact**: Scales the image by a factor of 0.5, helping it generalize to objects of different sizes.
6) **fliplr: 0.5**
   - **Impact**: Horizontally flips the image with a probability of 0.5, helping the model learn to recognize objects regardless of their horizontal orientation.
7) **mosaic: 1.0**
   - **Impact**: Uses mosaic augmentation, which combines four training images into one. This can help the model learn to detect objects in a more complex and varied context.
8) **erasing: 0.4**
   - **Impact**: Randomly erases parts of the image with a probability of 0.4, which can help the model become more robust to occlusions.

This augmentations help the model to work in different circumstances, not necessarily represented in the training data. Due to the dataset being of relative good size, data augmentation can be very useful. Contrary to augmentation in other deep learning models, which can add too much noise and hinder the model's performance, the ones provided here for YOLO are relatively simple, straightforward and are based on the ones suggested by Ultralytics [30] [31].

IV. Training

In this section, we'll be describing our training process, the models and hyper-parameters used. We also describe the results of this process based on the models performance metrics, which are generated based on data from the three stages, training, validation and testing.

The following metrics can be obtained during training and validation stages:

- **box_loss**: The bounding box regression loss, measures how accurately predicted bounding boxes align with ground truth boxes.
- **cls_loss**: The classification loss, evaluates the model's ability to correctly classify objects.
- **dfl_loss**: The Distribution Focal Loss, is used for improving regression tasks by focusing on key features.

The next metrics can be obtained during the validation/testing stage:

- **Intersection over Union (IoU)**: Measures the overlap between predicted bounding boxes and ground truth boxes to evaluate localization quality.
- **Precision(B)**: Precision is calculated across validation/testing batches, indicates the proportion of true positive predictions among all positive predictions.
- **Recall(B)**: Recall is calculated across validation/testing batches, measuring the proportion of true positive predictions among all actual positives.
- **Average-Precision (AP)**: The precision averaged over different recall levels
- **mean Average-Precision (mAP)**: The mean of average precision scores across all classes
- **mAP50(B)**: Mean Average Precision calculated at a fixed IoU threshold of 0.50 across validation/testing batches.
- **mAP50-95(B)**: Mean Average Precision calculated across multiple IoU thresholds (from 0.50 to 0.95, in increments of 0.05) across validation/testing batches, providing a comprehensive performance measure.
- **F1 Score**: The harmonic mean of precision and recall, offering a balanced evaluation of detection performance.
- **Confusion Matrix**: Presents the classification performance, comparing the models predictions (True Positives, True Negatives, False Positives or False Negatives).

A. Training time

Epochs represent complete passes through the training dataset, during which the model adjusts its weights to minimize errors. While fewer epochs can lead to underfitting, excessive epochs risk overfitting. Techniques like early stopping help maintain a balance [32]. Using our selected dataset of 1374 training images and a batch size of 16, each epoch involves 86 iterations. This means that for a train with 10 epochs, a total of 860 iterations will be made.

Training time depends mostly on the model, number of epochs and batch size. In Figure 2, we present the training time for all models, with the label of each entry, corresponding, from left to right, to the model, optimizer, learning rate, weight decay, and number of epochs. This results were gathered by training, using an Nvidia RTX 4000 Graphical Processing Unit (GPU) (20 GB Video Random-Access Memory (VRAM)).

B. YOLO Models

A wide variety of YOLO models were trained. Since our computational power was considerably high, we played a lot

