MÁSTER UNIVERSITARIO EN INGENIERÍA INDUSTRIAL

Trabajo Final de Máster

Análisis comparativo de datasets de imágenes reales vs. Imágenes sintéticas en entornos industriales

Autor:
Lionel Güitta López

Directores:
Álvaro López López
Ignacio de Rodrigo Tobías

Madrid
2022-2023

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título

"Análisis comparativo de *datasets* de imágenes

reales vs. Imágenes sintéticas en entornos

industriales"

en la ETS de Ingeniería - ICAI de la Universidad Pontificia Comillas en el

curso académico 2022-2023 es de mi autoría, original e inédito y

no ha sido presentado con anterioridad a otros efectos. El Proyecto no es plagio de otro, ni total ni parcialmente y la información que ha sido tomada

de otros documentos está debidamente referenciada.

Fdo.: Lionel Güitta López Fecha: 22 / 08 / 2023

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Álvaro Jesús López López

Firmado digitalmente por Álvaro Jesús López López
Fecha: 2023.08.29 14:41:31 +02'00'

Fdo.: Álvaro López López Fecha: ……/ ……/ ……

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO

Fdo.: Ignacio de Rodrigo Tobías Fecha: 23 / 08 / 2023

MÁSTER UNIVERSITARIO EN INGENIERÍA INDUSTRIAL

Trabajo Final de Máster

Análisis comparativo de datasets de imágenes reales vs. Imágenes sintéticas en entornos industriales

Autor:
Lionel Güitta López

Directores:
Álvaro López López
Ignacio de Rodrigo Tobías

Madrid
2022-2023

Este trabajo no habría sido posible sin el apoyo de toda mi familia y amigos a lo largo de todos los años del grado y el máster.

Del mismo modo, gracias a mis directores Nacho y Álvaro por todos estos años de trabajo en tantos proyectos que culminan con este proyecto de final de máster.

Muchas gracias a todos por estar ahí apoyándome siempre en todo.

Resumen

Análisis comparativo de datasets de imágenes reales vs. Imágenes sintéticas en entornos industriales

Autor: Lionel Güitta López
Director: López López, Álvaro
Codirector: de Rodrigo Tobías, Ignacio
Entidad Colaboradora: ICAI – Universidad Pontificia Comillas

En este proyecto se busca explorar los beneficios que ofrecen los datasets de imágenes sintéticas (*renders*) en entornos industriales. Así, se desarrollaron las herramientas necesarias (imágenes, etiquetas, etc.) para generar las muestras reales y sintéticas. A partir de los conjuntos de muestras reales y sintéticas se pudo comprobar cómo se adaptan a las condiciones de trabajo real los modelos entrenados con muestras sintéticas.

Palabras clave: Visión por ordenador, Regresión, Redes Neuronales, Aprendizaje Profundo, YOLO.

1. Introducción

Durante los últimos años, se están incorporando, cada vez más, nuevos elementos a la industria, provenientes de la llamada Industria 4.0. Estos son tales como las operaciones colaborativas en robótica, la inteligencia artificial, o el *Internet of Things*. El desarrollo de estas nuevas tecnologías y su comprensión es clave para el avance de la propia industria y del aumento de la eficiencia y eficacia a la hora de aplicarlas. De este modo, en la industria, destacan los modelos de inteligencia artificial y su uso en múltiples aplicaciones como detectores de objetos, detectores de defectos, mantenimiento predictivo, etc. Existen casos en los cuales, el entrenamiento de los modelos ha sido lo suficientemente generalista para una tarea y es posible adaptar esa solución a una nueva, reduciendo el tiempo de entrenamiento o suprimiéndolo. Para un entrenamiento, se requiere de un conjunto de muestras o *dataset* lo suficientemente grande para que el modelo sea capaz de aprender, pero no siempre es posible debido a lo costoso que puede resultar en tiempo la elaboración de estos o la complejidad que puede suponer. De este modo, el uso de datasets compuestos por muestras sintéticas, o mixtos, puede suponer una gran ayuda a la hora de la elaboración de estos, reduciendo la complejidad y tiempo de creación de los mismos.

2. Definición del proyecto

En este contexto, se busca explorar los beneficios que ofrecen los datasets de imágenes sintéticas (*renders*) en entornos industriales en aplicaciones de detección de objetos y regresión para determinar la orientación de un objeto.

Para ello, se estableció el flujo de trabajo expuesto en la Figura 1. Se puede observar que se comenzó obteniendo los *datasets* real y sintético. Una vez obtenidos, se entrenaron modelos de detección de objetos y de regresión con ambos *datasets*. Además, para el modelo de regresión se añadió un tercer conjunto de datos compuesto por las imágenes sintéticas tras ser postprocesadas en una red adversaria generativa que se tuvo que entrenar para que asemejasen las imágenes sintéticas a las imágenes reales.

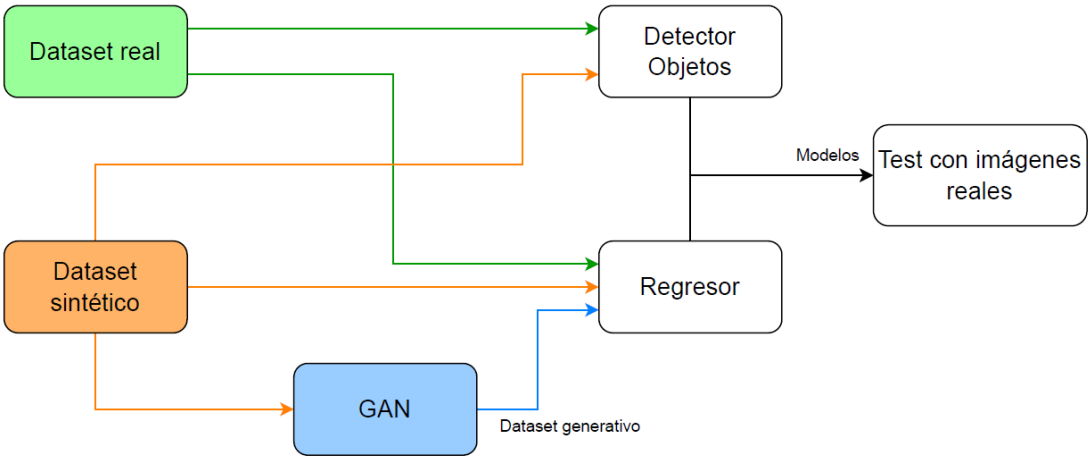


Figura 1: Pipeline del proyecto.

Una vez entrenados los modelos, tanto de regresión como de detección de objetos, estos se validaron con un dataset de imágenes reales para demostrar su eficacia en entornos de trabajo reales.

De esta forma, se desarrollaron las herramientas necesarias (imágenes, etiquetas, etc.) para generar las muestras sintéticas y reales. A partir de estas muestras, se pudo comprobar cómo se adaptan a las condiciones de un entorno de producción real los modelos entrenados con dichos datos.

3. Resultados

Siguiedo el flujo del trabajo del proyecto, se generó un *dataset* de 400 imágenes reales a partir del diseño de la infraestructura necesaría y un proceso automatizado de captura de imágenes. Por otro lado, a partir del software de renderizado de BlenderProc, se generó un dataset de más de 1.000 imágenes sintéticas que contenían el modelo CAD del objeto de interés en distintas posiciones. Debido a las diferencias entre las imágenes sintéticas y las reales, se entrenó una red adversaria generativa a partir de la cual se generó un *dataset* compuesto por imágenes sintéticas procesadas por esta para asemejarlas a las reales.

En el problema de detección y localización de objetos, los modelos entrenados con imágenes reales presentaron un desempeño mayor que los modelos de imágenes sintéticas a la hora de enfrentarlos al *dataset* de evaluación compuesto por imágenes reales. Sin embargo, los modelos entrenados con imágenes sintéticas presentaban un desempeño más que notable frente al conjunto de evaluación.

En el problema de regresión, al igual que en el anterior caso de uso, el modelo entrenado con imágenes reales presentaba un mejor desempeño que los entrenados con imágenes sintéticas o postprocesadas frente al conjunto de evaluación. Sin embargo, los otros modelos seguían mostrando un desempeño notable. Se pudo apreciar cómo las zonas de atención del modelo de imágenes postprocesadas eran más similares a las del modelo de imágenes reales que las del modelo de imágenes sintéticas. Gracias a los mapas de saliencia, que representaban la zonas de atención de los modelos de regresión, se pudo observar cómo las sombras y la iluminación de las imágenes eran aspectos clave en el caso de uso expuesto.

4. Conclusiones

Sin duda, un modelo dará los mejores resultados cuanto más fiel sea el conjunto de datos de entrenamiento al conjunto de datos del entorno de producción. Por lo tanto, los modelos entrenados con imágenes reales siempre darán mejores resultados en el entorno de producción. Sin embargo, cabe destacar que el desempeño de los modelos sintéticos, aún siendo peor, era similar y considerablemente bueno. Si se pueden conseguir las mismas condiciones en un entorno virtual que en el real, es una opción a considerar teniendo en cuenta el ahorro de recursos y tiempo que presenta. Si no se pueden conseguir directamente en un entorno virtual, la transferencia de estilos por medio de una GAN ofrece una posibilidad de aumentar la similitud de las condiciones con la que se pueda mejorar el modelo sintético.

Abstract

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

Author: Lionel Güitta López
Supervisor: López López, Álvaro
Cosupervisor: de Rodrigo Tobías, Ignacio
Collaborating Entity: ICAI – Universidad Pontificia Comillas

This project seeks to explore 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 will be possible to test how the models trained with synthetic samples adapt to real working conditions.

Palabras clave: 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. These include collaborative operations in robotics, artificial intelligence, or the Internet of Things. The development of these new technologies and their understanding is key to the advancement of the industry itself and to the increase of efficiency and effectiveness when applying them. Thus, in industry, artificial intelligence models stand out in their use cases in 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, a sufficiently large set of samples or datasets is required for the model to be able to learn, but this is not always possible due to the time-consuming and complex nature of the training. Thus, the use of datasets composed of synthetic or mixed samples can be a great help in the elaboration of these datasets, reducing the complexity and time of their creation.

2. Project definition

In this context, this project seeks to explore the benefits offered by synthetic image datasets in industrial environments in object detection and regression applications to determine the orientation of an object.

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

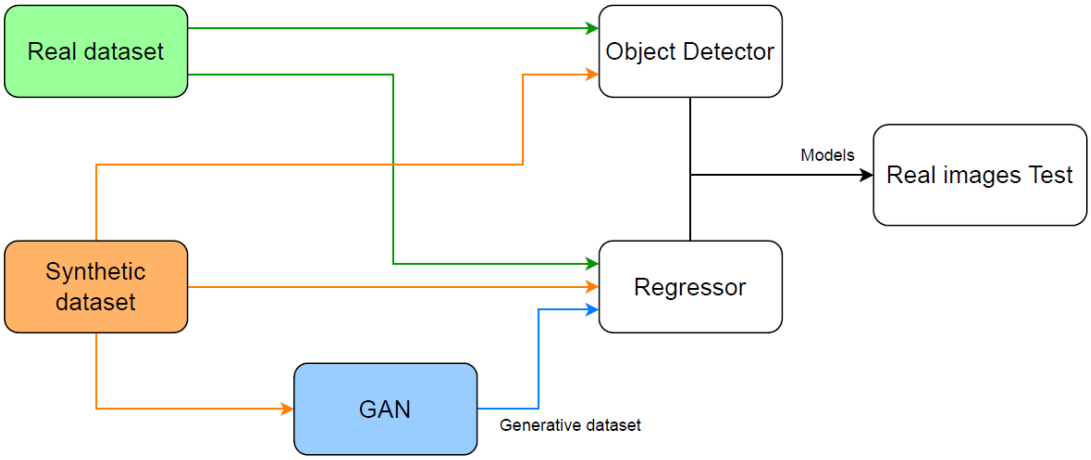


Figure 1: Pipeline del proyecto.

Once the models, both for regression and object detection, were trained, they were validated with a dataset of real images to demonstrate their effectiveness in real working environments.

In this way, the necessary tools (images, labels, etc.) were developed to generate the synthetic and real samples. From these samples, it was possible to test how the models trained with these data adapt to the conditions of a real production environment.

3. Results

Following the project pipeline, a dataset of 400 real images was generated from the design of the necessary infrastructure and an automated image capture process. On the other hand, from the BlenderProc rendering software, a dataset of more than 1,000 synthetic images containing the CAD model of the object of interest in different positions was generated. Due to the differences between the synthetic images and the real ones, a generative adversarial network was trained to generate a dataset composed of synthetic images processed by it to resemble the real ones.

In the object detection and localization problem, the models trained with real images performed better than the synthetic image models when confronted with the evaluation set composed of real images. However, the models trained with synthetic images presented a more than remarkable performance against the evaluation set.

In the regression problem, as in the previous use case, the model trained with real images performed better than those trained with synthetic or post-processed images against the evaluation set. However, the other models still showed remarkable performance. It could be seen how the attention areas of the postprocessed imagery model were more similar to those of the real imagery model than those of the synthetic imagery model. Thanks to the saliency maps representing the attention zones of the regression models, it could be seen how the shadows and illumination of the images were key aspects in the exposed use case.

4. Conclusions

Undoubtedly, a model will give the best results the more faithful the training dataset is to the production environment dataset. Therefore, models trained with real images will always give better results in the production environment. However, it should be noted that the performance of the synthetic models, even if 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 savings in resources and time 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.

Índice

Índice de figuras

Índice de cuadros

1. Introducción

Durante los últimos años, se están incorporando, cada vez más, nuevos elementos a la industria, provenientes de la llamada Industria 4.0. Estos son tales como las operaciones colaborativas en robótica, la inteligencia artificial o el *Internet of Things*.

En este punto, el desarrollo de estas nuevas tecnologías y su comprensión es clave para el avance de la propia industria y del aumento de la eficiencia y eficacia a la hora de aplicarlas. Todo ello, teniendo en cuenta la cantidad limitada de recursos de los que se dispone, incluyendo en estos el tiempo como un factor crítico en un contexto en el que los avances se dan casi a diario.

De este modo, en la industria, destaca la inteligencia artificial y la flexibilidad que presenta en su uso en múltiples aplicaciones, como detector de objetos, detección de defectos, mantenimiento predictivo, etc. Es suficiente con entrenar un modelo para realizar la tarea especificada y ponerlo en acción. Existen casos en los cuales, dicho entrenamiento ha sido lo suficientemente generalista para una tarea y es posible adaptar esa solución a una nueva. De este modo el tiempo de entrenamiento necesario se reduciría o incluso se podría llegar a suprimir.

Para los casos en los que se requiere un entrenamiento, ya bien porque se va a enseñar a un modelo desde cero, o bien porque se quiere ajustar un modelo existente a una nueva tarea, se requiere de un conjunto de muestras de entrenamiento o dataset lo suficientemente grande como para que el modelo sea capaz de aprender. Estas muestras pueden ser de una amplia variedad según la aplicación. Para usos de visión artificial, lo más probable es que sean imágenes, otros casos pueden requerir de entradas discretas de distintas características de un proceso, y así dichas muestras de entrada del modelo pueden ser del tipo de elección que se prefiera o según como esté definido el modelo ya existente.

La naturaleza de este conjunto de datos, *datasets*, debe de ser tal que, al entrenar el modelo, este se adapte lo máximo posible a la realidad, o en concreto al caso de uso en el que se vaya a aplicar el modelo, pero no siempre es posible debido a lo costoso que puede resultar en tiempo la elaboración de estos o la complejidad que puede suponer. De este modo, el uso de datasets compuestos por muestras

sintéticas, o mixtos, puede suponer una gran ayuda a la hora de la elaboración de estos, reduciendo la complejidad y tiempo de creación de los mismos.

En este contexto, en este proyecto se busca explorar los beneficios que ofrecen los datasets de imágenes sintéticas (*renders*) en entornos industriales en aplicaciones de detección de objetos y regresión para determinar la orientación de un objeto.

Para ello, se estableció el flujo de trabajo expuesto en la Figura 1. En este se puede observar que se comenzó obteniendo las imágenes que conformaron los datasets real y sintético. Una vez obtenidos se establecieron dos vías de trabajo para dos tareas diferentes e independientes. La primera tarea consiste en un entrenamiento de modelos de detección de objetos con imágenes reales y sintéticas que contenían varios objetos a detectar por imagen. Por otro lado, la segunda vía de trabajo consistía en el entrenamiento de modelos de regresión con sendos *datasets* y al que se añadió un tercero conformado por imágenes sintéticas postprocesadas por un modelo generativo que asemejaba estas a las imágenes reales.

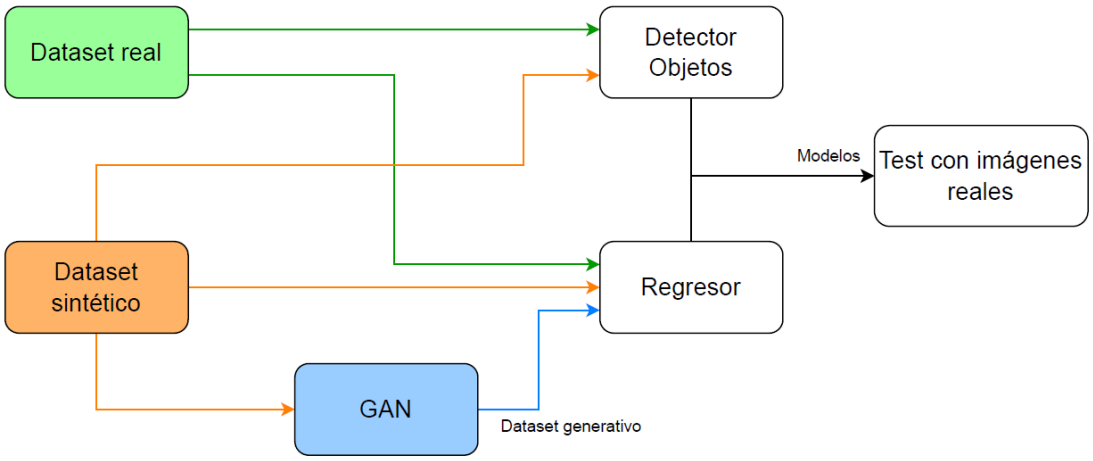


Figura 1: Pipeline del proyecto.

Una vez entrenados los modelos, tanto de regresión como de detección de objetos, estos se validaron con un dataset de imágenes reales para demostrar su eficacia en entornos de trabajo reales.

De esta forma, se desarrollaron las herramientas necesarias (imágenes, etiquetas, etc.) para generar las muestras sintéticas y reales. A partir de estas muestras se pudo comprobar cómo se adaptan a las condiciones de trabajo real los modelos entrenados con dichos datos.

1.1. Motivación

A medida que va creciendo el uso de la inteligencia artificial en aplicaciones de la industria, aumenta también la necesidad de la optimización del desarrollo de esta en todos los aspectos relacionados.

De este modo, la obtención del conjunto de muestras reales empleadas para el entrenamiento del modelo de inteligencia artificial es una parte clave del proceso. Este conjunto es determinante en el desempeño del modelo debido a que, en concreto, en aprendizaje profundo, la calidad del modelo resultante y sus resultados en inferencia en producción escalan con la cantidad de datos con la uqe se haya entrenado el modelo.

Sin embargo, la obtención de estas muestras puede ser una tarea complicada y costosa tanto en tiempo como recursos, convirtiéndose en un obstáculo para el desarrollo del modelo y bloqueando el proceso por completo. Por lo tanto, puede ser conveniente generar muestras sintéticas de forma que el consumo de tiempo y recursos se vea reducido y que estas sirvan, del mismo modo que las reales, para entrenar al modelo que luego será empleado en la aplicación real y recibirá muestras reales para inferencia.

1.2. Objetivos

Este trabajo abarca el desarrollo completo de distintos modelos de inteligencia artificial aplicados en dos problemas distintos relativos a la detección de objetos en imágenes y la obtención de la orientación de los mismos. Además, una vez desarrollados se realizará la comparación entre los mismos para extraer las conclusiones correspondientes.

El desarrollo completo de los modelos y su comparación lleva implícitos los objetivos descritos a continuación.

1. Creación de un *dataset* de 400 imágenes reales con su correspondiente etiquetado para detección de objetos y orientación de los mismos.
2. Creación de un *dataset* de imágenes sintéticas con su correspondiente etiquetado para detección de objetos y orientación de los mismos.
3. Creación de un *dataset* de imágenes sintéticas postprocesadas a través de una GAN para tener mayor similitud con el dataset real.
4. Completar los entrenamientos con los modelos *Nano, Medium* y *ExtraLarge* de YOLOv5 con los distintos datasets y sus posibles combinaciones.
5. Completar los entrenamientos del modelo de regresión con los distintos datasets y sus posibles combinaciones.
6. Detección de las características de las imágenes reales y sintéticas que más afectan a la similaridad entre las mismas y a los entrenamientos con ellas.
7. Obtención de las métricas de desempeño de los modelos en función de los *datasets* empleados y evaluación de los mismos
8. Validación de los modelos entrenados con *datasets* sintéticos con inferencia en entornos reales.

1.3. Alineación con los ODS

Actualmente, la sociedad está viviendo un momento crítico, atravesando una crisis energética y con una mayor escasez de recursos. Por lo tanto, el uso de estos ha de ser responsable ante todo. En la industria, como gran consumidor, más aún, y la nueva revolución que está sufriendo hace que sea el momento adecuado para impulsar entre otros este consumo responsable y que sea un objetivo común como se establece en el objetivo número 12 de producción y consumo responsables [1].

Además, apostando por la innovación y su implantación en la industria para aumentar la eficiencia de distintos procesos y reducir el gasto energético de recursos el proyecto se alinea con el objetivo número 9 de industria, innovación e infraestructuras.

Del mismo modo, reduciendo la carga de trabajo repetitivo, mecánico y poco ergonómico se apuesta por la salud y bienestar de las personas como se promueve en el objetivo de desarrollo sostenible número 8.

La introducción de modelos de inteligencia artificial en la industria favorece el uso de estos recursos de una manera eficiente. Así mismo, evita que operarios tengan que realizar tareas repetitivas que puedan conllevar un riesgo para su salud a largo plazo. Más allá de los procesos de producción, estos modelos pueden aplicarse para la correcta clasificación de residuos que después serán reciclados, para la identificación de defectos y su correspondiente solución o para mantenimiento predictivo que prolongue la vida útil de la maquinaria y evite que se deseche antes de lo que debería.

De este modo, esta nueva tecnología se incorpora a la industria mejorando la eficiencia de los procesos, reduciendo las tareas no convenientes para una persona y colaborando en que estos procesos formen parte de la economía circular.

2. Estado de la cuestión

Actualmente, en plena era de la información, todo está rodeado de datos que se deben procesar. Este contexto ha permitido la evolución de la inteligencia artificial en muchos ámbitos y seguirá evolucionando gracias a su gran potencial y a los beneficios que brinda a grandes empresas e industrias [2].

Dentro de este panorama destacan los sistemas de visión artificial, los cuales permiten detectar personas, objetos, desperfectos o características específicas que puedan ser más difíciles de localizar por una persona. En su mayoría, estos sistemas se basan en las redes neuronales y, en concreto, para el procesado de imágenes se suelen emplear las redes neuronales convolucionales.

2.1. Redes neuronales

Dentro de los sistemas de visión artificial predomina el uso de redes convolucionales para el procesado de las imágenes, la detección de objetos y la clasificación de los mismos.

Estas redes se basan en la aplicación de la operación de convolución, como indica su nombre, con un filtro o *kernel* sobre toda la matriz de entrada [3]. En el caso de imágenes en blanco y negro, es decir, con un solo canal, este filtro será bidimensional. Para el caso de imágenes a color, es decir, con más de un canal, dicho filtro será tridimensional. Estos filtros se desarrollarán con los entrenamientos de la red y terminarán generando las activaciones correspondientes para que la arquitectura procese la imagen correctamente en relación al problema que se desee solventar.

Los problemas principales que presentan y que se busca optimizar son la gran cantidad de imágenes y capacidad computacional que requiere su entrenamiento y evaluación y, por otro lado, la velocidad y desempeño que presentan en su aplicación real. Estas redes pueden estar compuestas por diversas arquitecturas cuyo desempeño puede variar [4] dando buenos resultados y soluciones a uno de los problemas mencionados, pero fallando en otro.

