# Comparative analysis of real image datasets vs. synthetic image datasets in industrial environments

**Author**
Lionel Güitta López

**Supervisor**
A. López López

**Cosupervisor**
N. de Rodrigo Tobías

**Entity**
ICAI- Universidad Pontificia Comillas

This project explores the benefits offered by synthetic image datasets in industrial environments. Thus, the necessary tools (images, labels, etc.) will be developed to generate real and synthetic samples. From the sets of real and synthetic samples, it was possible to test how the models trained with synthetic samples adapt to actual working conditions. These tests showed great performance of both real and synthetic datasets and the most relevant features of the images that helped the models achieve those results.

**Key Words:** Computer vision, Regression, Neural Networks, Deep Learning, YOLO.

## 1 Introduction

Over the last few years, new elements from the so-called Industry 4.0 are increasingly being incorporated into the industry, including collaborative operations in robotics, artificial intelligence, or IIoT. The development of these technologies and their understanding are key to the advancement of the industry itself and to the increase of efficiency and effectiveness. Thus artificial intelligence models stand out in their multiple applications such as object detection, defect detection, predictive maintenance, etc. There are cases in which the training of the models has been sufficiently generalist for a task and it is possible to adapt that solution to a new one, reducing the training time or eliminating it. For training the models, a large set of samples is required, but this is not always possible due to the time-consuming and complex nature of the generation of samples. Thus, the use of datasets composed of synthetic or mixed samples can be a great help in the elaboration of these, reducing the complexity and time of their creation.

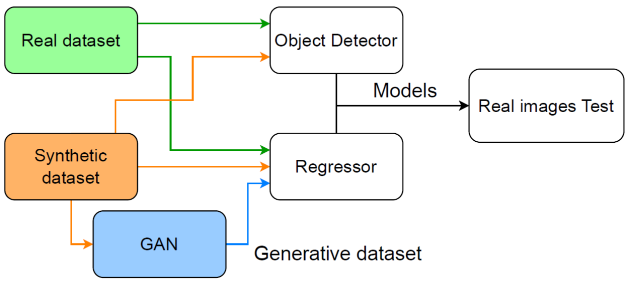


Figure 1: Project pipeline.

This project explores the benefits offered by synthetic image datasets in industrial environments in two applications. The first one detects several instances of an object in an image and the second one obtains a unitary vector perpendicular to the surface of an object shown in an image through regression.

For this purpose, the workflow shown in Figure 1 was established. The first step was to obtain the real and synthetic datasets. Then, object detection and regression models were trained with both datasets. In addition, for the regression model, a third dataset composed of the synthetic images, after being post-processed in a generative adversarial network that resembles the synthetic images to the real images, was added.

The models were validated with a set of real images to demonstrate their effectiveness in actual working environments. Also, the most relevant features of the images, the ones that provide the information to the model, were extracted.

## 2 Related work

**YOLO -** The YOLO architecture was first exposed in 2015 [1] and has evolved into subsequent versions [2] [3] [4] which included features such as classification tasks or complex architecture designs. The version at the beginning of this project was YOLOv5 [5] and is still under development. YOLO stands out due to its speed of inference [6], processing the image as a whole and allowing real-time processing. The lead developers of YOLOv5 offer five pretrained models with the MSCOCO dataset. These are the *Nano*, the *Small*, the *Medium*, the *Large* and the *XLarge*. Their differences lie in their complexity, capacity, performance, speed, and computational load. It is possible to adapt them to specific applications with transfer learning, saving training and validation time, as well as a considerable amount of energy.

**Regressor -** Another artificial intelligence application in the supervised learning field is regression. In Machine Learning, linear regression relates the dependent variable with the independent variables through the regression coefficients, without nonlinearities in the [7] model. For *deep learning* applications the model includes nonlinearities between input variables and outputs due to the activation functions. In addition, an image can be used as an input and CNN can be used. In this way, numerical predictions about values related to the images can be obtained [8] [9].

**GAN -** One type of neural networks are generative adversarial networks (GANs). These were introduced in 2014 and seek to train 2 models whose objectives are opposing [10]. The model G seeks to capture the essence of the distribution of the training data to generate a sample that can exist within that distribution. The model D, has the objective of knowing how to differentiate between real samples and the samples generated by the G model. The ultimate goal is to obtain a generating model that is good enough for the discriminator model to get its predictions right randomly, i.e. 50% of the time. The samples that compose the distributions can be any type of data in an n-dimensional latent space, determining the complexity of the training. Depending on the architecture used for G, it can generate samples from random noise, a specific set of values, or even an image.

**CycleGAN -** An application associated with the transfer of styles between images is the CycleGAN [11] network, where the style of a painting can be transferred to a photo and in the opposite direction for example. The architecture presents two generators in charge of translating the original pictures to the opposite style and two discriminators associated with each style. The loss function includes a term for the losses due to pitting a generator against a discriminator, where the Wasserstein loss [12] [13] or the MSE provides stability to the model. The cycle consistency loss term is added, based on the fact that when processing an image with one generator and processing it back with the other, the image should remain unchanged. To maintain the color composition of the images faithfully, identity mapping loss is introduced.

**Real datasets -** There are companies dedicated to the collection of real images for the creation of datasets and then offer them for the development of artificial intelligence models. Examples are the MSCOCO dataset [14], the Youtube-8M dataset [15] or the CelebA dataset [16]. The problem with such images is that, for specific applications, dedicated images must be generated for that problem, a process that can be complex and costly in both time, resources, and materials.

**Synthetic datasets -** As with real image datasets, there are companies dedicated to the generation of synthetic image datasets for specific applications [17] thanks to graphics rendering applications and video game engines such as Unity [18] or Blender [19], where the aim is to make them as similar as possible to what a real image would be.

**Compare Images -** In order to quantify the differences between images one way is comparing the pixel distributions between the 3 color layers of the image (red, green and blue). This comparison presents the problem that it is possible that the images themselves have the same amount of each color, but they are arranged in a completely different order. Another approach is to calculate the Wasserstein distance, also known as the EMD (Earth mover's distance) [20], between them, comparing distributions as probability distributions.

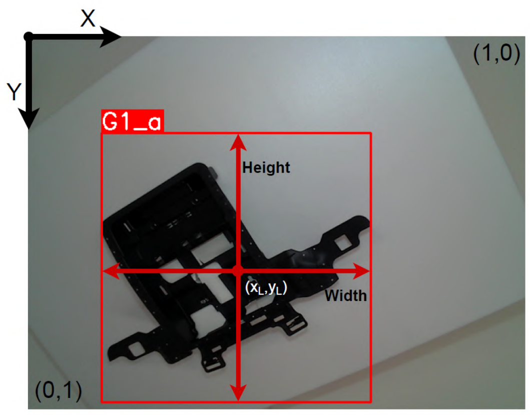


Figure 2: Label elements in YOLO image.

## 3 Dataset generation

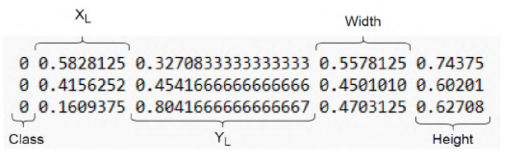
This section describes how the different datasets were obtained and then used in model training. It includes an explanation of the characteristics of the images and the labels that make up the dataset for each model, according to the requirements.

### 3.1 YOLO dataset

For the YOLO dataset, the coordinate base of an image is in the pixel located in the upper left corner of the image. From there, each pixel to the right adds one unit on the x-axis and each pixel down adds one unit on the y-axis. To standardize the measurement, all coordinates will be relative to the image size, being bounded between 0 and 1.

When training the model, it is necessary to define the classes of the objects to be classified and enumerate them in a configuration file. Each number in the configuration file will be the number to be assigned in the image label. Images do not need to have a specific dimension or a predetermined number of channels since they will be adapted later to be processed by the model. Each label consists of the location of the object in the image, defined by the center and size of the bounding box that frames it, and the class of the object itself. Each image will have a text file with the corresponding labels. An example of a labeled image and its labels can be seen in the Figure 2 and Figure 3.