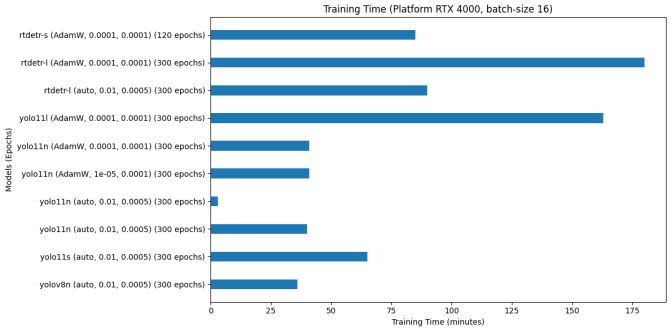


Fig. 2: Training time, based on parameters, model and hardware

with different models, like YOLO V11n, V11s, V11l and 8nu. All of these very popular and with very powerful tools, such as optimizers. Optimizers in YOLO are algorithms that adjust the model's weights during training in order to minimize the cost function. They are very important since they influence how efficiently the model learns from data. Some of the best available optimizers in YOLO are Adam and AdamW.

1) Adam and AdamW Optimizers: Adam (Adaptive Moment Estimation) is an optimization algorithm designed to improve gradient descent by adapting the step size for each parameter based on how the gradients change. It calculates two moving averages: the first moment ***m(t)*** (mean of gradients) and the second moment ***v(t)*** (uncentered variance of gradients), which are updated at each step. By combining these averages, Adam adjusts the learning rate dynamically, taking larger steps when gradients are stable and smaller steps when gradients vary significantly. The adaptability of Adam to the gradient shifts makes it a great choice.

$$
\begin{aligned}
x(t) &= x(t-1) - \alpha \nabla f \\
m(t) &= \beta\_1 m(t-1) + (1-\beta\_1) g(t) \\
v(t) &= \beta\_2 v(t-1) + (1-\beta\_2) g(t)^2 \\
x(t) &= x(t-1) - \alpha \cdot \frac{m(t)}{\sqrt{v(t)} + \epsilon}
\end{aligned}
$$

Fig. 3: Equations for adjusting model's weights, first and second moments and step size

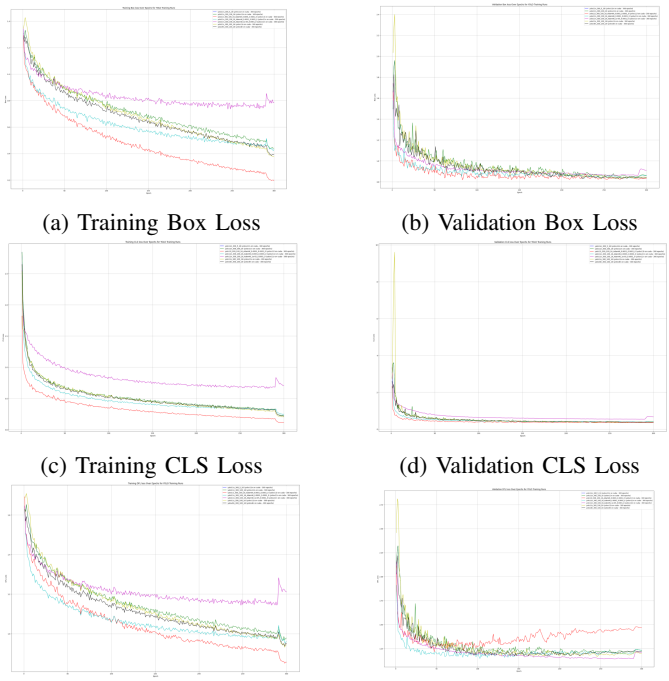
AdamW improves the Adam optimizer by fixing how weight decay is applied. In Adam, the regularization term is scaled by the gradient dynamics, making it less effective. AdamW separates weight decay from gradient updates, applying it directly to the weights after computing the step size. This ensures consistent regularization, leading to better generalization and making AdamW a powerful tool for modern deep learning tasks [33].

For this reason and corroborated by RT-DETR using AdamW as the default configurated optimizer [34], it was chosen as an optimizer for some of our models.

2) Batch Size: The batch size can be described as the number of training examples processed in one step of training. Bigger batch sizes provides a smoother convergence, while needing powerful computational resources and careful learning rate tuning. Since none of these characteristics were a problem, our models trained with a big batch size of 16 [35].