A medida que se fueron desarrollando distintos modelos para diversas aplicaciones en el procesado de imágenes, algunas arquitecturas predominaron frente a otras ofreciendo un mejor desempeño frente a los problemas expuestos. Destacan por su popularidad la *Region-proposal Convolutional Neural Network* (R-CNN) y su evolución, la *Faster Region-proposal Convolutional Neural Network* (FR-CNN) [5] o la *You Only Look Once* (YOLO) [6], todas ellas dedicadas a la detección de objetos en las imágenes procesadas.

2.1.1. YOLO

La arquitectura de YOLO se expuso por primera vez en 2015 [6] y, desde entonces, ha ido evolucionando hacia versiones posteriores [7] [8] [9] que incluyeron características como la clasificación de los objetos detectados o diseños de arquitectura cuya complejidad podía ser mayor o menor según la necesidad de la aplicación. La versión vigente al comienzo de este proyecto es la denominada YOLOv5 [10] cuya arquitectura se puede observar en la Figura 2 y actualmente se encuentra en desarrollo y constante evolución.

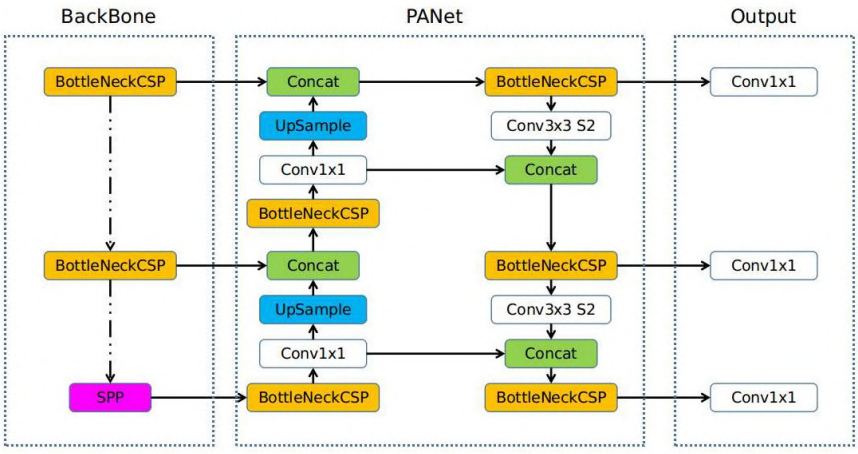


Figura 2: Arquitectura de YOLOv5. *Fuente:* [10]

YOLO destaca en gran medida por la velocidad de inferencia que presenta frente al resto de modelos [11]. En este sentido, permite ejecutar el procesado de imágenes en tiempo real para aplicaciones donde este recurso puede ser crítico.

Esta ventaja se debe a que la imagen se procesa en la red como un conjunto. Esto quiere decir que, en vez de dividir la imagen y procesar cada elemento por separado, se procesa la imagen completa por las distintas capas de la red neuronal. Por otro lado, otras arquitecturas de gran popularidad sí dividen la imagen y analizan estos fragmentos más pequeños por separado para la detección de objetos. Esto resulta en un mayor número de operaciones y, por consiguiente, un mayor tiempo de procesamiento.

Los desarrolladores principales de YOLOv5 ofrecen cinco modelos preentrenados con el *dataset* MSCOCO de más de 220.000 imágenes etiquetadas como las expuestas en la Figura 3 [12].Estos son el *Nano*, el *Small*, el *Medium*, el *Large* y el *XLarge*. La diferencia de estos modelos reside en la complejidad de los mismos afectando al desempeño, velocidad y carga computacional de estos.

Figura 3: Ejemplos del dataset MSCOCO. *Fuente:* [12]

Gracias a estos modelos preentrenados, es posible adaptarlos a aplicaciones específicas sin tener que entrenar un modelo propio desde cero. Esto conlleva el ahorro de una gran cantidad de tiempo de entrenamientos y validación, así como una considerable cantidad de energía que se emplearía en recursos computacionales.

Una práctica muy común consiste en la congelación de las diez primeras capas del modelo que se conocen como el *backbone* de YOLO. Esto implica no modificar los valores de los filtros de estas capas en el entrenamiento y modificar únicamente los de las siguientes capas para adaptar los pesos de las mismas a los objetos que se deseen detectar y clasificar. Este proceso es conocido como *transfer learning*.

2.1.2. Regresor

En el apartado anterior se ha comentado la aplicación de la inteligencia artificial como detector de objetos y clasificación de los mismos empleando YOLO, pero existen otras muchas aplicaciones de esta tecnología. Entre otras, dentro del aprendizaje supervisado, la regresión.

Cabe destacar que esta aplicación no es lo mismo que una regresión lineal que corresponde al campo de *Machine Learning*. En la regresión lineal se quiere relacionar la variable dependiente con las variables independientes a través de los coeficientes de regresión. El objetivo es determinar los coeficientes de regresión óptimos para ajustar la recta de regresión a la realidad. Las variables de entrada y la de salida se relacionan de forma lineal directamente a través de los coeficientes de regresión, no existe ninguna no linealidad en el modelo [13]. En la Ecuación 1 se puede observar cómo obtener la predicción del resultado con la regresión lineal. Donde β_p son los coeficientes de regresión, x_p son las variables independientes de entrada, ϵ es el error residual del modelo e $\hat{y}$ es la predicción del modelo.

$$\hat{y} = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + +\beta_p x_p + \epsilon \tag{1}$$

Para aplicaciones de *deep learning* el modelo incluye no linealidades entre las variables de entrada y las salidas debido a las funciones de activación, que típicamente incluyen esta característica. Además, mientras que las variables de entrada eran categóricas o numéricas para la regresión lineal, en los modelos de *deep learning* se puede emplear como variable de entrada una imagen y emplear una arquitectura convolucional. De este modo, a partir de imágenes se pueden obtener predicciones numéricas sobre valores relacionados con las mismas [14] [15]. En la Figura 4 se puede observar un ejemplo de arquitectura convolucional empleada para una aplicación de regresión a partir de imágenes.

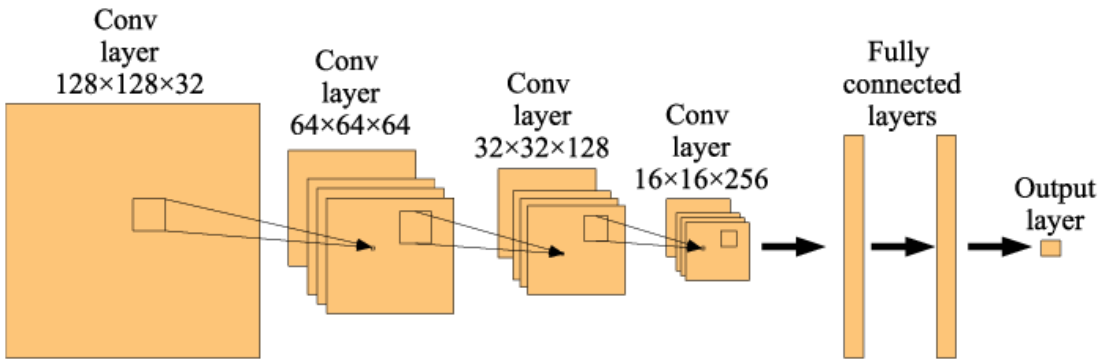


Figura 4: Arquitectura de una CNN para regresión. *Fuente:* [15]

En este proyecto se empleó una arquitectura del mismo estilo, combinando capas convolucionales y densas, para obtener las 3 componentes de un vector tridimensional que definen el vector unitario, normal a la superficie de un objeto. Con dichas componentes, un robot podría aproximarse con la dirección del vector resultante para recoger el objeto, por ejemplo, con una ventosa, donde la orientación de la misma es crítica para la efectividad del agarre.

2.2. Redes generativas adversarias (GAN)

Un tipo concreto de redes neuronales son las redes generativas adversarias, o GAN. Estas se introdujeron en 2014 y buscan entrenar 2 modelos cuyos objetivos son contrapuestos [16]. Estos objetivos consisten en que uno de los modelos, el modelo generador (G), trata de capturar la esencia de la distribución de los datos de entrenamiento para ser capaz de generar una muestra que pueda existir dentro de dicha distribución. Por otro lado, el otro modelo, el modelo discriminador (D), tiene el objetivo de saber diferenciar entre muestras reales, propias de la distribución de los datos de entrenamiento, y las muestras generadas por el modelo G.

El objetivo final del entrenamiento de este tipo de redes reside en obtener un modelo generador lo suficientemente bueno como para que el modelo discriminador acierte en sus predicciones de forma aleatoria, es decir, un 50 % de las veces. Sin embargo, un discriminador que no mejore confundirá las muestras aún cuando el generador no esté dando resultados se asemejen a las muestras reales. Por lo tanto, ambos modelos deben entrenar de forma simultánea tratando de superar al otro constantemente.

Relativo a las muestras que componen las distribuciones, estas no tienen por qué ser un dato unidimensional, pueden ser un vector, una imagen o una representación de otro tipo de datos en un espacio latente n-dimensional Z, como una palabra codificada.

Esto será determinante en la complejidad del entrenamiento del modelo G, que deberá generar una mayor o menor cantidad de datos en relación a la salida que se desee generar. Por ejemplo, no es lo mismo generar una imagen con una resolución de 224x224 píxeles que una de imagen de resolución 4k, siendo este último caso, mucho más complejo. Otro ejemplo bastante claro sería la generación de una imagen en blanco y negro o a color, donde, el segundo caso, requiere que el modelo G genere el triple de datos que el primero. Este aumendo de datos genenados tambien supone un incremento de la complejidad en la tarea del discriminador, pero, sin duda, el objetivo de este es más sencillo que el del generador, y tiene tendencia a tener errores más bajos.

Dependiendo de la arquitectura que se emplee para el modelo G, este podrá generar las muestras a partir de ruido aleatorio, un conjunto de valores específico o incluso una imagen; dando lugar a múltiples posibilidades a la hora de generar nuevas muestras.

En la Figura 5 se muestra la arquitectura básica de una GAN. En esta, el modelo generador obtiene las muestras falsas $\hat{X}$ a partir de una entrada ξ que, como se ha comentado anteriormente, puede ser de diversos tipos. Las muestras falsas generadas se juntarán con las reales X para pasar a ser las entradas del modelo D. Este tendrá que procesarlas y obtener la predicción $\hat{Y}$ sobre si cada muestra es verdadera o falsa.

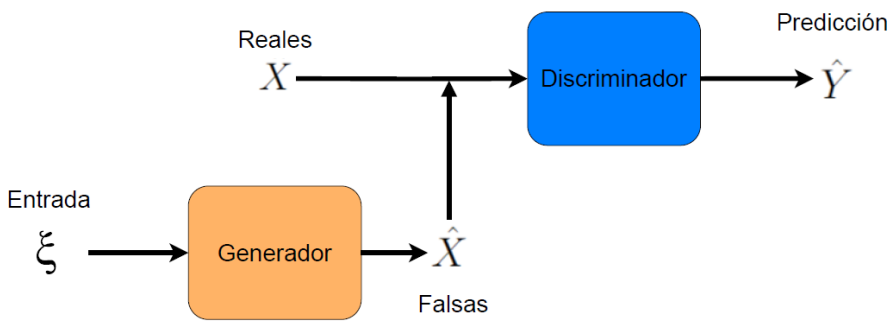


Figura 5: Arquitectura básica de una GAN.

2.2.1. Cambio en la función de pérdidas

Para que los modelos G y D puedan aprender de forma conjunta en un mismo entrenamiento, ambos deben trabajar con sus respectivas funciones de pérdidas. Estas funciones determinan el grado de error de cada modelo al procesar un número determinado de datos. En este caso, se busca que el modelo D maximice la probabilidad de asignar la etiqueta correcta, verdadera o falsa, a las muestras que recibe. Por otro lado, el modelo G busca minimizar las probabilidades de que el modelo D asigne la etiqueta correcta a las muestras que genera. Este razonamiento lleva a que el objetivo global pueda expresarse como se expone en la Ecuación 2, donde la primera parte del sumatorio representa la esperanza de que el modelo D acierte al asignarle la etiqueta a datos reales y la segunda parte del sumatorio representa la esperanza de que el modelo D de asigne la etiqueta correcta a muestras generadas por el modelo G.

$$\min_{G} \max_{D} V(G, D) = \mathbb{E}_{x \sim p_{data}(x)}[logD(x)] + \mathbb{E}_{z \sim p_z}[1 - logD(G(z))] \tag{2}$$

De este modo, como se puede observar, el modelo D va a querer maximizar dicha ecuación y, por lo tanto, su función de pérdidas se podrá expresar como dicha ecuación al completo. Por otro lado, el modelo G querrá minimizar dicha ecuación, pero solo tiene acceso al segundo sumatorio, por lo tanto, su función de pérdidas pasa a expresarse como en la Ecuación 3 que se puede transformar en la Ecuación 4

$$\min_{G} V(G) = \mathbb{E}_{z \sim p_z}[1 - logD(G(z))] \tag{3}$$

$$\max_{G} V(G) = \mathbb{E}_{z \sim p_z}[logD(G(z))] \tag{4}$$

Estas funciones se asemenjan a la popular *Binary Cross Entropy* (BCE), empleada para modelos de clasificación. Como se verá más adelante, estas funciones pueden ser modificadas por otras que aporten mayor estabilidad al modelo o que solventen ciertos problemas como la desaparición de gradientes o el colapso de modas.

2.2.2. Aplicaciones

Desde su desarrollo inicial, las GAN han evolucionado y han demostrado su validez en distintas aplicaciones como las siguientes:

- Generación de imágenes de diversos tipos, desde imágenes realistas de caras de personas a partir de ruido [17] a imágenes generadas según una entrada de texto [18]. Ejemplos de estas se pueden encontrar en la Figura 6 y en la Figura 7

Figura 6: Imágenes generadas por StyleGAN. Fuente [17]

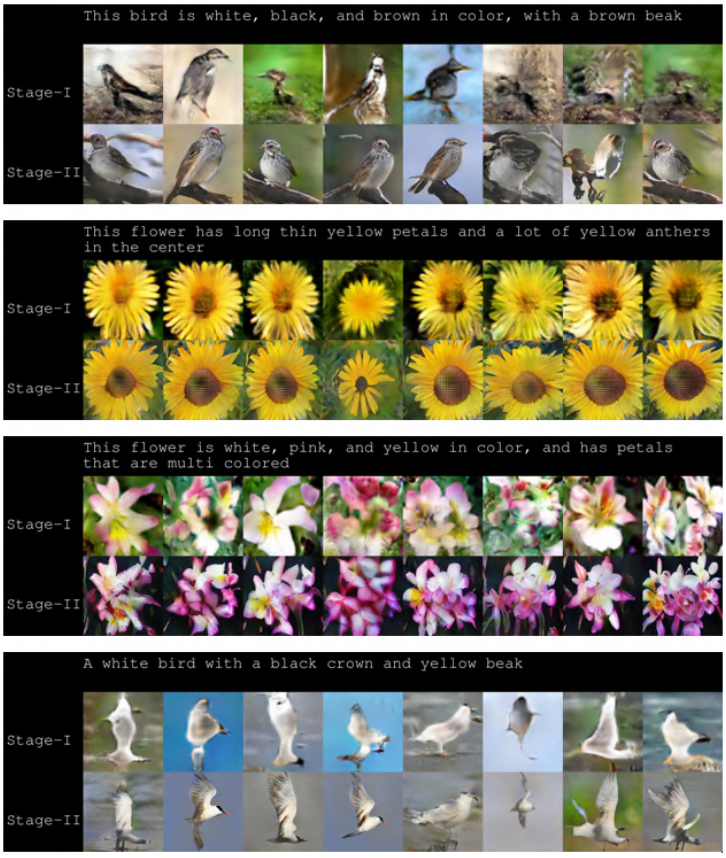


Figura 7: Imagenes generadas por StackGan desde texto. Fuente [19]

- Incremento de la resolución de imágenes para dar un resultado realista de mayor calidad [20], como la que se muestra en la Figura 8.

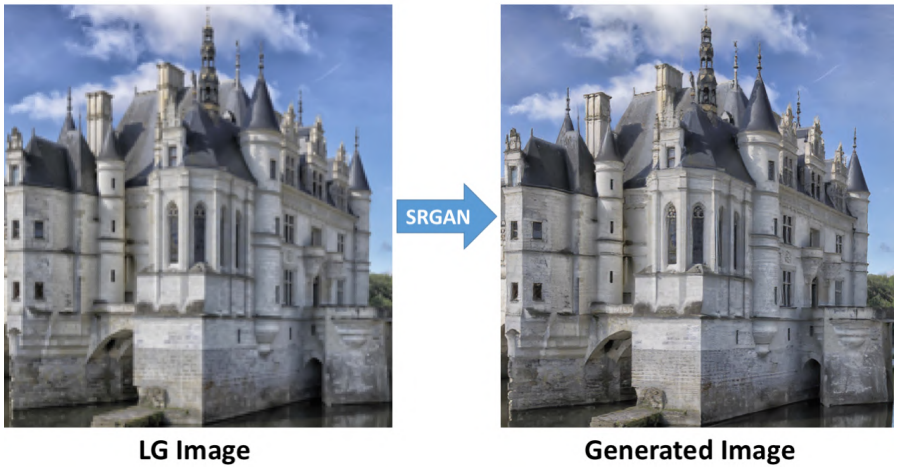


Figura 8: Imagen generada por SRGAN. Fuente [20]

- Aumento de muestras de *datasets* empleados para entrenar otros modelos de inteligencia artificial.

2.2.3. DCGAN

Una primera evolución a las primeras GANs fue la introducción de la DCGAN [21]. Esta combinó la arquitectura general de las GAN junto con las redes convolucionales (CNN) e introdujo un conjunto de reglas aplicables a la arquitectura a partir de las cuales se conseguía un entrenamiento más estable y eficiente. Estas reglas eran las siguientes:

- Reemplazar las capas de *pooling* con capas convolucionales con *stride* en el discriminador y con capas deconvolucionales en el generador.
- Utilizar normalización del lote tanto en el generador como en el discriminador.
- Sustituir todas las capas densas por una arquitectura más profunda, con más convoluciones.

- Utilizar la función de activación ReLU para el generador en todas las capas menos la última, que debe aplicar la tangente hiperbólica.
- Utilizar la función de activación de LeakyReLU para todas las capas del discriminador

2.2.4. WGAN y WGAN-GP

Uno de los problemas que presenta la función de pérdidas expuesta en la ecuación 2 es que el discriminador puede sufrir el problema de la desaparición de gradientes y dejar de aprender o ser penalizado por sus errores. Por otro lado, la definición de aprender cómo es una distribución reside en minimizar la distancia de la distribución real a una generada, pero el término de distancia puede tomar muchas formas. WGAN [22] y su evolución WGAN-GP [23] tratan de afrontar estos problemas.

La solución propuesta consiste en sustituir la función de pérdidas por una en la que se implemente el concepto de la distancia Wassertein, que de desarrollará posteriormente. Esto da como resultado la Ecuación 5.

$$\max_{\|f\|_L \leqslant 1} \mathbb{E}_{x \sim \mathbb{P}_r}[f(x)] - \mathbb{E}_{x \sim \mathbb{P}_\theta}[f(x)] \tag{5}$$

Donde:

- $\mathbb{P}_r$ es la distribución de muestras o imágenes reales.
- $\mathbb{P}_\theta$ es la distribución de muestras o imágenes generadas por la GAN.
- f es el discriminador o crítico.
- $\| f \|_L \leqslant 1$ representa que el discriminador tiene que cumplir la condición de continuidad 1-Lipschiz, que quiere decir, que su pendiente no debe ser mayor que 1.

Para cumplir con la condición de continuidad 1-Lipschiz los pesos tienen que estar limitados, lo que ocasiona que si en la fase de *backpropagation* se detecta

que estos cambian demasiado, se reducirá dicho cambio hasta el límite. Esto, en cierto modo, limita el aprendizaje del modelo, pero lo hace más estable. Además, la evolución de esta red, WGAN-GP añade un termino penalizador al gradiente como término regulador, dando lugar a la Ecuación 6.

$$\max_{\|f\|_L \leqslant 1} \mathbb{E}_{x\sim\mathbb{P}_r}[f(x)] - \mathbb{E}_{x\sim\mathbb{P}_\theta}[f(x)] + \lambda \mathbb{E}_{\hat{x}\sim\mathbb{P}_{\hat{x}}}[(\| \nabla_{\hat{x}} f(\hat{x}) \|_2 - 1)^2] \tag{6}$$

Donde:

- $\mathbb{P}_{\hat{x}}$ es la distribución de muestras generadas por interpolación de muestras reales y generadas por la GAN.
- λ es el hiperparámetro que especifica cuanto se desea aplicar la regulación.

Esto implica, entre otros aspectos, un entrenamiento más estable y un discriminador, o crítico, como se denomina en el artículo original, en el que se reduce el problema de la desaparición de gradientes. Además, la métrica de pérdidas se relaciona directamente con la distancia entre las distribuciones y lleva a una reducción del colapso de modas.

2.2.5. CycleGAN

Además de la generación de imágenes a partir de vectores de ruido, existe una aplicación asociada a la transferencia de estilos entre imágenes. Un ejemplo representativo de esta aplicación es la red de CycleGAN [24] y una muestra de su aplicación se puede ver en la Figura 9. En esta, se puede ver cómo el estilo de un cuadro se puede trasladar a una foto y en el sentido contrario, lo mismo ocurre con el estilo de una cebra a un caballo y viceversa.

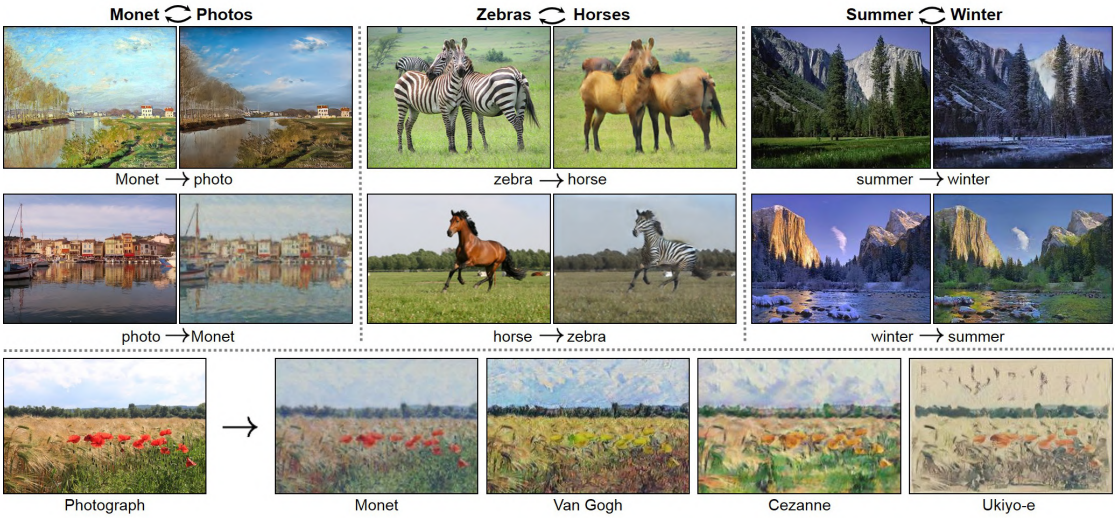


Figura 9: Aplicaciones CycleGAN. Fuente [25]

Para ello, esta arquitectura presenta dos generadores y dos discriminadores asociados a cada estilo, como se muestra en la Figura 10.

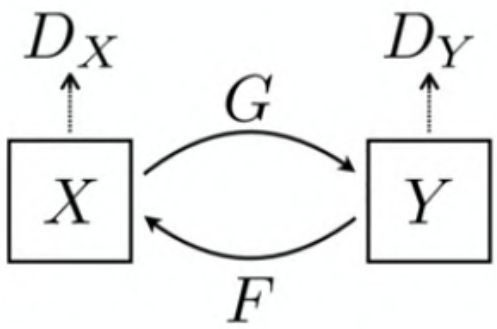


Figura 10: Arquitectura CycleGAN. Fuente [24]

Donde X e Y son imágenes con distintos estilos, G y F son los modelos generadores encargados de trasladar las imágenes originales al estilo opuesto, y D_X y D_Y son los modelos discriminadores asociados a las imágenes de cada tipo.