The content of each line of the text file must represent a complete label with the following values separated by a blank space:


$X_L$
Width
Class
$Y_L$
Height

```
0 0.5828125 0.3270833333333333 0.5578125 0.74375
0 0.4156252 0.4541666666666666 0.4501010 0.60201
0 0.1609375 0.8041666666666667 0.4703125 0.62708
```

Figure 3: Text file with YOLO labels.

1. Object class index.
2. Center of the bounding box at the x-coordinate.
3. Center of the bounding box at the y-coordinate
4. Width of the bounding box at the x-coordinate of the image.
5. Height of the bounding box at y-coordinate of the image.

In this project, color images were used, with different resolutions, with a single class. An example of this type of images can be seen in the Figure 4.

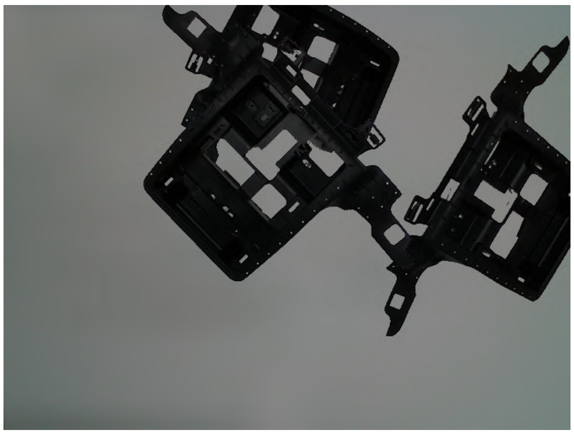


Figure 4: YOLO training sample.

### 3.2 Regression dataset

First, the point of interest from which we want to obtain the normal vector and the basis of coordinates of the image (X,Y) and the vector (u,v,w) were defined. These are shown in the Figure 5.

The input image is colored and has a resolution of 224x224 pixels. It will only be possible to extract one complete normal vector for each image, resulting in only one label per image. The labels will all be stored in a single text file like the one shown in Figure 6.

The header of the file shows the content of each column, which are:

- ***Name:*** Name of the file.
- ***X:*** $x$ coordinate of the point of interest.
- ***Y:*** $y$ coordinate of the point of interest.

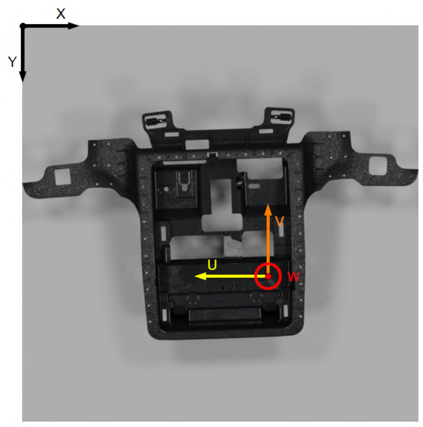
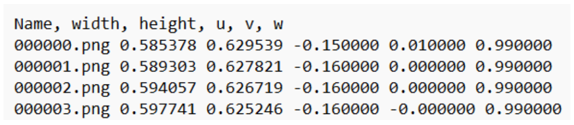


Figure 5: Regression basis of coordinates.



```
Name, width, height, u, v, w
000000.png 0.585378 0.629539 -0.150000 0.010000 0.990000
000001.png 0.589303 0.627821 -0.160000 0.000000 0.990000
000002.png 0.594057 0.626719 -0.160000 0.000000 0.990000
000003.png 0.597741 0.625246 -0.160000 -0.000000 0.990000
```

Figure 6: Regression labels file.

- ***U:*** $u$ component of the vector.
- ***V:*** $v$ component of the vector.
- ***W:*** $w$ component of the vector.

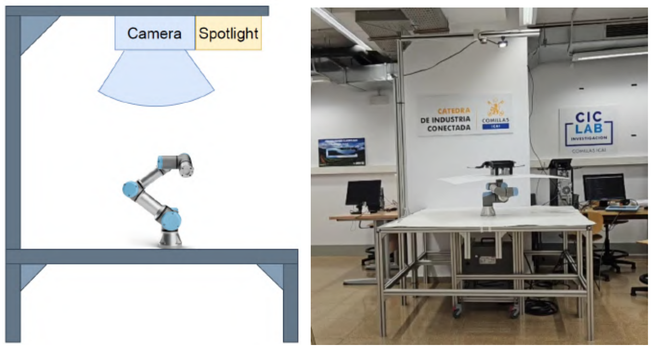
Despite adding the values of the coordinates $x$ and $y$, in this project, they will not be used and it is left for a future iteration the development of a model capable of obtaining these values too.

### 3.3 Real dataset

An Intel RealSense L515 [21] camera was used to capture the images, accompanied by an illumination system consisting of LED spotlights to maintain lighting conditions as constant as possible.

To obtain the necessary measurements and the images, the process was automated using a rudimentary object detection algorithm making use of color filters and performing the appropriate operations to obtain the bounding boxes. In addition, a Universal Robots UR3e [22], robotic arm was used to move the piece in the image and obtain the measurements of the vector normal to the surface.

The infrastructure for this process consisted of a working structure with the UR3e robot at the center. A 3D part was designed and printed to be mounted on the robot and on which the part of interest would be attached. The photo was captured from the upper view of the working environment. Both the schematic and the result of the design in the real environment are shown in Figure 7.



(a) Schematic (b) Real scenario

Figure 7: Image capture infrastructure.

The algorithm to capture the images and label them is shown in Figure 8 and is detailed below.

The system starts by establishing the connection to the camera and the robot. Then, the first axis and the wrist axes of the robot are moved to random positions within a range, covering as many positions as possible. An image is captured with the camera and the measurement of the vector normal to the TCP of the robot, which is equal to the one of interest, is obtained. A color filter is used to extract the label for the object detection model. The measurements of the bounding box will be adjusted so that the width and height of the bounding box have a ratio of 1:1. The label and image for the object detection dataset are completed. For the regressor, the remaining step is to crop the image obtained by the camera in the area defined by the bounding box. The process continues by moving the robot to a new random position and repeating the cycle as many times as necessary.

Figure 9 shows examples of one sample of the images obtained for each dataset.

#### 3.3.1 Object detection dataset modifications

As the pretrained models are really powerful, in order to take advantage of their full capacity and pose a suitable challenge, it was decided to alter the images of the obtained dataset. Since in this project, there is only one object of a single class, the classification task is left uncovered. Therefore, we went from having one piece per image to having between one and 4 pieces per image with their corresponding labels on a homogeneous white back-

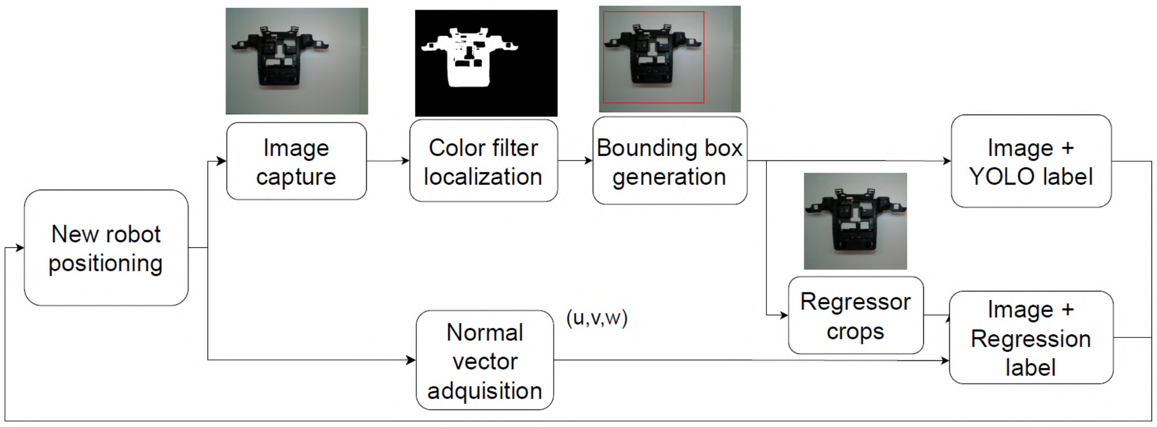
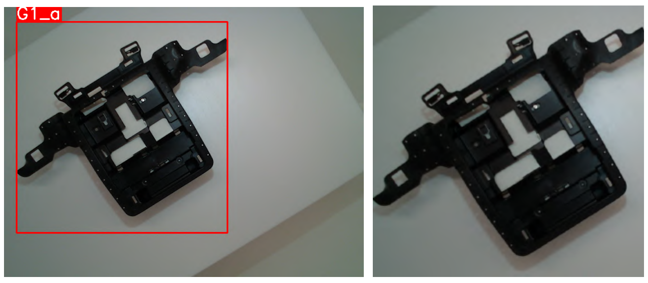


Figure 8: Image capture and labeling process.



