universidad de león

Escuela de Ingenierías

Industrial, Informática y Aeroespacial

GRADO EN INGENIERÍA EN ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

Trabajo de Fin de Grado

ELABORACIÓN DE UN SISTEMA DE DETECCIÓN DE PLÁSTICOS PET EN PLANTAS DE RESIDUOS MEDIANTE INTELIGENCIA ARTIFICIAL

DEVELOPMENT OF A PET PLASTICS DETECTION SYSTEM IN WASTE PLANTS BY ARTIFICIAL INTELLIGENCE

Autor: DANIEL COMESAÑA PEREIRA

Tutor: Ángela Díez Díez

(SEPTIEMBRE,2022)

UNIVERSIDAD DE LEÓN

Escuela de Ingenierías Industrial, Informática y Aeroespacial

GRADO EN INGENIERÍA EN ELECTRÓNICA INDUSTRIAL Y AUTOMÁTICA

Trabajo de Fin de Grado

ALUMNO: Daniel Comesaña Pereira

TUTOR: Ángela Díez Díez

TÍTULO: Sistema de detección de plásticos PET en plantas de reciclaje mediante inteligencia artificial

TITLE: PET plastics detection system in recycling plants by Artificial Intelligence

CONVOCATORIA: Septiembre, 2022

RESUMEN:

El plástico es uno de los materiales de uso cotidiano que más contamina. El Polietileno Tereftalato es el plástico con mayor facilidad para ser reciclado, debido a esto la recolección de los desechos PET en las plantas de residuos adquiere una gran importancia, ya que ayuda a prevenir la contaminación derivada de la elaboración y extracción de nuevos materiales. Los procedimientos que existen hoy en día para la separación de estos plásticos son la clasificación de manera manual realizada por los trabajadores de la planta o, excepcionalmente, mediante el uso de máquinas muy complejas y de gran coste. En esta tesis se realiza un estudio de la elaboración de un sistema de inteligencia artificial que tenga la capacidad de localizar este material por medio del uso de redes neuronales convolucionales basadas en regiones. Estas redes se entrenan con fotografías de los objetos que se desean detectar y tienen la capacidad de ubicarlos en una o varias zonas de la imagen mediante el análisis de la forma y los colores propios de estos. Entre los tipos de redes basadas en regiones existen dos que analizaremos y compararemos: las Faster RCNN, las cuales localizan los objetos mediante cuadros delimitadores y las Mask RCNN que son capaces de detectar los objetos dibujando su contorno mediante un polígono. El PET lo encontramos en gran medida en las botellas de bebidas y de agua, por lo que el objeto que debe detectar nuestra RCNN son estas botellas. La implementación de este método en una planta de residuos sería simple y económica debido a que solo haría falta una cámara para captar las imágenes de los residuos en cada momento y un ordenador con el sistema RCNN implementado que

detectase los materiales de PET.

En este trabajo vamos a analizar la capacidad de un sistema con tecnología RCNN, creado en este proyecto, de detectar residuos o materiales deformables, en este caso se analizará con las botellas de PET.

ABSTRACT:

Plastic is one of the most polluting household materials nowadays. Polyethylene Terephthalate is the plastic that is most easily recycled, which is why the collection of PET refuse in waste plants is of great importance, as it helps to prevent contamination derived from the production and extraction of new materials. The procedures that exist today for the separation of these plastics are manual sorting by plant workers or, exceptionally, by using very complex and expensive machines. In this thesis we study the development of an artificial intelligence system that could locate this material through the use of region-based convolutional neural networks. These networks are trained with pictures of the objects to be detected and could locate them in one or several areas of the image by analysing their shape and colours. Among the types of region-based networks there are two that we will analyse and compare: Faster RCNNN, which locate objects by means of bounding boxes, and Mask RCNNN, which are able to detect objects by drawing their outline using a polygon. PET is found to a large extent in drinks and water bottles, so the objects to be detected by our RCNNN are these bottles. The implementation of this method in a waste plant would be simple and economical because it would only require a camera to capture images of the waste in real time and a computer with the RCNN system implemented to detect the PET materials.

In this work we are going to analyse the capacity of a system with RCNN technology, created in this project, to detect waste or deformable materials, in this case it will be analysed with PET bottles.

Palabras clave: IA, Deep Learning, MRCNN, Faster RCNN, RCNN.

Firma del alumno:	**VºBº Tutor/es:**

ÍNDICE DE CONTENIDOS

Índice de figuras

Índice de tablas

Índice de ecuaciones

0 Introducción

La contaminación es uno de los grandes problemas que amenazan en la actualidad. Hoy en día la sociedad se rige por un consumismo masivo que acelera el agotamiento de algunas materias primas. La separación de residuos y su posterior reciclaje permite ampliar la vida del uso de los materiales en los residuos, utilizándolos para la creación de otros productos, esto a su vez ayuda a la prevención de los problemas de contaminación y agotamiento. En España en 2020 "*38,9 millones de ciudadanos afirmaron reciclar sus envases a diario*" (Ecoembes 2020), pero la separación que se realiza en las casas no es suficiente, una vez la basura llega a una planta de residuos hay que separarla en subgrupos más pequeños. La idea de este trabajo es realizar un sistema que sea capaz de detectar alguno de estos subgrupos de basura mediante inteligencia artificial para su fácil separación posterior. En este proyecto realizaremos un estudio investigando cómo funciona la inteligencia artificial, qué métodos existen para implementar esta tecnología en las imágenes y pondremos en marcha varios programas que localicen estos residuos con la ayuda de esta tecnología y compararemos su eficiencia.

La dificultad de la detección de muchos residuos reside en que estos son **deformables** y un mismo material lo podemos encontrar de numerosas formas, en este trabajo vamos a analizar la capacidad de un sistema con tecnología IA, creado en este proyecto, de detectar residuos o materiales deformables.

0.1 HISTORIA DE LA INTELIGENCIA ARTIFICIAL

> "*La Inteligencia Artificial (AI) es una rama de la Ciencia que se ocupa de ayudar a las máquinas a encontrar soluciones a problemas de una manera más humana*" (Rich and Knight 1991).

Según el libro *Artificial Intelligence A Modern Approach* (Russell y Norvig 2010) la historia de la inteligencia artificial se divide en las siguientes etapas:

0.1.1 GESTACIÓN DE LA INTELIGENCIA ARTIFICIAL (1943-1955)

En esta etapa se reconoce el primer trabajo de IA, el cual fue llevado a cabo por Warren McCulloch y Walter Pitts, los cuales, basándose en el conocimiento del funcionamiento de las neuronas, propusieron un modelo de neuronas artificiales, en el cual cada neurona tenía 2 estados on y off y un cambio de estado de una neurona creaba una respuesta de cambio de estado en algunas neuronas vecinas. Demostraron que cualquier función computable podía ser realizada por una red de neuronas conectadas y que todos los conectores lógicos pueden ser representados mediante una estructura de red neuronal. En 1950, dos estudiantes de Harvard, Marvin Minsky y Dean Edmonds, construyeron el primer ordenador de red neuronal, el cual simulaba una red de 40 neuronas. Posteriormente, hubo una serie de trabajos que abarcaron estudios de la Inteligencia Artificial, de los cuales destacamos el realizado por el matemático e informático, Alan Turing, el cual, en 1950, introduce en su artículo “Computing Machinery and Intelligence” términos como: machine learning, algoritmos genéticos y el Test de Turing; en el que se evaluaba la capacidad de una máquina de realizar actividades con un comportamiento inteligente similar al de un ser humano.

0.1.2 NACIMIENTO DE LA INTELIGENCIA ARTIFICIAL (1956):

En este año, el informático John McCarthy, reunió en Dartmouth, Hanover, a investigadores interesados en redes neuronales y autómatas, bajo la propuesta de, en 2 meses, realizar numerosos avances en ámbito de la Inteligencia Artificial. De este estudio nació en programa de razonamiento Logic Theorist (LT), desarrollado por Allen Newell y Herbert Simon; este programa imita el comportamiento del ser humano para solucionar problemas matemáticos y fue capaz de demostrar 38 de los 52 teoremas del libro *Principia Mathematica,* de Russell y Whitehead. El taller de Dartmouth sirvió para que se presentasen entre sí figuras importantes que dominarían el campo de la IA durante los próximos 20 años, en él se reunieron personas del MIT, CMU, Stanford e IBM. Además, el taller también sirvió para establecer la Inteligencia Artificial como un campo separado y una rama de la informática.

0.1.3 **GRANDES EXPECTATIVAS PARA LA IA (1952-1969)**

En los primeros años de la IA se consiguieron grandes objetivos. Las computadoras de aquel entonces eran consideradas máquinas que simplemente podían realizar operaciones aritméticas y era extraño que una computadora pudiese realizar una acción inteligente.

Posteriormente a la creación del Logic Theorist, Simon y Newell desarrollaron el General Problem Solver, o GPS; este programa imitaba los protocolos de resolución de problemas realizados por los humanos para resolver ciertos acertijos.

- En 1952, Arthur Samuel, creó un programa que aprendió a jugar las damas a nivel amateur.

- En 1958, John McCarthy, definió Lisp, el lenguaje de programación dominante durante los siguientes 30 años. Además, en 1959, desarrolló Advice Taker, un programa diseñado para, usando el conocimiento, resolver problemas, pero, a diferencia de los otros solucionadores de problemas realizados anteriormente, este programa debía incorporar el conocimiento general del mundo, por tanto, el Advice Taker se podía reprogramar para que incorporase nuevos conocimientos generales. Posteriormente, en 1965, se incorporaron en el Advice Taker aplicaciones lógicas.

- En 1959, Herbert Gelernter, construyó el Geometry Theorem Prover, el cual podía probar numerosos teoremas de matemáticas.
- Entre 1963 y 1968, se crearon numerosos programas que podían resolver operaciones matemáticas: James Slagle creó SAINT (capaz de realizar problemas de integración), Tom Evans desarrolló ANALOGY (capaz de resolver problemas de analogía geométrica) y Daniel Borrow creó STUDENT (capaz de resolver problemas de algebra).
- En esta época nació la Neurona de McCulloch y Pitts, que fue el primer intento de formalizar matemáticamente el funcionamiento de una neurona. Esto sirvió para asentar las bases de lo que hoy en día son las redes neuronales artificiales y el Deep Learnig.

0.1.4 **GOLPE DE REALIDAD (1966-1973)**

El inicial éxito de los proyectos de Inteligencia Artificial provocó que las predicciones sobre esta rama de cara al futuro fuesen muy ambiciosas. Hebert Simon predijo, en 1957, que dentro de 10 años un ordenador sería campeón de ajedrez y que un teorema matemático importante sería demostrado por una máquina, pero, estas predicciones llegaron 40 años más tarde y no 10. La mayoría de las actividades de IA de esta época fueron de gran dificultad en comparación a la simplicidad de las máquinas de aquellos tiempos, por este mismo hecho fracasaron grandes proyectos de la IA. Un ejemplo de estos proyectos fallidos fue el intento de un programa de traducción automática, fomentado por el Consejo de Investigación Nacional de los Estados Unidos con el objetivo de traducir documentos científicos rusos tras el lanzamiento del Sputnik en 1957. Este programa traducía de una manera literal y no se preocupaba la ambigüedad del texto, por tanto, daba lugar a numerosas incoherencias. Otra dificultad fue la imposibilidad de las máquinas de resolver de problemas extensos, ya que los programas de IA de la época resolvían los problemas probando diferentes combinaciones de pasos hasta encontrar la solución y estos problemas extensos requerían una gran cantidad de combinaciones. Se pensaba que la solución a este problema era una cuestión de hardware más rápido y memorias más grandes (nada más lejos de la realidad).

Aún con todo, la ilusión por un poder computacional ilimitado provocó el inicio de los primeros experimentos de **algoritmos genéticos**, que se basaban en la creencia de que, mediante una serie de combinaciones de pequeñas mutaciones en un programa, se puede generar un programa con buen rendimiento para cualquier tarea concreta.

0.1.5 **SISTEMAS BASADOS EN EL CONOCIMIENTO (1969-1979)**

En esta época cambió la idea preconcebida de que para que una máquina resolviese problemas debía encadenar pasos de razonamiento elementales para encontrar soluciones completas (era el denominado **método débil**) debido a, como ya se mencionó anteriormente, la imposibilidad de resolver problemas grandes o difíciles. La alternativa fue utilizar un conocimiento más potente y específico del dominio "*Para resolver un problema difícil, casi hay que conocer la respuesta*" (Russell y Norvig 2010).

- En 1969, en Stanford, Ed Feigenbaum, Burce Buchanan y Joshua Lederberg desarrollaron DENDRAL, un programa que permitía interpretar las estructuras moleculares el cual, mediante una serie de reglas, permitía reconocer a la molécula dentro de subgrupos que iban restringiendo el número de combinaciones que debía realizar el programa para hallar su estructura nuclear (Figura siguiente).

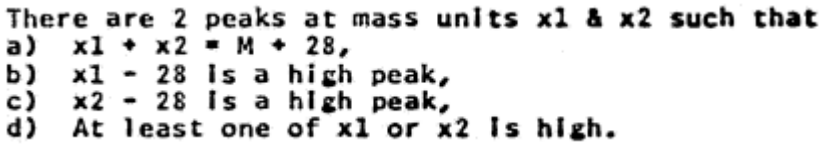

Figura 0.1. Ejemplo de una regla que tiene el programa DENDRAL para situar la molécula en un subgrupo de centonas; siendo M (masa de la molécula, si hay 2 picos de masa x1 y x2 y las condiciones de la a) a la d) se cumplen, entonces la molécula entra dentro del subgrupo de centonas. (Feigenbaum et al. 1970)

DENDRAL es el primer **sistema experto** creado, esto es, es el primer sistema que adquiere conocimiento humano y lo aplica para resolver problemas que normalmente serían solventados por una persona experta en el campo.

Esta metodología innovadora en el campo de la IA dio pie al desarrollo de otros programas:

- MYCIN: programa, creado por Feigenbaum, Buchanan y Edward Shortliffe, para diagnosticar infecciones sanguíneas. Fue el primer sistema de IA con un cálculo de incertidumbre, denominado **factores de certeza**, para reflejar el nivel de acierto del diagnóstico.
- SHRDLU: sistema, creado por Winograd, para la comprensión del lenguaje natural que era capaz de superar la ambigüedad y entender las referencias a pronombres, pero aun con todo, seguía dependiendo demasiado del análisis sintáctico y esto causaba algunos problemas cuando se aplicaba a trabajos de traducción automática.

0.1.6 **GRAN DESARROLLO (1980-ACTUALIDAD)**

Desde 1980 la IA ha ganado gran importancia en muchos ámbitos de la Industria y, durante esta época hasta la actualidad, ha sufrido el mayor desarrollo de todas las épocas:

- En 1986, McDermott, crea el primer sistema experto comercial de éxito, R1, un programa que utilizaba Digital Equipment Corporation para la configuración de pedidos de nuevos sistemas informáticos. En 1986, se calculó que, gracias a este programa, la empresa ahorraba 40 millones de dólares al año. Esto provocó una una gran financiación económica para el desarrollo de sistemas inteligentes por las empresas, estos sistemas inteligentes (sistemas expertos, sistemas de visión, robots, etc.). Debido al fracaso de numerosos proyectos, muchas empresas tuvieron grandes pérdidas de dinero, fue el periodo denominado como el "invierno de la IA".
- A mediados de los 80, se reinventó el algoritmo de aprendizaje por **retropropagación** (descubierto por Bryson y Ho en el 1969); este algoritmo conforma hoy en día la manera que tienen de aprender las redes neuronales artificiales. El algoritmo fue aplicado tanto en problemas de aprendizaje tanto en informática como en psicología.
- En esta época, la Inteligencia Artificial entra por fin al método científico; las hipótesis sobre IA, para ser aceptadas, deben someterse a experimentos empíricos rigurosos y sus resultados deben analizarse estadísticamente para determinar el nivel de importancia.

A mediados de los años 70, el **modelo oculto de Markov** (**HMM**) se empieza a aplicar en el campo del reconocimiento del habla. Este es un modelo probabilístico cuyo objetivo es determinar parámetros desconocidos (u ocultos) de una cadena a partir de parámetros observables. Este sistema es el que se utiliza actualmente para el reconocimiento del habla, el cual, nos lo encontramos, entre sus aplicaciones más comunes, en los dictados automáticos, en el control por comandos (Siri, Ok Google, etc.), en los sistemas diseñados para discapacitados, en la telefonía y en los sistemas portátiles (relojes o teléfonos móviles).

- A finales de los años 90, en la traducción automática se retoma un enfoque basado en secuencias de palabras con modelos que seguían los principios de la **Teoría de la Información** (*rama de la probabilidad que se encarga del transporte de información en los sistemas de comunicación* (M. Reza 1961)), los cuales habían quedado en desuso en los años 60. Este enfoque es el que domina el campo de la traducción automática actualmente.
- En el ámbito de las redes neuronales, en esta época nacen las **redes bayesanas**, un tipo de red que permite crear un modelo de probabilidad mediante pruebas observadas y registradas Estas redes se utilizan para realizar predicciones de situaciones. Como resultado de esto, la **minería de datos** adquirió gran importancia en la industria
- En la época de los 2000, se le resta importancia al estudio de los algoritmos en el campo de la Inteligencia Artificial y se empieza a dar más valor a los datos. Ejemplo que probaba esto fue un algoritmo (creado por Hays y Efros en 2007) para rellenar los huecos en una fotografía con Photoshop. El algoritmo funcionaba rellenando los huecos con imágenes parecidas al fondo del recorte, de esta manera no se notaba el rellenado en la fotografía. Estos descubrieron que el rendimiento de su algoritmo era pobre cuando utilizaban una colección de 10.000 imágenes, pero rendía en perfectas condiciones cuando aumentaban la colección a millones de imágenes.

La aplicación de Inteligencia Artificial también está presente en herramientas de internet (motores de búsqueda, sistemas de recomendación y agregadores de sitios web), sistemas sensoriales (visión, sonar reconocimiento de habla, etc), robótica, teoría del control, economía y automoción

0.1.7 ¿QUÉ PUEDE HACER LA INTELIGENCIA ARTIFICIAL EN LA ACTUALIDAD?

Hoy en día, la inteligencia artificial ha superado la capacidad humana en ciertas actividades; DEEP BLUE de IBM, creado por Goodman y Keene, consiguió derrotar en una partida de ajedrez al campeón mundial Garry Kasparov. Además se consiguió superar barreras; en el ámbito de la automoción, el vehículo autónomo STANLEY (creado en 2006, por Thrun) consiguió atravesar el desierto de Mojave a 22mph, en el campo del reconocimiento de voz, el programa MAPGEN (AI-Chang et al. , 2004) planifica las operaciones de los Mars Exploration de la NASA y en el área de la traducción automática, hoy en día, Google Translate (traductor de Google), es uno de los sistemas de traducción automática más potentes y es capaz de traducir automáticamente 109 idiomas.

Además, también han aumentado el número de ámbitos en los que está presente la IA; los algoritmos de aprendizaje clasifican al día más de mil millones de mensajes como spam para que el destinatario no pierda el tiempo eliminándolos, además en el ámbito doméstico, el uso de aspiradoras inteligentes se ha incrementado notablemente.

0.2 REDES NEURONALES ARTIFICIALES

Las redes neuronales son modelos simplificados del sistema nervioso biológico que sirven para representar la manera en la que funciona el cerebro desde el punto de vista de la computación (Yadav 2015). Esta representación del intelecto ha sido estudio desde el punto de vista de varias disciplinas como las matemáticas, la física, la estadística, la informática y la ingeniería. Las redes neuronales tienen varias aplicaciones en el área del modelado matemático, el procesamiento de señales, el análisis de series temporales, etc.