Those were the essential attributes manipulated on our training. Apart from those, the number of epochs was kept at 300 for being a reasonable amount of traversals through the entire dataset for the models to reach a convergence state where meaningful patterns were learned without simply memorizing the training data.

The following data shows the metrics obtained throughout the process of training, validation and testing of the YOLO models.



(a) Training Box Loss (b) Validation Box Loss

(c) Training CLS Loss (d) Validation CLS Loss

(e) Training DFL Loss (f) Validation DFL Loss

Fig. 4: Loss Functions across Trained Models

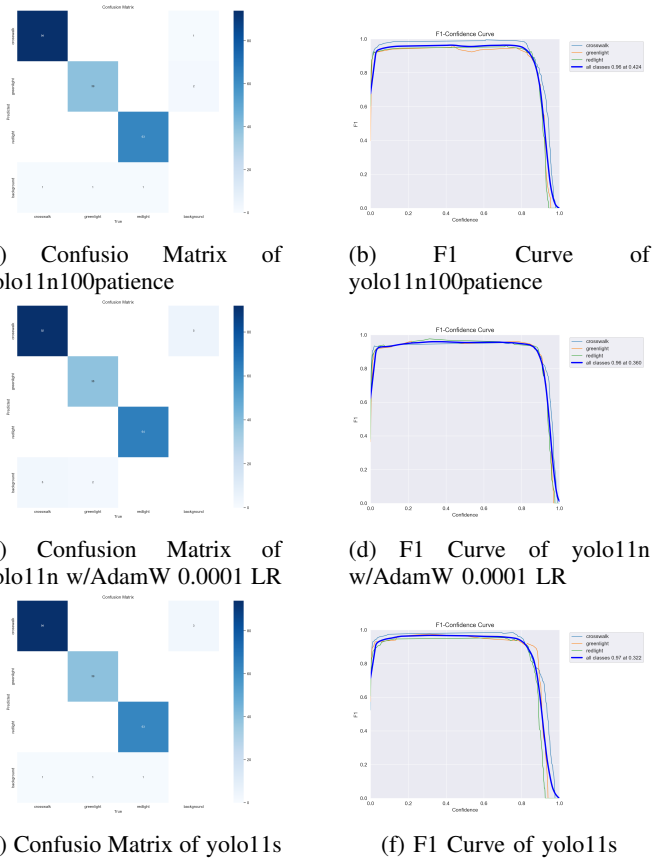
All models had similar levels of validation loss function values, except for some exceptions. The model with learning rate of 1e-05 had slightly bigger losses, and the model that used YOLOv11l had an interesting abnormal increase in the Validation DFL loss, which looks like a past convergence state/overfitting, while the training DFL loss of this model was the lowest of all. The other losses of this model were also considerably different between training and validation. Overall, there were some increases between the training and validation and these might be explained with reasons such as the overfitting, lack of data augmentation, training-validation mismatch in labeling, or even an inappropriate dataset. Our guess is that the dataset was too small for training with so many epochs, because all trainings get to a point around half

of the training were the slope of the functions gets near 0. Perhaps our small dataset didn't need that many epochs.

But, on the other side,few models, such as the YOLOv11l, with it's training/validation loss differences, show that the number of epochs was giving better results, but something made validation losses go up. And for that reason, it is also plausible to assume that there was something wrong going on between the training and the validation dataset, whether the labels had some considerable changes between these 2 we are not sure, it's just an assumption.