(a) Object detection (b) Regression

Figure 9: Real datasets samples.

ground equal to the one captured by the camera in the original images. Figure 10 shows an example of the final result.

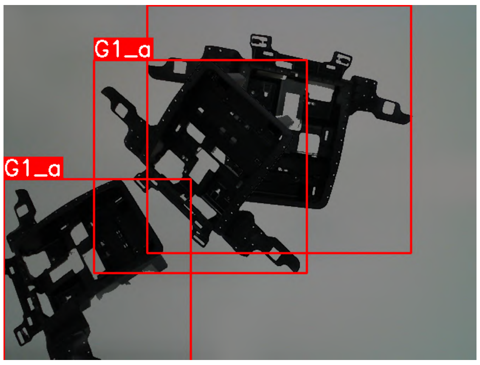
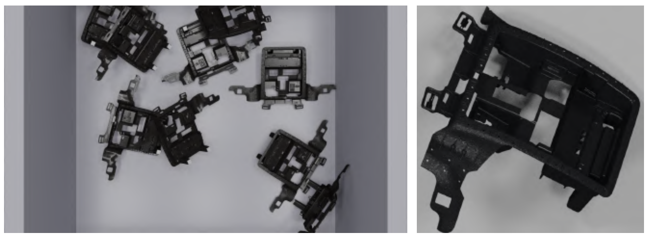


Figure 10: Object detection real dataset sample.

### 3.4 Synthetic dataset

The generation of this dataset was developed as an automated process in which, starting from a CAD model of the part, the Blenderproc tool [23] was used to obtain samples of both the object detection model and the regression model. A scenario was modeled consisting of a box in which numerous samples of the parts in different positions would be generated and a view from which an image would be rendered. This scenario allows complete control of the positions and orientations of the parts, unlike the real scenario, and allows taking more advantage of the image generation capacity and testing the limits of the developed model to a greater extent. The limitation in this case was computational time. The higher the resolution of the output images, the more computational resources and time it required. However, this is still slightly less than the full processing time of the real images.

Figure 11 shows samples obtained in this process. They are not exactly the same as the images obtained with the camera in real conditions and some differences between these and the real ones are remarkable.



(a) Object detection (b) Regression

Figure 11: Synthetic datasets samples..

### 3.5 Generative dataset

Due to the differences between the real and synthetic images, a third dataset was developed trying to increase the similarity between them. A GAN based on CycleGAN was trained to transfer the style of the real images to the synthetic ones. The first trainings were performed with the original architecture.

Figure 12 shows the curves of the loss values of the generator and the discriminator throughout the training. The loss associated with the generator decreases correctly up to a limit that it did not manage to overcome, possibly a local minimum. On the other hand, the loss associated with the discriminator increases slightly as the generator learns to generate images more similar to the real ones.

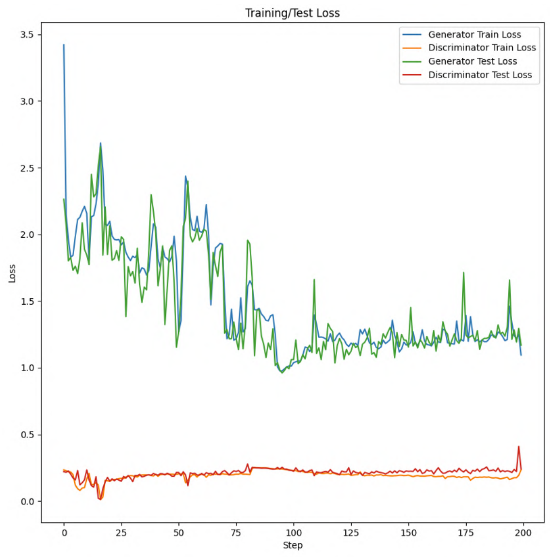


Figure 12: GAN Loss with original architecture.

It is worth noting that in the first trainings the generation of artifacts from a certain point in the process was observed. An example of this case can be seen in the Figure 13.

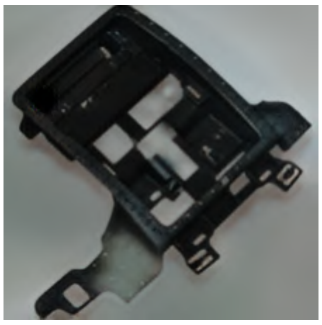


Figure 13: Artifact in generated sample.

To prevent these artifacts, the network was modified to implement changes that were introduced in StyleGANv2 [24] based on suppressing the batch normalization and convolutional layers by demodulated convolutional layers in the generator avoiding abrupt changes in the weights that may cause artifacts.

The loss curves associated with the new architecture are shown in Figure 14. It is possible to appreciate an evolution like the previous training and a sign of overfitting in the generator model from epoch 200 onwards.

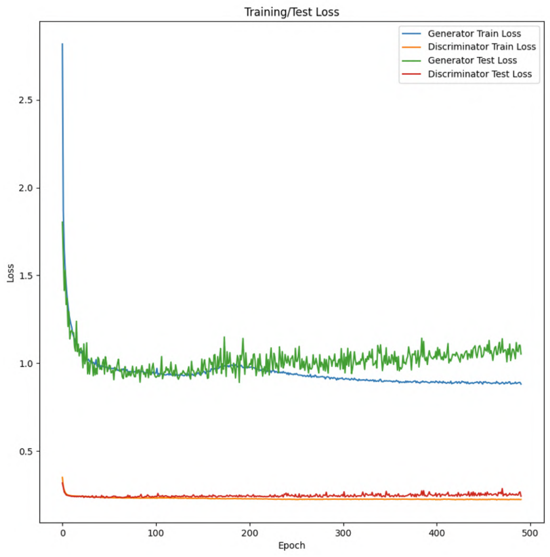


Figure 14: GAN Loss with modified architecture.

The problem of generated artifacts was solved. However, it was necessary to train with a lower learning rate and for a longer time. An example of the new images generated can be seen in the Figure 15.

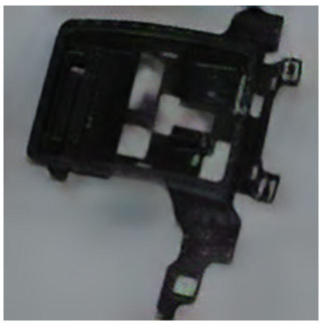


Figure 15: Generated sample without artifacts.

All the architectures used can be seen in Annex B.

## 4 Object detection

This section shows the training methodology of the models for the detection and localization of objects in images. We started with the pre-trained YOLO Nano, Medium, and Extra large models by

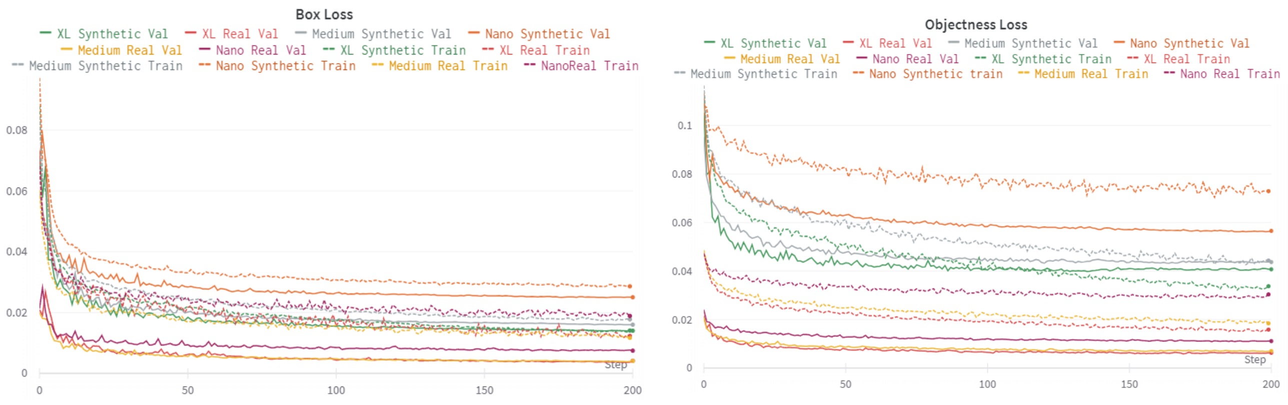


Figure 16: YOLO train losses.

freezing the backbone weights. Once the training was completed, an unseen data set of the models was used to perform inference on the models and evaluate them.