Desde el punto de vista del análisis de imágenes las redes neuronales nos sirven principalmente para el reconocimiento de patrones, esto se puede aplicar, por ejemplo, para el reconocimiento de rostros o de dígitos escritos a mano.

Para explicar cómo funcionan las redes neuronales para el reconocimiento de patrones nos ayudaremos de un ejemplo realizado en el libro *Neural Networks and Deep Learning* (Nielsen. Michael 2019), en esta obra nos encontramos una muestra de cómo funciona un sistema de reconocimiento de dígitos escritos a mano.

Para que un sistema de redes neuronales pueda reconocer dígitos escritos a mano la idea es tomar una gran cantidad de dígitos escritos a mano, conocidos como ejemplos de entrenamiento, y luego desarrollar un sistema que pueda aprender de esos ejemplos de entrenamiento, de manera que la red neuronal usa los ejemplos para inferir automáticamente reglas para reconocer dígitos escritos a mano. Además, al aumentar los ejemplos de entrenamiento, la red neuronal aprenderá más sobre los patrones de escritura y mejorará su precisión, por tanto, a mayor número de ejemplos, la red reconocerá con menor error los dígitos.

0.2.1 **PERCEPTRONES**

El **perceptrón** es un tipo de neurona artificial, desarrollada en la época de 1950, la cual funciona tomando varias entradas binarias, X_1, X_2, …, y una sola salida binaria:

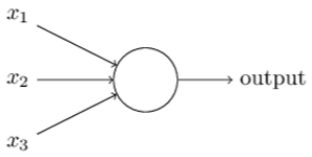

Figura 0.2. Representación de un perceptrón (Nielsen. Michael 2019)

La importancia de cada entrada se representa mediante números reales denominados pesos w_1, w_2, …. La salida de la neurona está determinada por la suma ponderada $\sum w_j X_j$ es mayor o menor que algún valor del umbral

$$producción = \begin{cases} 0 \text{ si } \sum w_j X_j \leq 1 \\ 1 \; si \; \sum w_j X_j > 1 \end{cases}$$

Fórmula 0.1. Salida de la neurona (Nielsen. Michael 2019)

Podemos simplificar la anterior fórmula de manera que $w \cdot X = \Sigma_{w_j} X_j$, donde w y X son los vectores cuyas componentes son los pesos y las entradas, respectivamente. Además, podemos añadir lo que se conoce como el **sesgo del perceptrón**, $b \equiv -\text{límite}$, el cual nos permite que la posibilidad de que un perceptrón sea 1 sea muy grande, cuando b es grande, y que la probabilidad de que un perceptrón sea 1 sea muy pequeña, cuando b es muy negativo:

$$producción = \begin{cases} 0 \;\; si \; w \cdot x + b \leq 0 \\ 1 \;\; si \; w \cdot x + b > 0 \end{cases}$$

Fórmula 0.2. Salida de la neurona simplificada (Nielsen. Michael 2019)

0.2.2 **NEURONAS SIGMOIDEAS**

Las neuronas sigmoideas son neuronas artificiales como los perceptrones, pero, al contrario que ellos, estas pueden modificar los pesos o sesgos de en la red neuronal de manera que no se cree una gran variación en la salida de la red, esta propiedad hace posible el aprendizaje de la máquina. Esta propiedad se debe a que las neuronas sigmoideas funcionan de manera diferente a los perceptrones. Estas, al igual que el perceptrón, tienen entradas, X_1, X_2, …, pero en vez de ser binarias, las entradas pueden tomar cualquier valor entre 0 y 1. Además también tienen pesos para cada entrada, w_1, w_2, … y un sesgo general, b. Pero la salida en vez de ser binaria, esta, al igual que sus entradas, puede tomar cualquier valor entre 0 y 1, y ese valor viene determinado por la función sigmoide $\sigma(w \cdot X + b)$, la cual tiene la siguiente definición:

$$\sigma(w \cdot X + b) = \frac{1}{1 + \mathrm{e}^{-w \cdot X - b}}$$

Fórmula 0.3. Función sigmoide (Nielsen. Michael 2019)

De manera que cuando $w \cdot X + b$ es grande y positivo, la salida de la neurona se aproxima a 1, cuando $w \cdot X + b$ es pequeño y negativo, la salida de la neurona se aproxima a 0 (de momento, estas propiedades son iguales a las de los perceptrones), pero cuando $w \cdot X + b$ es de tamaño normal, hay mucha desviación del modelo del perceptrón. Esto se entiende mejor con la representación de la función sigmoide:

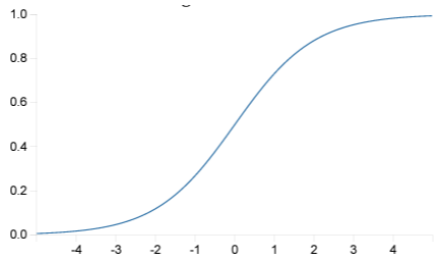

Figura 0.3. Representación de la función sigmoide (Nielsen. Michael 2019)

Esta es una función suavizada de una función escalón, la cual define a los perceptrones:

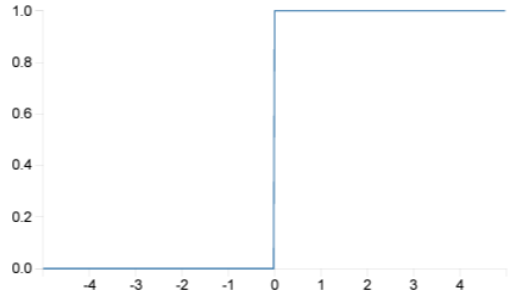

Figura 0.4. Representación de una función escalón (Nielsen. Michael 2019)

La suavidad de la función sigmoide permite que con pequeñas variaciones de los pesos o del sesgo se produzca, tan solo, una pequeña variación en la salida de la neurona. La variación de la salida se puede determinar por medio de una fórmula:

$$\Delta producción \approx \sum \frac{\partial producción}{\partial w_j} \Delta w_j + \frac{\partial producción}{\partial \mathrm{b}} \Delta \mathrm{b}$$

Fórmula 0.4. Fórmula de la salida de la neurona (Nielsen. Michael 2019)

Donde la producción es la salida, los pesos son w_j y el sesgo es b.

La presencia de derivadas parciales en la fórmula nos dice que la variación de la salida es una función lineal de las variaciones de sesgo y pesos.

Esta linealidad facilita la elección de pequeños cambios en los pesos y sesgos para lograr cualquier pequeño cambio deseado en la salida.

Otra cuestión que hay que tener en cuenta con las neuronas sigmoides es su representación en la salida, ya que el resultado no tiene por qué ser 1 en la salida. Por esto mismo, podemos establecer un **límite**, como por ejemplo 0,5, donde cualquier salida con valor menor a 0,5 es falsa y aquellas que valen 0,5 o mayores son verdaderas.

0.2.3 **ARQUITECTURA DE LAS REDES NEURONALES**

Una red neuronal es un grupo interconectado de neuronas artificiales que puede tener, por ejemplo, la siguiente estructura:

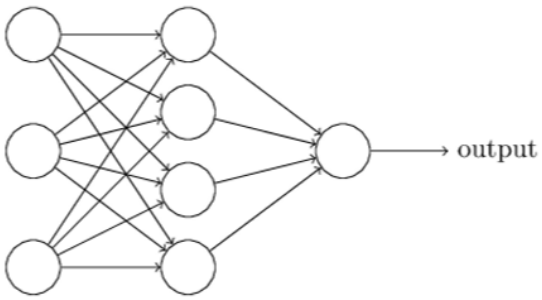

Figura 0.5. Ejemplo de red neuronal (Nielsen. Michael 2019)

En este ejemplo de estructura de una red neuronal, la capa más a la izquierda se llama **capa de entrada** y las neuronas dentro de esta capa se llaman **neuronas de entrada**. La capa más a la derecha es la **capa de salida** y contienen las neuronas son las **neuronas de salida**. La capa intermedia se denomina **capa oculta**.

0.2.4 **APRENDIZAJE CON DESCENSO DE GRADIENTE**

Con lo realizado hasta el momento, la red neuronal no tiene la capacidad de aprender. Lo primero que necesitamos, para que la red aprenda, es un conjunto de datos para educar a la red, estos se denominan conjunto de datos de entrenamiento. Por ejemplo, supongamos que queremos realizar una red neuronal capaz de reconocer dígitos escritos a mano, un conjunto de datos de entrenamiento para esta red sería una serie de dígitos escritos a mano con sus respectivas etiquetas del dígito al que se está refiriendo:

Figura 0.6. Conjunto de datos de entrenamiento para una red que reconoce dígitos escritos a mano (Cabrera et al. 2016)

Lo siguiente, sería poner en práctica la red con datos nuevos y observar con cuanta precisión esta clasifica los nuevos datos. Para enseñar al sistema los datos que está clasificando erróneamente, definiremos la **función de costos, costo cuadrático o error medio cuadrático**, esta funciona restando los valores que idealmente deberíamos obtener a los valores obtenidos y elevando esto al cuadrado. La fórmula de la función de costos es la siguiente:

$$C(w,b) = \frac{1}{2n}\sum_{x}||y(x) - a||^2$$

Fórmula 0.5. Fórmula de la función de costos (Nielsen. Michael 2019)

Donde *w* es la colección de todos los pesos en la red, b es el sesgo, *n* el número total de entradas de entrenamiento, *a* es el vector de salidas deseadas e *y(x)* es el vector de salidas obtenidas. La notación ||*v*|| denota la función de longitud habitual para un vector *v.*

El resultado de la función de costos *C(w,b)* será bajo cuando la red tiene un gran promedio de aciertos y alto cuando la red no tiene un gran número de aciertos.

Para minimizar el valor de la función de costos, de manera que la red tenga mayor número de aciertos, lo que necesitamos es encontrar algún mínimo en la función. Por ejemplo, si simplificamos la función de costos en *C(v)*, donde hemos reemplazado *w* y *b* en

la variable *v*, en el caso de una función de costos simple que tan solo tiene dos entradas v_1, v_2 (en el plano XY) y una salida *C* (en el eje Z):

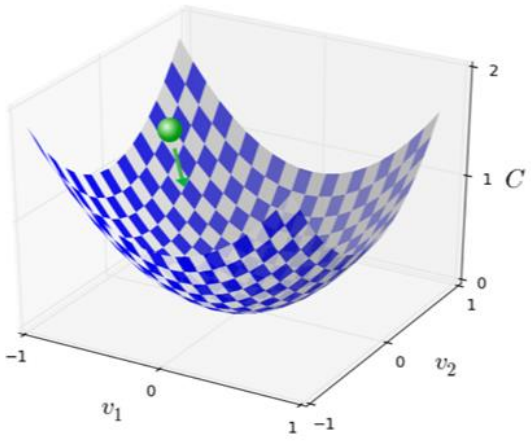

Figura 0.7. Ejemplo de una función de costos con dos entradas y una salida(Nielsen. Michael 2019)

En la imagen anterior, los pesos de la red no se encuentran en el mínimo de la función. Para encontrar la dirección en la que se encuentra el mínimo de la función, necesitamos encontrar el negativo del gradiente[1] $-\nabla C(w, b)$. El gradiente de *C* es un vector que se calcula de la siguiente manera:

$$\nabla C = \left(\frac{\partial C}{\partial v_1}, \ldots, \frac{\partial C}{\partial v_m}\right)^T$$

Fórmula 0.6. Fórmula del gradiente (Nielsen. Michael 2019)

T es la operación de transposición para convertir los vectores de fila en vectores de columna y los pesos y sesgos de cada neurona artificial están incluidos en las variables $v_1, \ldots, v_m$.

[1] El gradiente de una función da la dirección de ascenso más inclinada, por tanto, su negativo nos da la dirección en la cual la función disminuye más rápidamente.

El resultado del negativo del gradiente nos daría un vector con los valores que debería incrementar o decrecer cada peso w_i y sesgo b y cuanto (cuanto mayor sea el valor, más debe incrementarse v_i y cuanto menor sea el valor, más debe decrecer v_i).

0.2.5 **RETROPROPAGACIÓN**

Pongámonos en el caso de la red neuronal que no funciona correctamente (el error medio cuadrático es alto). El gradiente negativo de la función de costos $-\nabla C(v)$ nos da la información sobre lo que tenemos que cambiar en los pesos y los sesgos para que la red tenga un mejor rendimiento. Para ajustar el valor de la capa de salida de manera que se ajuste al resultado que queremos obtener, lo podemos hacer de tres formas:

- **Incrementando el sesgo b**.
- **Incrementando los pesos w_j;** los pesos de una capa se pueden incrementar de manera proporcional a las activaciones de la capa anterior a_j.
- **Cambiando las activaciones de la capa anterior a_j;** las activaciones de una capa no las podemos cambiar de manera manual como los pesos o los sesgos, pero estas pueden incrementar de manera proporcional a los pesos w_j que la preceden.

Estos cambios en la capa de salida provocan cambios proporcionales en las capas anteriores, de manera que todos los cambios necesarios en la capa de salida se informan a la capa anterior y esta, a su vez, informa de los cambios que necesita a la capa anterior y así consecutivamente. Este proceso es lo que denominamos **retropropagación** o **propagación hacia atrás** y es el método por el cual las redes neuronales aprenden. Además, su algoritmo se utiliza para calcular el gradiente de la función de costos $\nabla C(v)$.

0.2.6 **RED NEURONAL CONVOLUCIONAL**

Todo lo visto hasta ahora pertenece a las características de las **redes neuronales multicapa.** En este tipo de redes, los inputs que maneja el sistema es un vector con variables independientes, por lo tanto, ante una gran cantidad de datos de entrada se generarían unos tiempos de entrenamiento y de testeo inmensos. Esto limita el uso de redes neuronales multicapa para muchas actividades. Por ejemplo, en el ámbito del análisis de imágenes, las redes neuronales multicapa no pueden trabajar con imágenes con altas resoluciones.

Sin embargo, existen otro tipo de redes, denominadas **redes neuronales convolucionales,** las cuales pueden trabajar con grandes estructuras de datos de entrada, ya que son capaces de simplificar los datos de entrada y procesar la información realmente necesaria para obtener una buena respuesta.

Las redes convolucionales son un tipo especial de redes neuronales para procesar datos con tipología cuadriculada(Casas Roma et al. 2019). Debido a esto, las redes neuronales convolucionales son las más utilizadas para el análisis de imágenes.

El funcionamiento de las redes neuronales convolucionales se debe a los procesos que se producen en cada una de sus capas de red:

- **Capa de entrada**: normalmente la entrada de una red convolucional es una fotografía *m x n x r* donde *m* y *n* es tanto la altura como el ancho de la imagen y *r* es el número de **canales,** estos son los encargados de guardar la información del color de la imagen[2].
- **Capa convolucional:** esta etapa es de gran importancia para la síntesis de datos de entrada. Esto se debe a que en esta etapa se restringe el número de conexiones entre las neuronas de la capa oculta y los elementos de entrada, de manera que cada neurona de la capa oculta solo estará conectada a un pequeño subconjunto de elementos de la imagen de entrada.

  En la convolución[3] que existe en esta capa, el primer argumento (*f*) es la entrada y el segundo argumento (*g*) un **filtro** o **kernel** (vector multidimensional de parámetros). Esto nos da como resultado la función (*s*) la cual se refiere al **mapa de características** o **feature map**:

[2] Una imagen RGB tiene 3 canales: rojo, verde y azul, mientras que una imagen de escala de grises solo tiene 1 canal.

[3] La convolución es una operación matemática sobre dos funciones (f y g) que produce una tercera función (s) que expresa como la forma de una es modificada por la otra (Casas Roma et al., 2019).

$$s(t) = (f * g)(t) = \sum_{\tau=-\infty}^{\infty} f(\tau)g(t-\tau)$$

Fórmula 0.7. Mapa de características de una red convolucional (Nielsen. Michael 2019)

Donde * denota la operación de convolución y τ representa la posición en la que se está realizando la convolución. De manera que al aplicar la convolución entre una imagen *m x n x r* de entrada *A* de y un filtro *B* de fotografía *u x v x q* (normalmente *r* y *q* son iguales) obtenemos una matriz *C* de dimensiones *(m – u +1) x (n – v +1) x p*, donde *p* es el número de filtros que queremos aplicar, de manera que la fórmula de la convolución nos quedaría:

$$C(i,j) = (A * B)(i,j) = \sum_{m=0}^{i} \sum_{n=0}^{j} A(i+m, j+n)B(m,n)$$

Fórmula 0.8. Matriz de la convolución de píxeles (Nielsen. Michael 2019)

Así la imagen y el filtro se superponen y se calcula la convolución entre los respectivos elementos. Esto, aplicado a un ejemplo real, quedaría de la siguiente manera:

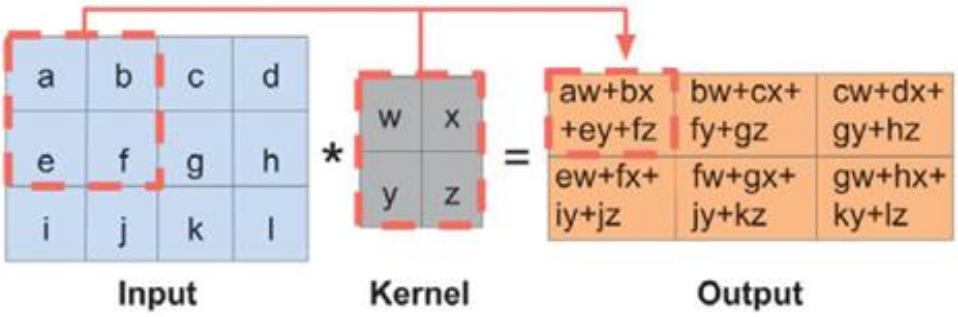

Figura 0.8. Ejemplo 1 de cómo funciona la convolución de imágenes (Casas Roma et al. 2019)

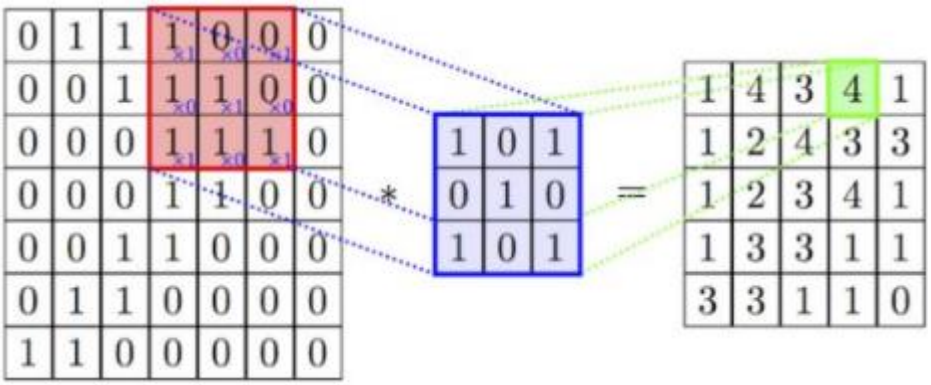

Figura 0.9. Ejemplo 2 de cómo funciona la convolución de imágenes (Traore et al. 2018)

El filtro se irá desplazando de manera iterativa sobre la imagen, hasta que, una vez haya recorrido toda la imagen, se obtendrá la **matriz de activación** que vemos como salida en las dos figuras anteriores.

De esta manera, en la capa convolucional se consigue reducir el número de elementos que conforman la red además de detectar las características útiles a la hora de analizar la imagen.