Model	TP	FP	FN	F1 Curve
yolo11l	197	6	11	0.95 at 0.32
yolo11n5patience	217	72	12	0.90 at 0.6
yolo11n100patience	196	3	3	0.96 at 0.42
yolo11nAdamW0.0001	191	5	5	0.96 at 0.36
yolo11nAdamW1e-05	196	22	2	0.96 at 0.64
yolo11s	196	3	3	0.97 at 0.32
yolo8n	195	12	3	0.96 at 0.68

TABLE IV: Test Phase Confusion Matrix(TP, FP, FN) and F1 Curve Data Across Models



(a) Confusio Matrix of yolo11n100patience

(b) F1 Curve of yolo11n100patience

(c) Confusion Matrix of yolo11n w/AdamW 0.0001 LR

(d) F1 Curve of yolo11n w/AdamW 0.0001 LR

(e) Confusio Matrix of yolo11s

(f) F1 Curve of yolo11s

Fig. 5: Test Phase Confusion Matrix and F1 Curve of best selected models

Out of all models, some were selected as "protagonists". YOLOv11n with 100 Patience, YOLOv11n with AdamW and 0.0001 learning rate and YOLOv11s models. This models show great confusion matrixes with minimal False Positives (FP) and False Negatives (FN). The results of the confusion matrixes makes it possible to reach the conclusion that the models perform in an almost perfect way, meaning they are well suitable for real-time scenarios. Also, the F1 curves tell us that there was an excellent trade of between precision and recall at low confidence levels, which goes well along with the previous mentioned feats.

C. Transfomers

The 3 models trained with RT-DETR did not overthrow YOLO. 2 of the models were trained using Ultralytic's transformers [36]. One of them was the worst trained model of all models. Just looking at the confusion matrixes, we can see a huge ammount of FP and FN. The other, which used AdamW as an optimizer, had some quite better results, with 2 FN and 21 FP and very very low loss function values. But, for our disappointment, the F1 score of this model was a bit bad, being the highest at higher confidence levels (78.7%). But apart from that, it really performed very well.

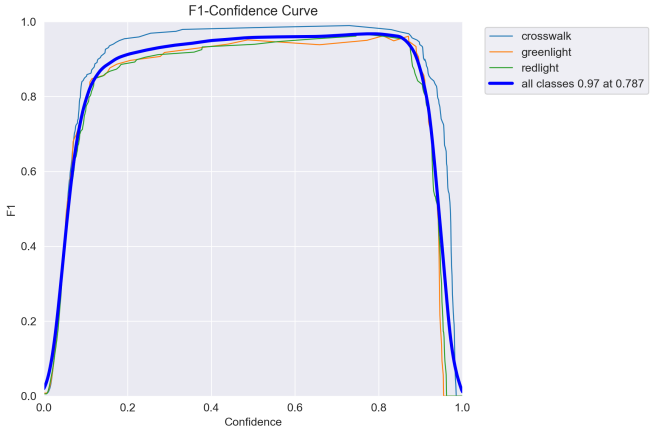


Fig. 6: F1 score of RT-DETR model with AdamW

The third model used a different transformer available at GitHub[23]. The results given by the model were astonishing, but unfortunately the logs that we have access to are very scarce. The only data we had was about Average Precision scores, which were great (AP50 of 0.97 and AP50-95 of 0.74) and at level with our best performing YOLO model, but unfortunately, it is not enough data to assume that it is better than the YOLO model. Other information, regarding loss, F1 score and confusion matrix data would be crucial to consider this kinds of decisions.

D. Prediction/Inference time

To better evaluate the performance of each model in a real-world environment, we decided to predict a set of 16 random images, taken from the test dataset and annotate the Inference times. Next, we processed the data by extracting the average and confidence interval for each model version (Note that we group by model version, because inference times are affected by model size, not training parameters). This test was performed using a laptop, equipped with an AMD Ryzen 7

7735HS, integrated GPU and 16 Gigabytes (GB) of Random-Access Memory (RAM), as to demonstrate a possible use-case of the model being present in an edge device. The results of this test are present in Figure 7.