The code for the train was extracted directly from the Ultralytics YOLOv5 repository on Github [5], adding some modifications on the validation code to obtain certain metrics that were extracted in the train. The modifications added are detailed in Annex A.

## 4.1 Trainings

The results of all the trainings developed for the object detection and localization model with the real dataset and with the synthetic dataset are presented in Figure 16.

Regarding the curves of the **Nano** models, it can be seen that the model trained with the real dataset achieves a lower loss as a whole. In addition, it can be seen that the values of the validation losses are lower than those of the training set due to the fact that in the training phase, the pictures are slightly modified by data augmentation techniques.

The curves of the **Medium** models curves show similar results to those of the Nano model. The real set has lower losses and validation has lower values than the training set. However, when the model has learned enough we see how the results of the training and validation set approach each other.

Regarding the curves of the **Extra Large** models, during these trainings, train, and validation losses, as well as with real and synthetic model losses, are closer than in the previous cases. The models have higher capacity and can learn more complex cases.

The models trained with the real images present better loss values than the models trained with the synthetic images. On the other hand, it can be seen that as the complexity of the model increases, the results improve. Finally, in the Figure 17, the values of precision, recall, $mAP_{0.5}$ and $mAP_{0.5:0.95}$ of the models throughout the training sessions are shown. In this case, no difference is seen between the models trained with different images. However, comparing the models with respect to their complexity improves the different performance metrics as the model complexity increases.

In Figure 18 it is possible to observe the predictions for two stages of the training in both datasets filtering the result to those predictions in which the model assigns a score of more than 0.5 to the obtained bounding box. By changing the acceptance threshold it is possible to obtain more bounding boxes than desired and it is a parameter to be adjusted during the model execution.

## 4.2 Evaluation

After training and validating the models, they were evaluated against a dataset of real images. For all inference processes in this dataset, a confidence threshold of 0.5 was set and predictions were limited to 10 per image.

The values of the performance metrics and the losses generated for the real image models can be seen in Table 1. On the other hand, the metrics for the synthetic models can be seen in Table 2

In both cases, the metrics improve the greater the

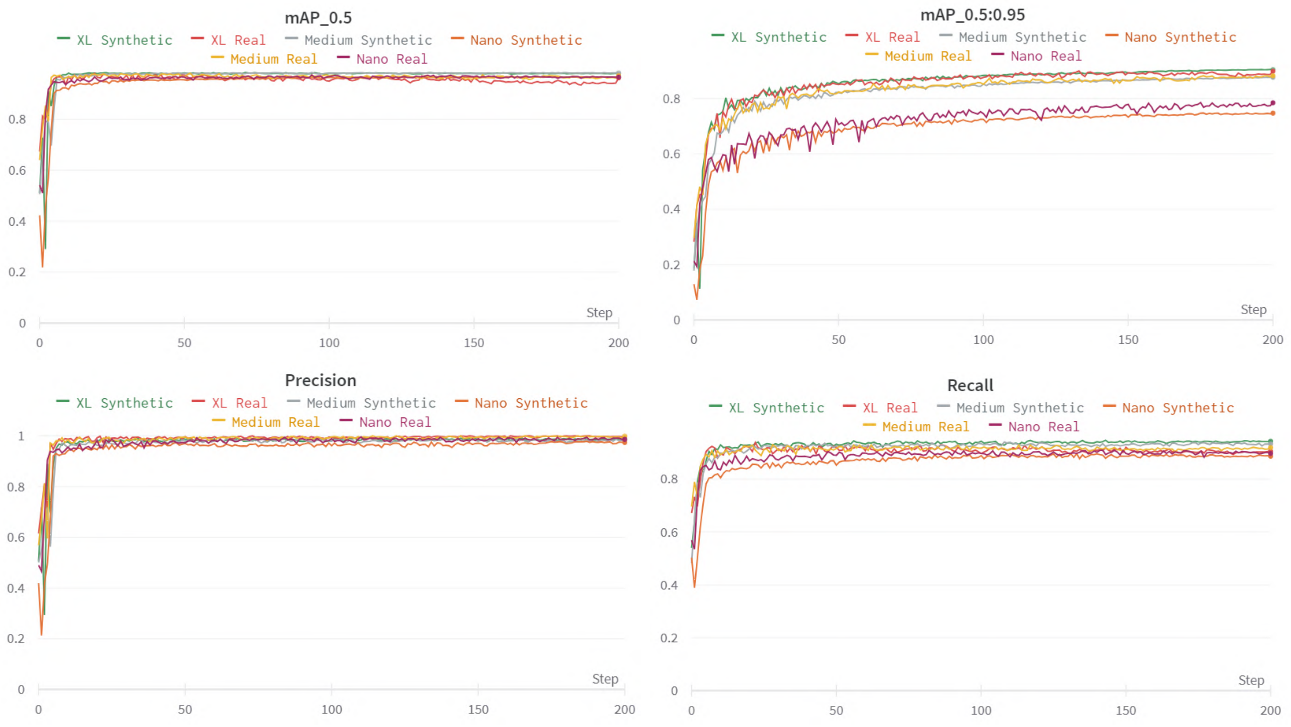


Figure 17: YOLO models performance metrics.

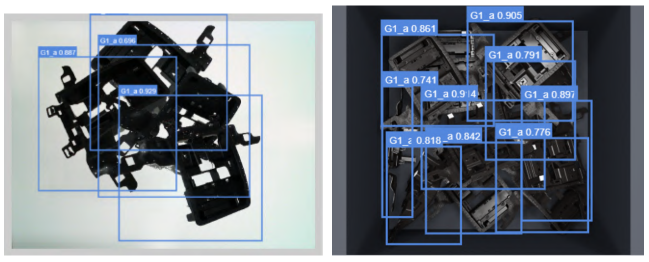


Figure 18: YOLO predictions.

| Test metrics real dataset models | | | |
|---|---|---|---|
| Metric | Nano | Medium | Extra Large |
| Precision | 0.964 | 0.962 | 0.963 |
| Recall | 0.849 | 0.860 | 0.858 |
| $mAP_{0.5}$ | 0.921 | 0.926 | 0.925 |
| $mAP_{0.5:0.95}$ | 0.916 | 0.919 | 0.920 |
| Box loss | 0.058 | 0.041 | 0.037 |
| Obj loss | 0.117 | 0.092 | 0.090 |

Table 1: Test metrics for real image models.

| Test metrics synthetic dataset models | | | |
|---|---|---|---|
| Metric | Nano | Medium | Extra Large |
| Precision | 1.000 | 1.000 | 1.000 |
| Recall | 0.824 | 0.822 | 0.824 |
| $mAP_{0.5}$ | 0.850 | 0.849 | 0.857 |
| $mAP_{0.5:0.95}$ | 0.850 | 0.849 | 0.855 |
| Box loss | 0.271 | 0.265 | 0.243 |
| Obj loss | 0.318 | 0.426 | 0.305 |

Table 2: Test metrics for synthetic image models.

complexity of the model. However, they do not represent a large improvement. This results in the fact that if moving to a real production environment, the Nano model should be used as it has much faster inference times and requires less computational power.

Models trained with real images give better results than those trained with synthetic images. However, this does not imply discarding this practice because the models trained with synthetic images present a great performance against the evaluation dataset composed of real images. This indicates that with a sufficiently faithful approximation of reality, the model is able to learn features from the images that help it to translate to a real environment.

# 5 Regression

This chapter describes the methodology for training the regression model to obtain the normal vector to a surface of interest. Once the trainings were completed, the models were evaluated with a dataset of real images.