- **Capa de agrupamiento (pooling)**: después de la capa de convolución, se encuentra la capa de agrupamiento o pooling. El objetivo de esta capa es simplificar aún más el número de datos que va a procesar el sistema, así como ayudar con la caracterización de la imagen obteniendo y localizando los rasgos predominantes en ella.

  Debido a que las imágenes poseen la propiedad de ser estacionarias[4] y en la capa convolucional se ha creado un mapa de características, el propósito de esta capa es analizar en que zonas se encuentran los rasgos predominantes. Para realizar este agrupamiento o *pooling*, de rasgos existen diferentes métodos de los cuales cabe destacar el ***max-pooling*** y el ***mean-pooling*** o ***average-pooling***.

  Por ejemplo, tenemos una matriz de entrada de 4x4 y queremos obtener una de salida de dimensiones 2x2. Para ello, cogemos la entrada y la dividimos en 4 regiones y

[4] Esto quiere decir que algunas de sus características que existan en alguna parte de la imagen pueden encontrarse en otra posición totalmente distinta.

dependiendo del *pooling* se hará lo siguiente: si es *max-pooling* se escogen los elementos de mayor valor de cada regíón y se guarda en nuestra matriz de salida como en la figura siguiente, si es *mean-pooling*, se realiza la misma operación, pero en vez de guardar los elementos de mayor valor se realiza una media de los valores de cada región y se guardan en la matriz de salida.

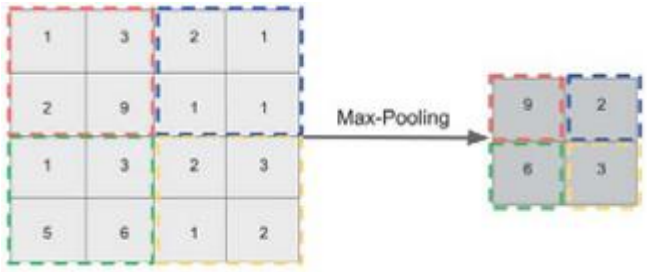

Figura 0.10. Ejemplo de una operación de max-pooling (Casas Roma et al. 2019)

- **Capa totalmente conectada (fully connected)**: esta capa se compone de una red neuronal multicapa, como las vistas anteriormente, encargada de procesar la salida de la capa de agrupamiento y dar como resultado la salida final.

Las redes neuronales convolucionales solo tienen una capa de entrada y una capa totalmente conectada de salida. Sin embargo, estas pueden tener varias capas convolucionales seguidas de capas de agrupamiento. De esta manera, un ejemplo de la estructura de una red neuronal convolucional es la que podemos encontrarnos a continuación en la figura siguiente:

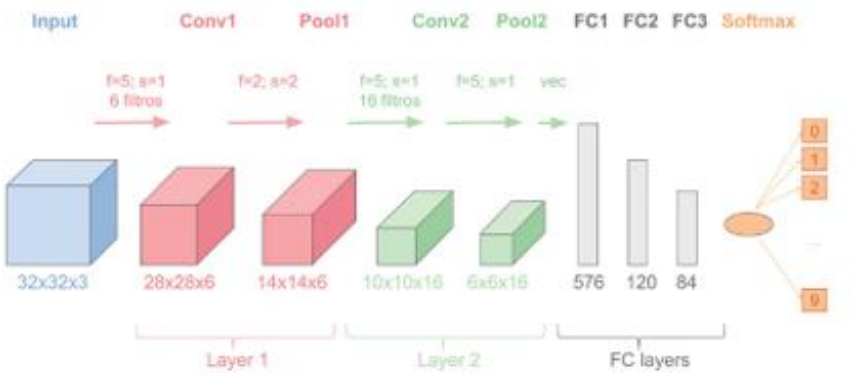

Figura 0.11. Ejemplo de la estructura de una red neuronal convoluciona (Casas Roma et al. 2019) l

1 Estado del arte

1.1 LÓGICA DIFUSA (FUZZY LOGIC) PARA EL PROCESAMIENTO DE IMÁGENES

1.1.1 INTRODUCCIÓN A LA LÓGICA DIFUSA

La teoría de los conjuntos difusos es una herramienta matemática para interpretar de manera computable los conceptos abstractos del lenguaje natural(Caponetti 2017). Las entidades que existen para computar estes conceptos se denominan **conjuntos difusos**, estos representan propiedades de los objetos, como, por ejemplo, *frío*, *grande*, *vacío*, etc. En la lógica difusa, al contrario que en la lógica tradicional[5], los valores varían de 0,0 (totalmente falso) a 1,0 (totalmente verdadero).

Los conjuntos difusos permiten recuperar datos incompletos, ruidosos o imprecisos, así como, desarrollar modelos de los datos que proporcionen más información. estes conjuntos se representan en una gráfica mediante **funciones de membresía o pertenencia**, estas funciones pueden ser de varias formas, pero las más comunes son la triangular, la trapezoidal y la gaussiana, como se representan en la figura siguiente.

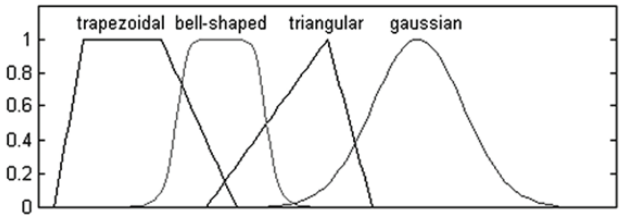

Figura 1.1. Principales funciones de membresía (Caponetti 2017)

[5] En la lógica tradicional, los valores solo pueden ser 0 (falso) ó 1 (verdadero)

Consideremos, por ejemplo, la variable difusa *temperatura*, una serie de conjuntos difusos de esta variable podría ser *frío (menos de 25ºC)*, *templado (entre 15ºC y 35ºC)* y *caliente (mayor de 25ºC)* y una representación posible sería la siguiente:

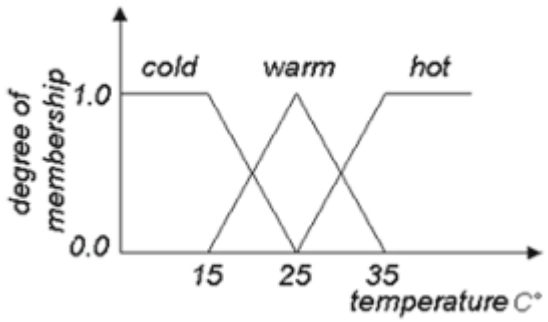

Figura 1.2. Ejemplo de conjuntos difusos para la variable temperatura(Caponetti 2017)

Estos sistemas, para representar la imprecisión del mundo real, utilizan las reglas IF-THEN, las cuales se expresan en lenguaje natural. La colección de las reglas se llama **base de conocimientos** y se utiliza para aproximar el mapeo de entradas y salidas desconocido. Un ejemplo de estas reglas es el siguiente:

R1: IF *temperatura* es *fría* THEN el calentador se *enciende*

Antes de hablar sobre las aplicaciones de la lógica difusa es necesario definir una serie de términos:

- La **fuzzificación** es el proceso que convierte las entradas nítidas en conjuntos difusos definidos en el espacio de entrada.
- La **desfuzzificacion** es el proceso de convertir los conjuntos difusos en unos nítidos. Este procedimiento es necesario debido a que algunos sistemas, por ejemplo, las aplicaciones de control difuso requieren una salida nítida.

1.1.2 **PROCESAMIENTO DE IMÁGENES DIFUSAS**

Se conoce por **procesamiento de imágenes difusas** a todas las actividades que representan y procesan imágenes, segmentos o características como conjuntos difusos.

Algunas de las actividades que pueden realizarse en imágenes mediante métodos difusos son:

- En un problema de mejora de contraste, definir un píxel más o menos brillante.
- En un problema de segmentación de imágenes, definir si un píxel está en el borde entre dos regiones.
- En un problema de análisis de escenas y de comprensión de imágenes, definir un rostro.

Las imágenes suelen tener cierta incertidumbre de los datos, esta imprecisión puede tratarse modelando un conjunto difuso. De esta manera, conceptos imprecisos como la oscuridad, el brillo, el contraste pueden ser expresados por medio de la lógica difusa. El problema de este método es que, debido a la aleatoriedad (la cual es cuestión de la Probabilidad) y de la ambigüedad (la cual es cuestión de los conjuntos difusos) de los datos, los resultados son inciertos.

Tradicionalmente, los datos de las imágenes se procesaban en el plano de niveles de gris, especificando el nivel de brillo. En cambio, en el procesamiento de imágenes difusas se mapea el plano de nivel de gris en el plano de pertenencia mediante la difuminación de la imagen.

Para la realización de la **fuzzificación de imágenes**, se modela la figura mediante conjuntos difusos del mismo tono (monótonos) sin modificar los píxeles. Por ejemplo, queremos definir en el conjunto *Dark* los niveles de gris *g* que comparten la propiedad de ser oscuros y queremos que este conjunto esté entre los niveles 0 y 100 de gris, para ello definimos dos parámetros, en este caso 50 y 100 y definimos:

- Los niveles $g \leq 50$ son elementos completos del conjunto *Dark*.
- Los niveles $g \geq 150$ no pertenecen al conjunto *Dark*.
- Los niveles $50 < g < 150$ tienen un grado parcial de pertenencia al conjunto *Dark*.

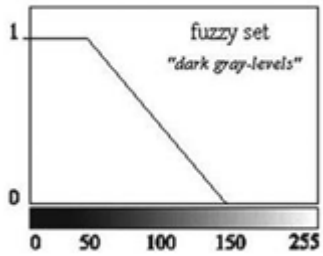

Figura 1.3. Conjunto difuso Dark (Caponetti 2017)

Para transformar la imagen original en una **imagen difusa** se utiliza una función de membresía adecuada denominada **fuzzifier.** Se pueden utilizar diferentes funciones de pertenencia, como la función gaussiana y la función sigmoide.

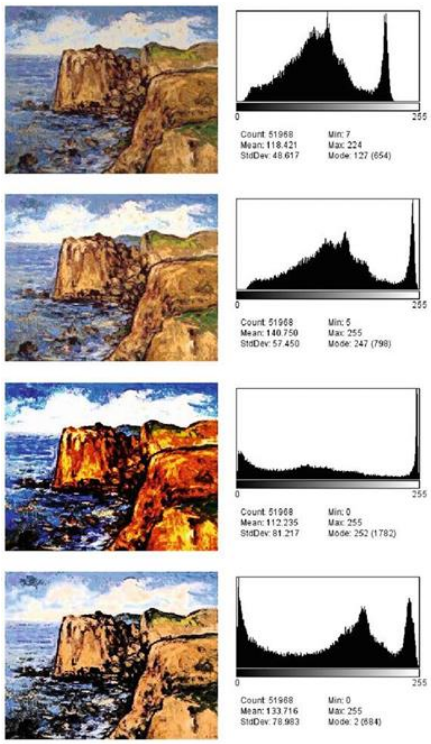

Figura 1.4. Ejemplos de modificación del contraste en una imagen utilizando diferentes funciones de pertenencia (Caponetti 2017).

Muchas veces es necesario una representación nítida en la salida o recuperar el conjunto nítido original ya que, aunque contenga menos información, suele ser más fácil de interpretar y comprender. Para ello es necesario realizar una **defuzzificación de la imagen**. El método más utilizado de la defuzzificación se denomina **método del centroide**

y consiste en realizar un corte en el centro del área de la función de membresía. Matemáticamente se expresa de la siguiente manera:

$$C = \frac{\int_s x\mu(x)\,dx}{\int_s \mu(x)\,dx}$$

Fórmula 1.1. Representación matemática del método del centroide (Caponetti 2017)

Donde $\mu(x)$ es la función de pertenencia del conjunto de salida, cuya variable es *x* y *S* es el dominio o rango de integración.

1.2 DETECCIÓN DE OBJETOS

Se denomina **detección de objetos** al conjunto de tareas para determinar diferentes tipos de elementos en una imagen con una cierta precisión. Estos elementos pueden ser tanto cuerpos (animales, letras, personas, autos, etc.) como anomalías (daños en materiales, irregularidades anatómicas, etc.).

En la detección de objetos podemos distinguir dos técnicas:

- **Clasificación de imágenes**: predecir el tipo o la clase de un objeto en una imagen. En este caso el input sería una imagen con un solo objeto y el output una etiqueta designando el que objeto hay en la imagen
- **Localización de objetos**: esta etapa se encarga de ubicar los distintos objetos en una imagen. En este caso la entrada es una imagen con uno o más objetos y la salida una máscara o cuadro delimitador indicando dónde están los distintos objetos.

La unión de estas dos técnicas da lugar a la detección de imágenes, en donde se detectan uno o varios objetos y su ubicación en una imagen.

Existen tres tipos de técnicas distintas para el reconocimiento de objetos:

- **Clasificación de imágenes**: los algoritmos de esta técnica crea una lista con los objetos existentes en la imagen.
- **Localización de un solo objeto**: los algoritmos producen una lista de objetos existentes en la imagen y la localización de las susodichas imágenes.

- **Detección de objetos**: los algoritmos producen, al igual que en la localización de un solo objeto, una lista de objetos existentes y su localización.

Tradicionalmente, los algoritmos de detección de objetos se realizaban a mano, con características para capturar la información relevante de las imágenes. Sin embargo, debido a que estos enfoques convencionales no pueden explotar en su totalidad volúmenes de datos extremadamente grandes, la detección de objetos empezó a llevarse a cabo en proyectos de *deep learning*, el cual, a diferencia del aprendizaje automático tradicional, una red neuronal se encarga de la extracción de características y la clasificación.

A continuación, analizaremos las redes neuronales convolucionales basadas en regiones, cómo funcionan, los tipos que hay y la manera que existe de medir su precisión

1.3 RCNN (REDES NEURONALES CONVOLUCIONALES BASADAS EN REGIONES)

Las **RCNN** son un conjunto de técnicas avanzadas de **deep learning** mucho más eficaces que las CNN convencionales en ciertas aplicaciones de la visión artificial (detección de objetos, categorización de imágenes, segmentación de imágenes, etc.). Esto se debe a que las redes neuronales convolucionales solo identifican la clase de un único objeto en la imagen mientras que las RCNN son capaces de hallar la ubicación de los objetos. Debido a que con el tiempo este tipo de redes neuronales convolucionales ha evolucionado, existen varios modelos de RCNN, los cuales veremos cómo funcionan a continuación:

1.3.1 RED BASE O BACKBONE

Las **redes base** o **backbone** son la arquitectura base de las R-CNN. Existen diferentes tipos de arquitectura de redes base que evolucionan todos los años. Algunas de estas aquitecturas están descritas a continuación:

- **LeNet**: diseñada en la epoca de los 90, fue una red diseñada para leer digitos escritos a mano. Es la primera arquitectura convolucional, la cual consiste en 2 capas convolucionales con ReLu[6] y capas de pooling, seguidas de otra capa convolucional, después dos capas fully-conected y por último una capa softmax.
- **AlexNet**: esta red neuronal es mucho más profunda que LeNet. En esta se utiliza una ReLu para añadir no linealidad, lo que permite la aceleración de la red. Esta red tiene 5 capas convolucionales, 3 capas fully-conected y una capa de salida. AlexNet permite 62,3 millones de parámetros. Esta red participó en el desafío de reconocimineto visual a gran escala imagenet (ILSVRC) en 2012, en el cual se examinan la detección de objetos y la clasificación de imágenes. AlexNet tuvo una tasa de error de clasificación del 15,3 %. Los campeones posteriores fueron ZF Net GoogleNet y VGGNet.
- **VGGNet**: esta red multicapa perteneciente al Visual Geometry Group en el cual nos encontramos VGG 16 (con 16 capas convolucionales) y VGG 19 (con 19 capas convolucionales). La arquitectura de la red consta de una capa de entrada, varias capas convolucionales (varían según el tipo como ya se ha mencionado previamente), capas ocultas con ReLu y 3 capas fully-conected (2 capas con 4096 canales y la tercera de ellas con 1000 canales (uno para cada clase))
- **GoogleNet**: esta red consigue una buena precisión, pero requiere de una gran potencia de cálculo debido a que el orden de cálculos que soporta es muy alto. Es capaz de clasificar imágenes en 1000 categorías de objetos. La arquitectura de esta red neuronal consta de 22 capas de profundidad con 27 capas de *pooling* incluidas.
- **ResNet**: Residual Network fue el vencedor del ILSVRC 2015. Las ResNets son los mejores modelos de redes neuronales convolucionales para las RCNN. En las redes anteriores, con el aumento de profundidad de la red, la precisión de esta también aumentaba. Este aumento de profundidad requiere un cambio en los pesos que hace que la predicción se vuelva pequeña en las capas iniciales. Además, este incremento requiere un enorme espacio de parámetros. Las Residual Networks solucionaron estos problemas introduciendo los bloques residuales que se basaban en el método de salto

[6] ReLu (unidad lineal rectificada) es una función de activación que en las redes neuronales artificiales es la responsable de procesar entradas ponderadas y ayudar a generar una salida

de conexiones, que permitía la conexión de unas capas con otras omitiendo algunas intermedias.

Esta red utiliza una arquitectura simple de 34 capas inspirada en VGG 19.

1.3.2 **RCNN**

La arquitectura RCNN incluye 3 pasos: primero se generan **propuestas de regiones independientes** de la categoría, después cada región pasa por una gran **red neuronal convolucional** (backbone) que extrae un vector de características y en el tercer paso la región pasa por una **SVM** (máquina de vectores de soporte) que la clasifica según sus características.

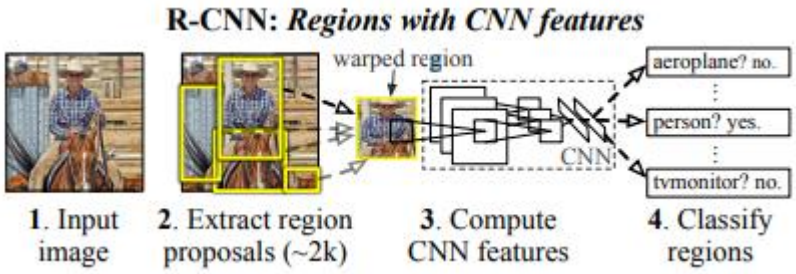

Figura 1.5. Arquitectura R-CNN (Brownlee 2019)

- **Propuestas de regiones**: estas son las regiones más pequeñas de la imagen que posiblemente contengan los objetos buscados en la imagen de entrada (puede haber un total de 2000 propuestas de región), en el caso de la RCNN básica, esta región se define con un marco denominado cuadro delimitador:

Figura 1.6. Cuadros delimitadores detectando una persona y una televisión (Girshick et al. 2014)

Sin embargo, las propuestas de regiones originales suelen ser menos precisas y venir definidas de la siguiente manera:

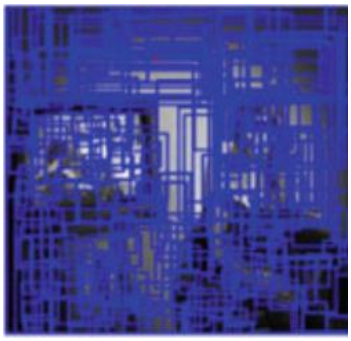

Figura 1.7. Conjunto inicial de propuestas de regiones en una RCNN (Girshick et al. 2014)

Para reducir las propuestas de región, la R-CNN utiliza un algoritmo denominado **búsqueda selectiva**, el cual funciona combinando regiones segmentadas más pequeñas para generar una propuesta de región y utilizando la segmentación de los colores en la imagen. Este algoritmo empieza creando muchas ventanas pequeñas de propuestas de regiones y utiliza el algoritmo voraz[7] para que las regiones crezcan y sean menos. Posteriormente localiza los colores similares en regiones y los fusiona.
La similitud entre regiones puede ser calculada de la siguiente manera:

$$S(a, b) = Stextura(a, b) + Stamaño(a, b)$$

Fórmula 1.2. Cálculo de la similitud entre regiones (Girshick et al. 2014)

En donde *Stextura(a, b)* es la similitud visual y *Stamaño(a, b)* es la similitud entre regiones.