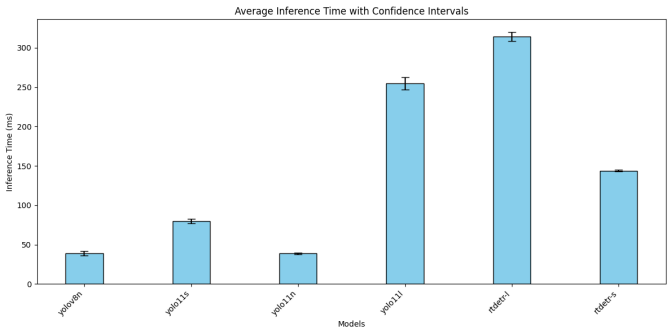


Fig. 7: Average Inference times, per model

It's evident that a balance between models capability and speed, needs to be measured, since using better, more accurate models will substantially worsen speed. For example, using the fastest model, YOLO11n, will take roughly 39 milliseconds (ms) per frame, meaning a frame processing time of 25 Frames per Second (fps). Whilst, using the slowest, will take in order of 315 ms, representing a frame processing time of 3 fps.

V. Results

With all of this said, the results show that the best model of all was the one trained with YOLO11s, 300 epochs, 100 patience and 16 of batch size. This model showed optimal decision taking at low confidence levels (97% F1 score at 32% confidence), very few bad predictions (3 FP and 3 FN) and a reasonable inference time (79.73ms) compared to the other models.

A. Check Predictions

In terms of showing the capabilities of the model, based on it's testing set, we used the sixteen random images shown previously and joined the YOLO11s predictions with the labels in the data. The results of this test can be observed in Figure 8. In here we can observe that there are some some missed predictions, but overall the bounding boxes look accurate and overlap with the label data, in exception with some differences in a small number of images.

B. Test with a video

In order to test the three best YOLO models in a scenario closer to the real-world, we decided to use it directly in two videos taken from Youtube (https://www.youtube.com/watch?v=6Ps2Qopf1VM and https://www.youtube.com/watch?v=Tei1StINC9c). Excerpts were taken from each video and processed offline. The results from this test can be viewed in Google Drive. In here, we can see that all the models managed to correctly identify all the classes, but they still had troubled with False Positive and Negatives. The best model appears to be the YOLO11s, with solid performance

Fig. 8: Test of Labels vs. Prediction of the model

and accuracy, however, it still suffered from multiple False Positives, mainly in handling the street light, as in Figure 9. This could be due to the light distortion and the frame quality, which leads YOLO to generalize more and not go based on color or figure in the light, but for light intensity. Some of this could possibly be mitigated by adjusting confidence levels in the prediction, however, as in Figure 9, it's prediction for the street light 50 % probability of being a red-light is already too high to be filtered. This scenario could also be considered too extreme, due to the low resolution and low-light conditions, however, the dataset contains examples in such conditions and such a system as the one proposed in this paper should be able to work in all conditions.

Fig. 9: False Positive of a street light identified as a red light

C. Comparisons with State Of Art

Our work demonstrates significant advancements pedestrian assistance, though the detection of Crosswalks, red and green traffic lights, by leveraging YOLO v11s on a diverse dataset encompassing varied condition. The results obtained, summarized in Table V, highlight the improvements achieved in precision, recall, and mAP, compared to other state-of-the-art methods.

Name	Type	Approach	Precision	Recall	F1-Score	mAP@0.5
chanyoung et al. [37]	Deep Learning	Roboflow 3.0 Object Detection (Fast)	95.2%	85.1%	N/A	91.5%
Zheng-De Zhang et al. [24]	Deep Learning	CDNet based on YOLOv5 (1 class)	N/A	N/A	94.83%	N/A
Mon Arjay Malbog et al. [25]	Deep Learning	Mask R-CNN (1 class)	97%	N/A	N/A	N/A
Alam et al. [38]	Deep Learning	YOLOv8l (3 class)	N/A	96.5%	N/A	65.3%
Samuel et al. [39]	Deep Learning	LYTNet based on CNN	97.54&	92.19%	94.79 %	N/A
Cheng et al. [40]	Feature Engineering	Color, shape, region, filtering, validation	97.35 %	88.87 %	92.91%	N/A
Our Work (YOLO v11n)	Deep Learning	YOLO11s (3 class)	**97.4%**	**96%**	**97%**	**97%**

TABLE V: Comparison of Methods for Crossroads and Traffic light detection

Despite this results, we acknowledge that, for a better use of our tools, more time and better conditions would be crucial, such as the opportunity for tests on real benchmarks, with real-time conditions and equipment.