Esta nueva arquitectura incluye cambios en la función de pérdidas. En primera instancia, para las perdidas debidas a enfrentar a un generador contra un discriminador, el término adversario (L_{GAN}), se puede seguir empleando el término expuesto en la Ecuación 2. Sin embargo, este se puede sustituir por el término expuesto en la Ecuación 6 o sustituir por la *Mean squared error* (MSE) que ha demostrado aportar estabilidad al modelo.

Por otro lado, al término anterior se añade uno nuevo denominado *cycle consistency loss*, expuesto en la Ecuación 7. Este término se basa en que al procesar una imagen con un generador y procesarla de vuelta con el contrario. Con este procesamiento, la imagen debería quedar inalterada respecto del punto de partida. Por lo tanto, si existen diferencias estas quedan cuantificadas en este nuevo término que penalizará el error.

$$L_{cyc}(G, F) = \mathbb{E}_{x \sim p_{data}(x)}[\| F(G(x)) - x \|_1] + \mathbb{E}_{y \sim p_{data}(y)}[\| G(F(y)) - y \|_1] \quad (7)$$

Esto da lugar a que, el término de pérdidas completo, y, por lo tanto, el objetivo del entrenamiento, incluya un termino adversario para cada pareja de generador y discriminador y además un término de consistencia como se puede ver en la Ecuación 8.

$$L(G, F, D_X, D_Y) = L_{GAN}(G, D_Y, X, Y) + L_{GAN}(F, D_X, X, Y) + \lambda L_{cyc}(G, F) \quad (8)$$

Por último, para mantener de una forma fiel la composición del color de las imágenes, se recomienda la introducción de un último término, el denominado *identity mapping loss*, que se expone en la Ecuación 9.

$$L_{identity}(G, F) = \mathbb{E}_{y \sim p_{data}(y)}[\| G(y) - y \|_1] + \mathbb{E}_{x \sim p_{data}(x)}[\| F(x) - x \|_1] \quad (9)$$

2.3. Generación de *datasets*

Como se ha comentado anteriormente, para que los modelos de inteligencia artificial aprendan a extraer características de la información de entrada y sean capaces de generalizar ese aprendizaje para cualquier entrada similar, se tiene que emplear un gran número de muestras. Todas las muestras forman el *dataset*. La práctica más habitual es la de dividir el conjunto completo en tres subconjuntos, uno mayoritario para entrenar el modelo, uno más pequeño para validarlo y un tercer conjunto para evaluarlo.

Por otro lado, las muestras dependen del problema en cuestión. Estas pueden tener diversos orígenes y ser de múltiples tipos, pero para visión por ordenador, detección de objetos y clasificación de los mismos, lo más común es que sean imágenes. Las imágenes empleadas deben contener la información correspondiente que se quiera obtener y con la que pueda trabajar el modelo a la hora de afrontar el problema en cuestión.

2.3.1. Imágenes reales

Hoy en día existen empresas dedicadas a la recopilación de imágenes reales para la creación de *datasets* para después ofrecerlos para el desarrollo de modelos de inteligencia artificial. Varios casos populares de este tipo son el *dataset* de MSCOCO con el que se entrenó YOLOv5 [12], el dataset de Youtube-8M que consta de 237.000 muestras de video etiquetadas [26], o el dataset de CelebA que emplea imágenes de caras de personalidades famosas y son etiquetadas según características de la imagen. Una muestra de este último ejemplo se puede observar en la Figura 11 [27]. Estas imágenes pueden ser de mayor o menor calidad, pero suele fijarse una resolución específica para todas ellas para luego procesarlas en los modelos.

Figura 11: Ejemplo de muestras del dataset de CelebA. *Fuente:* [27]

El problema que presenta este tipo de imágenes es que para aplicaciones específicas, se deben generar imágenes dedicadas a ese problema, un proceso que puede resultar complejo y costoso tanto en tiempo como en recursos y materiales.

2.3.2. Imágenes sintéticas

Al igual que con los datasets de imágenes reales, existen empresas dedicadas a la generación de *datasests* de imágenes sintéticas para aplicaciones específicas [28]. Estas surgieron con el objetivo de solventar los problemas que presenta la generación de *datasets* de imágenes reales y gracias a aplicaciones de renderizado de gráficos y motores de videojuegos como pueden ser Unity [29] o Blender [30].

Este tipo de imágenes son tratadas para que se asemejen lo máximo posible a lo que sería una imagen real, de esta forma el modelo sería capaz de hacer inferencia con imágenes reales habiendo sido entrenado con un *dataset* de imágenes sintéticas o con una combinación de sintéticas y reales.

2.3.3. Comparación

Para poder emplear un *dataset* sintético a la hora de entrenar un modelo de inteligencia artificial es necesario que las muestras sean cuanto más fieles a la realidad como sea posible. Para el ojo humano, esto puede ser obvio, pero se debe cuantificar para valorar la calidad de un *dataset* y extraer conclusiones sobre los resultados que aporta.

Para poder cuantificar las diferencias entre imágenes existen distintas opciones, entre ellas la más simple es comparar las distribuciones de píxeles entre las 3 capas de colores de la imagen (rojo, verde y azul). Esta comparación presenta el problema de que es posible que las propias imágenes tengan la misma cantidad de píxeles rojos, azules y verdes, pero que estén ordenados de forma completamente distinta y, por lo tanto, las imágenes sean completamente diferentes. Un ejemplo de esto se puede observar en la Figura 12.

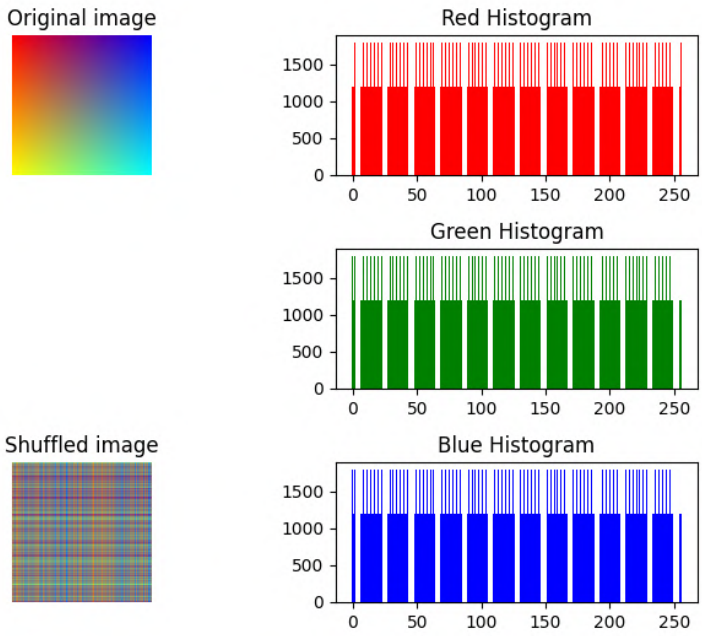


Figura 12: Ejemplo de imágenes diferentes con mismas distribuciones.

Otra aproximación, para cuantificar la semejanza entre *datasets* es calcular la distancia Wassertein entre los mismos. Esta distancia sirve para comparar distribuciones, siendo estas distribuciones de probabilidad. Es conocida como la EMD (*Earth mover's distance*), ya que se puede interpretar como si las distribuciones fuesen montañas de tierra y esta sea la energía necesaria para mover tierra de una a otra e igualarlas. En la Ecuación 10 se observa cómo se calcula para dos distribuciones F y G.

$$d_p(F,G) = \left(\int_0^1 |F^{-1}(u) - G{-}1(u)|^p du \right)^{1/p} \tag{10}$$

3. Generación de datasets

En este capítulo se desarrolla cómo se obtuvieron los distintos *datasets* que luego se emplearon en el entrenamiento de modelos. Además se incluye, una explicación de las características de las imágenes y las etiquetas que conforman el conjunto de datos para cada modelo, ya que, en cada caso, existen unos requerimientos u otros.

3.1. Dataset YOLO

Como se ha expuesto anteriormente, YOLO es una red neuronal convolucional capaz de localizar objetos en imágenes y clasificarlos. Para ello, es necesario crear un *dataset* de imágenes y sus correspondientes etiquetas para entrenar y validar los modelos. Cabe destacar que, es necesario tener en cuenta ciertos aspectos sobre las imágenes y los objetos a clasificar.

La base de coordenadas de una imagen se encuentra en el pixel situado en la esquina superior izquierda de la misma. Desde ahí, cada pixel hacia la derecha suma una unidad en el eje x y cada pixel hacia abajo suma una unidad en el eje y. Para estandarizar la medida, todas las coordenadas serán relativas al tamaño de la imagen, quedando acotadas entre 0 y 1. De esta forma, si se obtiene la medida en píxeles, se deberá dividir entre el correspondiente tamaño de la imagen en ese eje. Para una imagen de dimensiones X, Y, la coordenada (x_p, y_p) en píxeles se debería transformar como se muestra en la Ecuación 11 para obtener las unidades correspondientes a las etiquetas de las imágenes.

$$(x_l, y_l) = (\frac{x_p}{X}, \frac{y_p}{Y}) \tag{11}$$

A la hora de entrenar el modelo, es necesario definir las clases de los objetos que se quieren clasificar y enumerarlos en un archivo de configuración. Esto es importante debido a que el índice que acompañará a estos elementos será el número que se tendrá que asignar en la etiqueta de la imagen.

En este caso, las imágenes no necesitan tener una dimensión específica, ni un número de canales predeterminado. Pueden ser de cualquier forma, a color o en blanco y negro. Esto es debido a que se adaptarán posteriormente para poder ser procesadas por el modelo.

Sin embargo, para poder entrenar un modelo con un *dataset* propio, es necesario etiquetar las imágenes de una forma concreta. Cada etiqueta consta de la localización del objeto en la imagen, definida por el centro y tamaño de una *bounding box* que enmarca el objeto, y la clase del objeto en sí. Un ejemplo de imagen etiquetada se puede observar en la Figura 13

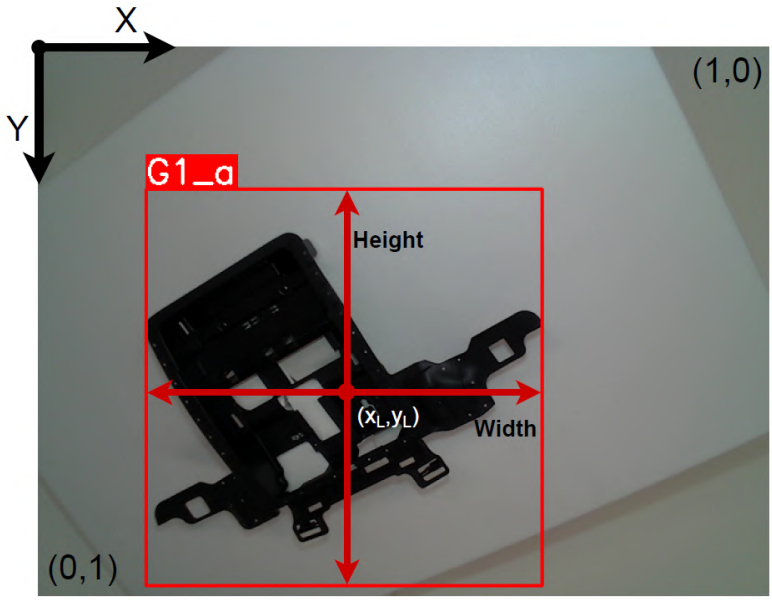


Figura 13: Elementos de una etiqueta en una imagen de YOLO.

Se debe generar un archivo de texto para cada imagen con las etiquetas correspondientes de la misma. Es decir, si en una imagen existen dos piezas que localizar y clasificar, se tendrá que generar un archivo de texto con el mismo nombre de la imagen en el cual existirán dos líneas, una para cada pieza.

El contenido de cada línea del archivo de texto debe representar una etiqueta al completo con los siguientes valores separados por un espacio en blanco:

1. Índice de la clase que representa el objeto.
2. Centro de la *bounding box* en la coordenada x de la imagen.
3. Centro de la *bounding box* en la coordenada y de la imagen.
4. Ancho de la *bounding box* en la coordenada x de la imagen.
5. Alto de la *bounding box* en la coordenada y de la imagen.

Como resultado, para una imagen con tres piezas, el archivo de texto debería estructurarse como el que se muestra en la Figura 14.

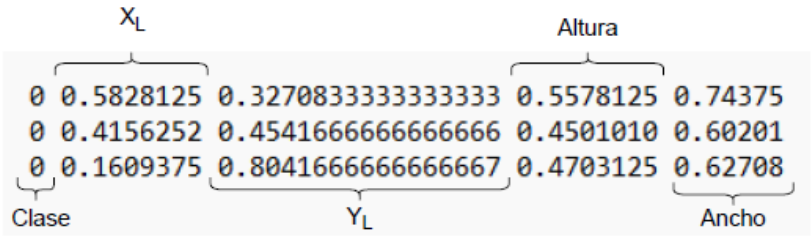


Figura 14: Archivo de texto con etiquetas de YOLO.

En este proyecto, para entrenar los modelos de YOLO, se emplearon imágenes a color, con distintas resoluciones. En ellas, solo existía una clase que debía detectar el modelo, centrando el objetivo en localizar el objeto en la imagen y no en clasificarlo. Sin embargo, dicho objeto podía aparecer en la imagen más de una vez y en diversas posiciones. En la Figura 15 se puede observar un ejemplo de este tipo de imágenes.

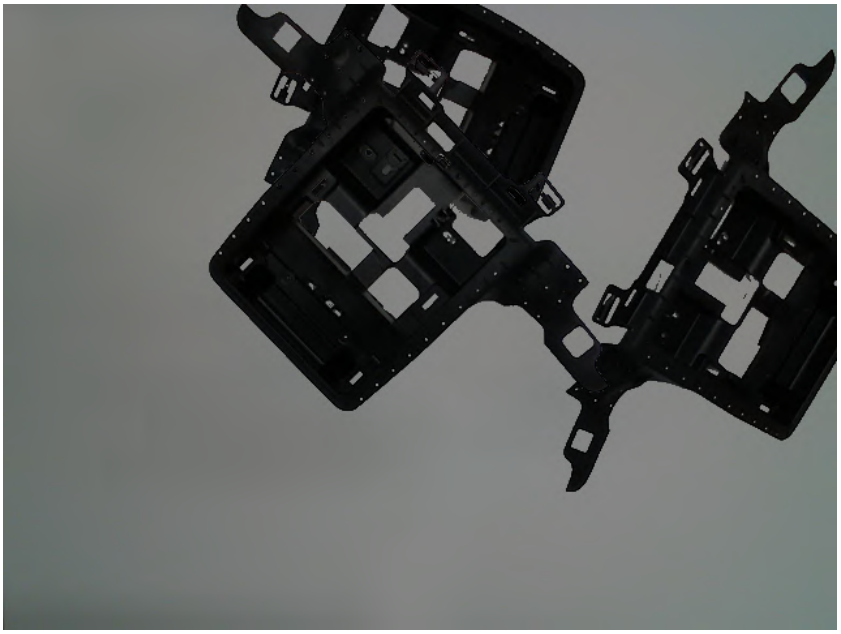


Figura 15: Imagen de entrenamiento para YOLO.

3.2. Dataset regresor

Como se ha comentado en el apartado 2.1.2, para este proyecto, el modelo de regresor que se entrenó y validó consta de una red convolucional que recibe una imagen como entrada y devuelve las 3 componentes de un vector unitario, normal a una superficie de interés del objeto de la imagen.

Al igual que con el modelo de YOLO, para este entrenamiento se debe tener en cuenta aspectos relativos a las características de las imágenes y las etiquetas de las mismas. En primera instancia, se definió el punto de interés del cual se quiere obtener el vector normal, expuesto en la Figura 16.

Figura 16: Punto de interés de la pieza G1 a.

Respecto a la imagen de entrada, el modelo programado espera imágenes con una resolución de 224x224 exclusivamente y a color, es decir, con 3 canales. Por lo tanto, todas aquellas imágenes que no tengan estas dimensiones deberán ajustarse a las mismas. Por otro lado, es importante definir la base de coordenadas de la imagen para tener una idea clara de qué representa cada componente de la etiqueta. Esto se puede observar en la Figura 17. Por último, estas imágenes representan lo que se puede interpretar como el resultado de localizar el objeto en una imagen de mayor tamaño como puede ser la salida del modelo de YOLO descrito anteriormente. Esto quiere decir que, las imágenes se pueden interpretar como las *bounding boxes* obtenidas de una imagen más grande con otras piezas. Sin embargo, en estas sólo existirá un punto de interés por cada imágen al tener solo un objeto y, por lo tanto, solo será posible extraer un vector normal completo por cada imagen, dando lugar a una sola etiqueta por imagen.

Respecto a las etiquetas, estas serán almacenadas en un único archivo de texto como el que se muestra en la Figura 18

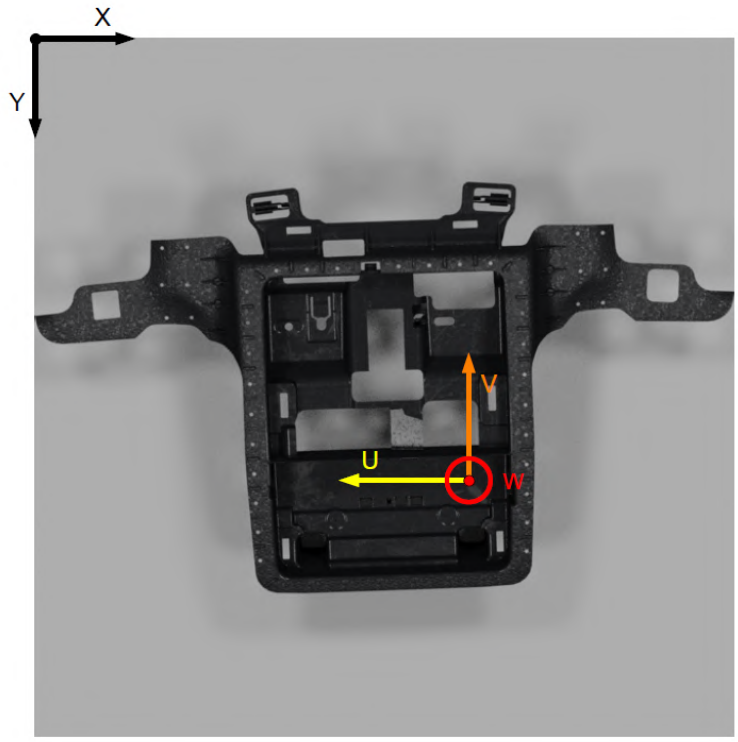


Figura 17: Base de coordenadas respecto del punto G1 a.

```
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
```

Figura 18: Archivo de etiquetas de regresor

En el ejemplo de archivo, se puede observar cómo va precedido por un encabezado, el cual indica qué significa cada valor separado por espacios en blanco en cada línea. A continuación, se expone una explicación de los mismos.

- ***Name:*** Nombre del archivo que representa la etiqueta.
- ***X:*** Coordenada x en la imagen del punto de interés.
- ***Y:*** Coordenada y en la imagen del punto de interés.
- ***U:*** Componente u del vector normal a la superficie de interés.
- ***V:*** Coordenada v del vector normal a la superficie de interés.

- ***W:*** Coordenada w del vector normal a la superficie de interés.

Cabe destacar que, a pesar de añadir los valores de las coordenadas x e y normalizadas según el tamaño de la imagen, en este proyecto no se va a hacer uso de ellas y queda para una futura iteración el desarrollo de un modelo capaz de obtener estos valores además de las componentes del vector normal a la superficie.

3.3. Dataset real

Una vez establecidas las características necesarias con las que debían cumplir las imágenes y las etiquetas de las mismas se procedió a desarrollar *dataset* de datos reales.

Por lo tanto, con el objetivo de capturar las imágenes se empleó una cámara Intel RealSense L515 [31] expuesta en la Figura 19. Esta se acompañará de un sistema de iluminación compuesto por focos LED para mantener unas condiciones de iluminación lo más constantes posibles.

Figura 19: Cámara Intel RealSense L515.

En primera instancia, se valoró cómo obtener las medidas necesarias, tanto la localización de los objetos en la imágenes como los valores de las componentes que conforman el vector normal a la superficie.

Para localizar el objeto u objetos en la imagen existían dos opciones. La primera se basaba en obtener un conjunto de imágenes y postprocesarlas posteriormente etiquetándolas manualmente con herramientas como Roboflow [32]. Por otro lado, la otra opción existente se basaba en automatizar este proceso empleando un algoritmo de detección de objetos rudimentario haciendo uso de filtros de color y realizando las operaciones convenientes para obtener las *bounding boxes*.

Por otro lado, las métricas necesarias para el modelo de regresión son más complejas de obtener. Debido a que realizar una medida manual del vector normal no es ni un proceso óptimo en cuanto a tiempo y recursos ni fiable, la única opción disponible es el uso de un brazo robótico con el que se puedan obtener dichas medidas. Para este proyecto se empleó el brazo UR3e de la marca Universal Robots [33], expuesto en la Figura 20

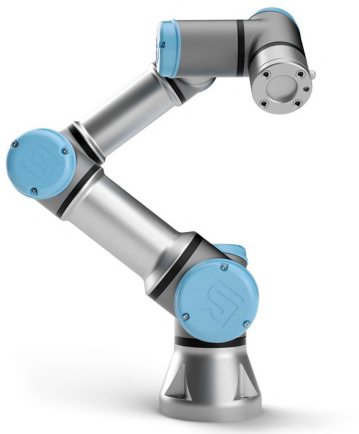
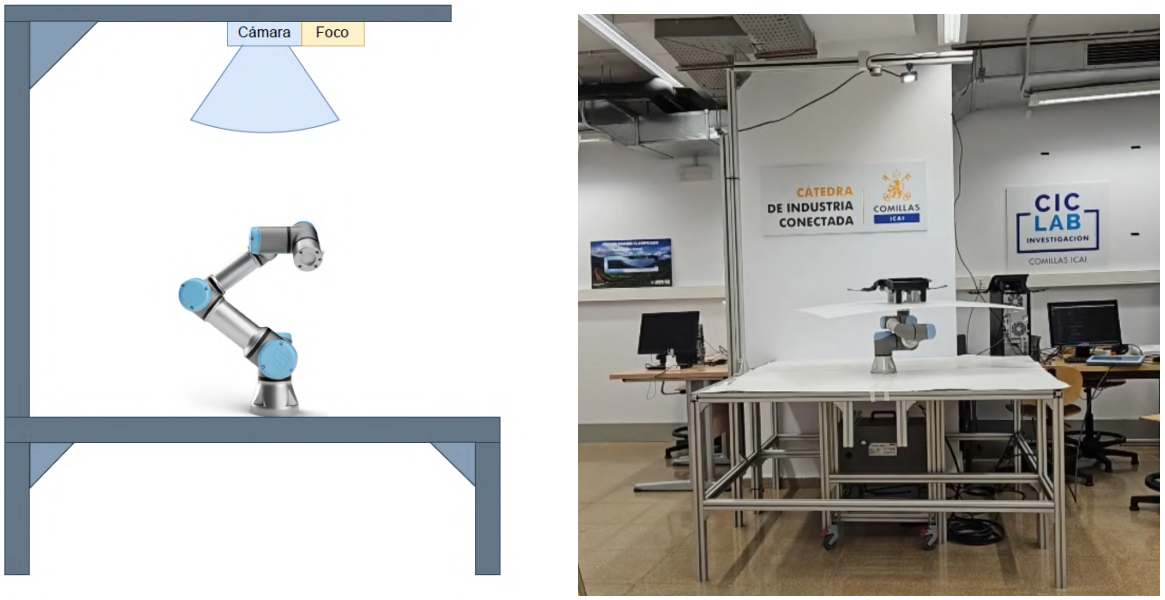


Figura 20: Robot UR3e de Universal Robots.

Con el objetivo de obtener ambos *datasets* de forma simultánea optimizando el tiempo y los recursos disponibles se procedió a combinar los procesos de captura de imágenes y extracción de las distintas métricas en un solo proceso.

La infraestructura de este proceso consistía en una estructura de trabajo con el robot UR3e en el centro donde se pudiese mover libremente. Sustituyendo la pinza que podía montarse sobre el mismo, se diseñó e imprimió una pieza 3D que se pudiese montar en el robot y sobre la que se fijaría la pieza de interés.

Para capturar las imágenes, se añadieron perfiles a la infraestructura de trabajo del robot de forma que se pudiese montar la cámara L515 y esta tomase la foto desde la vista de planta del entorno de trabajo. Como se ha comentado anteriormente, en estos perfiles se añadió también un sistema de iluminación LED para mantener una iluminación similar siempre que se capturasen imágenes. En la Figura 21 se muestran tanto el esquema como el resultado del diseño en el entorno real.