[7] El algoritmo voraz es en una estrategia de búsqueda que consiste en intentar obtener una solución óptima a partir de las opciones más óptimas

Figura 1.8. Propuestas de regiones utilizando el algoritmo de búsqueda selectiva (Girshick et al. 2014)

- **Red neuronal convolucional de características**: el segundo paso consiste en analizar las propuestas de regiones con la red base predeterminada, la cual nos dará como resultado un vector de características. El input de esta red neuronal sería el cuadro delimitador de una de las propuestas de regiones detectada. Poniendo de ejemplo una red base AlexNet, en esta el input determinado deberá tener de dimensiones 224x224 y ser una imagen a color (RGB, profundidad 3). Para ello el cuadro delimitador se redimensiona para entrar en la red convolucional y pasar por sus capas hasta obtener el vector de características de 4096 dimensiones.

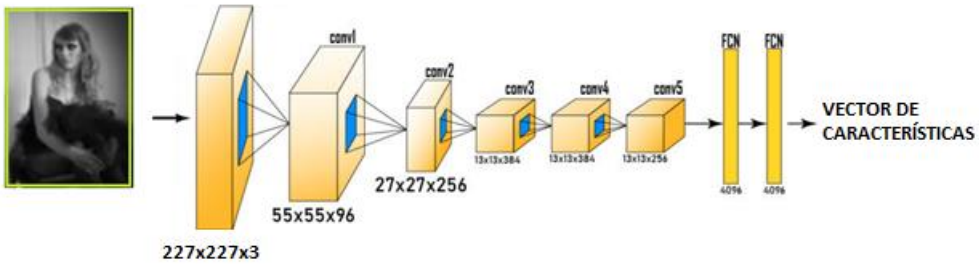

Figura 1.9. CNN AlexNet (Girshick et al. 2014)

- **SVM**: posteriormente el vector de características generado pasa por el SVM (máquina de vectores de soporte) el cual se trata de un hiperplano en un espacio de *N* dimensiones (*N=número de características*) cuyo objetivo es separar lo mejor posible los datos según sus características. De manera que si tenemos un sistema con dos objetos (*N=2*), un SVM para este podría ser el siguiente:

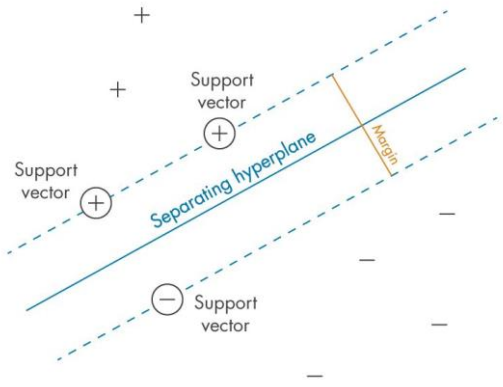

Figura 1.10. SVM de dos objetos (Matlab 2022)

Donde los puntos de datos, representados por los símbolos + y -, son los **vectores de características**. Estos están separados por el **hiperplano** en color azul, el margen es la anchura máxima de la región paralela al hiperplano dentro de la cual no hay puntos de datos.

Para mejorar la precisión de la ubicación el cuadro delimitador en la se utiliza el **regresor de cuadro delimitador** el cual consiste en modelo de regresión lineal que se entrena para predecir un nuevo cuadro delimitador.

1.3.3 **FAST RCNN**

La tradicional red neuronal convolucional basada en regiones tenía una serie de **limitaciones** en su uso:

- Detección de objetos lenta (la detección tarda 47s/imagen)
- Entrenamiento costoso en espacio y tiempo (con redes profundas como VGG16, este proceso tarda 2,5 GPU/días)
- El entrenamiento es un proceso de varias etapas

Fast RCNN es la mejora de RCNN y fue creado para solucionar estos problemas. La arquitectura de este sistema es la siguiente: la imagen entera es procesada por una **gran red neuronal convolucional** (backbone), la cual, mediante el **max-pooling** nos da como resultado un mapa de características convolucional con sus respectivas regiones de interés **RoI**. Estas regiones pasan por unas capas **fully-conected** y nos da como resultado un **vector de características**. Por último, este vector se ramifica en dos capas de salida: una,

mediante probabilidad softmax[8] , nos da la **información sobre la clase de objetos** L en la imagen, y la otra capa, mediante la regresión de caja delimitadora, genera 4 números (**coordenadas de la caja delimitadora**) para cada objeto.

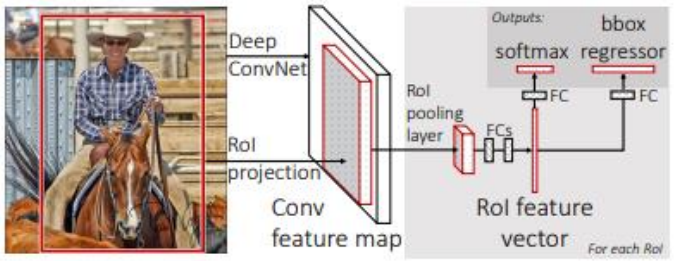

Figura 1.11. Arquitectura Fast RCNN (Verma 2021)

- **RoI pooling**: consiste en un max-pooling que permite obtener el mapa de características convolucional de una imagen a una escala muy pequeña Esta diminución de tamaño y del número de píxeles permite la convolución de todas las regiones de interés de una imagen a la vez en vez de tener que repetir el proceso de convolución 2000 veces (1 vez por cada región de interés) . Esto hace que Fast RCNN sea más rápida que R-CNN.

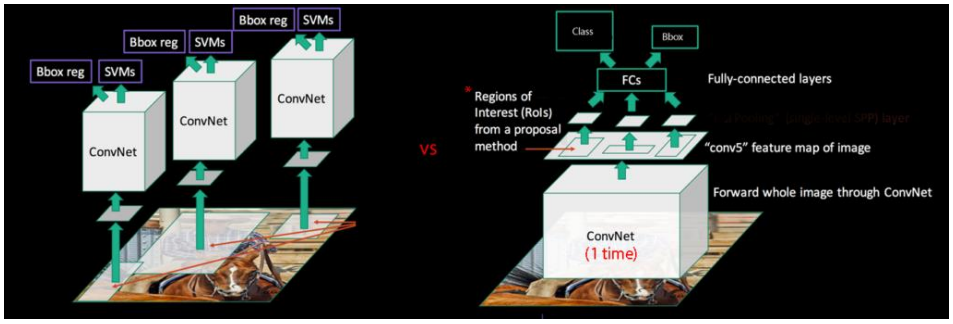

Figura 1.12. Comparativa RCNN vs Fast RCNN donde se puede visualizar que RCNN hace la convolución para cada región mientras que Fast RCNN la hace para toda la imagen completa (Verma 2021)

[8] Softmax: función matemática empleada para comprimir un vector de valores reales en uno en el rango [0, 1]

La capa **RoI pooling** utiliza el max pooling para convertir las características dentro de una región de interés en un pequeño mapa de características de dimensiones *H x W* (estas dimensiones son independientes de la región de interés). RoI puede ser representado como una ventana rectangular con cuatro parámetros *(r, c, h, w)*, en donde *(r, c)* son las coordenadas del punto superior izquierdo de la ventana y *(h, w)* son respectivamente la altura y el ancho.

RoI max pooling funciona dividiendo *h/H* y *w/W* y realizando el max-pooling a los valores de la ventana resultante.

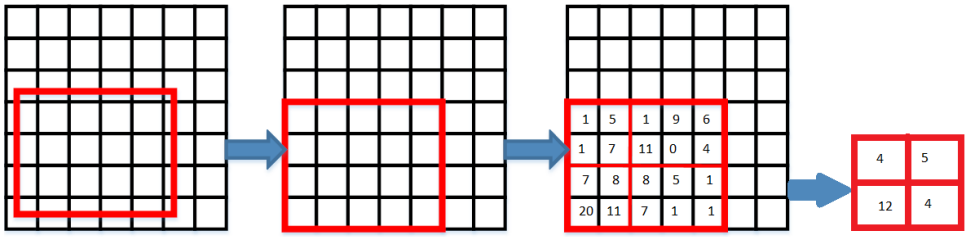

Figura 1.13. Ejemplo del funcionamiento del RoI pooling (Bai et al. 2020)

Si nos fijamos en el ejemplo de RoI pooling de la anterior figura podemos aclarar ciertos inconvenientes que este tiene. Al disminuir la escala del mapa de características, la región puede no coincidir con exactitud con los píxeles en el mapa y por tanto hay que hacer un redondeo con los pixeles cercanos. Además, al no ser cuadrada la región en este caso, el max-pooling en las celdas de la izquierda se hace con 4 pixeles mientras que en las dos celdas de la derecha se hace con 6 pixeles.

Debido a esto, se creó la implementación de **RoI Align**, un algoritmo capaz de solventar estos problemas de agrupamiento de regiones que no coinciden con los píxeles del mapa de características. Los primeros pasos de este algoritmo son los mismos que para el RoI pooling, se obtiene la región de interés y se sitúa en el mapa de características convolucional de manera que no coincide con los pixeles del mapa. Llegado a este punto, en vez de hacer un redondeo, dividimos la región de interés en la escala RoI pooling que deseemos y hacemos uso de la ecuación de interpolación lineal. Esto se trata de obtener cuatro puntos en cada uno de los cuadrantes de la RoI y poner un valor respecto al color y la distancia de los centroides de los píxeles cercanos

$$P = \frac{y_2 - y}{y_2 - y_1}\left(\frac{x_2 - x}{x_2 - x_1}Q_{11} + \frac{x - x_1}{x_2 - x_1}Q_{21}\right) + \frac{y - y_1}{y_2 - y_1}\left(\frac{x_2 - x}{x_2 - x_1}Q_{12} + \frac{x - x_1}{x_2 - x_1}Q_{22}\right)$$

Fórmula 1.3. Fórmula de RoI Align (Kemal 2020)

Siendo Q los valores de los pixeles cercanos al punto *P*, *(x,y)* las coordenadas del punto *P* y (x_1, y_1), (x_1, y_2), (x_2, y_1), (x_2, y_2) las coordenadas de $Q_{11}, Q_{12}, Q_{21}, Q_{22}$ respectivamente. De manera gráfica, el proceso que se realizaría sería el siguiente:

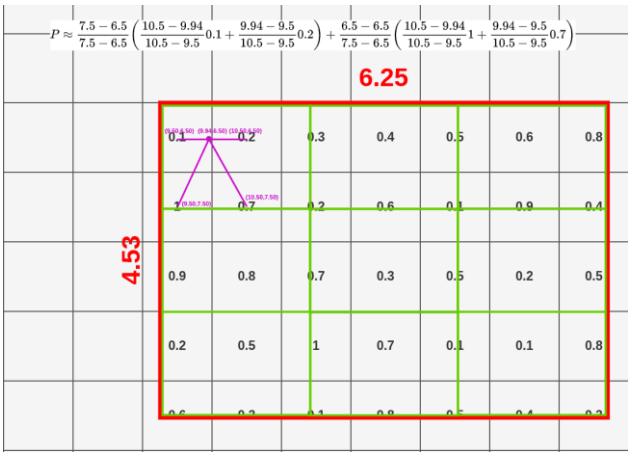

Figura 1.14. Ejemplo de RoI Align (Kemal 2020)

De esta manera, se crean 4 puntos equidistantes del centro de la celda con sus respectivos colores, se calcula la media entre sus colores y se puede hacer ya el RoI pooling.

1.3.4 **FASTER RCNN**

El sistema **Faster RCNN** se desarrolla como una mejora del Fast RCNN. Este funciona con la misma arquitectura que un Fast RCNN, pero a mayores, entre el mapa de características convolucionales y el RoI pooling, se implementa una **red de propuestas de regiones RPN**.

Red de propuestas de regiones (RPN): al contrario que RCNN y Fast RCNN, los cuales dependen del algoritmo de búsqueda selectiva para generar propuestas de región, Faster RCNN cuenta con un método para elegir regiones de interés denominado red de propuestas de regiones. Este funciona con una pequeña CNN entrenada para

buscar regiones en la imagen y utiliza la técnica 'atención'[9] para indicar al sistema de detección de objetos las regiones de interés encontradas por la red.

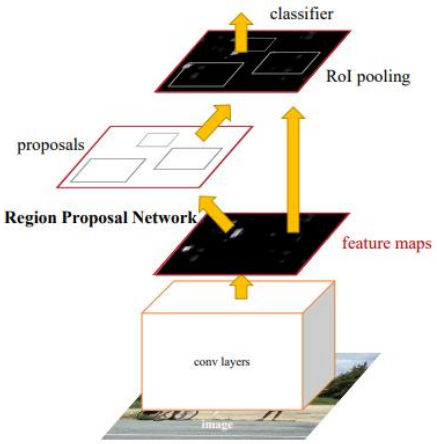

Figura 1.15. Implementación de la arquitectura RPN en un Faster RCNN (Ren et al. 2016)

Los modelos de detección que utilizan dos redes neuronales (backbone y RPN) se les denomina **modelos de detección de 2 etapas** y los que tan solo utilizan el backbone se les denomina **modelos de detección monoetapa.**

	RCNN	**Fast RCNN**	**Faster RCNN**
Método de propuestas de región	búsqueda selectiva	búsqueda selectiva	ROI
Tiempo de predicción	40-50s	2s	0,2s
Conjunto de datos de prueba (VOC 2007)	58,5	66,9	69,9

Tabla 1.1. Tabla modificada comparativa de los modelos RCNN, Fast RCNN y Faster RCNN (Verma 2021)

1.3.5 MASK RCNN

A diferencia de las RCNN vistas anteriormente, **Mask RCNN** se basa en la **segmentación de objetos** en vez de la caja delimitadora de objetos.

[9] En redes neuronales, la atención se refiere a la técnica que algunas de estas poseen para comunicarse con otras partes del sistema.

1.3.5.1 SEGMENTACIÓN DE OBJETOS

Se denomina **segmentación de objetos** al proceso de etiquetar cada píxel de una imagen en una clase particular. Los modelos de segmentación proporcionan el contorno exacto del objeto dentro de una imagen, es decir, proporcionan detalles píxel a píxel. De esta manera, la proposición de regiones de objetos en una imagen será mucho más precisa.

Distinguimos 3 tipos de segmentación de imágenes:

- **Segmentación semántica**: clasificación de los píxeles de una imagen en clases semánticas. El problema que conlleva este tipo de segmentación es que todos los píxeles pertenecientes a una clase se clasifican sin tener en cuenta más información. De manera que si tenemos una imagen con dos o más objetos de la misma clase que se superponen, este tipo de segmentación no distinguirá entre distintos objetos de la misma clase. Los fondos de una imagen se suelen segmentar de esta manera ya que la superposición no es un problema en estos casos.

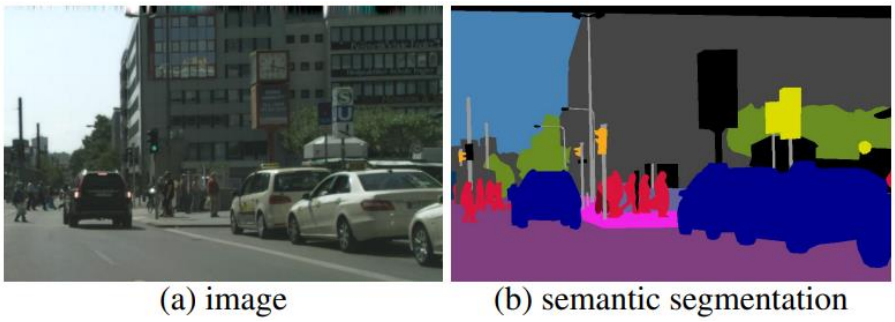

Figura 1.16. Ejemplo de segmentación semántica (Kirillov et al. 2021)

- **Segmentación de instancias**: en la segmentación de instancias se trata a múltiples objetos de la misma clase como objetos individuales distintos, de esta manera se evita la superposición de objetos de la misma clase existente en la segmentación semántica. Solo se suele utilizar para la segmentación de objetos (no de fondos).

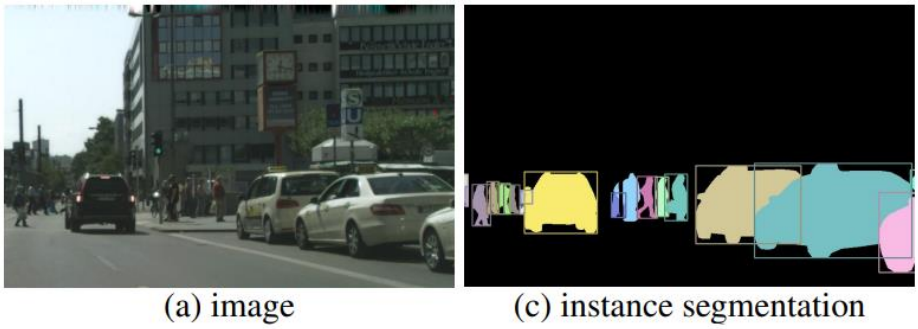

Figura 1.17. Ejemplo de segmentación de instancias (Kirillov et al. 2021)

- **Segmentación panóptica**: este tipo de segmentación se puede expresar como la combinación de la segmentación semántica (para los fondos) y segmentación de instancias (para los objetos). Esta es capaz de distinguir los objetos o instancias y a su vez reconoce las estructuras y texturas del fondo.

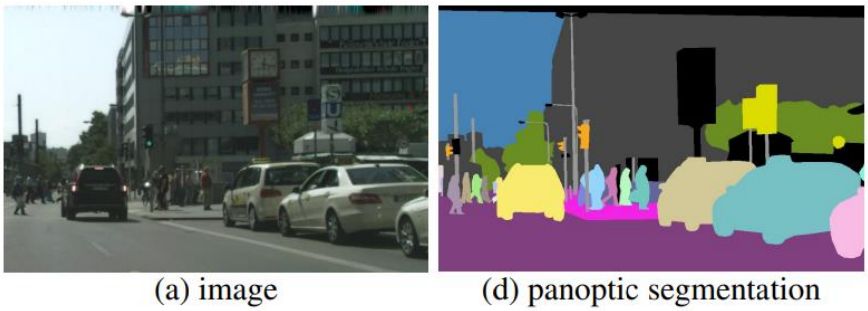

Figura 1.18. Ejemplo de segmentación panóptica (Kirillov et al. 2021)

1.3.5.2 ARQUITECTURA MASK RCNN

La arquitectura de la Mask RCNN es prácticamente idéntica a la de Faster RCNN, la única diferencia es la **segmentación de objetos.**

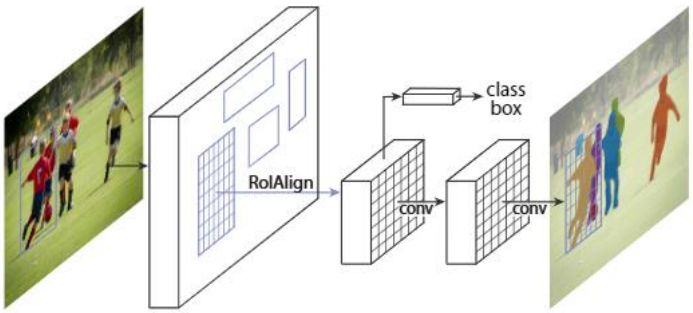

Figura 1.19. Arquitectura Mask RCNN (He et al. 2018)

1.4 TÉCNICAS PARA MEDIR LA PRECISIÓN DE LA DETECCIÓN DE OBJETOS

Las técnicas que existen para medir la precisión de un modelo de detección de objetos son las siguientes:

1.4.1 IOU (INTERSECCIÓN SOBRE UNIÓN)

La IoU una técnica utilizada para medir la precisión de un detector de objetos en un conjunto de datos. Para aplicar esta técnica necesitaremos **cuadros delimitadores ground-truth** (son aquellos que se trazan manualmente) y cuadros delimitadores predichos por nuestro modelo:

Figura 1.20. Ejemplo creado para representar cómo funciona IoU

La fórmula para el cálculo de la intersección sobre la unión es la siguiente:

$$IOU = \frac{Área\ de\ superposición}{Área\ de\ unión}$$

Fórmula 1.4. Fórmula del cálculo IoU (Rezatofighi et al. 2019)

Siendo el área de superposición y el área de unión las siguientes:

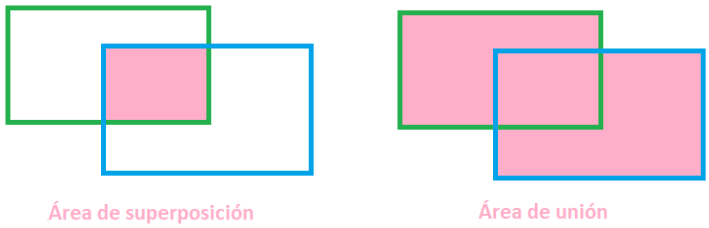

Figura 1.21. Ejemplo creado para representar el área de superposición y el área de unión

1.4.2 AP (PRECISIÓN PROMEDIO)

Cuando realizamos un proyecto de clasificación de imágenes, los resultados que obtendremos sobre la clasificación del objeto vendrán dados por un porcentaje de precisión el cual podemos programar para que llegado a cierto porcentaje nos indique el objeto.