Finally, the use of RT-DETR was important for discovering a new efficient way of creating object detection models apart from the famous YOLO, proving its value.

VI. Conclusion

This work achieved a model using YOLO v11s that is at level with the work of other authors. Our training using the described dataset, and after many trainings with different models/techniques, gave us a model that has an F1 score of 97 at 32.2% confidence, which is a very good result that concludes that this model could scale for the real-world, given the right resources and time. Also, it's worth nothing that YOLO has a lot of optimizations that make the training experience much smoother and less prone to bad results, something that is not as efficient (but still is at a certain level) with the used RT-DETR tools.

VII. Future Work

The work conducted in this report could be improved in the following ways:

- Experiment a system with two-stage operation. One with segmentation of the traffic light and another for assessing the light status. The zebra crossings could also use the same approach or stay the same. This approach would slow down operation of the system, but could help in having more data/control in identifying the correct signal
- Use a bigger dataset, or preferably create one dedicated to European/Portuguese traffic lights and roads.
- Further test more RT-DETRv2 models and parameters.
- Use a high resolution camera during the day and an Infra-Red one during the night (two models needed). This could help in improving low-light condition, since the colors of the traffic lights wouldn't get so "washed-out", due to the RGB camera low-light performance.

VIII. Contributions

The distributions for the work here conducted are as follows:

- Gonçalo Silva, 103244, 50%
- Samuel Teixeira 103325, 50%

IX. Acknowledgments

The research and writing of this paper were conducted with the use of Artificial Intelligence (AI) tools such as ChatGPT, Copilot, Gemini or Claude.

References

[1] W. contributors, *Pedestrian*, Jan. 2025. [Online]. Available: https://en.wikipedia.org/wiki/Pedestrian.

[2] M. L. Wells HL, *Distracted pedestrian behavior on two urban college campuses*, Feb. 2018. [Online]. Available: https://pmc.ncbi.nlm.nih.gov/articles/PMC5762430/.

[3] V. Australia, *New report reveals 1 in 12 pedestrians, being hit by motor vehicles and cyclists*, 2012. [Online]. Available: https://visionaustralia.org/news/2019-08-23/new-report-reveals-1-12-pedestrians-being-hit-motor-vehicles-and-cyclists.

[4] *Pedestrian deaths and other motor vehicle crash deaths*, 2022. [Online]. Available: https://www.iihs.org/topics/fatality-statistics/detail/pedestrians.

[5] *Global estimates of vision loss*, 2020. [Online]. Available: https://www.iapb.org/learn/vision-atlas/magnitude-and-projections/global/.

[6] *Understanding the impact of pedestrian accidents:* Feb. 2024. [Online]. Available: https://pinkstonlawgroup.com/understanding-the-impact-of-pedestrian-accidents/.

[7] X. Zou, "A review of object detection techniques," in *2019 International Conference on Smart Grid and Electrical Automation (ICSGEA)*, 2019, pp. 251–254. DOI: 10.1109/ICSGEA.2019.00065.

[8] D. G. Lowe, *Distinctive image features from scale-invariant keypoints - international journal of computer vision*. [Online]. Available: https://link.springer.com/article/10.1023/B:VISI.0000029664.99615.94#citeas.

[9] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *2014 IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 580–587. DOI: 10.1109/CVPR.2014.81.

[10] R. Girshick, J. Donahue, T. Darrell, and J. Malik, *Rich feature hierarchies for accurate object detection and semantic segmentation*, Oct. 2014. [Online]. Available: https://arxiv.org/abs/1311.2524.