(a) Esquema (b) Escenario real

Figura 21: Infraestructura de captura de imágenes.

Por otra parte, se programó un procedimiento para capturar las imágenes y procesarlas para obtener las etiquetas correspondientes. El flujo de este se puede observar en la Figura 22 y se detalla a continuación.

El sistema desarrollado se inicia estableciendo la conexión con la cámara L515 y el robot UR3e. Una vez confirmada dicha conexión, se mueven los ejes de la muñeca del robot a posiciones aleatorias dentro de un rango. Los ejes 4 y 5 se mueven ± 30 grados respecto de su posición base mientras que el eje 6 no tiene esta limitación y se puede mover a lo largo de todo el rango de giro, cubriendo los 360 grados posibles.

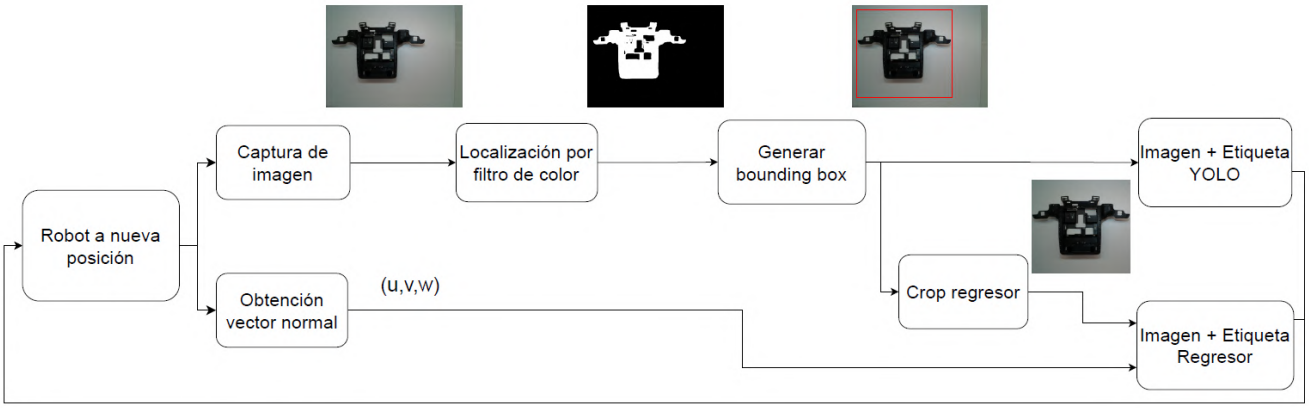


Figura 22: Proceso de obtención y etiquetado de imágenes.

Cuando el robot alcanza la posición de destino se captura una imagen con la cámara y se obtiene la medida del vector normal al TCP del robot. Dicho vector normal es el mismo que el de la superficie de interés debido a que la pieza de conexión entre el eje 6 del robot y el objeto de interés se diseñó con este propósito.

Por un lado, con el vector normal ya se tiene parte de la etiqueta necesaria para la imagen del regresor. Sin embargo, la imagen obtenida no corresponde a la necesaria para el modelo de regresión ya que debe ser un recorte de la imagen que capturado la cámara en el que solo esté presente la pieza. Además, aún no se tiene la localización del punto de interés en la imagen.

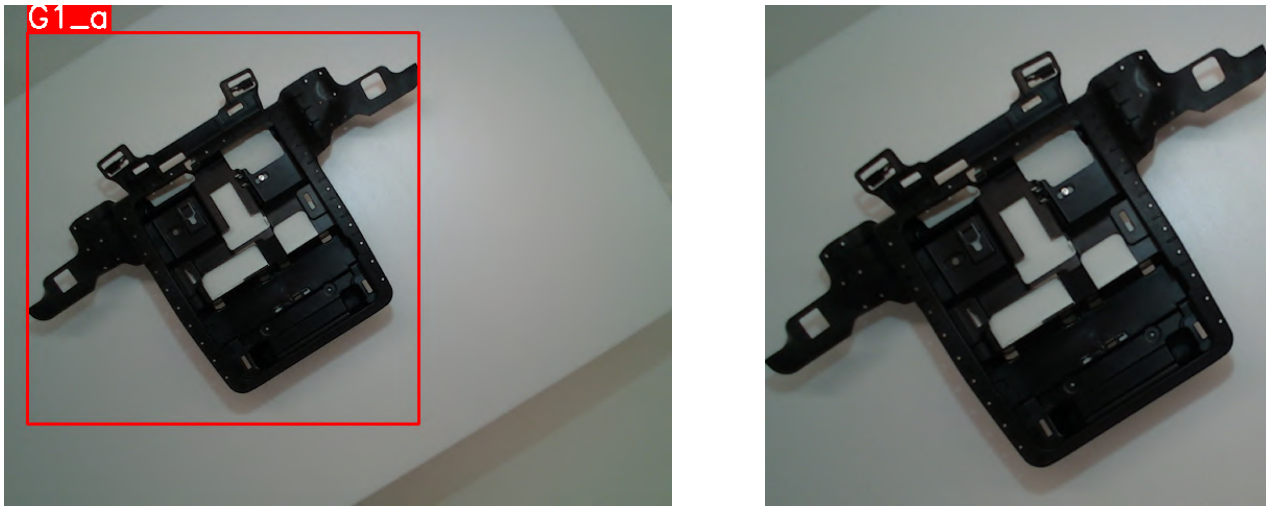
Por otra parte, para el *dataset* del modelo de detección y localización de objetos, esta imagen sí es válida, pero todavía no se tiene la etiqueta necesaria.

Por lo tanto, dado el color característico de la pieza y al contraste con el fondo, con un filtro de color se procede a extraer la etiqueta para el modelo de detección de objetos. Las primeras medidas que se obtienen generan una *bounding box* rectangular, perfectamente ajustada a la pieza. Para luego obtener el recorte correcto para el regresor, estas medidas se ajustarán para que el ancho y alto de la *bounding box* tengan una relación de 1:1 sin dejar fuera ninguna parte del objeto. Además, con estas medidas también quedan definidas las coordenadas del punto de interés de la pieza dentro de la *bounding box*.

Con las medidas obtenidas, ya se tiene la etiqueta completa y la imagen para el *dataset* del modelo de detección de objetos. No obstante, para el regresor, aunque ya se dispone de la etiqueta al completo, la imagen sigue sin corresponder. El último paso que falta es recortar la imagen obtenida por la cámara en la zona que define la *bounding box*.

Una vez completada esta iteración, ya se tiene una muestra completa para cada *dataset*. El proceso continua llevando el robot a una nueva posición aleatoria y repitiendo el ciclo tantas veces como sea necesario.

En la Figura 23 se pueden observar ejemplos de los dos tipos de imágenes obtenidos durante el proceso.



(a) Detección de objetos (b) Regresión

Figura 23: Imágenes reales obtenidas en el proceso de generación de *datasets*.

3.3.1. Modificaciones al *dataset* del modelo de detección de objetos

A pesar de haber obtenido un conjunto de muestras válido para el modelo de detección de objetos, en el entrenamiento se va a partir de modelos preentrenados, como se expondrá más adelante. Estos modelos son realmente potentes, y para poder aprovechar toda su capacidad y plantear un reto conveniente, se decidió alterar las imágenes del dataset obtenido.

La capacidad de este modelo reside en la detección y localización de múltiples objetos de distintas clases en distintas posiciones de la imagen. Como en este proyecto solo se va a trabajar con un objeto de una sola clase, este aspecto queda sin cubrir. Sin embargo, al igual que se ha hecho en el proceso anterior, de cada imagen se puede extraer la pieza deseada y agrupar varias en una misma foto.

Por lo tanto, aplicando el mismo filtro de color se pasó de tener una pieza por imagen a tener entre una y 4 piezas por cada imagen con sus correspondientes etiquetas. Cabe destacar que estas se podían superponer al igual que pudiese ocurrir en un caso real donde las piezas estuviesen en una caja dispuestas unas sobre otras.

Para evitar incongruencias en la imagen, las piezas se insertaron sobre un fondo blanco homogéneo igual al capturado por la cámara en las imágenes originales, pero cubriendo toda la imagen.

En la Figura 24 se puede observar un ejemplo de cómo son finalmente las imágenes del *dataset* del modelo de detección de objetos.

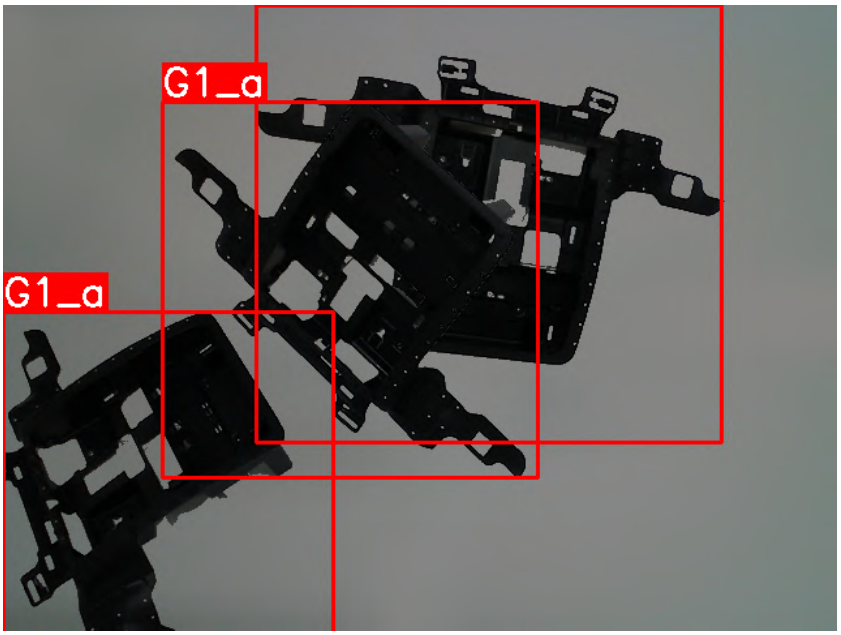


Figura 24: Muestra final del *dataset* del modelo de detección de objetos.

3.4. Dataset sintético

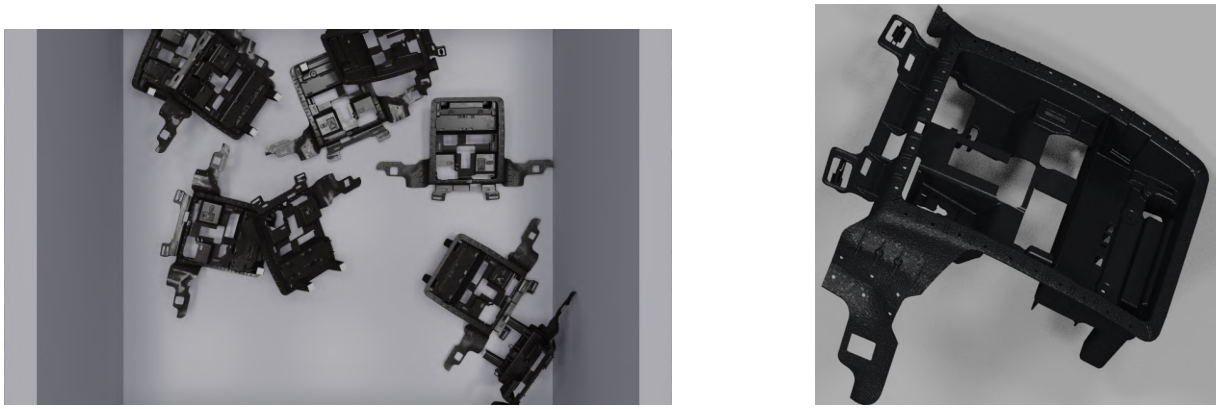
De forma paralela a la generación del dataset de muestras reales, se procedió a generar el dataset de muestras sintéticas. La generación de este dataset se desarrolló como un proceso automatizado en el que, partiendo de un modelo CAD de la pieza, se hizo uso de la herramienta de Blenderproc [34] para obtener muestras tanto del *dataset* del modelo de detección de objetos como para el de regresión.

El proceso consistía en la modelación de un escenario compuesto por una caja en la que se generarían numerosas muestras de las piezas en distintas posiciones. A partir de aquí, se puede establecer una vista desde la cual se quisiese renderizar una imagen para así obtenerla. Además, se tiene un control completo de las posiciones y orientaciones de las piezas, a diferencia del escenario real. Por lo tanto, la extracción de las medidas correspondientes para las etiquetas fue mucho más sencilla.

En este caso, se podían incluir muchas más piezas por imagen en el escenario. Lo cual supone una ventaja frente al escenario real, ya que, se puede aprovechar más la capacidad de generación de imágenes y se puede probar en mayor medida los límites del modelo desarrollado.

El único limitante en este escenario era el tiempo de computación. Cuanto mayor fuese la resolución de las imágenes de salida del proceso, mayores recursos de computación requería y mayor tiempo. Sin embargo, este sigue siendo ligeramente superior al tiempo de procesado completo de las imágenes reales.

En la Figura 25 se pueden observar ejemplos de las muestras sintéticas obtenidas en este proceso. Cabe destacar que no son exactamente iguales que las imágenes obtenidas con la cámara en condiciones reales y se aprecia diferencia entre estas y las de la Figura 23



(a) Detección de objetos (b) Regresión

Figura 25: Imágenes sintéticas obtenidas.

3.5. Dataset sintético generativo

Debido a las diferencias que existen entre las imágenes reales y las sintéticas se decidió desarrollar un tercer *dataset*. Este último *dataset* se generó tratando de aumentar la semejanza entre las imágenes sintéticas y reales debido a que las diferencias existentes entre las mismas pueden suponer problemas en la transición de un entorno sintético al real.

Para cumplir con este objetivo, se entrenó una red generativa adversaria basada en CycleGAN. De esta forma se puede transferir el estilo de las imágenes reales a las sintéticas. No se empleó una red generativa que generase imágenes a partir de ruido porque se perdía la seguridad de que la imagen que se obtuviese corresponda con la etiqueta correcta y además sería un entrenamiento mucho más complicado. Sin embargo, bajo la arquitectura de CycleGAN la parte de interés a la hora de pasar al entorno de producción es el modelo generador, que transforma imágenes sintéticas en reales. De esta forma, la entrada sería una imagen que tiene su etiqueta correctamente generada y asignada y esta se pudo mantener para la imagen de salida.

Inicialmente se realizaron entrenamientos con arquitecturas como las que se pueden observar en la Figura 26, la Figura 27 y la Figura 28. La imagen de partida con el estilo A, se procesa por la arquitectura del generador dando lugar a una imagen con el estilo B que servirá de entrada a la arquitectura del discriminador.

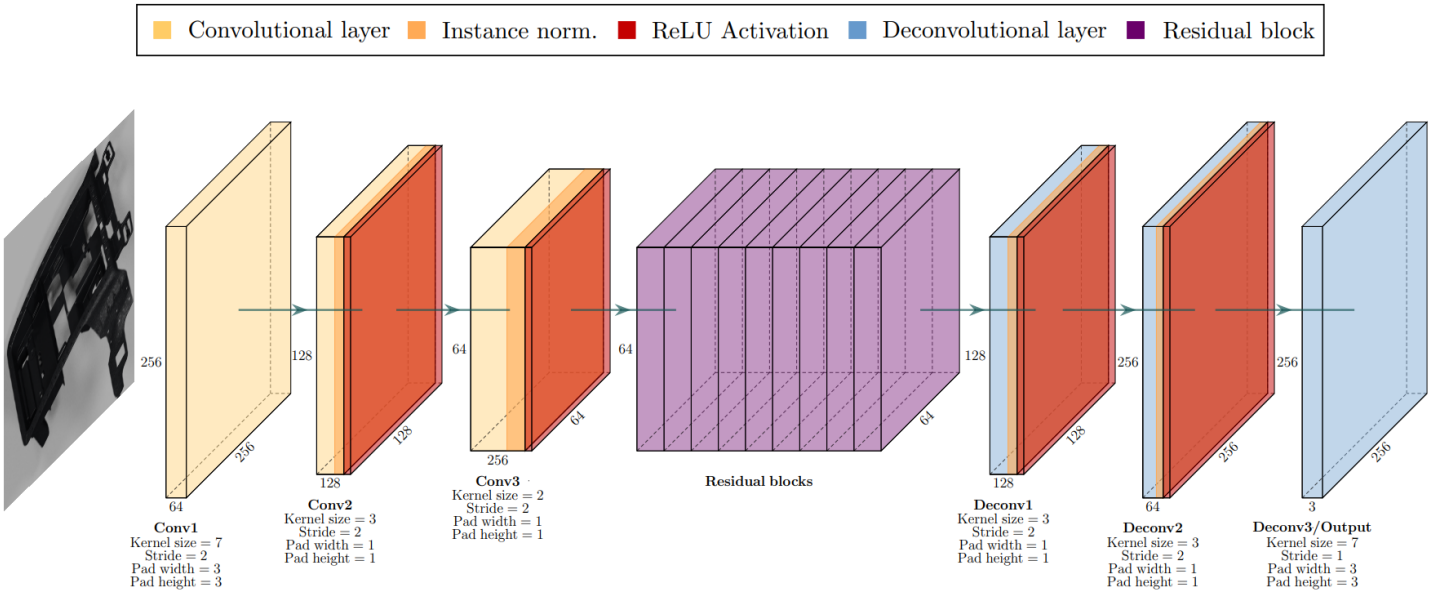


Figura 26: Arquitectura del generador en CycleGan.

Como se puede apreciar, el generador está basado en una estructura similar a la de un *autoencoder*. En este, se pueden identificar principalmente tres partes que componen la estructura completa.

La primera parte del generador consiste en una sucesión de capas convolucionales que van reduciendo las dimensiones de la imagen de entrada a medida que el procesamiento avanza por las capas de la red. En este bloque también encontramos operaciones de normalización tras varias operaciones de convolución que reduce el contraste excesivo de una imagen en concreto para que no afecte a la generalización del entrenamiento. En este bloque se parte de una imagen de 256x256 píxeles a color, es decir, con 3 canales que se reduce hasta un tensor dimensiones 256x64x64.

El siguiente componente del generador consiste en 9 bloques denominados residuales como el que se puede observar en la Figura 27. Cada uno de estos bloques está formado por dos capas convolucionales que procesan el tensor de entrada dando lugar a otro tensor con las mismas dimensiones. La salida del bloque consiste en la suma elemento a elemento del tensor de entrada y el tensor procesado por las dos capas convolucionales. Como se puede observar, en cada bloque se implementa la normalización de instancia. Al final, como salida del conjunto de bloques residuales se obtiene un tensor como el de entrada con dimensiones de 256x64x64, pero más procesado.

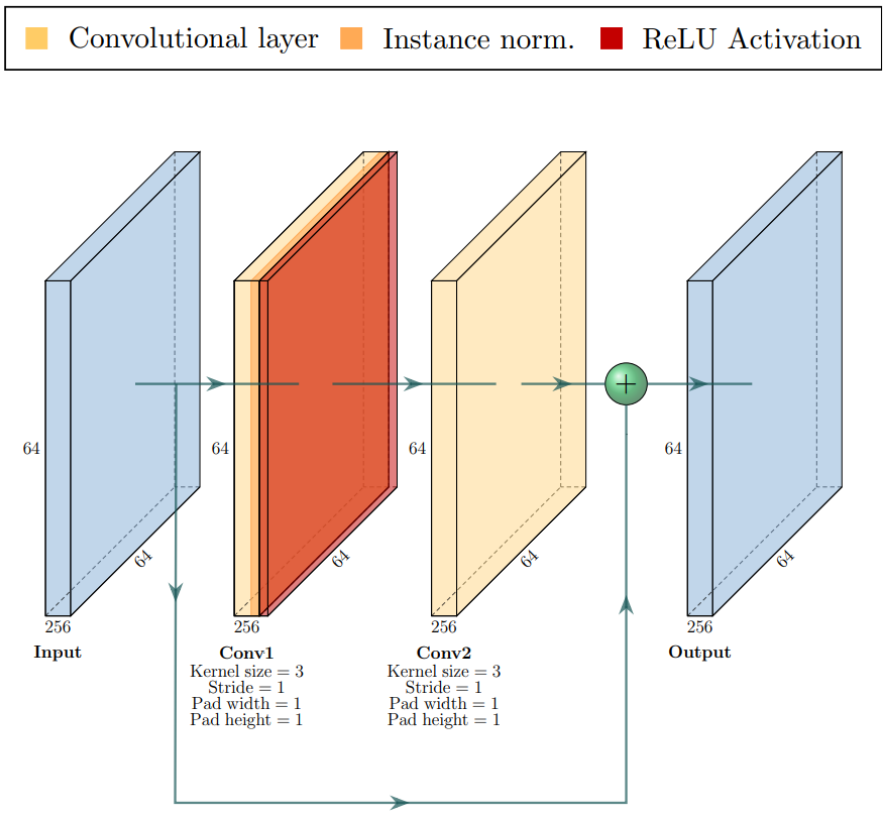


Figura 27: Arquitectura del bloque residual en CycleGan.

Por último, tras el conjunto de bloques residuales, el generador presenta una sucesión de capas deconvolucionales que aumentan las dimensiones del tensor de entrada hasta las mismas que la imagen de entrada original. Del mismo modo que en casos anteriores, se observan normalizaciones de instancia para realizar un procesamiento que beneficie la generalización en los datos de entrenamiento. Como salida se obtiene una imagen con las mismas dimensiones que la imagen original, pero, mientras que la imagen original se identificaba con el estilo A, la imagen se identifica con el estilo B si el entrenamiento ha sido efectivo.

Por otro lado, la arquitectura del discriminador, visible en la Figura 28 es común a cualquier clasificador convolucional. Toma una imagen de entrada y esta se procesa por varias capas convolucionales con sus respectivas activaciones hasta un resultado final. En este caso, en vez de reducir la imagen a una salida única como puede ser un valor binario de verdadero o falso, el discriminador produce una matriz de predicciones de dimensiones de 30x30 píxeles. En esta matriz de predicciones cada elemento hace referencia a una zona de la imagen y lo que representa

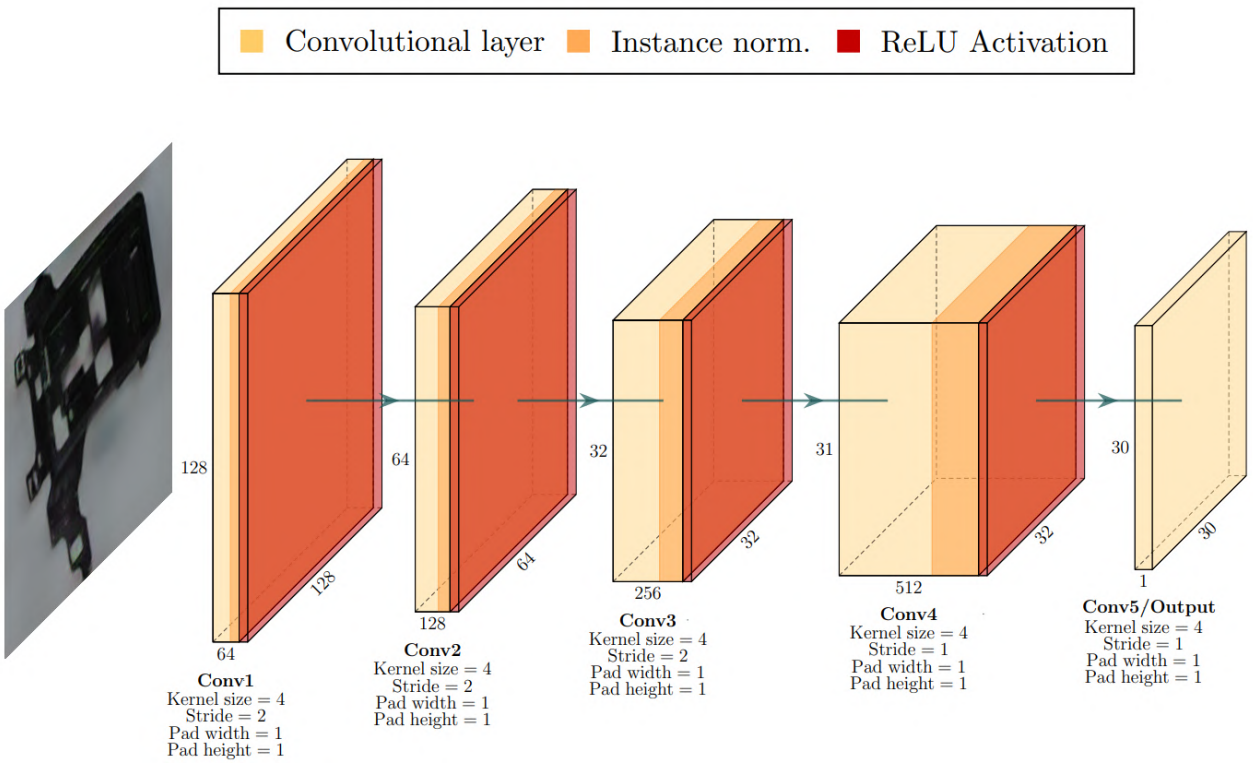


Figura 28: Arquitectura del discriminador en CycleGan.

es si clasifica esa zona como parte de la imagen original o no. El objetivo ideal es que, a medida que avance el entrenamiento y el generador mejore lo suficiente, el discriminador no sepa identificar correctamente ninguna de las zonas de la imagen generada por el generador, indicando que la imagen se parece a las del estilo que se quiere obtener.