En proyectos con poca precisión, muchas veces la clasificación será errónea. En estos casos hablamos de **falsos positivos** (*false positive (FP)*), si la clasificación es correcta será un **verdadero positivo** (*true positive (TP)*) y en el caso en el que no se clasifique un objeto se produce un **falso negativo** (*false negative (FN)*). La precisión sería la proporción TP y la recuperación sería la proporción de TP entre los posibles positivos:

$$Precisión = \frac{TP}{TP + FP}$$

$$Recuperación = \frac{TP}{TP + FN}$$

Fórmula 1.5. Fórmula precisión y recuperación en el AP (Lai y Lepetit 2016)

La definición de la precisión promedio es el área que se produce debajo de la curva formada en la gráfica de la precisión con respecto a la recuperación:

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

Fórmula 1.6. Definición de la precisión promedio (Lai y Lepetit 2016)

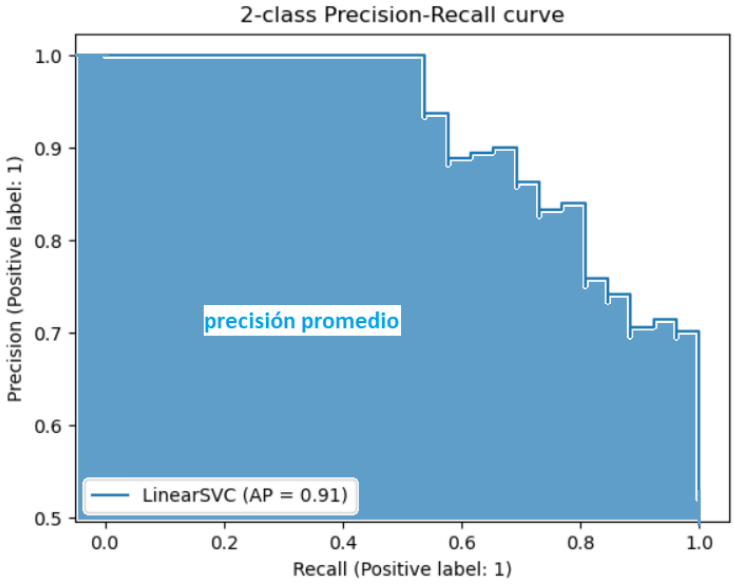

Figura 1.22. Ejemplo de una gráfica precisión-recuperación donde se indica el área de la precisión promedio (Scikit-learn 2022)

1.4.3 MAP (PRECISIÓN PROMEDIO MEDIA)

La mAP calcula la media de la precisión promedio (AP) de todas las clases de la imagen, por tanto, su fórmula es la siguiente:

$$mAP = \frac{1}{N}\sum_{i=1}^{N} AP_i$$

Fórmula 1.7. Fórmula para calcular el mAP (Lai y Lepetit 2016)

Siendo *N* el número de clases y *AP* cada uno de los *average precision*.

1.5 RECICLAJE DE RESIDUOS

El **reciclaje de materiales** es una tarea de gran importancia en la actualidad ya que aminora grandes problemas los cuales fomentan la contaminación, estos son: la necesidad de extraer nuevos materiales (minería, tala de árboles, etc.) y la refinación y procesamiento de materias primas. El reciclaje de los materiales está constituido en una serie de etapas:

- **Separación**: esta es el número de residuos que los habitantes separan en distintas clases para su posterior recogida.

- **Recogida** selectiva de residuos: conjunto de actividades que permiten llevar estos residuos del punto de separación a una planta de residuos
- **Selección**: esta es la clasificación mediante operaciones automáticas y manuales el material de la recogida selectiva. Existen varios tipos de selección que son necesarios analizar para la realización de este proyecto. Cuando los residuos llegan a una planta de selección, el primer paso es **abrir las bolsas** mediante una serie de cuchillas y posteriormente pasan por una serie de pasos que cuyo orden sería el siguiente

1. **Separación manual**: la separación se realiza manualmente por una cadena de trabajadores encargados de separar los distintos residuos.
2. **Separación automática**:

    1. **Separación volumétrica o densimétrica**: como su nombre indica, es la separación por forma o tamaño:

        - **Separación por aspiración**: aspiración de residuos flexibles y poco densos mediante una campana extractora.

    2. **Separación magnética**: recogida de residuos de acero o férricos mediante imanes permanentes o electroimanes.
    3. **Separación por inducción**: captación de residuos de aluminio mediante la repulsión que estos sufren ante las corrientes de Focault permitiendo así su separación.
    4. **Separación óptica**: detección de algunos residuos plásticos y de cartón mediante los distintos tipos de reflexión ante la luz que ofrecen estos materiales. Su recogida se realizará con un compresor el cual explusará aire para impulsar el residuo especificado.

- **Reciclado**: los residuos clasificados se transportan a plantas especializadas donde se someten a una serie de procesos para transformarlos en materia prima.

1.6 TIPOS DE PLÁSTICOS RECICLABLES

Como este proyecto se basa en la separación de botellas de plástico, vamos a ver que tipos de plásticos que son reciclables y su clasificación numérica la cual va vinculada a la facilidad de reciclaje que estos plásticos tienen:

- **PET** (Tereftalato de polietileno): es el tipo de plástico más fácil de reciclar y su clasificación numérica es 1. Podemos encontrar este plástico en las botellas de agua o de bebidas.
- **HDPE** (Polietileno de alta densidad): es un tipo de plástico altamente reciclable, con una clasificación numérica de 2 y podemos encontrarlo en muchos envases como botellas de leche o aceite de motores.
- **PVC** (Policloruro de vinilo): tiene una clasificación numérica de 3, por lo tanto, su reciclaje es más complejo que los vistos anteriormente. Lo encontramos en tuberías, pieles sintéticas, marcos de puertas y ventanas, tarjetas de crédito y revestimiento de cables.
- **LDPE** (Polietileno de baja densidad): es difícil de reciclar, tiene una clasificación numérica de 4 y lo encontramos en las bolsas, plástico de burbujas y aislantes.
- **PP** (Polipropileno): es difícil de reciclar, tiene una clasificación numérica de 5 y mayormente lo encontramos en tapones de botellas, pajitas y tupperwares.
- **PS** (Poliestireno): con una clasificación numérica de 6 es posible pero bastante difícil de reciclar. Se encuentra en hueveras, perchas, aislantes, rellenos para embalaje y materiales térmicos.
- Los plásticos con **clasificación numérica de 7** son una mezcla de los anteriores y son muy difíciles de reciclar debido a que no se puede tener certeza de las cantidades de resinas que contienen ya que no hay certeza de las proporciones de plásticos.

1.7 DETECTORES DE RESIDUOS QUE EXISTEN HOY EN DÍA

Hoy en día existen, algunos sistemas para la separación de residuos con plástico PET. Los inconvenientes de estos son que son muy costosos y complejos o que todavía están en desarrollo. Entre estos, podemos encontrar:

- **Separador óptico**: como vimos en el apartado de reciclaje de residuo, un clasificador óptico es un dispositivo que clasifica automáticamente los objetos en función de criterios visuales. Estas máquinas son capaces de clasificar cereales, alimentos y plástico. Los separadores ópticos utilizan varias tecnologías que incluyen cámaras, iluminación y software de aprendizaje automático.

  Este artefacto es capaz de detectar, mediante luz reflejada, colores no deseados, defectos en los materiales y materiales extraños, lo que permite la clasificación y a su vez un control de calidad de los materiales.

  La arquitectura que tiene este sistema es la siguiente; primero el objeto a clasificar entra por una tolva a una cinta deslizante que lleva el objeto hacia el sistema de clasificador óptico, el cual mediante un sistema de iluminación para que se vea con nitidez los inputs por una cámara o sensor capaz de detectar los objetos deseados o no deseados, posteriormente unas paletas son capaces de impulsar el input deseado y desear el otro para que quede clasificado en un divisor.

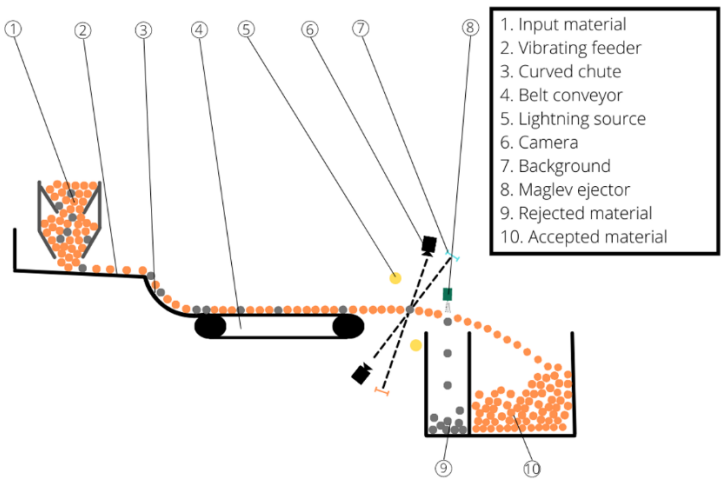

Figura 1.23. Arquitectura del funcionamiento de un separador óptico (MEYER Company 2022)

Los separadores ópticos más populares hoy en día son MACH Hyspec del grupo Machinex (hecha para clasificar productos de plástico productos de fibra) y otra realizada en España Ecoglass del grupo PICVISA (desarrollada para la recuperación de vidrio).

Entre los problemas de los separadores ópticos nos encontramos que son **máquinas complejas y costosas**, debido a que necesitan cámara con grandes objetivos que permitan ver las propiedades de los materiales de los residuos que está analizando (el tipo de grano en metales, aleaciones, etc.), lo que implica que existan pocas plantas de selección de residuos que utilicen estas máquinas e imposibilita que estas lleguen a países en vías de desarrollo, en algunos de los cuales la cantidad de residuos es muy elevada. Además, no tienen una precisión perfecta, debido a esto, posteriormente a la separación óptica se necesita una revisión manual.

- **MAX AI**: es una inteligencia artificial del grupo BHS – Bulk Handling Systems que a través de deep learning identifica materiales reciclables y otros elementos para su recuperación. Max emplea redes neuronales de múltiples capas y un sistema de visión para ver e identificar objetos. Esta máquina utiliza la arquitectura Mask RCNN para la búsqueda de los objetos. Este sistema todavía está en desarrollo. Puede clasificar de forma autónoma bandejas termoformadas, aluminio, fibra y botellas de PET. Después de la detección de los residuos mediante deep learning, un brazo robótico mediante ventosas y un sistema de aspiración de aire coge los residuos que detecta y los clasifica según su material.

Figura 1.24. Funcionamiento de la tecnología MAX AI (Max AI 2022)

Este sistema **todavía se está desarrollando** debido a que la tecnología que utiliza es muy innovadora.

1.8 PROGRAMAS Y LENGUAJES PARA LA PROGRAMACIÓN DE SISTEMAS DE VISIÓN ARTIFICIAL

Existen varios programas y lenguajes para realizar proyectos de visión artificial, entre los cuales destacan:

1.8.1 PROGRAMAS

- **Anaconda**: distribución de código abierto para realización de la ciencia de datos y el aprendizaje automático con Python y R que tiene como principal objetivo simplificar la administración y la implementación de paquetes. La distribución de Anaconda viene con más de 250 paquetes instalados automáticamente y se pueden instalar más de 7500 paquetes de código abierto adicionales desde PyPI. Entre los paquetes más importantes de Anaconda nos encontramos:

  - **Jupyter notebook**: entorno de desarrollo interactivo basado en la web de código abierto que se puede usar para crear y compartir documentos que contienen código en vivo, ecuaciones, visualizaciones y texto. El nombre viene de los principales lenguajes de programación que admite: Julia, Python y R.
  - **Spyder**: entorno de desarrollo integrado (IDE) dedicado para Python diseñado por y para científicos. Incluye un depurador, un generador de perfiles, una terminal de IPython, un explorador de archivos y numerosas capacidades más.
  - **Anaconda prompt**: terminal de línea de comandos propia de Anaconda. Es similar a la terminal de Windows, pero en ella se pueden utilizar los comandos de anaconda y Conda sin tener que cambiar los directorios o su ruta.
  - **Anaconda Navigator**: es una interfaz gráfica de usuario que permite iniciar aplicaciones, administrar paquetes, entornos y canales de Conda sin necesidad de usar comandos.

- **Google Colab**: perteneciente a Google Research, es un entorno de cuaderno Jupyter gratuito el cual se ejecuta totalmente en la nube. Se pueden realizar un gran número de tareas con Google Colab:

  - Crear/Cargar/Compartir cuadernos.
  - Escribir y ejecutar código de Python.
  - Importar/Guardar cuadernos de Google Drive
  - Servicio gratuito en la nube con GPU gratis
  - Tiene integrado PyTorch, TensorFlow, OpenCV
  - Importar conjuntos de datos externos (por ejemplo, desde Kaggle)

1.8.2 **LENGUAJES**

- **Python**: creado en 1990 por Guido van Rossum, Python es un lenguaje de código abierto orientado a objetos con el que se pueden realizar un amplio abanico de contenidos: aplicaciones web, automatización, aplicaciones big data, aplicaciones IA etc. Este lenguaje es compatible con numerosas librerías de inteligencia artificial (entre las fundamentales nos encontramos TensorFlow, Scikit-learn, Keras). Además, Python es muy flexible y con una sintaxis simple la cual es muy similar al idioma inglés. Entre sus características podemos destacar:

  - Compatible con varias plataformas (Windows, Mac, Linux, Raspberry Pi, etc.)
  - Opera en un sistema de interpretación que permite que el código se ejecute de inmediato
  - Se puede manejar de manera procesal, orientada a objetos o funcional.

- **C++:** es un lenguaje de programación orientado a objetos de propósito general creado en 1980 por Bjarne Stroustrup. Es parecido al lenguaje C, pero mucho más seguro y estructurado que este, ya que está basado en OOP (object-oriented programming). Sus principales usos se dan en máquinas virtuales Java, navegadores, marcos de aplicación y web. Es compatible con Microsoft Windows, Mac OS y Linux.
- **Matlab**: es una plataforma diseñada para científicos e ingenieros con un lenguaje propio basado en matrices permitiendo así una sintaxis computacional matemática.

Esta plataforma puede realizar numerosas tareas basadas en el análisis de datos, el desarrollo de algoritmos y la creación de modelos y aplicaciones. Podemos destacar:

- Facilidad de uso, con numerosas herramientas para su fácil desarrollo (editor, documentación y manuales en línea, navegador de espacio de trabajo, etc.).
- Independencia de la plataforma, debido a su compatibilidad con diferentes sistemas informáticos.

1.9 LIBRERÍAS

- **Tensorflow**: es una biblioteca de código abierto para la computación numérica y aprendizaje automático a gran escala creada en 2015 por el equipo de Google Brain. Esta biblioteca agrupa gran cantidad de modelos y algoritmos de aprendizaje automático y deep learning. Es compatible con Python 3.7-3.10, Ubuntu 16.04 o posterior y Windows 7 o posterior. Es capaz de entrenar y ejecutar redes neuronales profundas para la clasificación de dígitos escritos a mano, reconocimiento de imágenes, incrustación de palabras, procesamiento de lenguaje natural, etc.

  Las aplicaciones de Tensorflow se pueden ejecutar en gran cantidad de dispositivos: una máquina local, un clúster en la nube, dispositivos Android e iOS, CPU o GPU.

  Tensorflow permite a los desarrolladores crear gráficos de flujo de datos, estos son estructuras que describen cómo se mueven los datos a través de un gráfico o una serie de nodos de procesamiento.

  Los nodos y tensores en Tensorflow son objetos de Python y las aplicaciones de Tensorflow son en sí mismas aplicaciones de Python. El trabajo de alto nivel (crear nodos y capas y vincularlos entre sí) utiliza la biblioteca **Keras.**

  Tensorflow, al igual que Keras, es compatible con la red neuronal **ResNet**.
- **Keras**: es una interfaz de programación de aplicaciones (API) para Python relacionada con Tensorflow, que se utiliza para la creación de modelos de aprendizaje automático. Esta API se ejecuta sobre Tensorflow y sirve para simplificar la implementación de redes neuronales complejas con su marco fácil de usar. Básicamente, Keras permite el desarrollo de redes neuronales con Tensorflow para principiantes.
- **NVIDIA CUDA**: creada en 2003 por un equipo de investigadores dirigidos por Ian Buck, esta plataforma informática y modelo de programación desarrollado por Nvidia

permite a los desarrolladores acelerar las aplicaciones de cómputo intensivo aprovechando el rendimiento de las GPU. La combinación de GPU, CUDA y Nvidia domina varias áreas de aplicación, incluido el deep learning.

- **PyTorch**: es una librería para Python que facilita la creación de proyectos de aprendizaje profundo. Es compatible con MacOS y se recomienda su uso junto una GPU NVIDIA para aprovechar la potencia de la compatibilidad con CUDA de Pytorch. Las principales características de PyTorch son:

  - Computación de tensores con soporte de GPU para acelerar los procesos
  - Diferenciación automática para crear y entrenar redes neuronales profundas

- **Numpy**: biblioteca de Python que proporciona objetos de matriz multidimensional, varios objetos derivados y una variedad de rutinas para operaciones rápidas en matrices (lógica, transformadas discretas de Fourier, etc.).
- **Scikit-learn:** esta API proporciona interfaces de Python para muchas de las funciones en el tiempo de ejecución/dispositivo **CUDA.**
- **OpenCV**: es una biblioteca de software de aprendizaje automático y visión artificial de código abierto. Esta biblioteca cuenta con más de 2500 algoritmos, los cuales se pueden usar para detectar y reconocer rostros, identificar objetos, clasificar acciones humanas en videos, etc.

  Esta es compatible con los lenguajes de programación C++, Python, Java y Matlab y con los sistemas operativos Windows, Linux, Android y Mac OS.