[11] A. Vaswani *et al.*, *Attention Is All You Need*, arXiv:1706.03762 [cs], Aug. 2023. DOI: 10.48550/arXiv.1706.03762. [Online]. Available: http://arxiv.org/abs/1706.03762 (visited on 01/15/2025).

[12] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*, arXiv:1810.04805 [cs], May 2019. DOI: 10.48550/arXiv.1810.04805. [Online]. Available: http://arxiv.org/abs/1810.04805 (visited on 01/15/2025).

[13] A. Radford, K. Narasimhan, T. Salimans, and I. Sutskever, "Improving Language Understanding by Generative Pre-Training," en,

[14] A. Dosovitskiy *et al.*, *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*, arXiv:2010.11929 [cs], Jun. 2021. DOI: 10.48550/arXiv.2010.11929. [Online]. Available: http://arxiv.org/abs/2010.11929 (visited on 01/15/2025).

[15] *Tecnologia NVIDIA DLSS 4*, pt-br. [Online]. Available: https://www.nvidia.com/pt-br/geforce/technologies/dlss/ (visited on 01/15/2025).

[16] *NVIDIA DLSS 4 Introduces Multi Frame Generation & Enhancements For All DLSS Technologies*, en-us. [Online]. Available: https://www.nvidia.com/en-us/geforce/news/dlss4-multi-frame-generation-ai-innovations/ (visited on 01/15/2025).

[17] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, *You only look once: Unified, real-time object detection*, May 2016. [Online]. Available: https://arxiv.org/abs/1506.02640.

[18] R. Kaur and S. Singh, "A comprehensive review of object detection with deep learning," *Digital Signal Processing*, vol. 132, p. 103 812, Nov. 2022. DOI: 10.1016/j.dsp.2022.103812.

[19] Ultralytics, *Detect*, en. [Online]. Available: https://docs.ultralytics.com/tasks/detect (visited on 01/15/2025).

[20] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko, *End-to-End Object Detection with Transformers*, arXiv:2005.12872 [cs], May 2020. DOI: 10.48550/arXiv.2005.12872. [Online]. Available: http://arxiv.org/abs/2005.12872 (visited on 01/15/2025).

[21] Y. Zhao *et al.*, *DETRs Beat YOLOs on Real-time Object Detection*, arXiv:2304.08069 [cs], Apr. 2024. DOI: 10.48550/arXiv.2304.08069. [Online]. Available: http://arxiv.org/abs/2304.08069 (visited on 01/12/2025).

[22] W. Lv, Y. Zhao, Q. Chang, K. Huang, G. Wang, and Y. Liu, *RT-DETRv2: Improved Baseline with Bag-of-Freebies for Real-Time Detection Transformer*, arXiv:2407.17140 [cs], Jul. 2024. DOI: 10.48550/arXiv.2407.17140. [Online]. Available: http://arxiv.org/abs/2407.17140 (visited on 01/12/2025).

[23] lyuwenyu, *Lyuwenyu/RT-DETR*, original-date: 2023-05-10T06:35:56Z, Jan. 2025. [Online]. Available: https://github.com/lyuwenyu/RT-DETR (visited on 01/12/2025).

[24] Z.-C. L. H.-C. L. L. P. Zheng-De Zhang Meng-Lu Tan and W.-X. Yu, "Cdnet: A real-time and robust crosswalk detection network on jetson nano based on yolov5," *Neural Computing and Applications*, 2022. DOI: 10.1007/s00521-022-07007-9.

[25] M. A. Malbog, "Mask r-cnn for pedestrian crosswalk detection and instance segmentation," in *2019 IEEE 6th International Conference on Engineering Technologies and Applied Sciences (ICETAS)*, 2019, pp. 1–5. DOI: 10.1109/ICETAS48360.2019.9117217.

[26] *Pedestrian light, crosswalk Dataset ¿ Overview*, en. [Online]. Available: https://universe.roboflow.com/chanyoung/pedestrian-light-crosswalk (visited on 01/15/2025).