The training and evaluation code, as well as the model architecture, are own developed. In addition, features were added to the codes to be able to visualize saliency maps with model attention areas and relevant image details.

The architecture of the convolutional network can be seen in the Annex B. It takes as input a 224-pixel color square image and is processed by a series of convolutional layers with ReLU activation functions and a max pooling layer. After three convolution layers, it is processed by a series of dense layers producing 3 output values. To ensure that the output is a unit vector these values are normalized.

## 5.1 Loss function

To compare two vectors $A$ and $A'$ with components $u, v, w$ each one, the difference between them results in a third vector with 3 components that represent the difference between the initial components, the degree of error between the vector predicted by the model and the real vector. To implement it as a loss function it must be formulated as a single value. Therefore, one solution is to obtain the modulus of the previous operation, the distance between the two vectors. However, the ability to evaluate the components of the vector separately is lost. The goal now is that the distance between the two vectors is 0 and therefore, the actual label of the images is 0. Applying this form of error to all the vectors obtained from a batch of N images and averaging results in the Ecuation 1 which is equals to the MSE function.

$$Loss = \frac{\sum_i^N (u_i - u_i')^2 + (v_i - v_i')^2 + (w_i - w_i')^2}{N} \qquad (1)$$

## 5.2 Trainings

The results of the training performed on the real, synthetic and post-processed dataset in the GAN are presented in Figure 19, Figure 20 y la Figure 21

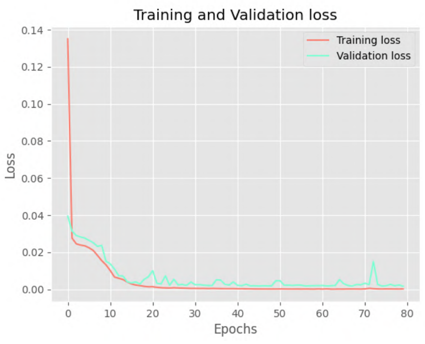
The first training developed was with the **real image set**. This consisted of 300 images which were divided into a training and a validation set with a ratio of 80% and 20% respectively. The value of the loss function, shown in Figure 19a, is reduced to almost exactly 0. Figure 20b shows how the model learns to look for information on the contour of the part and its shadows, indicating the relevance of these in the image, while the interior is practically ignored. Figure 21a shows a sample of the real vector and the one predicted by the model when processing an image. For the evaluation process, the models with the best performance against the validation set were studied.

The next training developed was the **synthetic image set**. This consisted of 633 images which were divided into a training set and a validation set with a ratio of 80% and 20% respectively. Figure 19b shows how the value of the loss function is reduced to almost 0. Figure 20d shows how the model focuses on the part contour, interior details, and its shadows, indicating, the relevance of these features. Compared to the previous case, it has a more general perspective of the image and not of a specific area. Figure 21b shows both the real vector and the one predicted by the model for the same image. For the evaluation process, the models with the best performance against the validation set were studied.

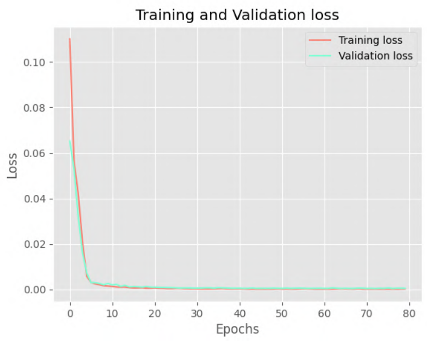
The last training set developed was with the **synthetic image set with the added GAN postprocessing**. Like the previous one, this consisted of 633 images that were divided into a training and a validation set with a ratio of 80% and 20% respectively. Figure 19c shows the curves with the loss function throughout the training and how it reduces to almost 0. Figure **??** shows how the model learns to focus on part details, the part outline and its shadows, indicating, again, the relevance of these in the image. However, compared to the previous cases, it can be seen that it has a more general perspective of the image with respect to the real image model, like the synthetic image model, but still maintains attention to some particular areas. Figure 21c shows both the actual vector and the one predicted by the model for the same image. For the evaluation process, the models with the best performance against the validation set were studied.

## 5.3 Evaluation

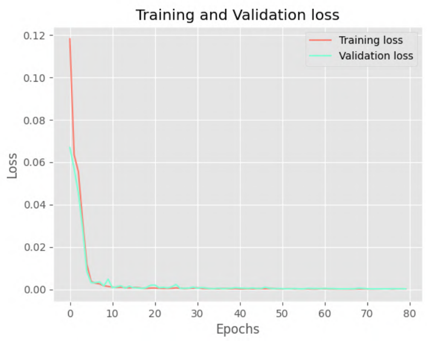
Finally, the three trained models were confronted with a dataset they had never seen before. This dataset consists of a set of 100 real images. In this way, the generalization capability of the models would be tested against a real environment.



(a) Real.

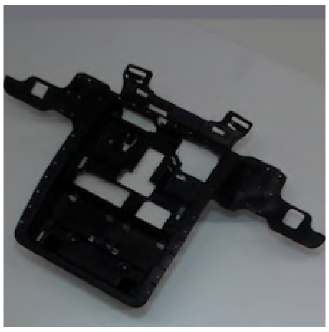


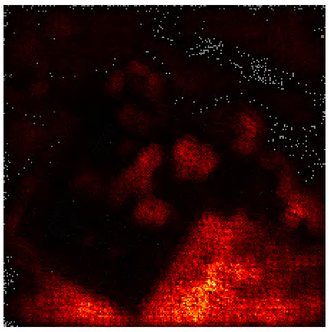
(b) Synthetic.



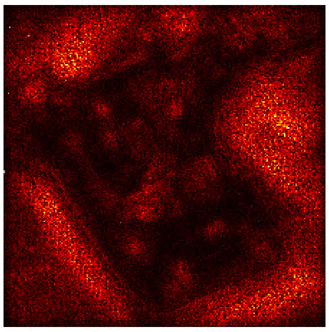
(c) GAN.

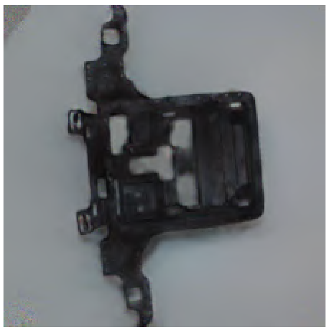
Figure 19: Regression train losses.


(a) Real image


(b) Real saliency map

(c) Synthetic image


(d) Synthetic saliency map


(e) GAN image

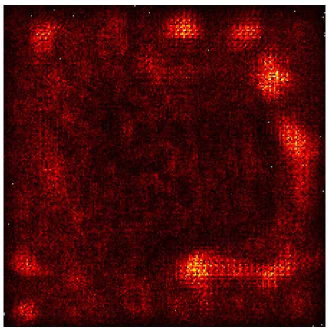
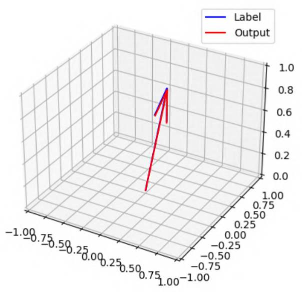
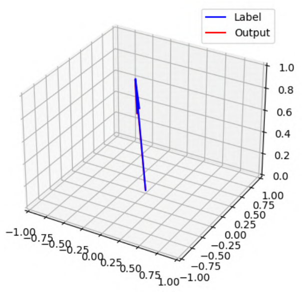

(f) GAN saliency map

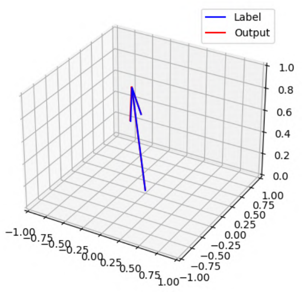
Figure 20: Saliency maps for the trainings.



(a) Real.

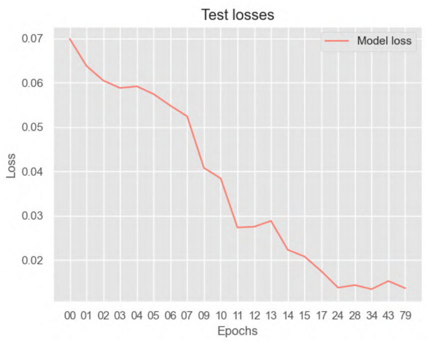


(b) Synthetic.