1.10 DATASET PARA ENTRENAR LA RCNN

Para el desarrollo de un sistema RCNN necesitamos entrenar el sistema con un conjunto de imágenes las cuales estén segmentadas, etiquetadas y con sus respectivas características puestas de manera manual en archivos **XML** o **JSON** para que el programa sea capaz de interpretar los datos y de esta manera la red neuronal pueda aprender a detectar los objetos mediante deep learning. Para ello, existen herramientas para segmentar las imágenes manualmente o datasets con sus características respectivas anotadas previamente listos para ser utilizados y entrenados en un sistema de visión artificial:

- **Labelme**: es una herramienta de anotación de imágenes gráficas escrita en Pyhton para crear datos de imágenes para la investigación de visión artificial. Esta herramienta crea un archivo XML como el siguiente:

```
<annotation>
  <folder/>
  <filename>2011_000003.jpg</filename>
  <database/>
  <annotation/>
  <image/>
  <size>
    <height>338</height>
    <width>500</width>
    <depth>3</depth>
  </size>
  <segmented/>
  <object>
    <name>bottle</name>
    <pose/>
    <truncated/>
    <difficult/>
    <bndbox>
      <xmin>191</xmin>
      <ymin>107</ymin>
      <xmax>313</xmax>
      <ymax>329</ymax>
    </bndbox>
  </object>
  <object>
    <name>person</name>
    <pose/>
    <truncated/>
    <difficult/>
    <bndbox>
      <xmin>365</xmin>
      <ymin>83</ymin>
      <xmax>500</xmax>
      <ymax>333</ymax>
    </bndbox>
  </object>
</annotation>
```

Figura 1.25. Ejemplo de datos de un documento Labelme XML de nuestro trabajo

Como podemos observar este documento nos da la información del nombre de la imagen, el tamaño, el tipo de objeto, las coordenadas de su caja delimitadora, pero no nos da información sobre las coordenadas del objeto segmentado, esto se debe a que estos datos están pensados para proyectos de Faster RCNN.

El resultado de la caja delimitadora en la imagen es el siguiente:

Figura 1.26. Representación con cuadro delimitador de un archivo XML de Labelme de nuestro trabajo

Sin embargo, también podemos crear un archivo JSON como el siguiente:


```
{
  "version": "4.2.9",
  "flags": {},
  "shapes": [
    {
      "label": "Clear_plastic_bottle",
      "line_color": "null",
      "fill_color": "null",
      "points": [
        [
          427,
          393
        ],
        [
          459,
          389
        ],
        [
          439,
          537
        ],
        [
          427,
          393
        ]
      ],
      "group_id": "null",
      "shape_type": "polygon"
    }
  ],
  "imagePath": "Clear_plastic_bottle.5fba48db-24bc-11ed-be73-309c23179b39.jpg",
  "imageHeight": 3264,
  "imageWidth": 2448
}
```


Figura 1.27. Ejemplo de datos de un documento Labelme JSON de nuestro trabajo

Aquí podemos comprobar que todas las características están inscritas en un diccionario JSON y en vez de las coordenadas de la caja delimitadora, nos da las coordenadas de la segmentación del objeto. Este tipo de documento se desarrolla para proyectos con Mask RCNN.

El resultado de la segmentación del objeto es la siguiente:

Figura 1.28. Segmentación con Labelme de nuestro trabajo

- **VGG Image Annotator**: es un software de anotación manual simple e independiente para imágenes, audio y video. Cabe destacar en su anotación de imágenes que crea un archivo JSON en formato **COCO** para todo el dataset que estemos segmentando. Un ejemplo de JSON de VGG Image Annotator sería el siguiente:

```
{
  "Clear_plastic_bottle.5fba48db-24bc-11ed-be73-309c23179b39.jpg7990272": {
    "filename": "Clear_plastic_bottle.5fba48db-24bc-11ed-be73-309c23179b39.jpg",
    "size": 7990272,
    "regions": [
      {
        "shape_attributes": {
          "name": "polygon",
          "all_points_x": [
            427,
            459,
            487,
            439,
            427
          ],
          "all_points_y": [
            393,
            389,
            541,
            537,
            393
          ]
        },
        "region_attributes": {
          "rubbish": "Clear_plastic_bottle"
        }
      },
      {
        "shape_attributes": {
          "name": "polygon",
          "all_points_x": [
            911,
            933,
            953,
            963,
            993,
            1029,
            1069,
            1093,
            1124,
            1157,
            1161,
```


Figura 1.29. Ejemplo de datos de un documento VGG Image Annotator JSON de nuestro trabajo.

El resultado de la segmentación de imágenes en VGG Image Annotator es el siguiente:

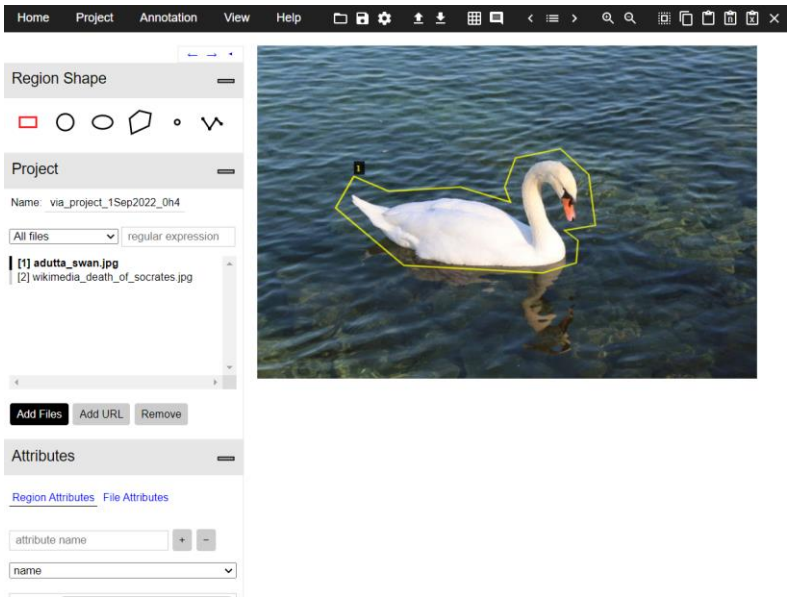

Figura 1.30. Ejemplo de segmentación de VGG Image Annotator(VGG Image Annotator 2019)

- **COCO**: es un conjunto de datos de detección, segmentación, detección de puntos clave y subtítulos de objetos a gran escala. Además, en este nos podemos encontrar unos **pesos preentrenados** que también sirven para acelerar el proceso de entrenamiento de una RCNN con pocos datos de entrenamiento. Este conjunto de datos tiene anotaciones para:

  - Detección de objetos: cuadros delimitadores y máscaras de segmentación por instancia con 80 categorías de objetos.
  - Detección de puntos clave: contiene más de 200.000 imágenes y 250.000 instancias de personas etiquetadas con puntos clave.
  - Segmentación de imágenes de objetos: máscaras de segmentación con 91 categorías.
  - Segmentación de escena completa, con 80 categorías de objetos y un subconjunto de 91 categorías de fondos.

  El conjunto de datos COCO está definido en un tipo de archivo JSON como el que vimos en VGG Image, además ese formato de archivo se denomina COCO.

Entre los datasets para la clasificación de residuos, los más importantes que nos encontramos son:

- **Trash-ICRA19**: es un conjunto de datos etiquetado con cuadro delimitador de basura en el fondo submarino. Cuenta con 5700 imágenes submarinas.
- **TACO**: conjunto de datos de 1500 imágenes de 28 categorías y 60 subcategorías detalladas de desechos en la naturaleza etiquetadas con segmentación de objetos.
- **UAVVaste**: tecnología inteligente de detección de basura de drones. Consta de 772 imágenes y 3716 anotaciones tomadas de imágenes realizadas por drones con segmentación de objetos.
- **Trashnet:** este conjunto de datos abarca 6 clases: vidrio, papel, cartón, plástico, metal y basura. Actualmente consta de 2527 imágenes. Sus anotaciones no contienen coordenadas de una máscara o una caja delimitadora, solo es un dataset formado para la clasificación de objetos
- **TrashCan 1.0**: conjunto de datos de imágenes del fondo marino con 7212 imágenes con 34 categorías etiquetadas con segmentación de imágenes.

Name	No. categories	No. subcategories	No. images	Annotation	Comment	Website
TrashCan 1.0	3	34	7 212	Instance-Segmentation	Underwater images	website
Trash-ICRA19	3	34	5 700	Detection	Underwater images	website
TACO	28	60	1 500	Segmentation	Waste in the wild	website
UAVVaste	1	-	772	Segmentation	Drone dataset	github
Trashnet	6	-	2 527	Classification	Clear background	github

Tabla 1.2. Distintos datasets para la clasificación de residuos (Majchrowska et al. 2022)

2 Objetivos

Creación de un sistema Mask RCNN para la detección de residuos PET.

Análisis de los sistemas RCNN para la detección de residuos deformables, en este caso con botellas PET deformables

Comparación y análisis de distintos sistemas de RCNN.

Investigación sobre la Inteligencia Artificial y su historia.

3 Descripción del sistema

En esta sección vamos a ver el procedimiento que hemos seguido para decidir y realizar nuestro sistema detector de residuos de plástico PET.

3.1 PROGRAMAS, SISTEMAS Y TIPO DE RESIDUO A DETECTAR

Para la elaboración de un sistema de detección del residuo se ha decidido utilizar la tecnología de las RCNN, las cuales pueden detectar el objeto y a su vez localizarlo en la imagen con ciertos métodos. Utilizaremos dos tipos de **RCNN** y las compararemos entre sí para ver cuál es más eficaz, estos tipos serán las **Mask RCNN** y las **Faster RCNN**, las cuales constituyen las 2 RCNN más innovadoras y con mayor calidad a la hora de la detección y localización de objetos.

El residuo que se ha decidido utilizar son las **botellas PET**, estas nos van a permitir analizar la calidad de los sistemas RCNN implementados de detectar **objetos deformables** y a su vez, como ya hemos visto, el PET es el plástico más fácil de reciclar y por tanto en el que la selección adquiere mayor importancia

Para la realización de este proyecto se decidió la realización con el lenguaje **Python**, ya que, como bien hemos visto en el apartado librerías, este lenguaje es compatible con Tensorflow, Keras, Numpy, Pytorch, NVIDIA CUDA, etc. Esto nos permite el uso de un amplio abanico de herramientas para la inteligencia artificial, deep learning y visión artificial. Además, partiendo de la base del poco conocimiento sobre programación,

también es de ayuda que el lenguaje sea didáctico y para esto es perfecto Python ya que está basado en la sintaxis inglesa lo que conlleva una sencilla compresión de su implementación.

En comparación con Matlab y C++, Python es más popular y usado que cualquiera de los otros dos, lo que implica que a la hora de buscar documentos debido a dudas de implementación o funcionamiento nos encontraremos con mayor cantidad de información de Python que de los otros dos lenguajes.

Para adquirir el conocimiento necesario de programación básica con Python se ha realizado el curso de la plataforma coursera denominado *Python for Data Science and AI* (Santarcangelo, sin fecha).

La implementación de Python en nuestro sistema se ha llevado a cabo mediante el desarrollo interactivo **Jupyter Notebook** el cual es una herramienta de la distribución **Anaconda**. De esta distribución utilizaremos otras herramientas como **Anaconda Prompt** o **Anaconda Navigator** a medida que realizamos el sistema como veremos posteriormente en los siguientes apartados.

3.2 RECOPILACIÓN DE IMÁGENES

Para la recopilación de imágenes se barajaron distintas posibilidades entre las vistas en el apartado Datasets para entrenar la RCNN. El conjunto de imágenes debía permitir el entrenamiento con una Mask RCNN, por tanto, debía implementar **segmentación de objetos**, debía ser un **conjunto grande de imágenes** y los datos debían tener **distintos fondos y otros objetos** todo esto para realizar un entrenamiento que nos diese una buena precisión de detección.

Finalmente se decidió escoger el conjunto de datos **TACO** debido a que contiene segmentación de objetos, las imágenes tomadas son de residuos en la naturaleza (con diferentes fondos y con varios objetos en las imágenes) y contiene un total de 1500 imágenes.

Una vez descargado el conjunto de datos TACO podemos ver detalladamente sus especificaciones en un cuaderno Jupter:

- El dataset TACO está dividido en una serie de categorías de conjuntos generales que a su vez tienen unas subcategorías más específicas

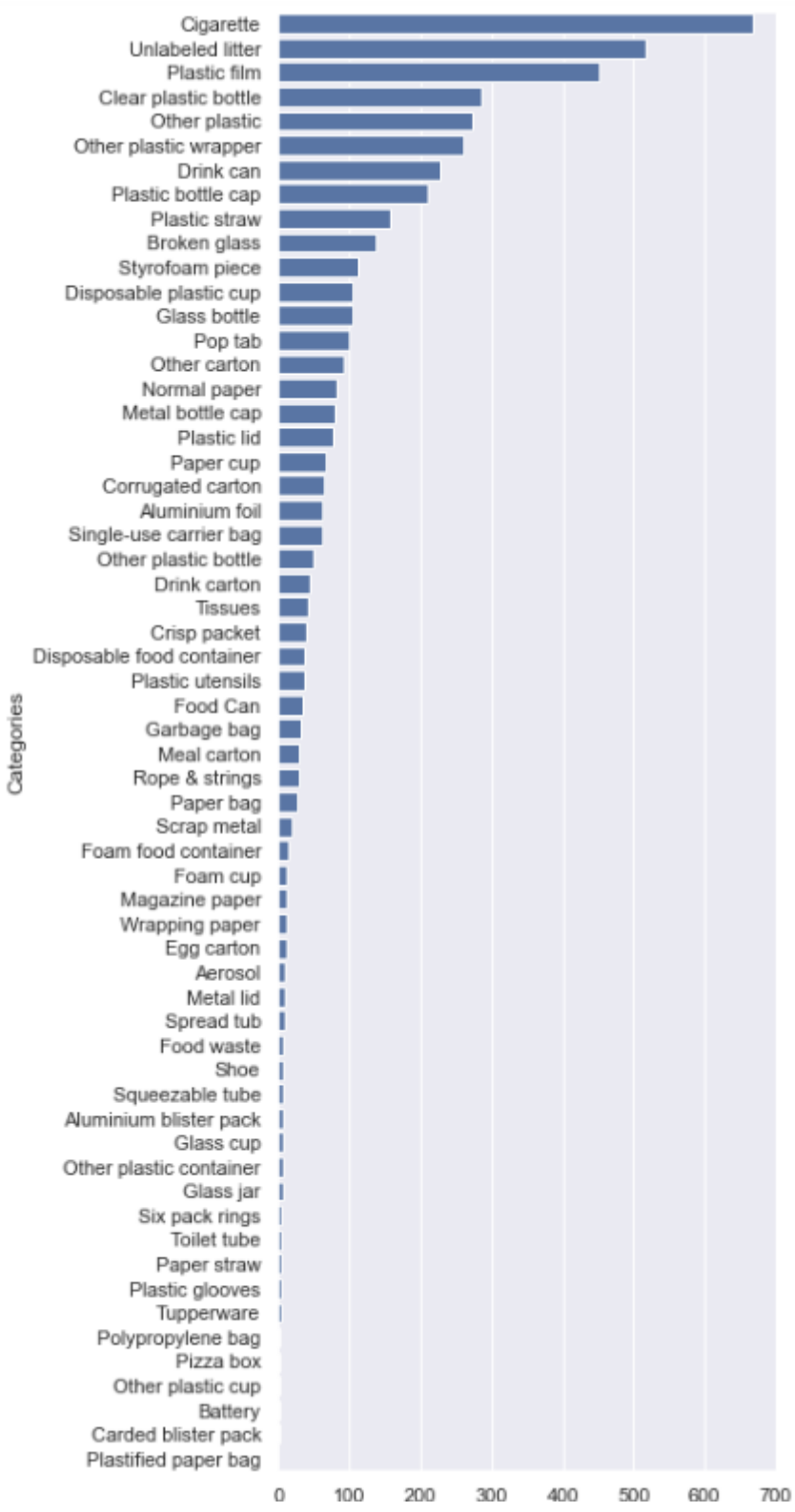

Figura 3.1. Subcategorías del conjunto TACO (TACO trash annotations 2022)

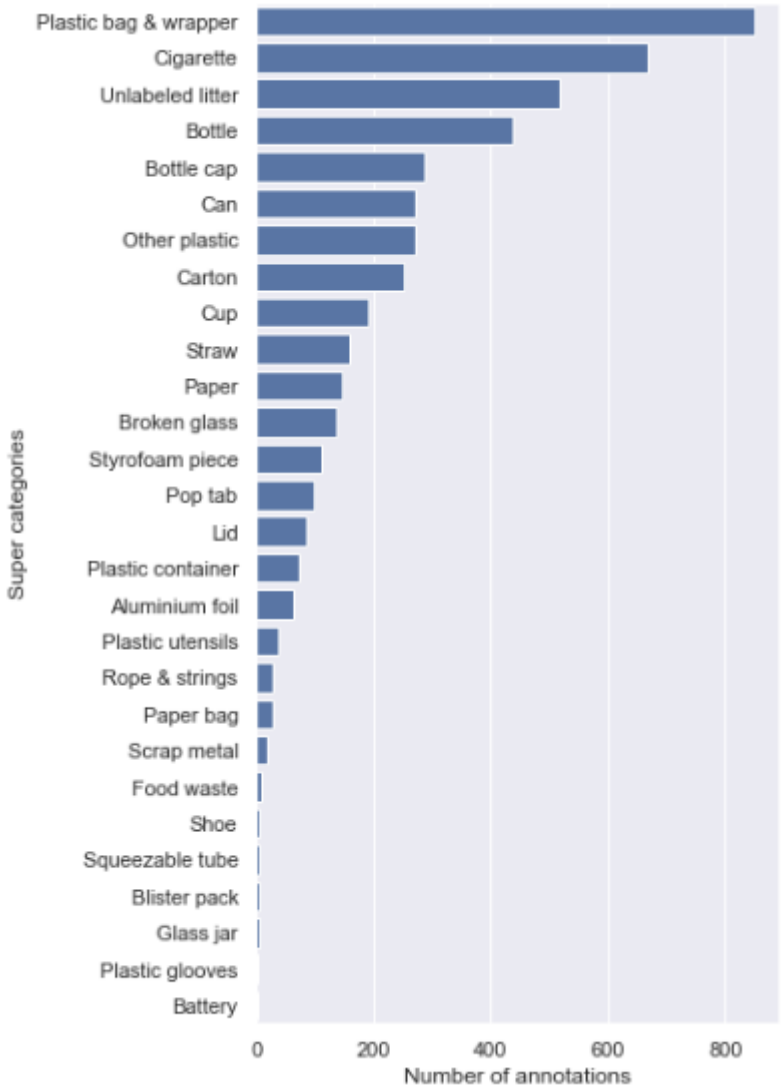

Figura 3.2. Categorías del conjunto TACO (TACO trash annotations 2022)

Según estas 2 gráficas podemos ver, en lo que respecta al PET, que el conjunto de anotaciones para la categoría botellas es de 450 más o menos y para la subcategoría *Clear Plastic Bottle* es de casi 300. Este conjunto de anotaciones es interesante para iniciar los entrenamientos de la red.