En la Figura 29 se pueden observar las curvas de los valores de pérdidas del generador y el discriminador a lo largo del entrenamiento para el conjunto de imágenes dedicadas a optimizar el modelo y el conjunto de validación.

Como se puede observar, la pérdida asociada al generador desciende de forma correcta a medida que avanza el entrenamiento hasta un límite que no consiguió superar, posiblemente un mínimo local. Por otro lado, la pérdida asociada al discriminador aumenta ligeramente ya que el generador va aprendiendo a generar imágenes más parecidas a las reales.

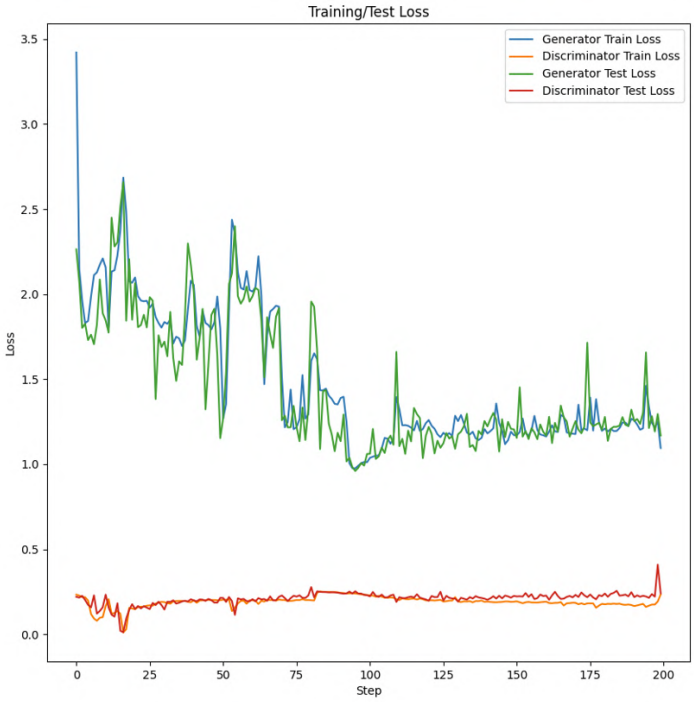


Figura 29: Pérdidas de GAN con arquitectura de StyleGAN.

Cabe destacar que en los primeros entrenamientos se observó la generación de artefactos a partir de cierto punto del proceso. Un ejemplo de este caso se puede observar en la Figura 30.

Figura 30: Imagen generada con artefacto.

Tras investigar sobre cómo prevenir estos artefactos, se procedió a modificar la red para implementar algunos cambios que se introdujeron en la evolución de la red de StyleGAN a StyleGANv2.

Estos cambios se basaban en suprimir el *batch normalization* y las capas convolucionales por capas convolucionales demoduladas en el generador. Este tipo de capas añade transformaciones a los pesos de la capa para evitar cambios abruptos que puedan ocasionar artefactos. Dicha transformación se rige por la Ecuación 12.

$$w'_{i,j} = \frac{w_{i,j}}{\sqrt{\sum_{i,j} w_{i,j}^2 + \epsilon}} \tag{12}$$

Por lo tanto, la arquitectura del generador modificado sería como la que se muestra en la Figura 31 donde cada bloque residual es como el que se muestra en la Figura 32.

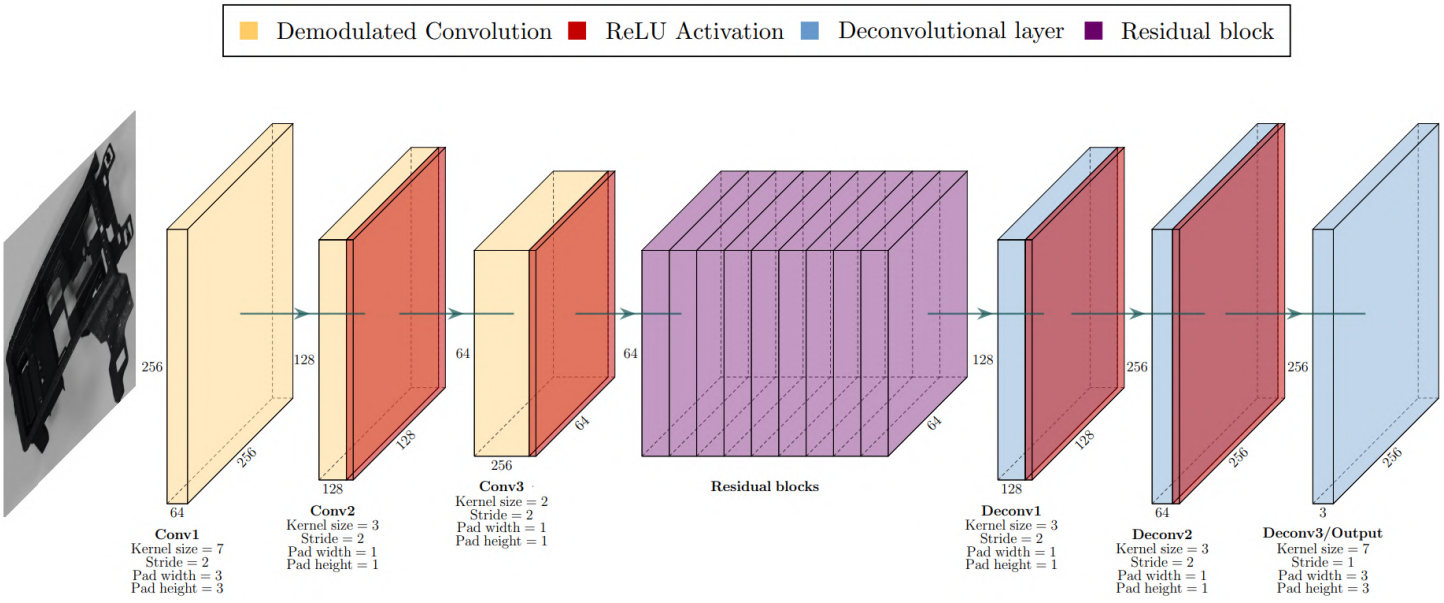


Figura 31: Arquitectura del generador modificado.

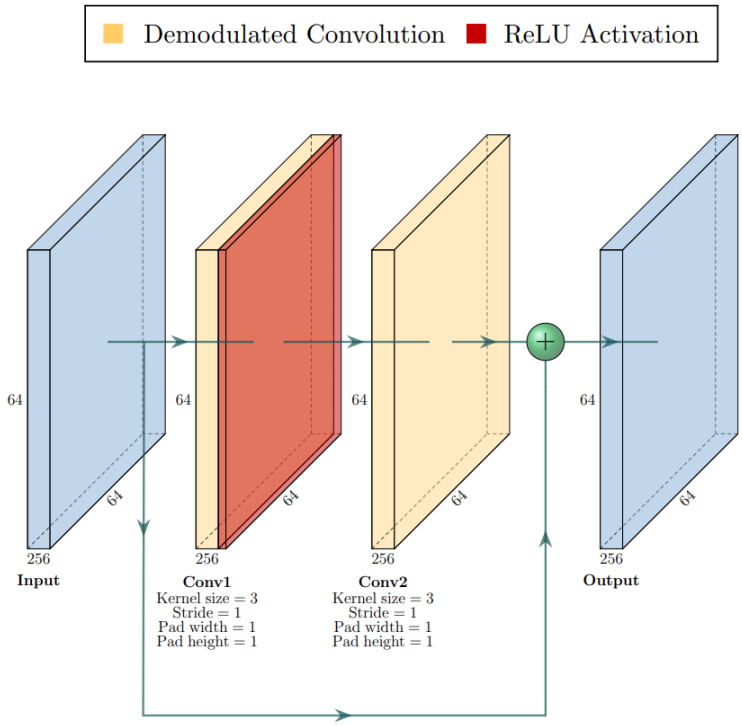


Figura 32: Arquitectura del bloque residual modificado.

A continuación, en la Figura 33 se exponen las curvas de pérdidas asociadas al generador y discriminador para el entrenamiento de la nueva arquitectura. Se puede apreciar una evolución como la del anterior entrenamiento en la que el generador reduce sus pérdidas hasta un límite mientras que el discriminador las aumenta ligeramente. Además cabe destacar un signo de overfitting en el modelo del generador a partir de la época 200, donde la pérdida de los conjuntos de entrenamiento y validación se empiezan a separar.

De esta manera se consiguió solventar el problema de los artefactos generados anteriormente. Sin embargo, se debía entrenar con un *learning rate* más bajo y durante más tiempo para obtener resultados aparentemente correctos y sin la presencia de los artefactos. Un ejemplo de las nuevas imágenes generadas se puede observar en la Figura 34.

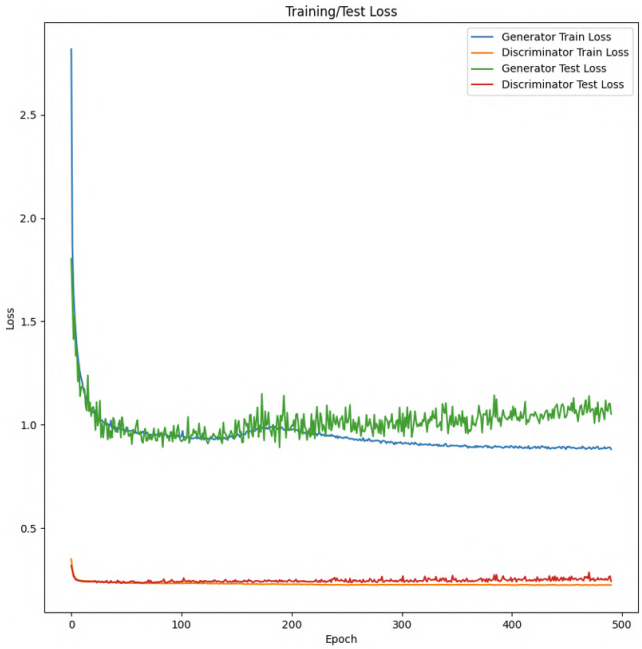


Figura 33: Pérdidas de GAN con arquitectura de StyleGAN modificada.

Figura 34: Imagen generada sin artefacto.

Con esto se completó el *dataset* sintético generativo y el conjunto de *datasets* global. Se disponía de uno real, un *dataset* generado en un software de renderizado y un último *dataset* igual al anterior, pero con un postprocesado extra para el aumento de la similaridad con el dataset de imágenes reales. De esta forma, se procedió con los entrenamientos en los distintos casos de uso para verificar la validez de los distintos conjuntos de datos.

4. Caso de uso: Detección de objetos en imágenes

En este capítulo se va a describir en detalle la metodología de los entrenamientos de los modelos de detección y localización de objetos en las imágenes a partir de la arquitectura de YOLO. En primera instancia se comentarán los modelos entrenados, seguido de una explicación de los entrenamientos ejecutados sobre el *dataset* real y el *dataset* sintético. Finalmente, una vez completados los entrenamientos, con un conjunto de datos nunca visto por los modelos se realizaró inferencia sobre los mismos. Se expondrán tanto los resultados de los entrenamientos como los de inferencia y se detallarán unas breves conclusiones.

El código empleado para los entrenamientos de este modelo se extrajo directamente del repositorio de YOLOv5 de Ultralytics en Github [10]. Cabe destacar que se hicieron algunas modificaciones sobre el código de validación de los modelos para obtener ciertas métricas que se extraían en los entrenamientos, pero que no estaban completamente implementadas en la validación. Las modificaciones incluidas se detallan en el Anexo A.

Debido a que la arquitectura de YOLO ha sido ya probada en numerosos casos y se ha comprobado su capacidad detectando y clasificando objetos de más de mil clases distintas, en este estudio no se van a entrenar los modelos desde cero. Para este proyecto se va a implementar la técnica de *transfer learning* con la cual se va a adaptar un modelo existente, ya entrenado, capaz de detectar una gran diversidad de objetos en distintas imágenes, al problema de detección que se está tratando.

La arquitectura de YOLO se puede dividir en dos partes principalmente, el *backbone* y el *head*. En este caso, para todos los entrenamientos, se comenzó con el modelo preentrenado y se congelaron los pesos del *backbone* se mantuvieron constantes a lo largo del entrenamiento. Sin embargo, los pesos del resto de la arquitectura se configuraron para ser optimizados a lo largo del entrenamiento. De esta forma, se reduce la capacidad computacional necesaria y el tiempo de entrenamiento ya que la optimización se limita a un número menor de parámetros en vez de al modelo completo.

4.1. Modelo

En este estudio entrenaron tres variaciones de la arquitectura de YOLO. Estas variaciones se basan en cambios en la complejidad del modelo. De esta forma, un modelo menos complejo tiene menos profundidad de capas y menos canales por capa.

De los modelos existentes, se realizaron los experimentos con los modelos *Nano*, *Medium* y el *ExtraLarge*, aumentando la complejidad del modelo en cada caso y con ello su capacidad. Para los 3 modelos, la arquitectura base es la misma. Sin embargo, están parametrizados con multiplicadores de profundidad y de amplitud del modelo, lo que se traduce en un aumento del número de capas y del número de canales en las capas respectivamente.

El modelo *Nano* consta de casi 2 millones de parámetros en total siendo el más pequeño de todos los modelos disponibles. Congelando el *backbone*, el número de parámetros entrenables se reduce a algo más de 800 mil parámetros. Esto reduce en gran medida el número de cálculos a realizar en la optimización y acelera el entrenamiento. Al ser el modelo con el menor número de parámetros, tiene la característica de ser el más rápido a la hora de entrenar y al realizar inferencia sobre las imágenes, ya que tiene un tiempo de procesamiento más rápido que el resto de los modelos. Sin embargo, de los modelos preentrenados, es el modelo que peor desempeño mostró en los entrenamientos con datasets heterogéneos de numerosas clases.

El modelo *Medium* es el modelo intermedio entre el modelo más pequeño y el más grande de los disponibles. Este consta de más de 21 millones de parámetros en total, más de diez veces el número de parámetros que tiene el modelo más pequeño. Una vez congelado el *backbone* de la arquitectura, el número de parámetros queda reducido a poco más de 9 millones de parámetros que van a ser optimizados, menos de la mitad del total. Al tener una mayor capacidad que el modelo *Nano*, en estudios previos, evaluando ambos modelos contra el mismo *dataset*, el modelo *Medium* ha mostrado un mejor desempeño que la versión reducida. Sin embargo, el tiempo de procesamiento y entrenamiento también se vieron afectados resultando en valores más altos.

Por último, en este estudio se probó la versión más compleja de los modelos disponibles, el modelo *ExtraLarge*. Este modelo consta de un total de más de 86 millones de parámetros en total, siendo cuarenta veces más grande que el más pequeño. el número de parámetros entrenables después de congelar el *backbone* se reduce a 37.455.165 en total, un valor más grande que el número total de parámetros del modelo *Nano* y el modelo *Medium*. Esto hace que a la hora de procesar las imágenes y entrenar el modelo, los tiempos sean mucho más altos, pero al igual que en caso anterior, en estudios previos, se demostró que el desempeño del modelo mejora respecto a las versiones más pequeñas.

4.2. Métricas

A la hora de entrenar los distintos modelos es de gran importancia conocer lo que representan las métricas de desempeño obtenidas y comprender por qué evolucionan de un modo u otro. En caso de la detección y localización de objetos existe un conjunto de medidas comúnmente empleado que permite evaluar y comparar distintos modelos. Estas medidas serían las siguientes.

- ***Intesection over Union* (IoU)**: Representa la superposición de dos áreas. En detección de objetos se emplea para medir la calidad de las *bounding boxes* y para definir un umbral de descarte o aceptación de *bounding boxes*. Se rige por la Ecuación 13, donde A y B son dos *bounding boxes* de una imagen.

$$IoU = \frac{A \cap B}{A \cup B} \tag{13}$$

- **Precisión**: Mide el porcentaje de predicciones que han sido correctas de todas las predicciones positivas.

- ***Recall***: Mide el porcentaje de predicciones correctas de todos los casos positivos reales.

- ***Average Precision* (AP)**: Mide el área bajo la curva que enfrenta los valores de Precisión y *Recall* para distintos valores de IoU especificados como umbral.

- ***Mean Average Precision 0.5*** ($mAP_{0.5}$): Mide el AP para cada clase de objeto existente en el *dataset* dando por válidas las predicciones con un IoU mayor que 0.5 y realiza la media sobre estos valores.
- ***Mean Average Precision 0.5:0.95*** ($mAP_{.5:.95}$): Mide el AP para valores del IoU de 0.5 a 0.95 con saltos de 0.05 y realiza la media de estos.

Como en este estudio se va a tratar de detectar un objeto de una sola clase, las métricas de ($mAP_{0.5}$) y ($mAP_{.5:.95}$) quedan reducidas a la precisión media para los distintos valores del umbral sin tener en cuenta el conjunto de valores obtenido por emplear más de una clase.

Los modelos de YOLO emplean hasta 3 funciones de pérdidas para optimizar la red en función de las distintas tarea del modelo, detectar, localizar y clasificar. Estas funciones se definen como *Box Loss*, *Objectness Loss* y *Classification Loss*.

La primera de las tres funciones representa la diferencia entre la *bounding box* obtenida mediante regresión por el modelo y la *bounding box real*, es decir, como de superpuestas están la *bounding box* predicha y la *bounding box* real, lo que se traduce en $1 - IoU$. Es decir, esta función de pérdidas hace que el modelo genere mejores *bounding boxes*.

Por otro lado, el modelo asigna una puntuación denominada *objectness score* a cada *bounding box*. Esta va de 0 a 1 dependiendo de cómo de cierto es que exista un objeto dentro de la misma. Es decir, dicha puntuación representa la probabilidad de que haya un objeto dentro de la región delimitada, lo que se puede interpretar como una predicción de la IoU. Por lo tanto, la *Objectness Loss* hace que el modelo genere mejores predicciones respecto a si hay un objeto o no dentro de la *bounding box*. En este caso buscará aumentar dicha puntuación en las mejores *bounding boxes* generadas y reducirla en las peores.

Por último, la *Classification Loss* representa el error en la clasificación de los objetos detectados y se rige por la *Cross Entropy*. En este estudio se emplea solo una clase de objeto y, por lo tanto, no se va a tener en cuenta en el análisis de los resultados.

4.3. Entrenamientos

A continuación, se van a exponer los resultados de todos los entrenamientos desarrollados para el modelo de detección y localización de objetos. Como se ha comentado anteriormente, estos entrenamientos se realizaron congelando las capas del *backbone* de la arquitectura para que mantengan los pesos de entrenamientos anteriores. Por otro lado, para cada modelo se realizan dos entrenamientos distintos, uno con el dataset real y otro con el dataset sintético.

4.3.1. Modelo *Nano*

Los resultados de las funciones de pérdidas para el modelo nano entrenado con ambos datasets se pueden observar en la Figura 35.

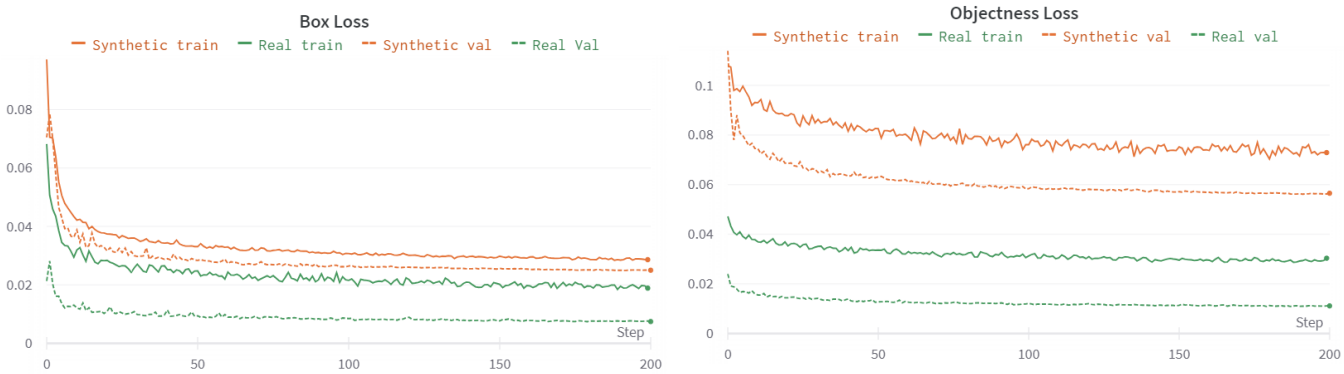


Figura 35: Pérdidas del modelo *Nano*.

Como se puede apreciar, en este caso, el modelo entrenado con el *dataset real* alcanza una menor pérdida en su conjunto. Además, se puede ver que los valores de las pérdidas de validación son menores que los del conjunto de entrenamiento. Sin embargo, esto es normal debido a que a la hora de optimizar los pesos se modifican las fotos ligeramente mediante técnicas de *data augmentation*. Esto hace que con el conjunto de entrenamiento el modelo no sea capaz de realizar predicciones tan buenas como en el de validación cuyas imágenes quedan inalteradas y requieren menor dificultad.

4.3.2. Modelo *Medium*

Los resultados de las funciones de pérdidas para el modelo *Medium* entrenado con ambos datasets se pueden observar en la Figura 36.

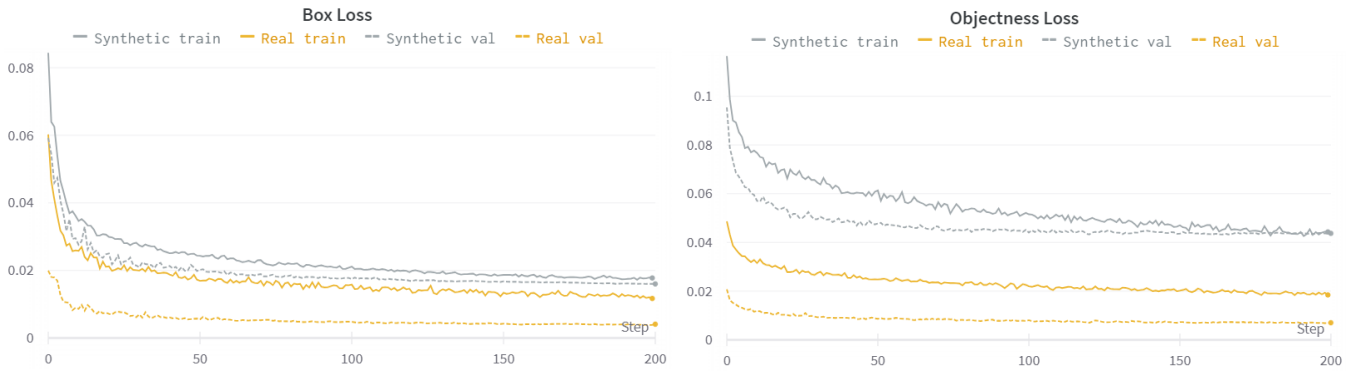


Figura 36: Pérdidas del modelo *Medium*.

En estas curvas de pérdidas se pueden apreciar resultados similares a los producidos con el modelo *Nano*. El conjunto real tiene menores pérdidas, es más simple y el modelo aprende mejor a realizar las predicciones. Además, la validación tiene unos valores menores que el conjunto de entrenamiento en casi todo el entrenamiento. Sin embargo, cuando el modelo ha aprendido lo suficiente vemos como los resultados del conjunto de entrenamiento y de validación se aproximan.