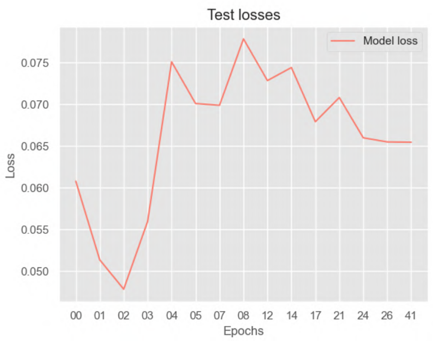


(c) GAN.

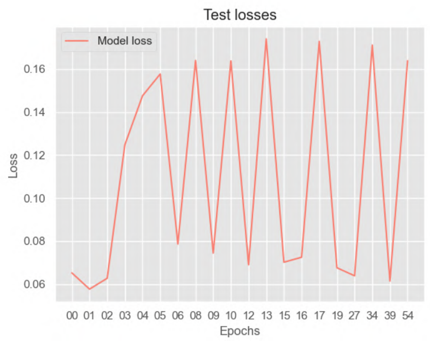
Figure 21: Regression training results samples.



(a) Real.



(b) Synthetic.



(c) GAN.

Figure 22: Regression test losses.

### 5.3.1 Test losses

Figure 22 shows the values of the loss function when pitting the best models of the validation set against the evaluation dataset. The real model improves as the epochs increase. This is contrary to what happens with the models of the other two image types where the value of the losses increases or even oscillates. Furthermore, for these image types, the models that perform best in the evaluation set are the ones that were trained less. For the rest of the evaluation, the models considered were the last model trained with real images and a model trained with few epochs of those obtained with the training of synthetic images and postprocessed images.

### 5.3.2 Atention areas

Figure 23 shows the saliency maps obtained from the three models when processing the same test image. The saliency map of the real image model is very similar to the one obtained during its training, it does not pay much attention to the part details. However, the other two differ more from those obtained during training, with a higher amount of noise. The post-processed image model has a greater similarity to the one obtained previously and, in addition, it is more similar to the one obtained with the real image model, focusing the attention on the contour of the part. Moreover, in the training images of the real dataset, the textures of the part are generally not visible. However, in the images of the other datasets, more details of the part can be seen and the models learn to obtain information from the part as well.

### 5.3.3 Prediction errors

Figure 24 shows the error distributions obtained by processing the complete evaluation dataset with the three models. The figure shows the error expressed as the angle of difference between the actual and the predicted vectors against the vertical projection of the actual vector. The real images model has the lowest errors and the lowest variance of the 3 models, serving as a reference for the other two, which are similar. While the synthetic image model achieves slightly lower errors, the post-processed image model has a much lower variance. This can mean that the synthetic image model is more random and unpredictable, whereas the other can be tuned to reduce the errors it already exhibits.

## 6 Conclusions

Through automation, a dataset of 400 real images was generated. To achieve this, resources and time had to be invested in the design of the image capture infrastructure, the design and printing of 3D parts to manipulate the object of interest, planning, the development of the dataset generation algorithm, and finally the generation of the dataset.

Using BlenderProc as the rendering software, a synthetic image dataset was generated from a 3D model of the object of interest. To achieve this, it was necessary to learn how to manipulate the rendering software and design the virtual image capture scenario. It does not consume as many physical resources as those needed for the generation of the real dataset, but it requires high computational power to achieve sufficiently low generation times. Therefore, without a GPU with sufficient resources, this method would not be feasible. In this study, an NVIDIA GeForce RTX 2080 [25] GPU was available and it offered better image generation times than those of the real imaging method.

Due to the differences between synthetic and real images, a dataset composed of synthetic images postprocessed by a GAN was generated. The GAN would resemble the real ones by transferring features such as shadows or illumination. Time was consumed training the model and modifying the architecture to achieve the expected results. However, in a production environment, this process has sub-second inference times and the bulk of this resource is consumed training the

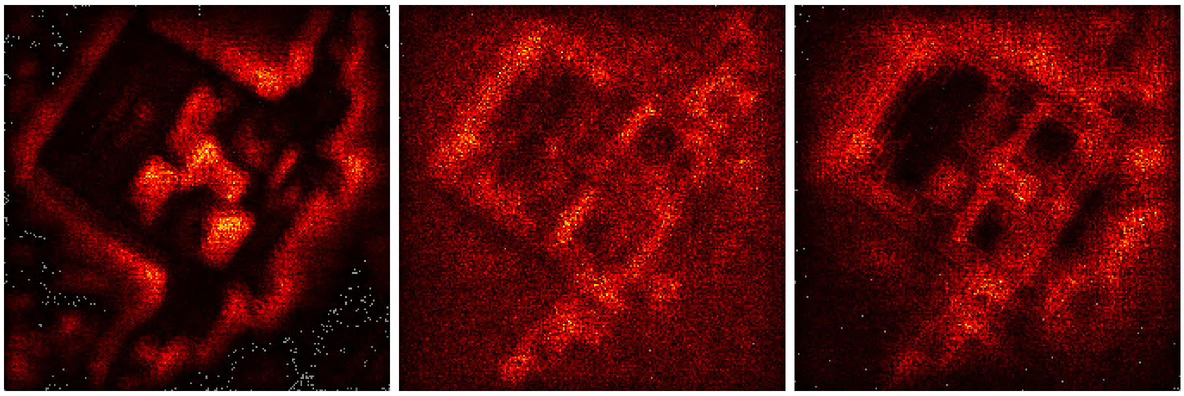
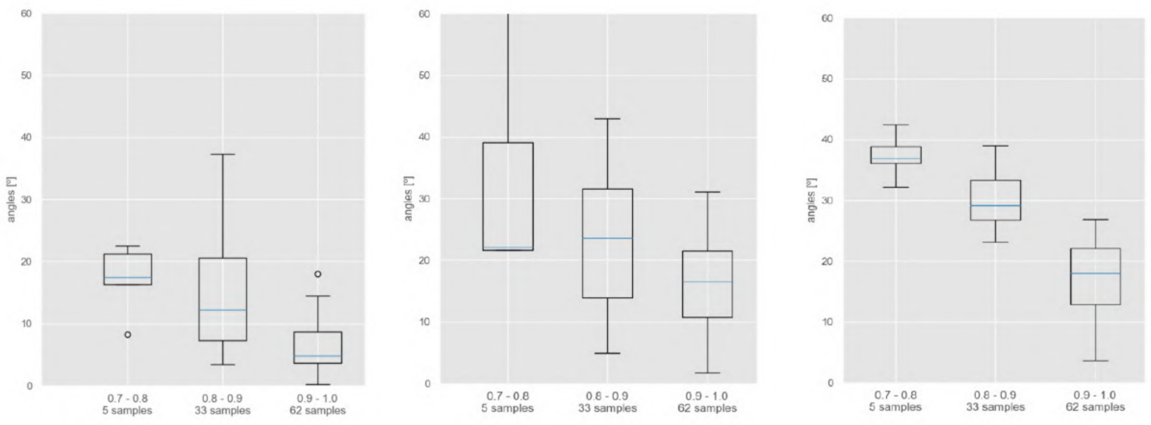

(a) Real (b) Synthetic (c) GAN

Figure 23: Saliency maps from test.



(a) Real (b) Synthetic (c) GAN

Figure 24: Test errors.

GAN. It requires a high computational load and therefore requires the use of a sufficiently powerful GPU.

In the object detection and localization problem, the models trained with real images performed better than the synthetic image models in the evaluation. This was an expected result. What is remarkable about this study is that despite being worse, the models trained with synthetic images presented a great performance against the evaluation set. In addition, for the problem at hand, the YOLO Nano model was sufficiently complex to provide correct predictions in reduced times.