- La siguiente gráfica expresa la cantidad de imágenes con los diferentes fondos existentes en TACO

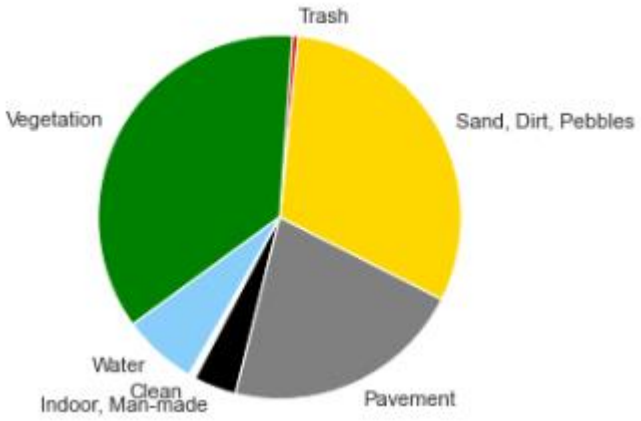

Figura 3.3. Representación de la cantidad de imágenes con un fondo específico en TACO (TACO trash annotations 2022)

Podemos ver que la mayoría de las imágenes están tomadas en un entorno de vegetación (fondo verde) y de arena, suciedad o piedras (fondo amarillo), seguidas de las tomadas en pavimento (fondo gris) y por último las tomadas en agua, basura y otros fondos.

Es de gran importancia que las imágenes estén tomadas con distintos fondos ya que, si todas las imágenes para entrenar fueran similares, el sistema sería poco capaz de adaptarse a la diversidad, por ejemplo, si se toman todas las imágenes en un fondo verde, el sistema estará acostumbrado a detectar objetos en fondos verdes y su precisión será muy baja cuando se cambie de color de fondo.

- El siguiente diagrama permite visualizar a que categoría pertenecen las subcategorías

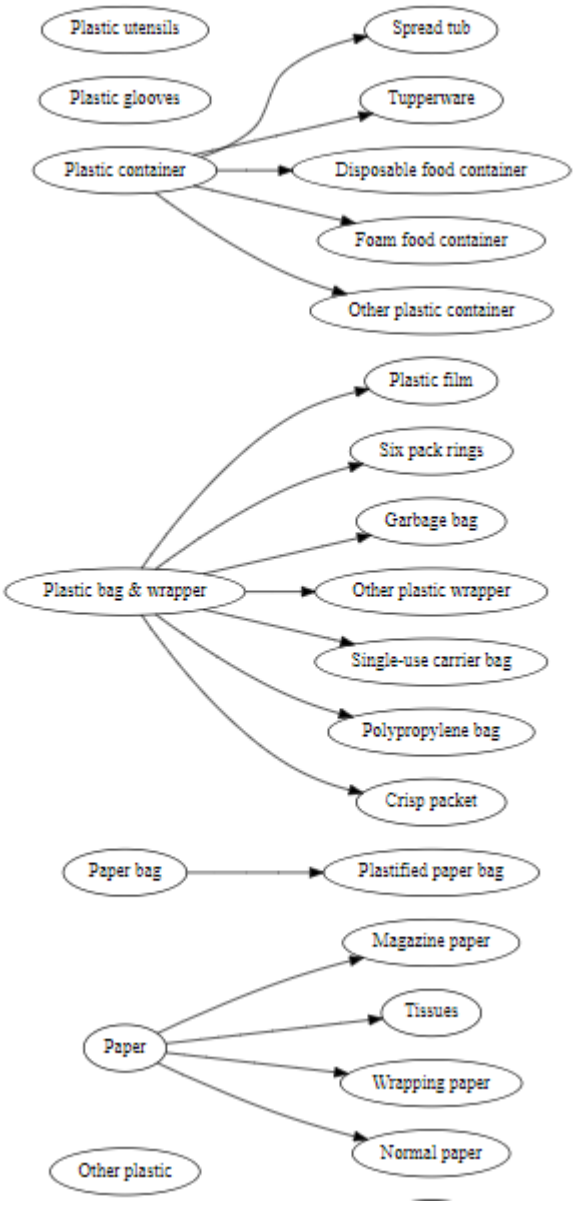

Figura 3.4. Diagrama de categorías y subcategorías de los plásticos del dataset TACO (TACO trash annotations 2022)

Solo se muestra el diagrama para los plásticos debido a que es el que nos conviene para este proyecto.

Como podemos comprobar existe una gran cantidad de categorías que nos podrían servir para implementar un sistema de clasificación y detección de varios residuos a

mayores de los plásticos PET, pero más adelante veremos la serie de problemas que estos datos han conllevado al desarrollo de nuestro sistema.

3.3 IMPLEMENTACIÓN DE FASTER RCNN PREENTRENADO

Para la simplificación de la explicación de la implementación de un **sistema Faster RCNN** se han creado una serie de **diagramas de flujo** a través de la herramienta diagrams.net, disponible de manera online.

La implementación de Faster RCNN se realiza en 2 apartados distintos que veremos a continuación.

3.3.1 RECOPILACIÓN DE IMÁGENES TACO

Vamos a ver cómo funciona este apartado a través del siguiente diagrama de flujo:

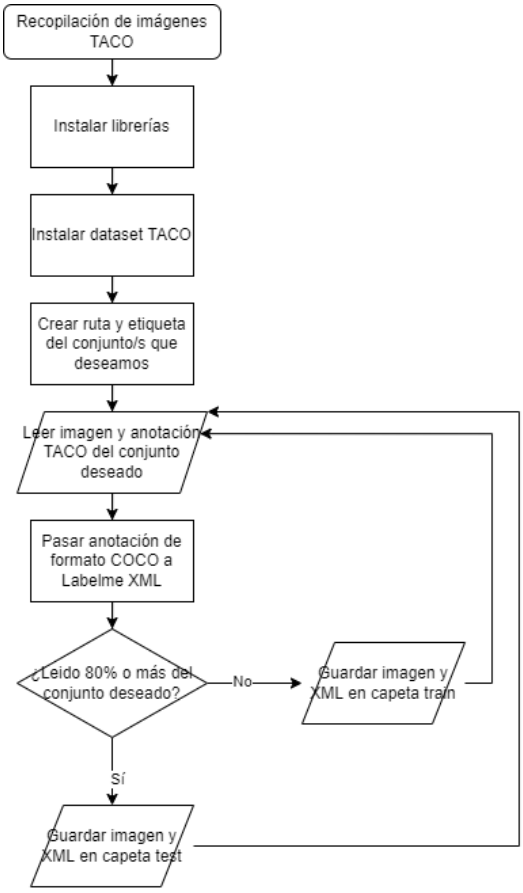

Figura 3.5. Diagrama de flujo de recopilación de datos para Faster RCNN

Como podemos ver en este diagrama, el apartado se divide en una serie de pasos:

- **Librerías**: las librerías utilizadas para este apartado son las siguientes:

```
import cv2
import uuid
import os
import numpy as np
import pandas as pd
import json
import matplotlib.pyplot as plt
from pycocotools.coco import COCO
import xml.etree.ElementTree as ET
from xml.etree.ElementTree import ElementTree, tostring
```

Figura 3.6. Librerías utilizadas para la recopilación de imágenes

  - **Cv2 o OpenCV y Numpy** son dos librerías ya definidas en el apartado de Librerías para la Visión Artificial.
  - **Uuid** (código identificador estándar empleado en el proceso de construcción de software): biblioteca empleada para crear identificadores únicos para el reconocimiento único de objetos en el sistema.
  - **Os** (interfaces misceláneas del sistema operativo): proporciona funcionalidades dependientes del sistema operativo al programa.
  - **Pandas**: conjunto de herramientas de análisis de datos (nos servirá para ver las gráficas de conjunto TACO).
  - **Json**: librería que sirve para abrir editar y crear archivos JSON
  - **Matplotlib**: biblioteca multiplataforma para la visualización y edición de datos, gráficos e imágenes.
  - **Pycocotools**: API de Python que ayuda a cargar, visualizar y analizar las anotaciones en el formato COCO.
  - **Xml.etree.ElementTree**: biblioteca empleada para crear, editar y analizar archivos XML.

- **Instalar TACO dataset**: esto se puede realizar abriendo y guardando en una lista el archivo en formato JSON con todas las anotaciones de todo el dataset TACO mediante la librería json (*json.loads()*).
- **Crear rutas y etiquetas para los outputs**: se crea la etiqueta específica para las botellas PET que como ya se vio anteriormente, el nombre que estas tenían en el

dataset TACO es *clear plastic bottle*. Además, también se crean las rutas para guardar los datos de salida.

```
conjuntos = ['Clear plastic bottle']
RUTA_IMAGENES_TRAIN = os.path.join('Tensorflow','espacio_de_trabajo', 'imagenes'
RUTA_IMAGENES_TEST = os.path.join('Tensorflow','espacio_de_trabajo', 'imagenes',
```

Figura 3.7. Rutas y etiqueta de la recopilación de imágenes

- **Leer imágenes y anotaciones del conjunto TACO**: para ello hacemos uso de las librerías cv2 para abrir las imágenes (con el comando *cv2.imshow()*) y de pycocotools para abrir las anotaciones en formato COCO (con *coco.getImgIds()*).
- **Pasar de COCO a Labelme**: las imágenes y anotaciones se leerán de objeto en objeto dentro de un bucle, de esta manera podemos transformar los datos de las anotaciones de objeto en objeto. Los datos de entrada de las redes neuronales están divididos para entre los datos de **entrenamiento** (suele ser el **80%** del total) y los datos para **validar** la precisión de la red neuronal (suele ser el **20%**). Debido a esto, antes de llegar al 80% de las imágenes, estas junto a sus respectivas anotaciones se guardarán en la carpeta designada en *RUTA_IMAGENES_TRAIN*. Una vez llegado al 80% el resto de los datos se guardarán en la carpeta designada en *RUTA_IMAGENES_TEST*.

Finalmente, con esta recopilación, conseguimos obtener una cantidad de 225 imágenes, las cuales se repartirán entre la carpeta test y la carpeta train.

3.3.2 **ENTRENAMIENTO DE FASTER RCNN PREENTRENADA**

El siguiente diagrama de flujo representa como se ha implementado el entrenamiento del sistema Faster RCNN:

Figura 3.8. Flujograma entrenamiento de Faster RCNN preentrenada.

Como podemos observar, la realización de este apartado se divide en una serie de etapas:

- **Librerías**: En este apartado, las versiones de las librerías son importantes para una correcta compatibilidad y que funcione todo bien en el sistema. Por esto mismo, se ha creado un ambiente virtual desde cero en el cual se han instalado las versiones necesarias. Este ambiente virtual tiene la versión de **Python 3.9.12.**

  Las versiones de las librerías que estamos utilizando en el ambiente virtual se pueden visualizar en la siguiente imagen:

```
numpy==1.23.2
scipy==1.9.1
Pillow==9.2.0
cython==0.29.32
matplotlib==3.5.3
scikit-learn==1.1.2
tensorflow==1.15.0
keras==2.2.5
object-detection==0.0.3
opencv-python==4.6.0.66
h5py==3.7.0
IPython[all]
```

Figura 3.9. Librerías y sus versiones para la implementación de Faster RCNN

Además, también se han importado al cuaderno de Jupyter una serie de librerias:

```
import os
import tensorflow
import object_detection
import cv2
import numpy as np
import matplotlib
import wget
```

Figura 3.10. Librerías importadas a Jupyter

Se puede observar que en esta parte de la implementación de Faster RCNN se utilizan librerías ya nombradas en la recopilación de imágenes o en el apartado de librerías para la visión artificial (cv2, numpy, tensorflow, matplotlib, etc.) y nuevas librerías que no habíamos nombrado antes:

- **Scipy**: librería de código abierto que se utiliza para resolver problemas matemáticos, técnicos, científicos o de ingeniería permitiendo la manipulación de datos y su visualización.
- **Pillow**: esta librería contiene todas las funciones básicas del procesamiento de imágenes de Python. Permite la extracción de datos estadísticos de la imagen y la mejora automática de contraste.

- **Cython**: API de Python que funciona como un lenguaje de programación capaz de realizar procesos de Python con un rendimiento similar al de C.
- **H5py**: paquete que proporciona una interfaz de alto y bajo rendimiento para la biblioteca HDF5 [10]de Python.
- **Wget**: librería de python que sirve para descargar archivos de la red URL en segundo plano.
- **Object_detection**: API de Tensorflow que facilita la construcción, el entrenamiento y la implementación de modelos de detección de objetos.

- Rutas del sistema: los directorios y carpetas del sistema estarán distribuidos de la siguiente manera:

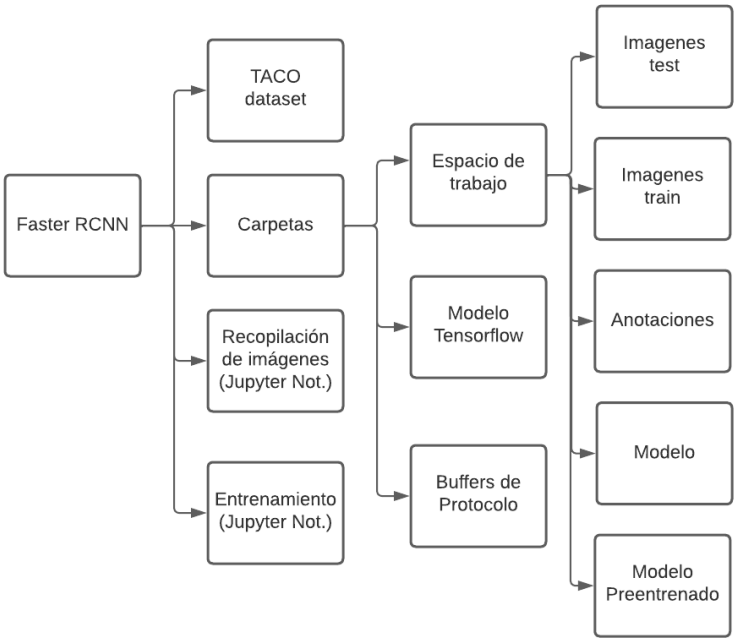

Figura 3.11. Diagrama en árbol de la distribución de ficheros

[10] HDF5 es una biblioteca de Software de datos de alto rendimiento HDF (formato de datos jerárquicos)

Como se puede visualizar, todo el sistema Faster RCNN está dentro de la carpeta *Faster RCNN* donde nos encontramos el conjunto de datos TACO (*TACO dataset*), sus imágenes y XML obtenidos para el entrenamiento y validación (*Imágenes train e Imágenes test*) y los cuadernos Jupyter donde se recopilan las imágenes (*Recopilación de imágenes (Jupyter Not.)*) y donde se implementa el entrenamiento (*Entrenamiento (Jupyter Not.)*)

En este apartado hemos definido algunos directorios o carpetas, pero la gran mayoría se irán nombrando a medida que explicamos la implementación del entrenamiento.

- **Descargar modelo preentrenado, instalar la API de modelos de Tensorflow e instalar los buffers de protocolo:** para la implementación de Faster RCNN, debemos implementar una red neuronal para el reconocimiento de imágenes mediante cajas delimitadoras. Para ello existe un conjunto de modelos de detección entrenados previamente en el conjunto de datos de COCO denominados modelos de detección de **TensorFlow 2 Detection Model Zoo**, estos están dentro del directorio de **github** (Birodkar 2021). Los modelos que nos interesan para nuestro sistema son los de Faster RCNN y SSD:

Model name	Speed (ms)	COCO mAP	Outputs
SSD MobileNet v2 320x320	19	20.2	Boxes
SSD MobileNet V1 FPN 640x640	48	29.1	Boxes
SSD MobileNet V2 FPNLite 320x320	22	22.2	Boxes
SSD MobileNet V2 FPNLite 640x640	39	28.2	Boxes
SSD ResNet50 V1 FPN 640x640 (RetinaNet50)	46	34.3	Boxes
SSD ResNet50 V1 FPN 1024x1024 (RetinaNet50)	87	38.3	Boxes
SSD ResNet101 V1 FPN 640x640 (RetinaNet101)	57	35.6	Boxes
SSD ResNet101 V1 FPN 1024x1024 (RetinaNet101)	104	39.5	Boxes
SSD ResNet152 V1 FPN 640x640 (RetinaNet152)	80	35.4	Boxes
SSD ResNet152 V1 FPN 1024x1024 (RetinaNet152)	111	39.6	Boxes
Faster R-CNN ResNet50 V1 640x640	53	29.3	Boxes
Faster R-CNN ResNet50 V1 1024x1024	65	31.0	Boxes
Faster R-CNN ResNet50 V1 800x1333	65	31.6	Boxes
Faster R-CNN ResNet101 V1 640x640	55	31.8	Boxes
Faster R-CNN ResNet101 V1 1024x1024	72	37.1	Boxes
Faster R-CNN ResNet101 V1 800x1333	77	36.6	Boxes
Faster R-CNN ResNet152 V1 640x640	64	32.4	Boxes
Faster R-CNN ResNet152 V1 1024x1024	85	37.6	Boxes
Faster R-CNN ResNet152 V1 800x1333	101	37.4	Boxes
Faster R-CNN Inception ResNet V2 640x640	206	37.7	Boxes
Faster R-CNN Inception ResNet V2 1024x1024	236	38.7	Boxes

Figura 3.12. Modelos SSD y Faster RCNN preentrenados de Tensorflow (Birodkar 2021)

Como podemos comprobar, todos estos modelos tienen implementada una ResNet o Mobilnet como **backbone**, pero únicamente las SSD tienen implementado una red de propuesta de regiones (**RPN**). Al lado de la columna del nombre, están las de la velocidad en la que realiza la detección de objetos en milisegundos y la de la precisión mediante la precisión promedia media (**mAP**). Se desea un modelo que tenga una buena precisión y que no detecte los objetos de manera lenta, por eso mismo, se ha escogido como modelo a utilizar. Si analizamos la tabla, las SSD son óptimas que las Faster RCNN. Por todo esto se ha escogido para nuestro sistema **SSD MobileNet V2 FPNLite 320x320.** La instalación de este modelo se hará en la carpeta *Modelo preentrenado*.

Una vez realizada la instalación del modelo preentrenado el siguiente paso es la descargar un repositorio de Tensorflow denominado **Model Garden for TensorFlow** (Google's protocol buffers 2022). Este repositorio contiene una serie de

implementaciones para TensorFlow (incluida la API **object_detection**) con las que podremos visualizar el entrenamiento de nuestro modelo y mejorar su rendimiento. Este repositorio se descargará en la carpeta *Modelo TensorFlow*.

El último paso de este apartado es la instalación de los **buffers de protocolo**; estos son un mecanismo para serializar datos estructurados, en nuestro caso los necesitaremos para serializar los XML de las imágenes para que queden estructurados en un archivo más pequeño y por tanto más rápido de ejecutar por el programa (esto lo veremos en el apartado de XML a TFRecord). Estos buffers se descargarán a través de github (Kim 2022) mediante el comando *wget* y su ubicación estará dentro de *Carpeta*.

- **Mapa de etiquetas**: este mapa de etiquetas es similar al realizado en el apartado de recopilación de imágenes. En este, las etiquetas de los objetos tendrán un nombre y un número identidad mayor que uno, ya que el 0 está reservado para el fondo de la imagen. Como nuestro único objeto son la botella PET, creamos el respectivo mapa de etiquetas y lo guardamos en *Anotaciones*.

```
item {
    name:'Clear_plastic_bottle'
    id:1
}
```

Figura 3.13. Mapa de etiquetas creado

- **Modificar las imágenes de XML a formato TF Record**: para que el entrenamiento se produzca de manera rápida conviene el cambio de los archivos XML de las imágenes a TF Records. TF Record es un formato de documento binario que ocupa muy poco espacio y por tanto aumenta el rendimiento de importación de archivos y disminuye el tiempo de entrenamiento de nuestro sistema. Los nuevos archivos quedarán guardados en *Anotaciones*.
- **Ajustar los parámetros del entrenamiento**: estos parámetros vienen por defecto en un archivo con el nombre "*pipeline.config*" dentro de la carpeta del modelo preentrenado *SSD MobileNet V2 FPNLite 320x320*. Las variables que dependen del rendimiento del sistema son las siguientes:

  - "***num_classes***": número de los objetos que van a ser detectados (en nuestro caso 1).
  - "***fixed_shape_resize***": modifica las dimensiones de las imágenes al tamaño deseado sin deformarlas (cubriendo los espacios con marcos). En nuestro caso, ya que

estamos utilizando "*SSD MobileNet V2 FPNLite 320x320*" las redimensionará a *320x320* píxeles.