[27] Ultralytics, *Tips for best training results*, Sep. 2024. [Online]. Available: https://docs.ultralytics.com/yolov5/tutorials/tips_for_best_training_results/.

[28] Ultralytics, *A guide on model testing*, Oct. 2024. [Online]. Available: https://docs.ultralytics.com/guides/model-testing/#signs-of-underfitting.

[29] Ruman, *Yolo data augmentation explained*, Jun. 2023. [Online]. Available: https://rumn.medium.com/yolo-data-augmentation-explained-turbocharge-your-object-detection-model-94c33278303a.

[30] Ultralytics, *Augment*. [Online]. Available: https://docs.ultralytics.com/reference/data/augment/.

[31] Ultralytics, *Train*, Nov. 2024. [Online]. Available: https://docs.ultralytics.com/modes/train/#train-settings.

[32] Ultralytics, *Epoch*, Oct. 2024. [Online]. Available: https://www.ultralytics.com/glossary/epoch.

[33] F. Graetz, *Why adamw matters*, Jun. 2018. [Online]. Available: https://towardsdatascience.com/why-adamw-matters-736223f31b5d.

[34] *RT-DETR/rtdetrv2_pytorch at main · lyuwenyu/RT-DETR*, en. [Online]. Available: https://github.com/lyuwenyu/RT-DETR/tree/main/rtdetrv2_pytorch (visited on 01/15/2025).

[35] Ultralytics, *Batch size*, Oct. 2024. [Online]. Available: https://www.ultralytics.com/glossary/batch-size.

[36] Ultralytics, *RT-DETR (Realtime Detection Transformer)*, en. [Online]. Available: https://docs.ultralytics.com/models/rtdetr (visited on 01/13/2025).

[37] chanyoung, *Model & API*, en. [Online]. Available: https://universe.roboflow.com/search?q=class%3Acrosswalk (visited on 01/15/2025).

[38] M. F. Alam, *Farukalamai/traffic-lights-detection-and-color-recognition-using-yolov8*, original-date: 2023-09-12T13:54:28Z, Dec. 2024. [Online]. Available: https://github.com/farukalamai/traffic-lights-detection-and-color-recognition-using-yolov8 (visited on 01/15/2025).

[39] S. Yu, H. Lee, and J. Kim, "LYTNet: A Convolutional Neural Network for Real-Time Pedestrian Traffic Lights and Zebra Crossing Recognition for the Visually Impaired," in vol. 11678, arXiv:1907.09706 [cs], 2019, pp. 259–270. DOI: 10.1007/978-3-030-29888-3_21.

[Online]. Available: http://arxiv.org/abs/1907.09706 (visited on 01/15/2025).

[40] R. Cheng, K. Wang, K. Yang, N. Long, J. Bai, and D. Liu, "Real-time pedestrian crossing lights detection algorithm for the visually impaired," en, *Multimedia Tools and Applications*, vol. 77, no. 16, pp. 20 651–20 671, Aug. 2018, ISSN: 1573-7721. DOI: 10.1007/s11042-017-5472-5. [Online]. Available: https://doi.org/10.1007/s11042-017-5472-5 (visited on 01/15/2025).

X. Acronyms

AI Artificial Intelligence

ML Machine Learning

R-CNN Region Based Convolutional Neural Networks

CNN Convolutional Neural Networks

RPN Region Proposal Networks

YOLO You Only Look Once

IoU Intersection over Union

AP Average-Precision

mAP mean Average-Precision

ms milliseconds

fps Frames per Second

FP False Positives

FN False Negatives

CV Computer Vision

RT-DETR Real-Time Detection Transformer

DETR DEtection TRansformer

NLP Natural Language Processing

ViT Vision Transformer

RAM Random-Access Memory

GPU Graphical Processing Unit

GB Gigabytes

VRAM Video Random-Access Memory