In the regression problem, the model trained with real images performed better than those trained with synthetic or postprocessed images against the evaluation set. The synthetic image model and the post-processed image model performed well in the real environment when they had not been trained for a prolonged period of time. It could be seen that the attention areas of the post-processed image model were more similar to those of the real image model than those of the synthetic image model, demonstrating that they learned in a similar way. While the real image model focused attention on the part outline and shadows, ignoring the interior possibly due to lack of textures, the other models also extracted information from it. However, as they had less information from the interior of the part when facing real images, their performance was worse. Finally, the variance of the errors in the post-processed image model is much lower than that of the synthetic image model and resembles the real errors.

The saliency maps exposed that how shadows and image illumination were key aspects in the images, and, therefore, resembling these conditions in a synthetic environment would benefit the resulting model.

To sum up, a model will give the best results the more faithful the training data set is to the production environment data set. Models trained with real images will always give better results in the production environment. However, the performance of the synthetic models, even if somewhat worse, was similar and considerably good. If the same conditions can be achieved in a virtual environment as in the real environment, it is an option to consider considering the resource and time savings it presents. If they cannot be achieved directly in a virtual environment, the transfer of styles by means of a GAN offers a possibility to increase the similarity of the conditions with which the synthetic model can be improved.

## 7 Future development

It is proposed to continue and evolve this study through the following ways:

- Evolve the regression model architecture increasing its complexity and making it deal with images of different resolutions.
- Evolve or change the GAN architecture to a diffusion model that transfers the style of real images or specific lighting conditions.
- Pretrain synthetic models so that they learn to pay attention to the same areas as the real model.
- Transfer learning from one model trained with synthetic images to another to be trained with real images for less time.
- Mix real and synthetic image datasets to explore the proportion of real and synthetic images needed to perform well in the real production environment.

## References

[1] Joseph Redmon et al. “You Only Look Once: Unified, Real-Time Object Detection”. In: *CoRR* abs/1506.02640 (2015). arXiv: 1506.02640. URL: http://arxiv.org/abs/1506.02640.

[2] Joseph Redmon and Ali Farhadi. “YOLO9000: Better, Faster, Stronger”. In: *CoRR* abs/1612.08242 (2016). arXiv: 1612.08242. URL: http://arxiv.org/abs/1612.08242.

[3] Joseph Redmon and Ali Farhadi. “YOLOv3: An Incremental Improvement”. In: *CoRR* abs/1804.02767 (2018). arXiv: 1804.02767. URL: http://arxiv.org/abs/1804.02767.

[4] Alexey Bochkovskiy, Chien-Yao Wang, and Hong-Yuan Mark Liao. “YOLOv4: Optimal Speed and Accuracy of Object Detection”. In: *CoRR* abs/2004.10934 (2020). arXiv: 2004.10934. URL: https://arxiv.org/abs/2004.10934.

[5] Glenn Jocher. *YOLOv5 by Ultralytics*. Version 7.0. May 2020. DOI: 10.5281/zenodo.3908559. URL: https://github.com/ultralytics/yolov5.

[6] Shrey Srivastava et al. “Comparative analysis of deep learning image detection algorithms”. In: *Journal of Big Data* 8.1 (May 2021), p. 66. DOI: 10.1186/s40537-021-00434-w. URL: https://doi.org/10.1186/s40537-021-00434-w.

[7] CFI Education Inc. *Multiple Linear Regression*. URL: https://corporatefinanceinstitute.com/resources/data-science/multiple-linear-regression.

[8] Aiden Nibali et al. “Numerical Coordinate Regression with Convolutional Neural Networks”. In: *CoRR* abs/1801.07372 (2018). arXiv: 1801.07372. URL: http://arxiv.org/abs/1801.07372.

[9] Tomoyoshi Shimobaba, Takashi Kakue, and Tomoyoshi Ito. “Convolutional Neural Network-Based Regression for Depth Prediction in Digital Holography”. In: *2018 IEEE 27th International Symposium on Industrial Electronics (ISIE)* (2018), pp. 1323–1326.

[10] Ian J. Goodfellow et al. *Generative Adversarial Networks*. 2014. arXiv: 1406.2661 [stat.ML].

[11] Jun-Yan Zhu et al. *Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks*. 2020. arXiv: 1703.10593 [cs.CV].

[12] Martin Arjovsky, Soumith Chintala, and Léon Bottou. *Wasserstein GAN*. 2017. arXiv: 1701.07875 [stat.ML].

[13] Ishaan Gulrajani et al. *Improved Training of Wasserstein GANs*. 2017. arXiv: 1704.00028 [cs.LG].

[14] Tsung-Yi Lin et al. “Microsoft COCO: Common Objects in Context”. In: *CoRR* abs/1405.0312 (2014). arXiv: 1405.0312. URL: http://arxiv.org/abs/1405.0312.

[15] Sami Abu-El-Haija et al. “YouTube-8M: A Large-Scale Video Classification Benchmark”. In: *CoRR* abs/1609.08675 (2016). arXiv: 1609.08675. URL: http://arxiv.org/abs/1609.08675.

[16] Ziwei Liu et al. “Deep Learning Face Attributes in the Wild”. In: *Proceedings of International Conference on Computer Vision (ICCV)*. Dec. 2015.

[17] SBX Robotics. *Build Better Vision Models With SYNTHETIC DATA*. URL: `https://www.sbxrobotics.com/old-home`.

[18] Unity Technologies. *Unity*. URL: `https://unity.com/es`.

[19] Blender. *Blender*. URL: `https://www.blender.org/`.

[20] University of Edinburgh. *The Earth Mover's Distance*. URL: `https://t.ly/Ck0Tg`.

[21] © Intel Corporation. *Cámara LiDAR Intel® RealSense™ L515*. URL: `https://www.intel.la/content/www/xl/es/products/sku/201775/intel-realsense-lidar-camera-l515/specifications.html`.

[22] Universal Robots. *Brazo Robótico UR3e*. URL: `https://www.universal-robots.com/es/productos/robot-ur3/`.

[23] Maximilian Denninger et al. "BlenderProc". In: *arXiv preprint arXiv:1911.01911* (2019).

[24] Tero Karras et al. *Analyzing and Improving the Image Quality of StyleGAN*. 2020. arXiv: `1912.04958 [cs.CV]`.

[25] NVIDIA Corporation. *GeForce RTX2080*. URL: `https://www.nvidia.com/es-la/geforce/graphics-cards/compare/?section=compare-20`.

## A YOLO code modifications

The following are the modifications made to the original Ultralitycs YOLO code to show the value of the loss functions in the validation script.

In `val.py` the following lines get the loss values [lines 212-214 on GitHub]:

```
if compute_loss:
    loss += compute_loss(train_out, targets)[1]
```

However, `compute_loss` must be initialized before the for loop for the batches. To do that, import `ComputeLoss` from `utils.py` and pass `model.model` as the argument. In`val.py` model is a distributed object that raises an error if it is not done like this. Modify lines 191-198 of the original file, the ones that define the progress bar and add the `compute_loss` initialization and the losses to the progress bar.

```
s = ('%22s' + '%11s' * 9) % \
    ('Class','Images','Instances','P','R','mAP50',
    'mAP50-95','box_loss','obj_loss','class_loss')


tp,fp,p,r,f1,mp,mr,map50,ap50,map = 0.0,0.0,0.0,
                                    0.0, 0.0, 0.0,
                                    0.0, 0.0, 0.0,
                                    0.0
dt = Profile(), Profile(), Profile()
loss = torch.zeros(3, device=device)
jdict, stats, ap, ap_class = [], [], [], []
callbacks.run('on_val_start')
pbar = tqdm(dataloader, desc=s,
        bar_format='{l_bar}{bar:10}{r_bar}{bar:-10b}')

compute_loss = ComputeLoss(model.model)
```

Now the variable `loss` stores the 3 losses accumulated with each batch. Where the for loop ends the get loss lines were added before printing the results [281-285 of the original file]:

```
# Get Loss
box_loss=loss.cpu().numpy()[0]*batch_size/len(dataloader)
obj_loss=loss.cpu().numpy()[1]*batch_size/len(dataloader)
cls_loss=loss.cpu().numpy()[2]*batch_size/len(dataloader)

# Print results
pf = '%22s'+'%11i'*2+'%11.3g'*7
LOGGER.info(pf%('all',seen,nt.sum(),mp,mr,map50,
                map, box_loss, obj_loss, cls_loss))
if nt.sum() == 0:
    LOGGER.warning(f'WARNING: no labels found in {task}
                      set, can not compute metrics
                      without labels')
```

Multiplying by the batch size and dividing by the dataloader length it results in the average loss per image as the loss items are the total losses accumulated during the processing of the whole dataset.