4.3.3. Modelo *Extra Large*

Los resultados de las funciones de pérdidas para el modelo *Extra Large* entrenado con ambos datasets se pueden observar en la Figura 37

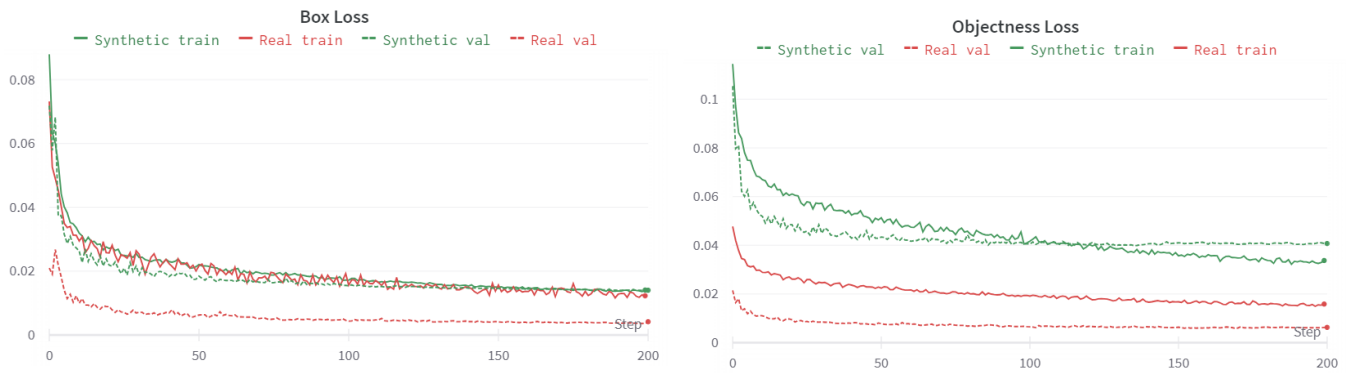


Figura 37: Pérdidas del modelo *Extra Large*.

Como se puede observar, durante estos entrenamientos las curvas de pérdidas de entrenamiento y validación están más próximas que en los casos anteriores. Los modelos tienen mayor capacidad y, por lo tanto, pueden aprender casos más complejos. Además, las curvas del entrenamiento con el *dataset* real y el *dataset* sintético también representan valores similares.

4.3.4. Comparación entrenamientos

A continuación, en la Figura 38, se exponen valores de las funciones de pérdidas para el conjunto de datos de validación de todos los modelos.

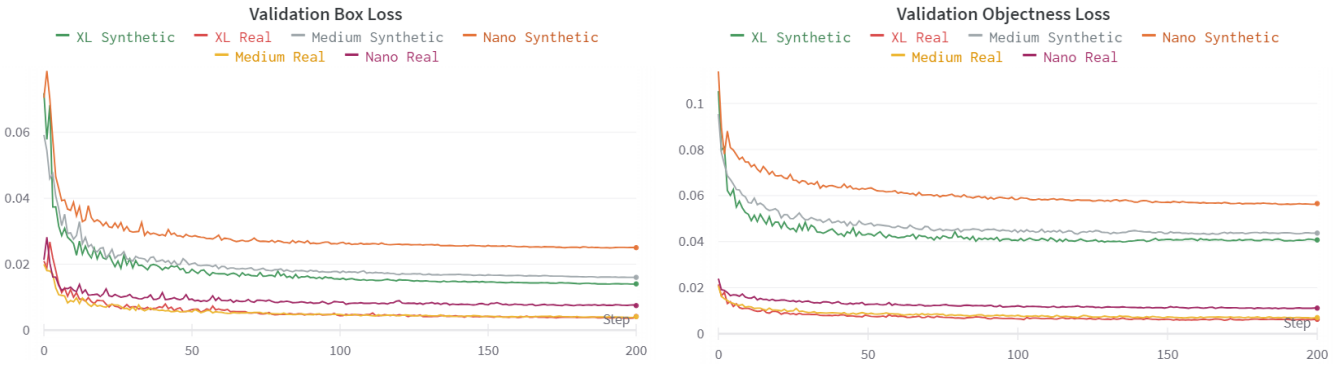


Figura 38: Pérdidas de validación de los modelos de YOLO.

Como se puede observar, de forma general, los modelos entrenados con las imágenes reales presentan mejores valores de pérdidas que los modelos entrenados con las imágenes sintéticas. Como se ha comentado, esto es debido a la complejidad que presentan las imágenes sintéticas frente a las reales.

Por otro lado, se puede apreciar que al aumentar la complejidad del modelo, los resultados mejoran. Esto se ve claramente enfrentando las curvas de los modelos entrenado con el *dataset* sintético.

Por último, en la Figura 39 se exponen los valores de precisión, *recall*, $mAP_{0.5}$ y $mAP_{0.5:0.95}$ de los modelos a lo largo de los entrenamientos.

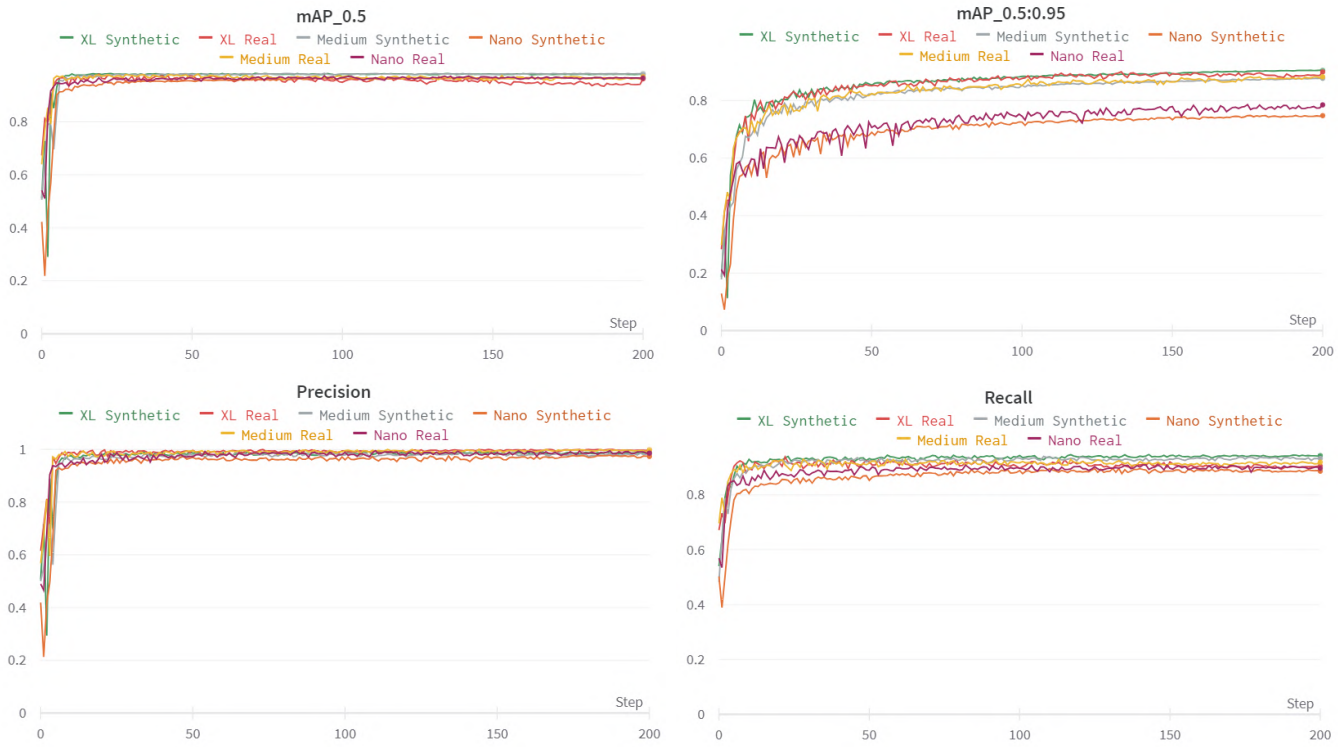
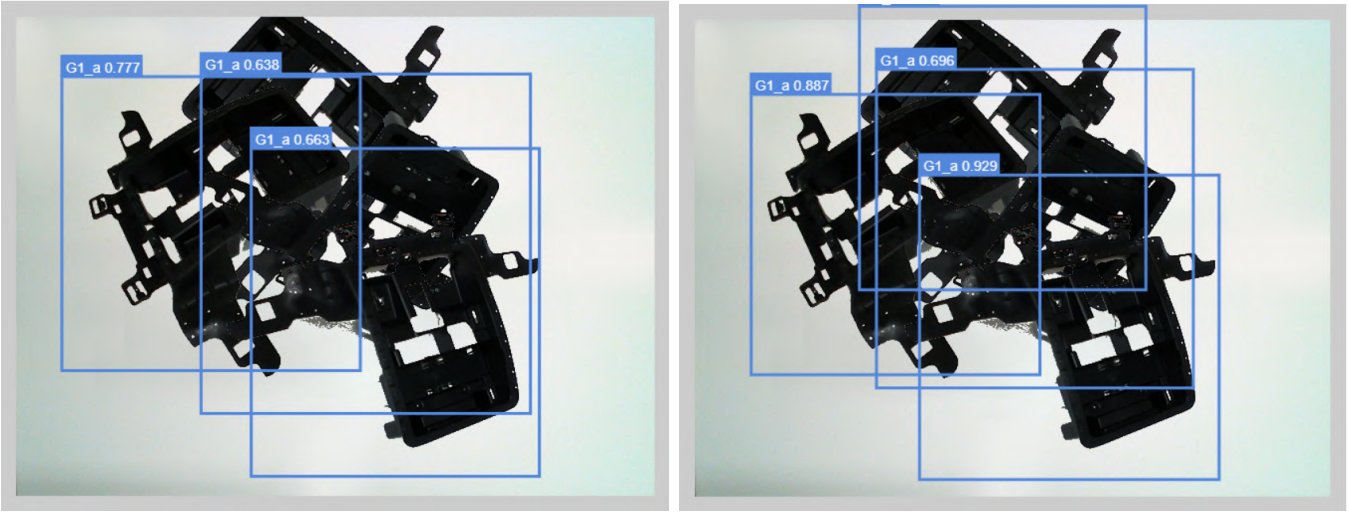


Figura 39: Métricas de desempeño los modelos de YOLO.

Se puede apreciar que todos los modelos obtienen un gran desempeño desde épocas tempranas del entrenamiento. En este caso no se aprecia diferencia entre los modelos entrenados con distintas imágenes ya que los valores son muy similares. Sin embargo, comparando los modelos respecto a su complejidad destaca cómo mejoran las distintas métricas de desempeño según aumenta la complejidad del modelo. Esto es claro observando los valores de $mAP_{0.5:0.95}$.

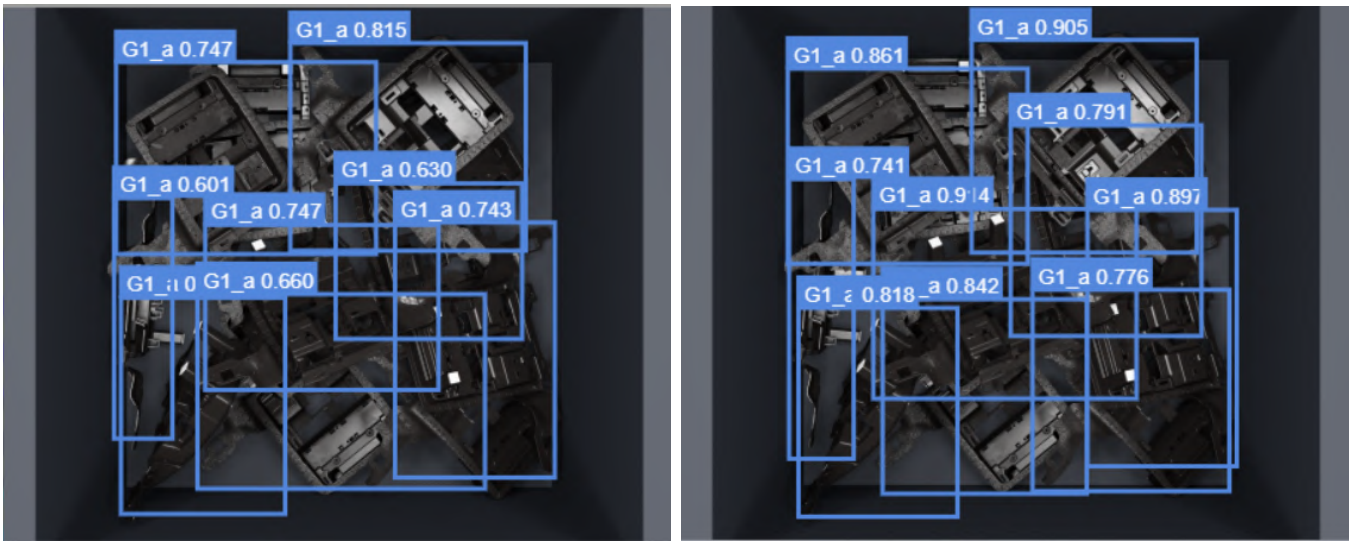
4.3.5. Evolución en las predicciones

Además de los valores numéricos de las funciones de pérdidas es conveniente estudiar las predicciones que se obtienen de los modelos y observarlas en los resultados. De este modo en la Figura 40 y la Figura 41 se pueden observar las predicciones para dos etapas del entrenamiento en ambos datasets. Cabe destacar que en este caso se ha filtrado el resultado a aquellas predicciones en las que el modelo asigna una puntuación de más de 0,5 a la *bounding box* obtenida.



(a) Época 19 (b) Época 199

Figura 40: Predicciones en *dataset* real en distintas épocas.



(a) Época 19 (b) Época 199

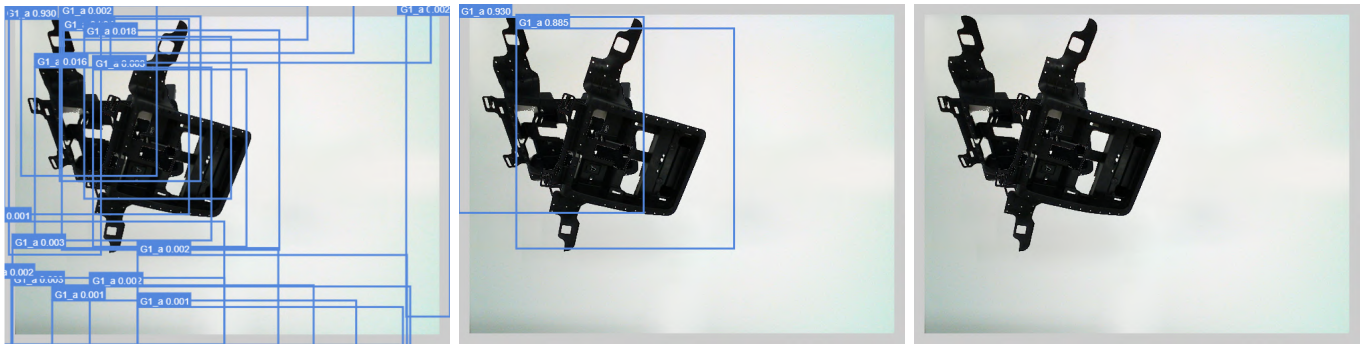
Figura 41: Predicciones en *dataset* sintético en distintas épocas.

Como se puede ver, para etapas temprana del entrenamiento la puntuación asignada a las *bounding box* es más baja y esta se ajusta peor a cada pieza que cuando ha aprendido durante más tiempo. Además, hay algunas piezas que al inicio del entrenamiento no detecta por estar ocultas tras otras mientras que más adelante sí.

4.3.6. Impacto del umbral de aceptación

Como se ha mencionado anteriormente, los resultados obtenidos a la hora de realizar inferencia con el modelo dependen en gran medida del umbral de aceptación especificado para la puntuación de las predicciones. Cuanto menor sea el umbral, mayor será el número de predicciones obtenidas dado que se aceptarán más *bounding boxes* con una puntuación baja. Por este motivo, este parámetro es crítico a la hora de evaluar los modelos y su desempeño.

Como se puede ver en la Figura 42 un umbral demasiado bajo genera demasiadas predicciones y convendría un postprocesado que escoja únicamente las mejores. Sin embargo, se puede observar cómo un umbral un poco más elevado limita los resultados a las predicciones que mejor se ajustan. Por último, un umbral en el que se exija una predicción perfecta, es decir, que la *bounding box* tenga una puntuación de 1, hace que no se obtenga ningún resultado debido a que el modelo no es capaz de generar predicciones con esa puntuación.



(a) Umbral > 0.0001 (b) Umbral > 0.5 (c) Umbral $= 1$

Figura 42: Predicciones para distintos umbrales de aceptación.

4.4. Evaluación

Tras realizar los entrenamientos se probaron los mejores modelos obtenidos de cada uno frente a un último dataset para evaluarlos. Este dataset está compuesto por imágenes reales, pero dichas imágenes no se han empleado para entrenar o validar cualquiera de los modelos entrenados con imágenes de este estilo. Es decir, los modelos entrenados con imágenes reales nunca han visto estas, lo cual servirá para evaluar dicho modelo frente a imágenes ligeramente distintas a lo que ha aprendido a procesar. Del mismo modo, realizar inferencia sobre imágenes reales con modelos entrenados con imágenes sintéticas servirá para comprobar la efectividad de estos modelos en entornos reales de producción. En la Figura 43 se puede observar una muestra de este *dataset*.

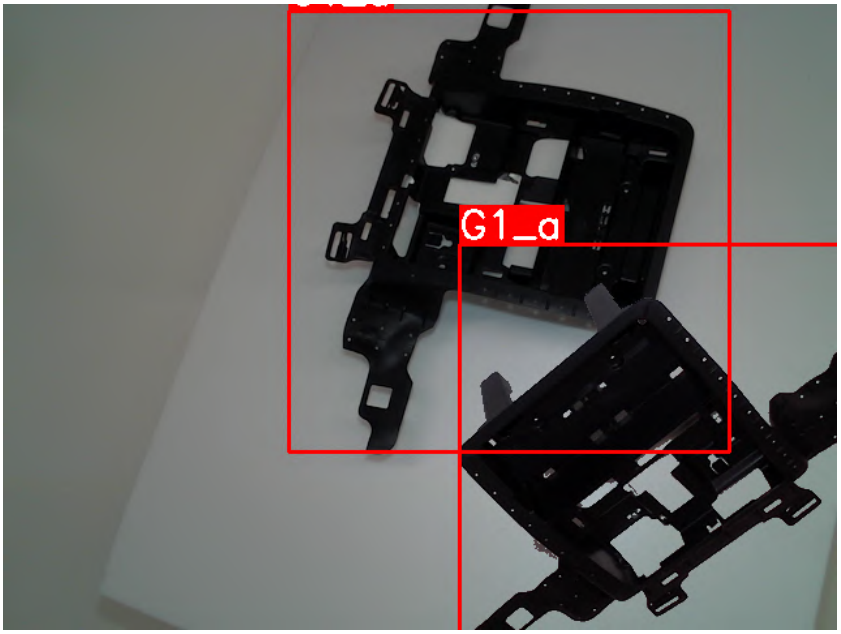


Figura 43: Muestra del *dataset* de evaluación de detección de objetos.

Cabe destacar que, para todos los procesos de inferencia de este dataset, se estableció un umbral de confianza de 0,5 y se limitaron las predicciones a 10 por imagen. Esto quiere decir que, en primera instancia se aceptaría todas las predicciones a las que el modelo da una puntuación de más de 0,5. Sin embargo, de todas las aceptadas se escogerán las 10 mejores.

A continuación, en la Tabla 1 se pueden observar los valores de las métricas de desempeño y las pérdidas generadas tras enfrentar los modelos entrenados con imágenes reales frente al dataset de evaluación.

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

Tabla 1: Métricas de los modelos de imágenes reales en el conjunto de test.

Como se ha visto anteriormente en los entrenamientos, los valores de las pérdidas mejoran cuanto mayor es la complejidad del modelo. Sin embargo, el resto de métricas de dempeño, a pesar de que mejoran con la complejidad en casi todos los casos, no representan una mejora cuantiosa. Esto da lugar a que si se pasa a un entorno real de producción convendría emplear el modelo *Nano* ya que tiene unos tiempos de inferencia mucho más rápidos y requiere de una menor capacidad computacional.

Por otra parte, en la Tabla 2 se pueden observar los valores de las métricas de desempeño y las pérdidas generadas tras enfrentar los modelos entrenados con imágenes sintéticas frente al dataset de evaluación.

Test metrics synthetic dataset						
Model	Precision	Recall	$mAP_{0.5}$	$mAP_{0.5:0.95}$	Box loss	Obj loss
Nano	1.000	0.824	0.850	0.850	0.271	0.318
Medium	1.000	0.822	0.849	0.849	0.265	0.426
ExtraLarge	1.000	0.824	0.857	0.855	0.243	0.305

Tabla 2: Métricas de los modelos de imágenes sintéticas en el conjunto de test.

Al igual que en el caso anterior, aumentar la complejidad del modelo mejora los valores de las funciones de pérdidas, pero en el resto de métricas de desempeño no se ve reflejada esta mejora. Nos encontreríamos ante el caso de tener que emplear también el modelo *Nano* a la hora de pasar a un entorno de producción si se consideran lo suficientemente buenos.

Como se puede apreciar, los modelos entrenados con imágenes reales dan resultados mejores que los entrenados con imágenes sintéticas como era de esperar. Sin embargo, esto no implica descartar esta práctica debido a que, independientemente de la diferencia entre ambos, los modelos entrenados con imágenes sintéticas presentan un gran desempeño frente al dataset de evaluación compuesto de imágenes reales. Por lo tanto, esto es indicador de que con una aproximación de la realidad lo suficientemente fiel, el modelo es capaz de aprender características de las imágenes que le sirven para trasladarse a un entorno real.

5. Caso de uso: Obtención del vector normal a una superficie en una imagen

En este capítulo se va a describir en detalle la metodología de los entrenamientos del modelo de regresión del vector normal a una superficie de interés. En primera instancia se comentará la arquitectura que tendrá el modelo a entrenar con los distintos *datasets*. Tras esto, se ofrecerá una explicación de los entrenamientos ejecutados sobre el *dataset* real, el *dataset* sintético y el *dataset* sintético generativo. Una vez completados los entrenamientos, con un conjunto de datos nunca visto por los modelos, se realizará inferencia sobre los mismos. Finalmente, se expondrán tanto los resultados de los entrenamientos e inferencia y se detallarán unas breves conclusiones sobre los mismos.

En este caso, el código de entrenamiento y evaluación, así como la arquitectura del modelo, son propios. Empleando la librería de Pytorch se desarrolló una arquitectura convolucional que recibía como entrada una imagen y ofrecía como salida las 3 componentes del vector unitario, normal a la superficie de interés. Además, se añadieron características a los códigos para poder visualizar los detalles más relevantes de las imágenes en los casos en los que el modelo ofrecía los mejores resultados. Esto hace referencia a las zonas de atención de los modelos, las cuales se represetnan en los mapas de saliencia.

5.1. Modelo

Para este estudio se desarrolló una arquitectura de red convolucional como la que se puede ver en la Figura 44.

Esta arquitectura toma como entrada una imagen cuadrada de 224 píxeles de resolución a color, con 3 canales. Inicialmente la entrada se procesará por una serie de capas convolucionales con funciones de activación ReLU. Tras cada capa convolucional se incluye una capa de *pooling* en la que cada *kernel* procesado se reduce a su valor máximo. Tras tres capas de convolución y *pooling* se reorganiza el tensor resultante de dimensiones de 128x18x18 como un vector unidimensional para procesarlo por una serie de capas densas que producen 3 valores de salida.