## B Model architectures

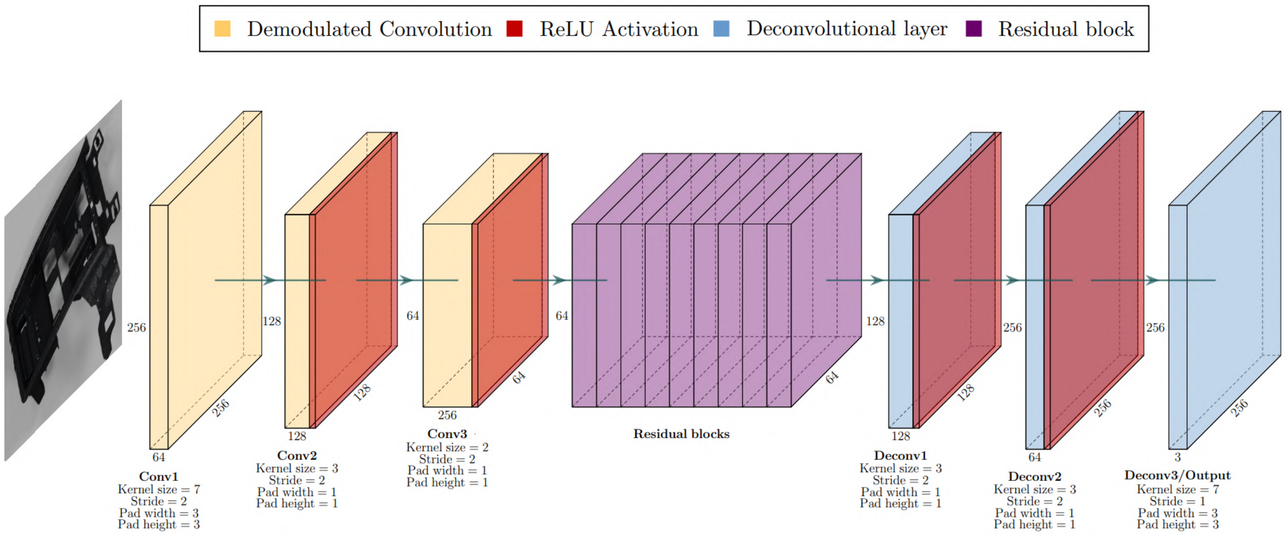


Figure 25: GAN generator architecture.

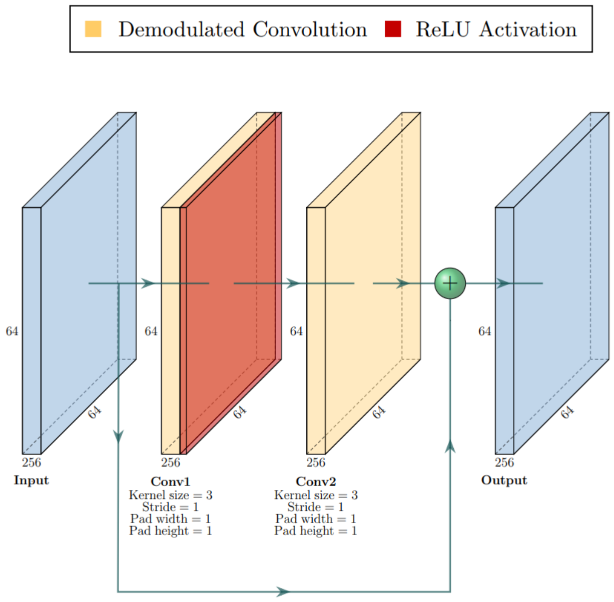


Figure 26: GAN residual block architecture.

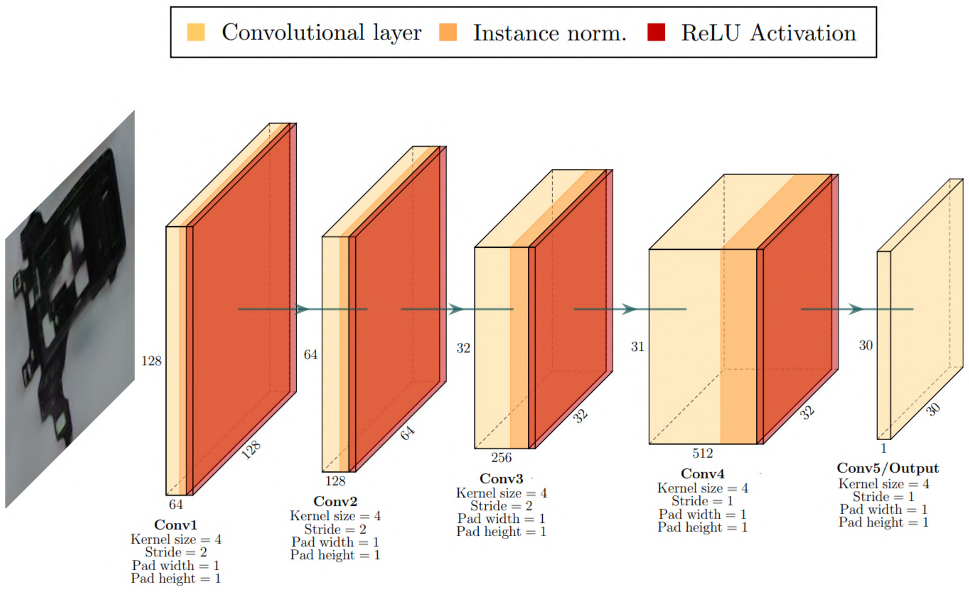


Figure 27: GAN discriminator architecture.

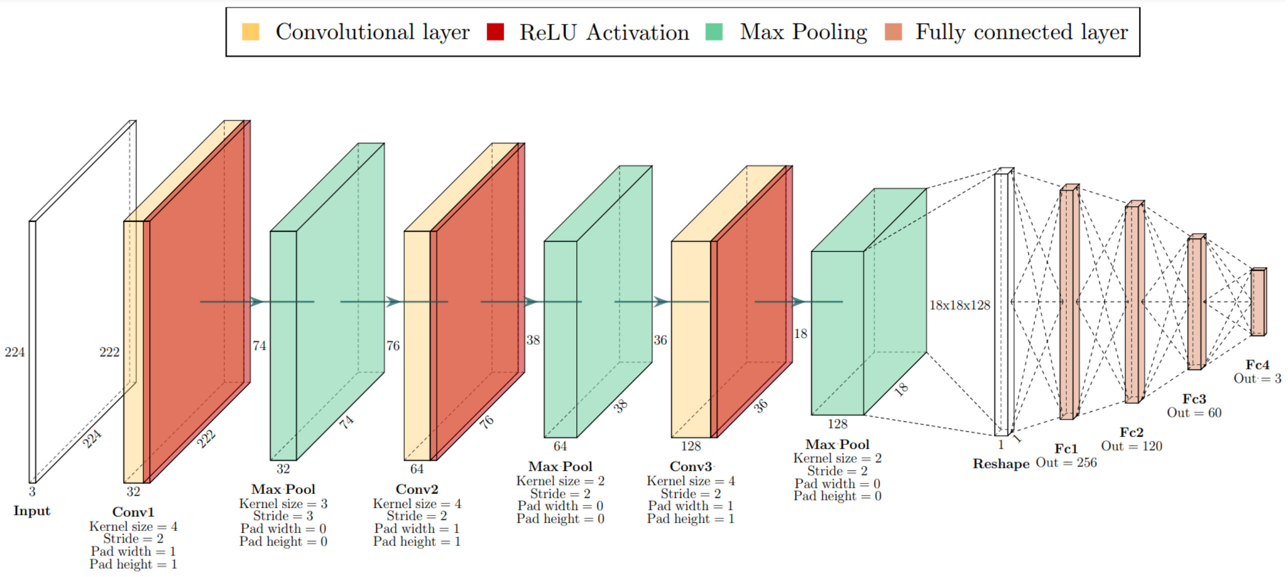


Figure 28: Regression model architecture.