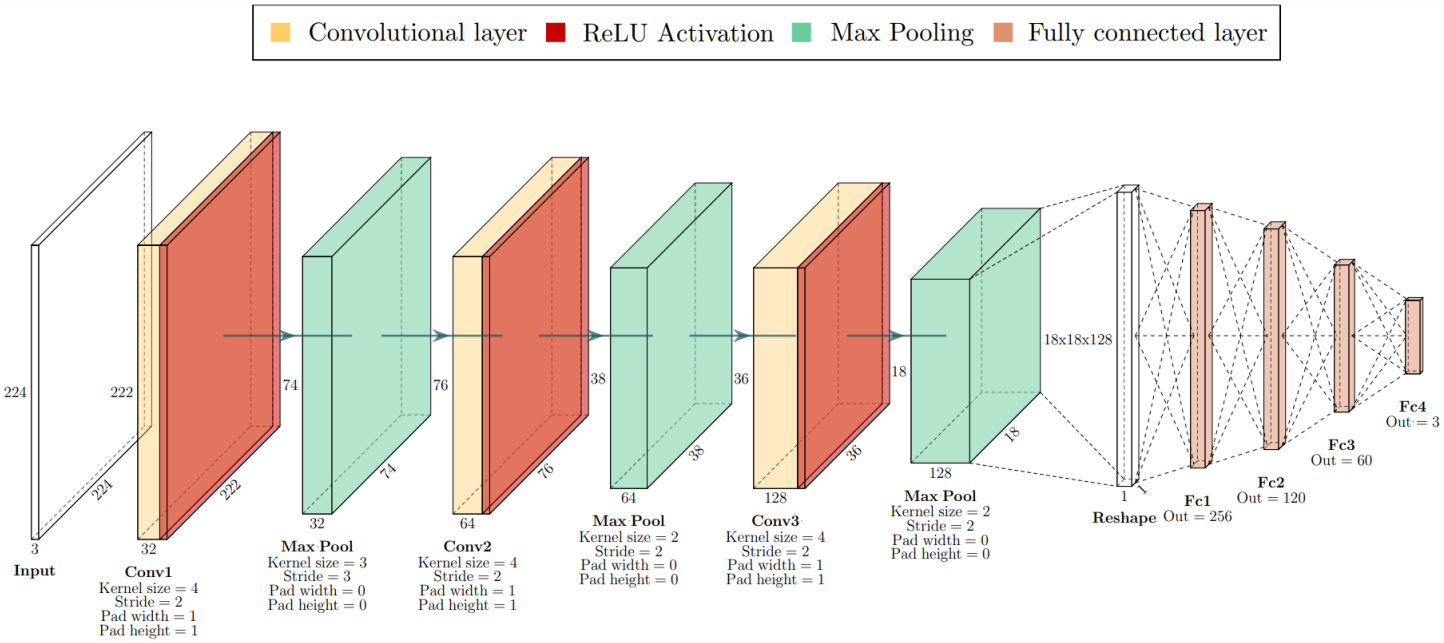


Figura 44: Arquitectura del modelo de regresión.

Los valores de salida que produce la arquitectura son las 3 componentes del vector normal a la superficie de interés. Sin embargo, para asegurar que el resultado es un vector unitario se normalizan estos valores.

En total, la arquitectura completa consta de más de 10,5 millones de parámetros que fueron entrenados desde cero para todos los entrenamientos hasta alcanzar los valores óptimos que minimicen la función de pérdidas.

5.2. Métricas

Como se ha expuesto anteriormente, el objetivo de este caso de uso es la obtención de un vector unitario normal a una superficie de interés en una imagen. Dicho vector se puede definir a partir de 3 componentes que representan su proyección en 3 planos ortogonales. Los valores de estas componentes se definen en las etiquetas de cada imagen del conjunto de datos con el que se realiza cada entrenamiento.

Para poder optimizar el modelo se debe escoger una función de pérdidas adecuada. Además, con el objetivo de aumentar la transparencia del modelo y la explicabilidad es conveniente que dicha función de pérdidas tenga sentido dentro del problema descrito. Por lo tanto, a continuación se desarrolla el razonamien-

to por el cual se concluyó que la función de pérdidas del error cuadrático medio (MSE) es la adecuada para este problema.

En primera instancia se procedió a evaluar cómo comparar dos vectores A y A' con componentes u, v, w cada uno. De este paso se concluyó que podía realizarse la diferencia de los mismos y se obtiene un tercer vector con 3 componentes que representan la diferencia entre las componentes iniciales como se muestra en la Ecuación 14.

$$\overrightarrow{A} - \overrightarrow{A'} = [u - u', v - v', w - w'] \tag{14}$$

De esta forma, cada componente del vector resultante muestra el grado de error entre el vector predicho por el modelo y el vector real.

Sin embargo, para poder implementarlo como función de pérdidas debe formularse como un valor único. Por lo tanto, una solución es obtener el módulo del vector obtenido anteriormente, como se muestra en la Ecuación 15.

$$|\overrightarrow{A} - \overrightarrow{B}| = \sqrt{(u_i - u'_i)^2 + (v_i - v'_i)^2 + (w_i - w'_i)^2} \tag{15}$$

El valor del módulo del vector resultante de la resta de dos vectores se puede interpretar como la distancia entre ambos vectores. Por ello, cuanto menor sea el valor obtenido en la Ecuación 15, más se parecerán los vectores. Sin embargo, se pierde la capacidad de evaluar las componentes del vector por separado.

El objetivo entonces, será que la distancia entre los dos vectores sea 0 y por lo tanto, la etiqueta real de las imágenes es 0. Definiendo el error como $(x_i - x'_i)$, x_i representa la etiqueta real, el 0, mientras que x'_i representará el cálculo de la distancia entre los dos vectores. Aplicando esta forma de error a todos los vectores obtenidos de un lote de N imágenes y haciendo la media se obtiene como resultado la función de la Ecuación 16.

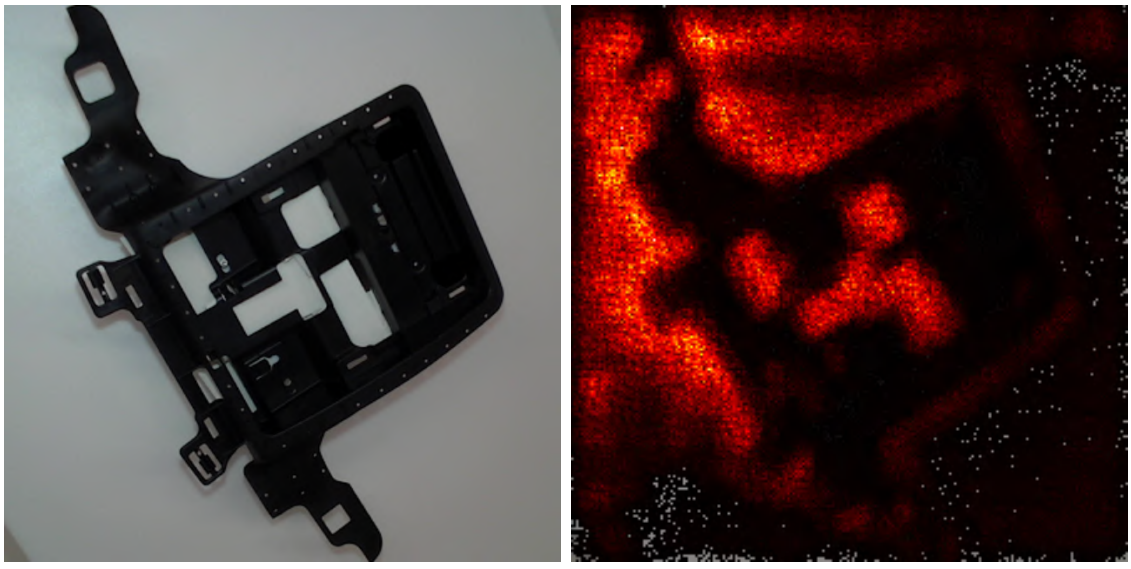
$$Loss = \frac{\sum_i^N (x_i - x'_i)^2}{N} = \frac{\sum_i^N (0 - \sqrt{(u_i - u'_i)^2 + (v_i - v'_i)^2 + (w_i - w'_i)^2})^2}{N} \tag{16}$$

Desarrollando y simplificando la función se llega a la Ecuación 17.

$$Loss = \frac{\sum_i^N (u_i - u_i')^2 + (v_i - v_i')^2 + (w_i - w_i')^2}{N} \tag{17}$$

La ecuación anterior muestra el resultado de emplear la distancia entre vectores como métrica de comparación de los mismos y como función de pérdidas para optimizar el modelo. Cabe destacar que la formulación que se muestra es el equivalente a emplear la función de pérdidas del error cuadrático medio sobre los valores obtenidos como predicciones del modelo y los valores reales de las componentes del vector normal.

Por otro lado, se incluye la visualización de los mapas de saliencia para las imágenes procesadas. El objetivo del mapa de saliencia es representar la conspicuidad, o saliencia, de cada lugar del campo visual mediante una cantidad escalar y guiar la selección de los lugares atendidos. Lo cual quiere decir que cada pixel de la imagen tendrá un valor de 0 a 1 dependiendo de la relevancia de este a la hora de procesar la imagen. A mayor puntuación, más relevancia tiene el pixel en la imagen a la hora de ser procesado y obtener la salida. Con esto se obtienen las zonas de la imagen donde se está fijando el modelo mayoritariamente para obtener las salidas correctas. Un ejemplo de este mapa para una imagen procesada por un modelo ya entrenado se puede observar en la Figura 45



(a) Imagen original (b) Mapa de saliencia

Figura 45: Ejemplo de mapa de saliencia para una imagen.

5.3. Entrenamientos

A continuación, se exponen los resultados de todos los entrenamientos desarrollados para el modelo de regresión. Todos los entrenamientos se realizaron desde cero sin pesos preentrenados. Se realizó un entrenamiento del modelo con cada conjunto de imágenes obtenido, el conjunto real, el conjunto sintético y el conjunto sintético postprocesado. Posteriormente, todos fueron evaluados bajo un mismo *dataset* de imágenes reales.

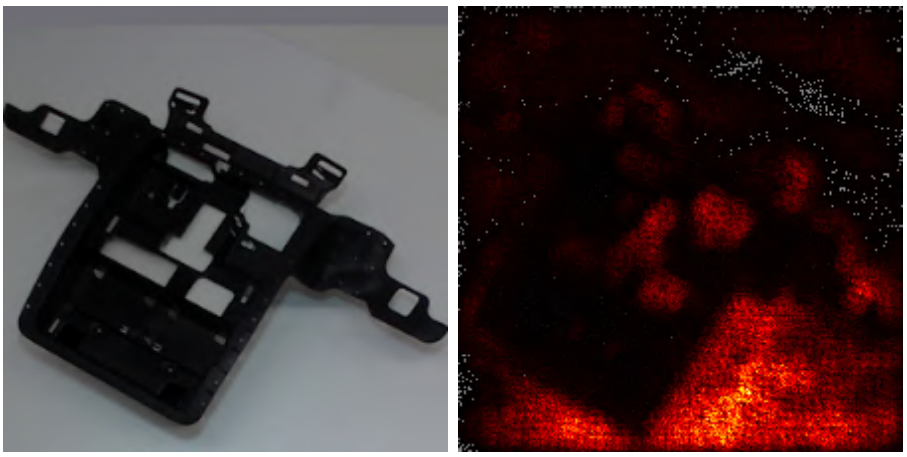
5.3.1. Modelo con imágenes reales

El primer entrenamiento desarrollado fue con el conjunto de imágenes reales. Este constaba de 300 imágenes que se dividieron en un conjunto de entrenamiento y otro de validación con una proporción del 80 % y del 20 % respectivamente. A continuación, en la Figura 46, se muestran las curvas con la función de pérdidas a lo largo del entrenamiento para ambos conjuntos de datos.

Figura 46: Pérdidas del modelo de regresión de imágenes reales.

Como se puede observar, el modelo aprende de forma correcta en ambos conjuntos de imágenes, tanto las de entrenamiento como las de validación. El valor de la función de pérdidas se reduce de forma correcta hasta casi un 0 exacto. Cabe destacar, que para el proceso de evaluación se estudiaron los modelos con el mejor desempeño frente al conjunto de validación.

A continuación, en la Figura 47 se puede observar cómo el modelo aprende a fijarse en el contorno de la pieza y sus sombras, lo cual indica la relevancia de estas en la imagen, mientras que el interior de es prácticamente ignorado. Por otro lado, en la Figura 48 se pueden observar tanto el vector real como el predicho por el modelo y cómo estos son casi iguales.



(a) Imagen original (b) Mapa de saliencia

Figura 47: Mapa de saliencia para modelo de imágenes reales.

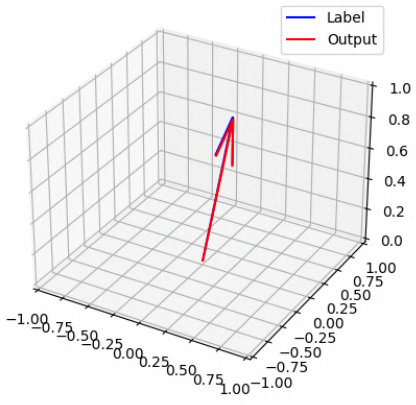


Figura 48: Vector real y salida del modelo de imágenes reales.

5.3.2. Modelo con imágenes sintéticas

El siguiente entrenamiento desarrollado fue con el conjunto de imágenes sintéticas. Este constaba de 633 imágenes que se dividieron en un conjunto de entrenamiento y otro de validación con una proporción del 80 % y del 20 % respectivamente. A continuación, en la Figura 49, se muestran las curvas con la función de pérdidas a lo largo del entrenamiento para ambos conjuntos de datos.

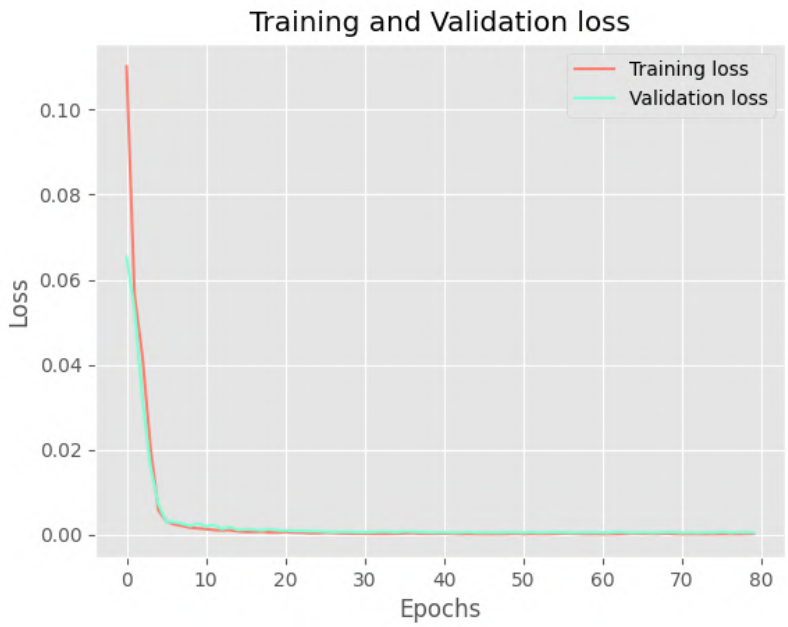
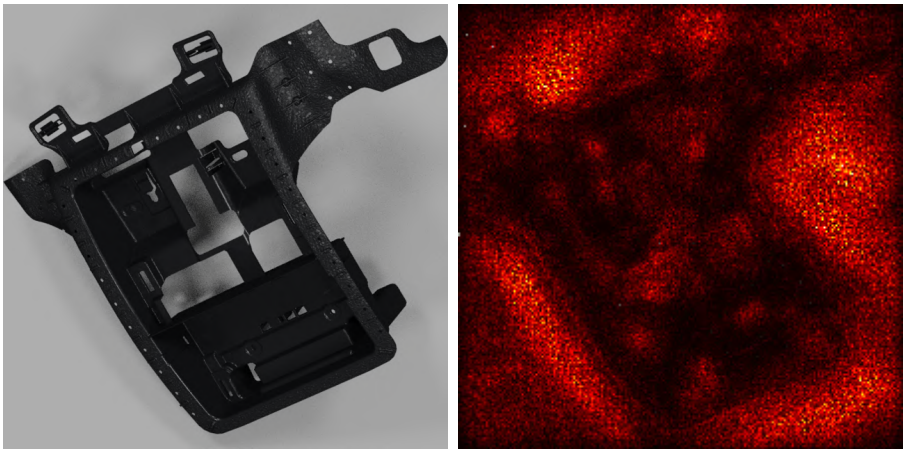


Figura 49: Pérdidas del modelo de regresión de imágenes sintéticas.

Como se puede observar, el modelo aprende de forma correcta en ambos conjuntos de imágenes, tanto las de entrenamiento como las de validación. El valor de la función de pérdidas se reduce de forma correcta hasta casi un 0 exacto. Cabe destacar, que para el proceso de evaluación se estudiaron los modelos con el mejor desempeño frente al conjunto de validación.

A continuación, en la Figura 50 se puede observar cómo el modelo aprende a fijarse en el contorno de la pieza, detalles del interior y sus sombras, lo cual indica, de nuevo, la relevancia de estas en la imagen. Sin embargo, comparándolo con el caso anterior, se puede ver que tiene una perspectiva más general de la imagen y no de una zona en concreto. Por otro lado, en la Figura 51 se pueden observar tanto el vector real como el predicho por el modelo y cómo estos son casi iguales.



(a) Imagen original (b) Mapa de saliencia

Figura 50: Mapa de saliencia para modelo de imágenes sintéticas.

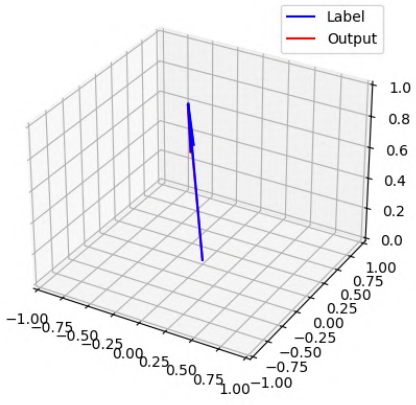


Figura 51: Vector real y salida del modelo de imágenes sintéticas.

5.3.3. Modelo con imágenes sintéticas postprocesadas

El último entrenamiento desarrollado fue con el conjunto de imágenes sintéticas con el postprocesado añadido de la red generativa. Al igual que el anterior, este constaba de 633 imágenes que se dividieron en un conjunto de entrenamiento y otro de validación con una proporción del 80 % y del 20 % respectivamente. A continuación, en la Figura 52, se muestran las curvas con la función de pérdidas a lo largo del entrenamiento para ambos conjuntos de datos.

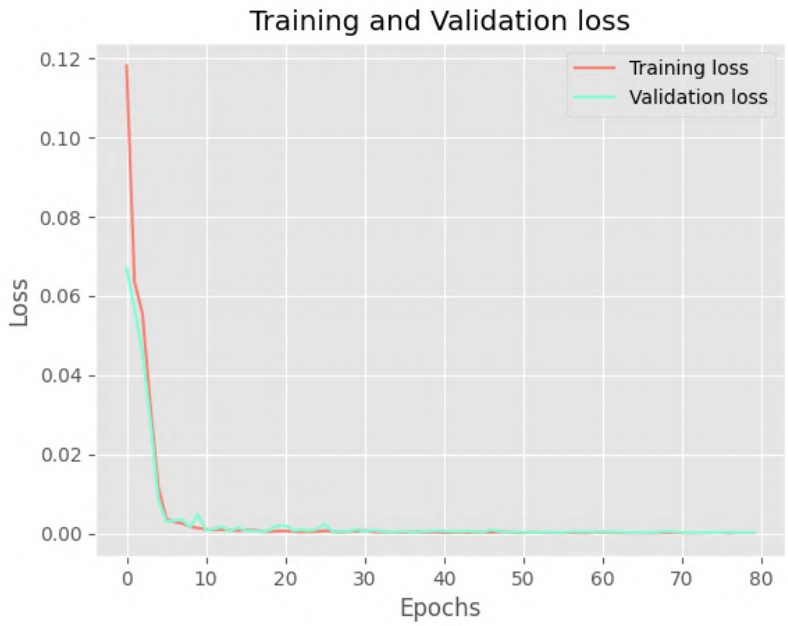
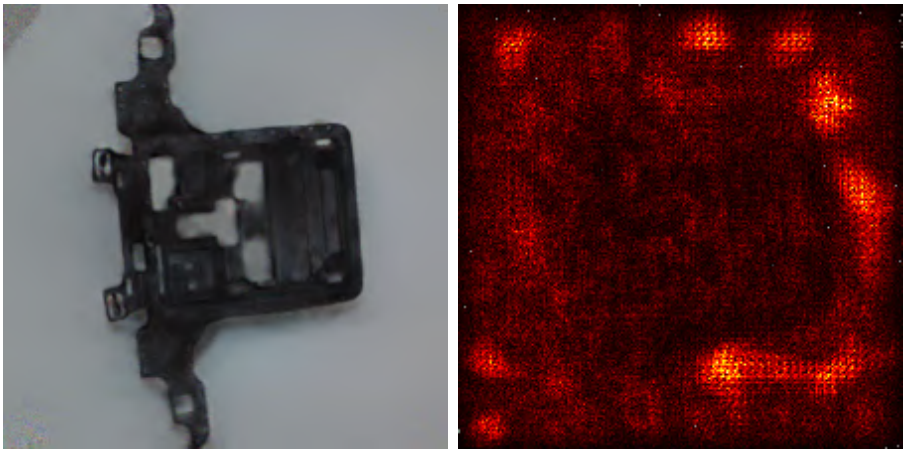


Figura 52: Pérdidas del modelo de regresión de imágenes postprocesadas.

Como se puede observar, el modelo aprende de forma correcta en ambos conjuntos de imágenes, tanto las de entrenamiento como las de validación. El valor de la función de pérdidas se reduce de forma correcta hasta casi un 0 exacto. Cabe destacar, que para el proceso de evaluación se estudiaron los modelos con el mejor desempeño frente al conjunto de validación.

A continuación, en la Figura 53 se puede observar cómo el modelo aprende a fijarse en detalles de la pieza, el contorno de la misma y sus sombras, lo cual indica, otra vez, la relevancia de estas en la imagen. Sin embargo, comparándolo con los casos anteriores, se puede ver que tiene una perspectiva más general de la imagen respecto al modelo de imágenes reales, como el modelo de imágenes sintéticas, pero sigue manteniendo la atención de una zona en concreto. Por otro lado, en la Figura 54 se pueden observar tanto el vector real como el predicho por el modelo y cómo estos son casi iguales.



(a) Imagen original (b) Mapa de saliencia

Figura 53: Mapa de saliencia para modelo de imágenes postprocesadas.

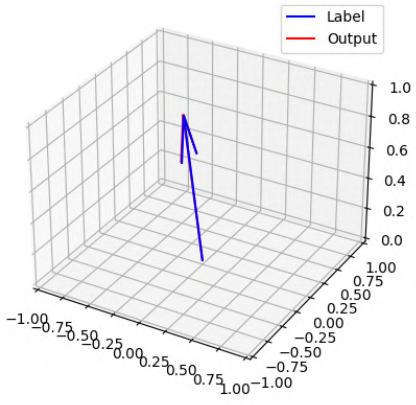


Figura 54: Vector real y salida del modelo de imágenes postprocesadas.

5.4. Evaluación

Por último, se enfrentaron los tres modelos entrenados frente a un *dataset* nunca visto por estos. Este dataset consiste en un conjunto de 100 imágenes reales. De esta forma, se comprobaría la capacidad de generalización de los modelos frente a un entorno real. A continuación, se exponen los resultados obtenidos y la comparación del desempeño de todos los modelos frente al *dataset* de evaluación.

5.4.1. Evolución de las pérdidas

Como se ha mencionado anteriormente, durante el entrenamiento de los modelos se anotaban los modelos que mejoraban el error de validación frente a sus respectivos *datasets*. A continuación, en la Figura 55, Figura 56 y la Figura 57 se exponen los valores de la función de pérdidas resultante al enfrentar estos modelos contra el *dataset* de evaluación.

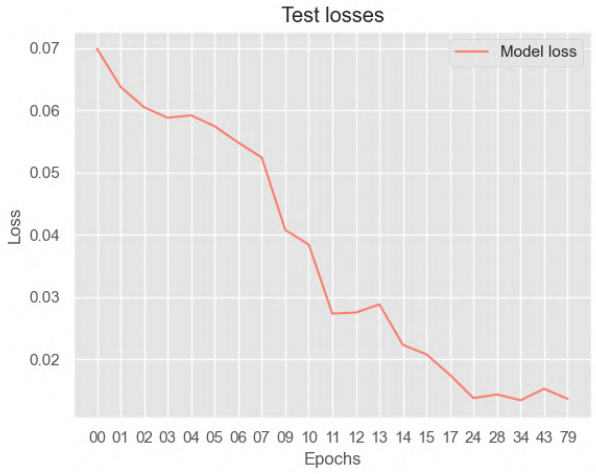


Figura 55: Pérdidas de evaluación del modelo de imágenes reales.

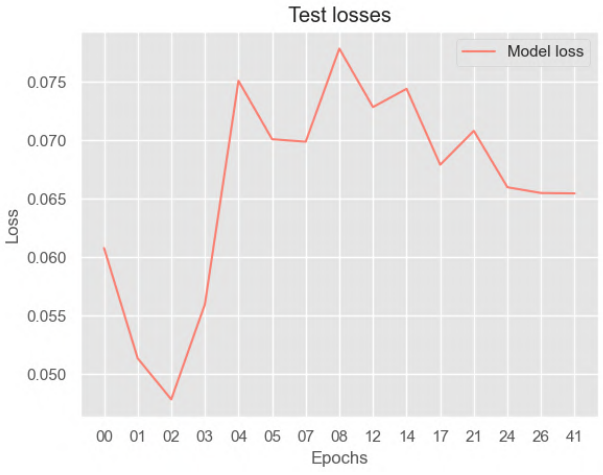


Figura 56: Pérdidas de evaluación del modelo de imágenes sintéticas.

Figura 57: Pérdidas de evaluación del modelo de imágenes postprocesadas.

Como se podía esperar, en la Figura 55 se puede observar que, a medida que las épocas aumentan, los modelos dan mejores resultados al igual que lo hizo durante el entrenamiento.

Sin embargo, esto es contrario a lo que sucede con los modelos de los otros dos tipos de imágenes. En estas podemos ver que el valor de las pérdidas aumenta o incluso oscila. Además, para estos tipos de imágenes, los modelos que mejor desempeño tienen en el conjunto de evaluación son los que están poco entrenados.

Por lo tanto, para el resto de la evaluación se optó por considerar únicamente el último modelo entrenado con imágenes reales y un modelo entrenado con pocas épocas de los obtenidos con los entrenamientos de imágenes sintéticas y de imágenes postprocesadas.

5.4.2. Zonas de atención

En esta sección se exponen los mapas de saliencia obtenidos de los tres modelos al procesar la imagen expuesta en la Figura 58. De esta forma se podrá observar si estos se fijan en los mismos aspectos de la imagen o no saben cómo procesarla. Los resultados se muestran en la Figura 59.

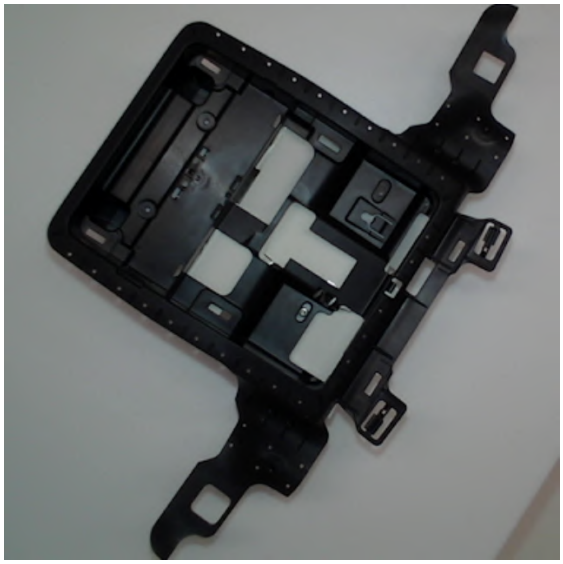
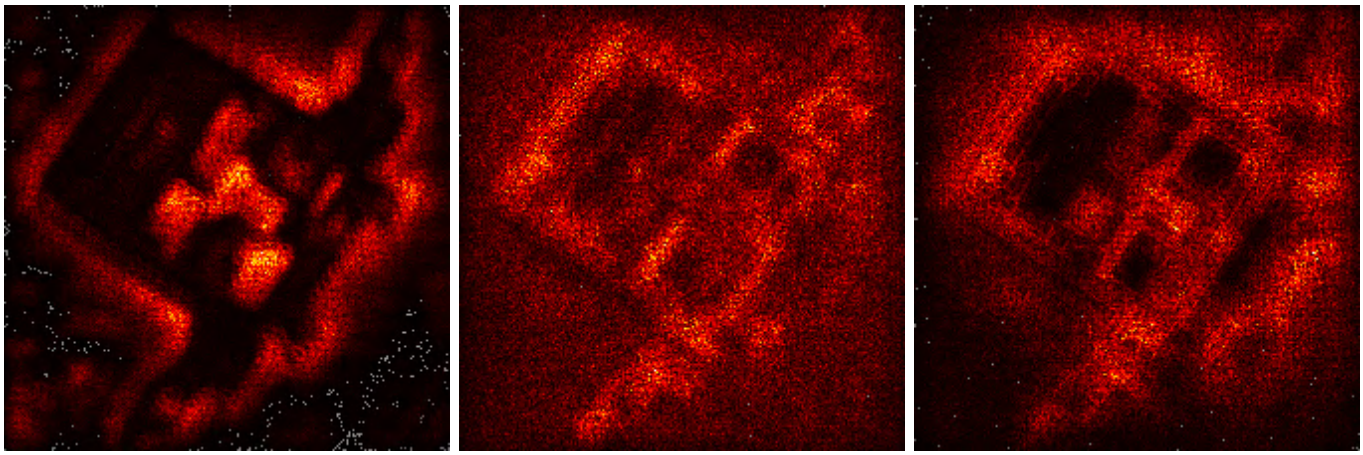


Figura 58: Imagen original de evaluación.



(a) Real (b) Sintéticas (c) Postprocesadas

Figura 59: Mapas de saliencia en evaluación.

Se puede observar que el mapa de saliencia del modelo de imágenes reales es muy similar al obtenido durante su entrenamiento, no presta mucha atención al interior de la pieza. Sin embargo, los otros dos parece que distan un poco más de los obtenidos durante su entrenamiento, apreciándose una mayor cantidad de ruido. Cabe destacar que, el modelo de imágenes postprocesadas tiene una mayor similitud al obtenido previamente y, además, se parece en mayor medida al obtenido con el modelo de imágenes reales, centrando la atención en el contorno de la pieza. Además, observando las imágenes de entrenamiento, en el *dataset* real, no se pueden apreciar las texturas de la pieza salvo en ciertas posiciones, siendo esta

un conjunto de píxeles mayoritariamente negros. Sim embargo, en las imágenes de los otros datasets se pueden apreciar más detalles de la pieza y, como se ha visto, los modelos aprenden a obtener información de la misma también.

5.4.3. Diferencia de vectores

En esta sección se exponen las distribuciones de los errores obtenidos al procesar el *dataset* de evaluación completo con los tres modelos. Estos resultados se pueden observar en laFigura 60, Figura 61 y la Figura 62 que enfrentan el error expresado en forma de ángulo de diferencia entre el vector real y el predicho y la proyección vertical del vector real.

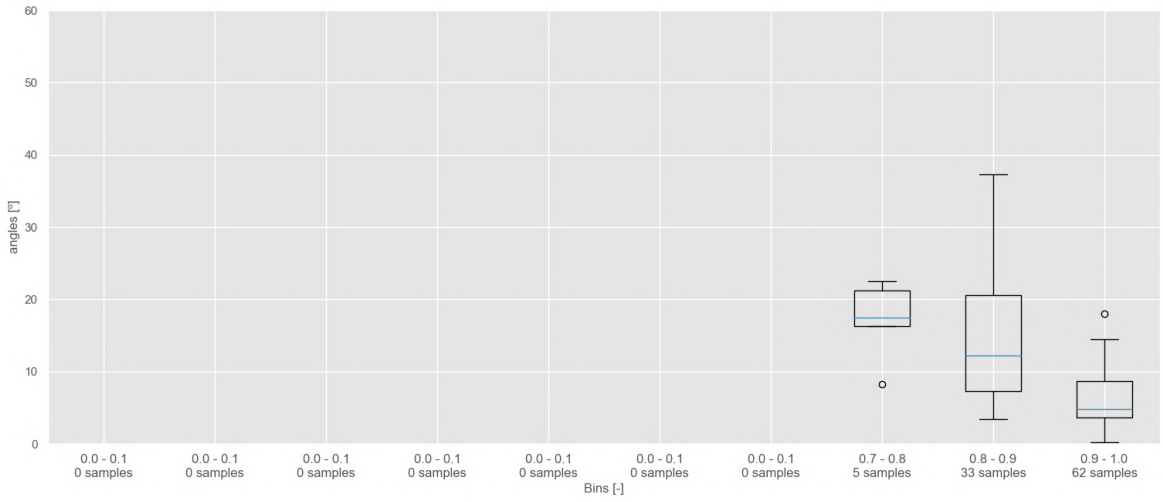


Figura 60: Distribución de errores del modelo de imágenes reales.

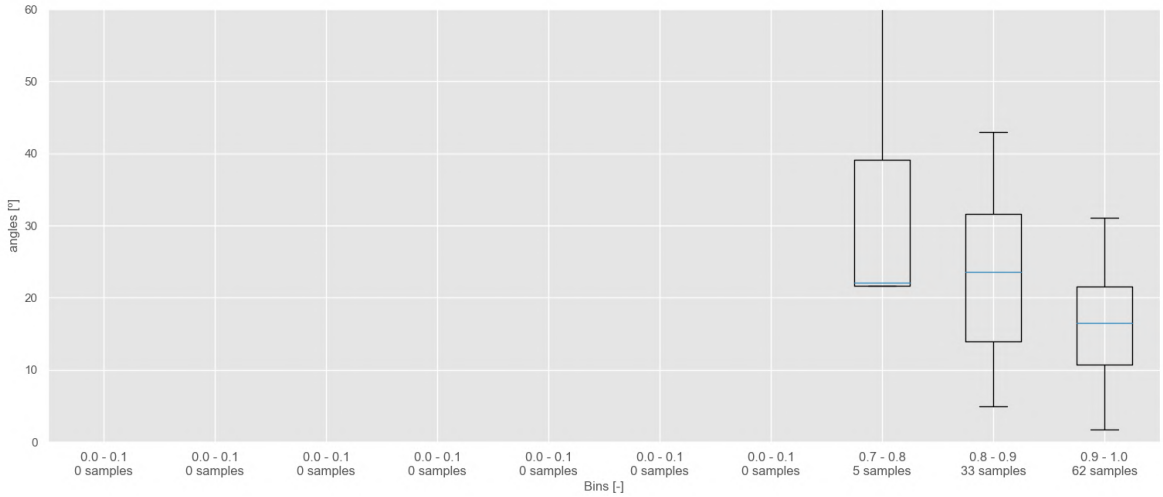


Figura 61: Distribución de errores del modelo de imágenes sintéticas.

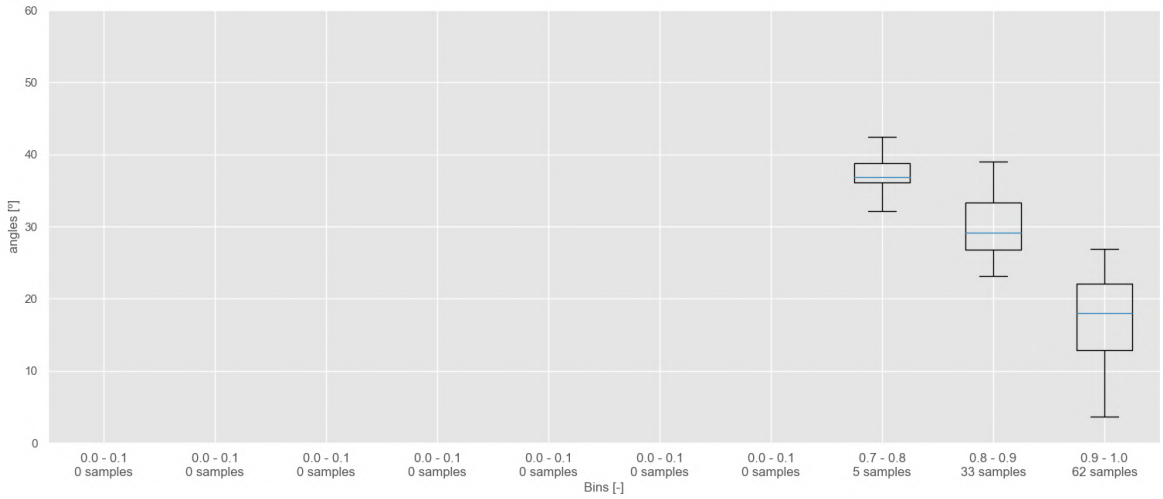


Figura 62: Distribución de errores del modelo de imágenes postprocesadas.

Se puede apreciar que, el modelo de imágenes reales tienes los errores más bajos y con menor varianza de los 3 modelos. Este sirve como referencia para los otros dos, los cuáles son similares.

Mientras que el modelo de imágenes sintéticas alcanza errores más bajos, el modelo de imágenes postprocesadas tiene una varianza mucho menor. Esto se puede interpretar como que el modelo de imágenes sintéticas es más aleatorio e impredecible y es posible que haya alcanzado los valores reducidos por suerte. Sin embargo, el otro puede afinarse para ir reduciendo los errores que ya presenta.

6. Conclusiones

En este capítulo se recopilan las conclusiones relativas a los resultados expuestos a lo largo de la memoria y se evalúa el cumplimiento de los objetivos expuestos en el apartado 1.2.

Por medio de la automatización, se generó un *dataset* de 400 imágenes reales que posteriormente se dividirían en dos distintos. Uno con 300 imágenes para los entrenamientos de los modelos y otro con 100 imágenes para evaluarlos. Para conseguirlo, se tuvieron que invertir recursos y tiempo en el diseño de la infraestructura de captura de imágenes, diseño e impresión de piezas 3D para manipular el objeto de interés, planificación y desarrollo del algoritmo de generación del *dataset* y, por último la generación del dataset.

A partir del software de renderizado de BlenderProc se generó un *dataset* de imágenes sintéticas a partir de un modelo 3D del objeto de interés. Para conseguirlo se tuvo que aprender a manipular el software de renderizado y diseñar el escenario virtual de captura de imágenes. Por otro lado, este método, si bien no consume tantos recursos físicos como los necesarios para la generación del *dataset* real, requiere de una capacidad computacional elevada para lograr tiempos de generación lo suficientemente bajos. Por ello, sin una GPU con los suficientes recursos este método no sería viable. Sin embargo, en este estudio se disponía de una tarjeta gráfica NVIDIA GeForce RTX 2080 [35] con la cual se consiguieron tiempos de generación de imágenes mejores que los del método de obtención de imágenes reales.

Dado que las diferencias entre las imágenes sintéticas y las reales eran significativas, se generó un *dataset* compuesto por imágenes sintéticas postprocesadas por una GAN que las asemejarían a las reales trasladando características como las sombras o la iluminación. Debido a la innovación que presentaba este método, se consumió tiempo entrenando el modelo y modificando la arquitectura para lograr los resultados esperados. Sin embargo, en un entorno de producción este proceso tiene unos tiempos de inferencia de menos de un segundo y el grueso de este recurso se consumiría en el entrenamiento de la GAN. Al igual que en el caso anterior, este método requiere una elevada carga computacional por la que hace necesario

el empleo de una GPU suficientemente potente.

Con todo los conjuntos de imágenes generados, se procedió con los entrenamientos de los modelos para ambos casos de uso.

En el problema de detección y localización de objetos, los modelos entrenados con imágenes reales presentaron un desempeño mayor que los modelos de imágenes sintéticas a la hora de enfrentarlos al *dataset* de evaluación compuesto por imágenes reales. Sin embargo, esto era un resultado que se podía esperar. Lo destacable de este estudio es que, aún siendo peor, los modelos entrenados con imágenes sintéticas presentaban un desempeño más que notable frente al conjunto de evaluación. Además, se observó que, para el problema en cuestión, el modelo *Nano* de YOLO era lo suficientemente complejo para aportar predicciones correctas en tiempos reducidos.

En el problema de regresión, al igual que en el anterior caso de uso, el modelo entrenado con imágenes reales presentaba un mejor desempeño que los entrenados con imágenes sintéticas o postprocesadas frente al conjunto de evaluación. En este caso, el modelo de imágenes sintéticas y el de imágenes postprocesadas ofrecían buenos resultados en el entorno real cuando no habían sido entrenados durante un periodo prolongado de tiempo. Además, se podía apreciar cómo las zonas de atención del modelo de imágenes postprocesadas eran más similares a las del modelo de imágenes reales que las del modelo de imágenes sintéticas, demostrando que estos aprendían de una forma similar. Cabe destacar que, mientras que el modelo de imágenes reales centraba la atención en el contorno de la pieza, posiblemente por falta de texturas de la misma, los otros modelos aprendieron a extraer información de la misma ya que en sus imágenes de entrenamiento, la pieza se podía apreciar con mayor detalle. Por lo tanto, los modelos de imágenes sintéticas y postprocesadas, al enfrentarse al conjunto de datos de evaluación trataban de buscar información en en interior de la pieza además de en el contorno y sombras de la misma. Sin embargo, al tener menos información del interior de la pieza, su desempeño era resultaba ser peor que el del modelo de imágenes reales. Por último, se observó que la varianza de los errores en el modelo de imágenes postprocesadas es mucho menor que la del modelo de imágenes sintéticas asemejándose a los errores reales.

Gracias a los mapas de saliencia que representaban la zonas de atención de los modelos de regresión, se pudo observar cómo las sombras y la iluminación de las imágenes eran aspectos clave en el caso de uso expuesto, y, por lo tanto, asemejar estas condiciones en un entorno sintético beneficiaría al modelo resultante.

Como resumen, sin duda, un modelo dará los mejores resultados cuanto más fiel sea el conjunto de datos de entrenamiento al conjunto de datos del entorno de producción. Por lo tanto, los modelos entrenados con imágenes reales siempre darán mejores resultados en el entorno de producción. Sin embargo, cabe destacar que el desempeño de los modelos sintéticos, aún siendo peor, era similar y considerablemente bueno. Si se pueden conseguir las mismas condiciones en un entorno virtual que en el real, es una opción a considerar teniendo en cuenta el ahorro de recursos y tiempo que presenta. Si no se pueden conseguir directamente en un entorno virtual, la transferencia de estilos por medio de una GAN ofrece una posibilidad de aumentar la similitud de las condiciones con la que se pueda mejorar el modelo sintético.

7. Futuros desarrollos

Una vez finalizado el desarrollo del proyecto, en este capítulo se exponen las posibles vías de desarrollo que se presentan para continuar con este tema de estudio.

- Evolución de la arquitectura del modelo de regresión que permita aumentar su complejidad y tratar con imágenes de distintas resoluciones.
- Cambios de la arquitectura de red adversaria generativa por un modelo de difusión que transfiera el estilo de las imágenes reales o unas condiciones de iluminación concretas.
- Preentrenar los modelos sintéticos para que aprendan a poner la atención en las mismas zonas que el modelo real.
- Realizar técnicas de *transfer learning* de un modelo preentrenado con imágenes sintéticas que adquiera las características principales de la imagen a otro para que se entrene con imágenes reales durante menos tiempo.
- Mezclar conjuntos de datos de imágenes reales y sintéticas para explorar la proporción de las mismas necesaria para tener buenos resultados en el entorno de producción real.

Referencias

[1] ONU. *Objetivos y metas de desarrollo sostenible*. URL: https://www.un.org/sustainabledevelopment/es/objetivos-de-desarrollo-sostenible/.

[2] Tania Babina et al. «Artificial intelligence, firm growth, and industry concentration». En: *Firm Growth, and Industry Concentration (November* 22 (2020), pág. 2020.

[3] Keiron O'Shea y Ryan Nash. «An Introduction to Convolutional Neural Networks». En: *CoRR* abs/1511.08458 (2015). arXiv: 1511.08458. URL: http://arxiv.org/abs/1511.08458.

[4] Sergey Soltan et al. «Deep Learning-Based Object Classification and Position Estimation Pipeline for Potential Use in Robotized Pick-and-Place Operations». En: *Robotics* 9.3 (2020). ISSN: 2218-6581. DOI: 10.3390/robotics9030063. URL: https://www.mdpi.com/2218-6581/9/3/63.

[5] Ross Girshick. «Fast R-CNN». En: *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*. Dic. de 2015.

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

[7] Joseph Redmon y Ali Farhadi. «YOLO9000: Better, Faster, Stronger». En: *CoRR* abs/1612.08242 (2016). arXiv: 1612.08242. URL: http://arxiv.org/abs/1612.08242.

[8] Joseph Redmon y Ali Farhadi. «YOLOv3: An Incremental Improvement». En: *CoRR* abs/1804.02767 (2018). arXiv: 1804.02767. URL: http://arxiv.org/abs/1804.02767.

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

[10] Glenn Jocher. *YOLOv5 by Ultralytics*. Ver. 7.0. Mayo de 2020. DOI: 10.5281/zenodo.3908559. URL: https://github.com/ultralytics/yolov5.

[11] Shrey Srivastava et al. «Comparative analysis of deep learning image detection algorithms». En: *Journal of Big Data* 8.1 (mayo de 2021), pág. 66. DOI: 10.1186/s40537-021-00434-w. URL: https://doi.org/10.1186/s40537-021-00434-w.

[12] Tsung-Yi Lin et al. «Microsoft COCO: Common Objects in Context». En: *CoRR* abs/1405.0312 (2014). arXiv: 1405.0312. URL: http://arxiv.org/abs/1405.0312.

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

[14] Aiden Nibali et al. «Numerical Coordinate Regression with Convolutional Neural Networks». En: *CoRR* abs/1801.07372 (2018). arXiv: 1801.07372. URL: http://arxiv.org/abs/1801.07372.

[15] Tomoyoshi Shimobaba, Takashi Kakue y Tomoyoshi Ito. «Convolutional Neural Network-Based Regression for Depth Prediction in Digital Holography». En: *2018 IEEE 27th International Symposium on Industrial Electronics (ISIE)* (2018), págs. 1323-1326.

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

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

[18] Han Zhang et al. *StackGAN++: Realistic Image Synthesis with Stacked Generative Adversarial Networks*. 2018. arXiv: 1710.10916 [cs.CV].

[19] Han Zhang et al. *StackGAN*. URL: https://github.com/hanzhanggit/StackGAN.

[20] Christian Ledig et al. *Photo-Realistic Single Image Super-Resolution Using a Generative Adversarial Network*. 2017. arXiv: 1609.04802 [cs.CV].

[21] Alec Radford, Luke Metz y Soumith Chintala. *Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks*. 2016. arXiv: 1511.06434 [cs.LG].

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

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

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

[25] Jun-Yan Zhu Taesung Park Phillip Isola Alexei A. Efros. *Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks*. URL: https://junyanz.github.io/CycleGAN/.

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

[27] Ziwei Liu et al. «Deep Learning Face Attributes in the Wild». En: *Proceedings of International Conference on Computer Vision (ICCV)*. Dic. de 2015.

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

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

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

[31] © 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.

[32] © 2023 Roboflow. *Roboflow*. URL: https://roboflow.com/.

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

[34] Maximilian Denninger et al. «BlenderProc». En: *arXiv preprint arXiv:1911.01911* (2019).

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

A. Modificación código YOLO

A continuación se exponen las modificaciones realizadas al código original de YOLO de Ultralitycs para mostrar el valor de las funciones de pérdidas en la validación.

En `val.py` las siguientes líneas sirven para obtener las pérdidas [líneas 212-214 en la versión GitHub]:

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

Sin embargo, `compute_loss` debe inicializarse antes del bucle for de los lotes. Para ello, se importa `ComputeLoss` del fichero `utils.py` y se pasa `model.model` como argumento. Esto es así porque en `val.py` model es un objeto distribuido que lanza un error si no se hace así. Se modifican las líneas 191-198 del archivo original, las que definen la barra de progreso. Se añadió la inicialización de `compute_loss` y las pérdidas a la barra de progreso

```
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()  # profiling times
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)

for batch_i,(im,targets,paths,shapes) in enumerate(pbar):
```

Ahora en la variable `loss` se tienen las 3 pérdidas acumuladas con cada lote. Cuando termina el bucle for se deben añadir las siguientes líenas antes de imprimir los resultados [281-285 del fichero orinial]:

```
# 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  # print format
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')
```

Se multiplica por el tamaño del lote y se divide por la longitud del cargador de datos para obtener la pérdida media por imagen, ya que los elementos de pérdida son las pérdidas totales acumuladas durante el procesamiento de todo el conjunto de datos.