- "***feature_extractor***": indica el sistema extractor de características. En nuestro caso "*SSD MobileNet V2 FPNLite 320x320*".
- "***anchor_generator***": indica una serie de características de las cajas delimitadoras:
  - "***scale***": máxima y mínima escala de las cajas delimitadoras.
  - "***aspect_ratios***": relación alto y ancho de las cajas delimitadoras.
- "***score_threshold***": umbral de puntuación para la supresión no máxima de la RPN.
- "***iou_threshold***": umbral de la intersección sobre la unión (IOU) para realizar la supresión no máxima (NMS) en las casillas predichas por la Red de propuesta de región (RPN).
- "***batch_size***": es el número de imágenes o inputs que se analizan por cada paso de la red. Debido a que este entrenamiento se ha realizado con la CPU del ordenador, el batch size se ha acotado a 4 imágenes por lote. Si el entrenamiento se realizase con una GPU potente, el batch size podría aumentar a varios inputs por paso.
- "***num_steps***": es el número de pasos que se realizan por cada época de entrenamiento, cuanto mayor sea el número de pasos, el entrenamiento se realizará antes. Por ejemplo, si se ha ajustado el número de pasos a 50.000. Sabiendo que el sistema analiza 4 imágenes por paso, por cada época se analizarán un total de 200.000 imágenes.
- "***data_augmentation_options***": mediante esta opción se puede aumentar drásticamente el conjunto de imágenes. Esta funciona cogiendo el conjunto de datos originales, modificándolo (volteando, girando ejes, difuminando la imagen…) y creando nuevas imágenes con esas modificaciones que servirán de igual manera para el entrenamiento de la red neuronal.

- **Entrenamiento del modelo**: una vez realizados los pasos anteriores, se puede llevar a cabo el entrenamiento de Faster RCNN.

3.3.3 ENTRENAMIENTOS DE FASTER RCNN

3.3.3.1 ENTRENAMIENTO CON 225 IMÁGENES Y 2000 ÉPOCAS

Para el análisis del entrenamiento se utiliza la herramienta **Tensorboard** de Tensorflow; esta nos proporciona la visualización de gráficos y otras herramientas para el estudio del entrenamiento de una red neuronal. Mediante Tensorboard podemos visualizar las siguientes gráficas:

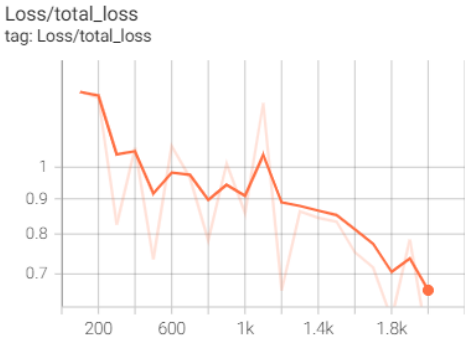

Figura 3.14. Gráfica de los errores totales por época

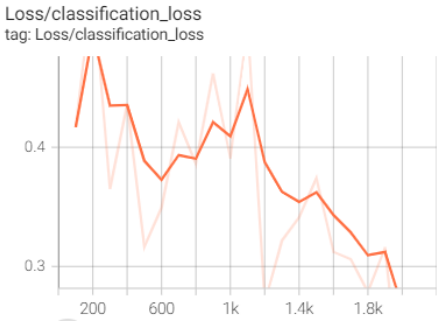

Figura 3.15. Gráfica de los errores de clasificación de imágenes

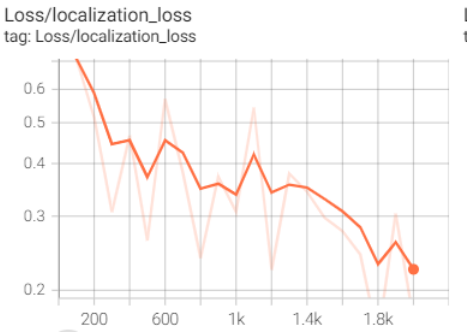

Figura 3.16. Gráfica de los errores de la caja delimitadora

En la última época, el **error total es de 0,6331**, el **error de clasificación es de 0,2412** y el **error de la caja delimitadora 0,308.**

Como podemos ver, existen una gran cantidad de variaciones en las gráficas y por lo tanto no se puede tomar una decisión clara del rendimiento del sistema.

3.3.3.2 ENTRENAMIENTO CON 225 IMÁGENES Y 22000 ÉPOCAS

Tras el aumento de épocas, podemos comprobar que los errores se han atenuado. En las gráficas observamos 2 colores, ya que el color naranja representa el entrenamiento de 2000 épocas y el azul el de 22000 épocas. En este entrenamiento se han implementado los pesos preentrenados del entrenamiento de 2000 épocas para aumentar la agilidad del proceso.

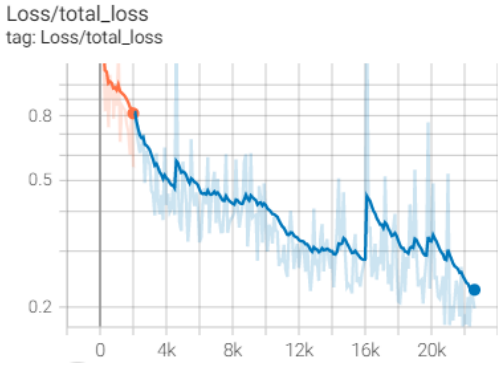

Figura 3.17. Gráfica de los errores totales por época

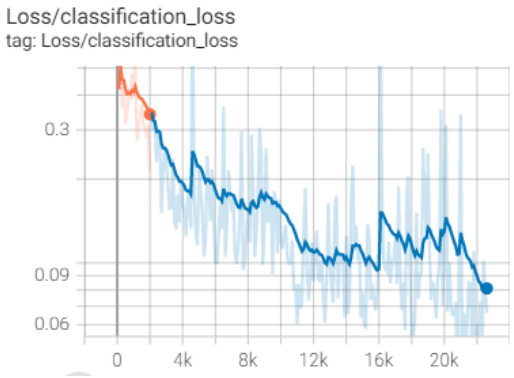

Figura 3.18. Gráfica de los errores de clasificación de imágenes

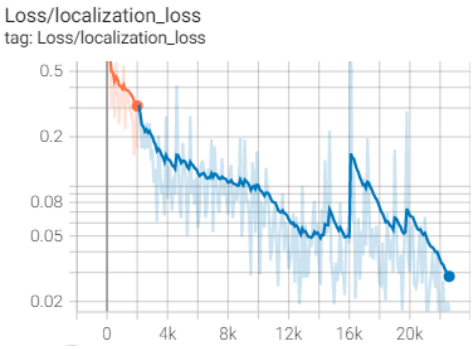

Figura 3.19. Gráfica de los errores de la caja delimitadora

En este entrenamiento, en la última época el error total es de 0,2267, el error de clasificación es de 0,078 y el error de caja delimitadora es de 0,029.

Con el aumento de épocas el **AP** fue del 23,4%, la **recuperación** de 24,1% y un **mAP** de **15,2%**, lo cual es muy poco.

Estos malos resultado se deben a que las botellas de PET son **deformables** y por lo tanto es muy difícil entrenar un modelo Faster RCNN datos de entrada deformables.

Esto se puede comprobar al analizar la detección del sistema:

Figura 3.20. Detección correcta de la botella PET (derecha) por el modelo Fast RCNN

Figura 3.21. Detección errónea de la botella PET (derecha) por el modelo Fast RCNN

El sistema es más preciso cuando se trata de botellas enteras y muy poco preciso a la hora de analizar botellas deformadas. Esto se debe a que en el conjunto de datos existe una gran cantidad de botellas sin deformar.

También cabe destacar que obtenemos unos errores tan bajos debido a que el sistema se ha memorizado los datos de entrada, pero la detección fuera de esos datos es extremadamente baja.

Necesitamos un modelo en el que la detección de los objetos sea más precisa. Para ello, en el próximo sistema haremos uso de la segmentación de objetos de la Mask RCNN, en la cual además de analizar ciertas características similares en las regiones de los objetos (como en las cajas delimitadoras), también se delimita el objeto con un polígono para obtener mayor precisión a la hora de localizar el objeto.

3.4 IMPLEMENTACIÓN DE MASK RCNN PREENTRENADA (ERRÓNEO)

En este entrenamiento se ha intentado implementar una Mask RCNN preentrenada basándonos en el entrenamiento de Faster RCNN preentrenado.

En este los diagramas de flujo son muy similares a los de Faster RCNN, únicamente cambian una serie de etapas:

- **Recopilación de imágenes TACO**:

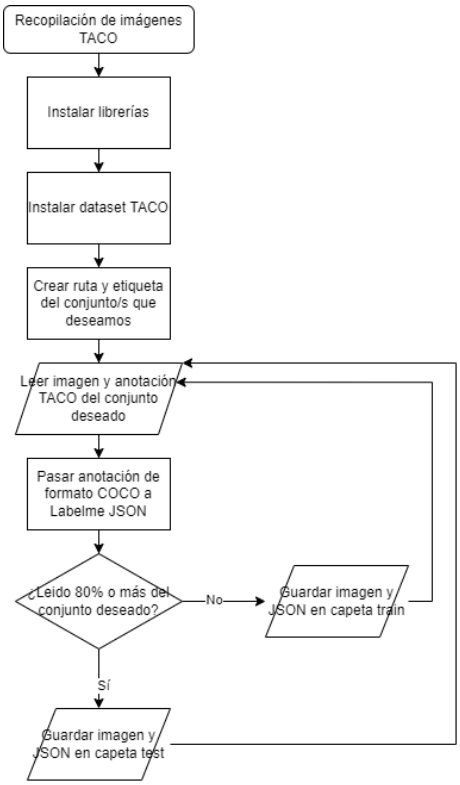

Figura 3.22. Diagrama de flujo de la recopilación de imágenes en Mask RCNN preentrenada

Podemos ver en la anterior imagen que el único cambio en la recopilación de imágenes en comparación con Faster RCNN es que las anotaciones en vez de modificarse a XML ahora pasan al formato JSON de Labelme. Esto se debe a que el XML de Labelme no contenía los valores de las coordenadas de segmentación de objetos y el JSON si los contiene.

- **Entrenamiento de Mask RCNN preentrenada**:

Figura 3.23.Diagrama de flujo del entrenamiento de Mask RCNN preentrenada

En el entrenamiento, podemos ver que los cambios producidos es el de los archivos JSON en vez de XML y además que ahora el modelo preentrenado será una Mask RCNN. La única Mask RCNN preentrenada que se puede obtener de TensorFlow 2

Detection Model Zoo es “Mask R-CNN Inception ResNet V2 1024x1024” con las características de la imagen:

Model name	Speed (ms)	COCO mAP	Outputs
Mask R-CNN Inception ResNet V2 1024x1024	301	39.0/34.6	Boxes/Masks

Figura 3.24. Mask RCNN de TensorFlow 2 Detection Model Zoo (Birodkar 2021)

Lamentablemente, una vez realizado todo el sistema, en el momento del entrenamiento, este daba un fallo:

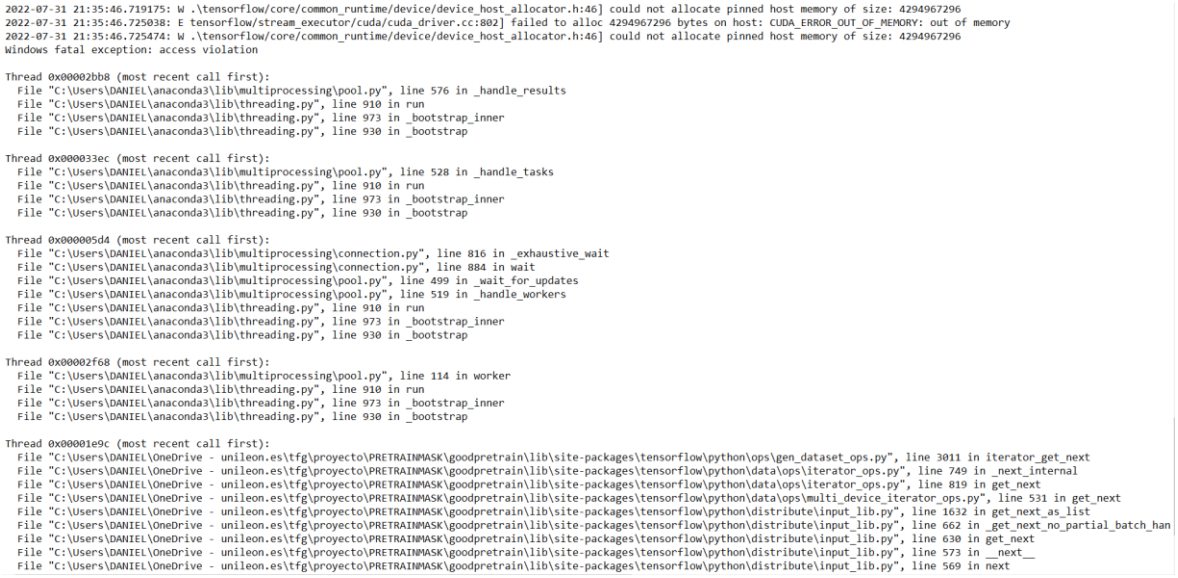

Figura 3.25. Fallo del sistema durante el entrenamiento de Mask RCNN preentrenada

Como se puede observar, el entrenamiento de este sistema daba como resultado “**Windows fatal exception: access violation**”, lo que implica una incompatibilidad de las librerías utilizadas y el sistema, lo que resultó extraño debido a que las librerías utilizadas fueron las mismas que en la implementación de Faster RCNN preentrenado. Este problema no fue posible de resolver.

3.5 IMPLEMENTACIÓN DE MASK RCNN PERSONALIZADA

Debido al intento fallido de realizar el entrenamiento de una Mask RCNN preentrenada, se decide la implementación de una Mask RCNN personalizada, la cual podemos cambiar los parámetros como se desee.

La implementación de este sistema será distinta a lo visto anteriormente. Algunos parámetros de los sistemas anteriores se modificarán, sustituirán o eliminarán.

La Mask RCNN personalizada permite un entrenamiento a medida, el cual podremos realizar tan solo con nuestro conjunto de datos, con pesos preentrenados de coco o como se pretenda.

3.5.1 **REALIZACIÓN DEL ENTRENAMIENTO**

La puesta en marcha de este entrenamiento se realiza de la siguiente manera:

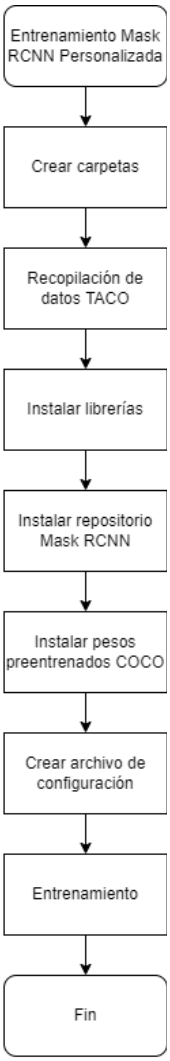

Figura 3.26. Realización del sistema Mask RCNN Personal

- **Directorios**: para la realización de este entrenamiento se han creado una serie de carpetas para guardar todos los ficheros. La ubicación de estos es la siguiente:

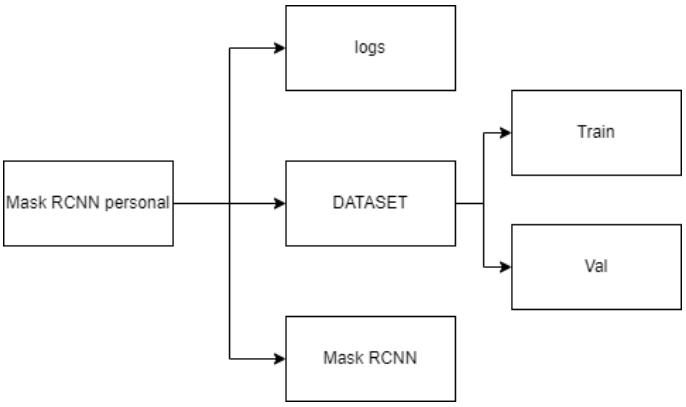

Figura 3.27. Diagrama en árbol indicando la ubicación de los ficheros del sistema

Todo está dentro de la carpeta *Mask RCNN personal*, dentro de la carpeta *DATASET* está la ubicación de los archivos (*imágenes y anotaciones*) para entrenar y evaluar el sistema, en *logs* se guardará el resultado del entrenamiento y en *Mask RCNN* se guardarán el resto de los archivos necesarios para la implementación del sistema.

- **Recopilación con TACO:** la recopilación de datos se hace de igual manera que en los anteriores sistemas, solo que ahora las anotaciones deben estar en formato VGG Image Annotator JSON, este formato es muy parecido al formato de COCO JSON con unas ligeras modificaciones.

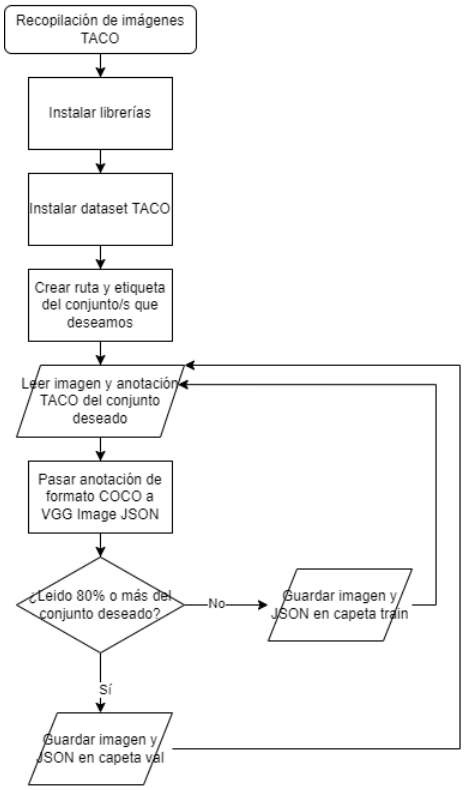

Figura 3.28. Diagrama de flujo indicando la recopilación del conjunto de imágenes TACO en formato VGG Image

- **Librerías**: al ser una implementación distinta a las anteriores, las versiones de las librerías serán distintas:

```
numpy==1.21.6
scipy==1.7.3
Pillow==9.2.0
cython==0.29.32
matplotlib==3.5.3
scikit-image==0.16.2
tensorflow==1.15.0
tensorflow-gpu==1.15.0
keras==2.2.5
opencv-python==3.5.3
h5py==2.10.0
imgaug
IPython[all]
```

Figura 3.29. Librerías del sistema Mask RCNN personal

Como podemos comprobar, algunas librerías como *Object_detection* no aparecen en este nuevo sistema. Esto se debe a que estas eran librerías para la implementación de RCNN preentrenadas de Tensorflow. Además, hay una nueva librería *tensorflow-gpu*, esta librería funciona como TenorFlow, pero en vez de realizar las funciones de rendimiento en la CPU, las realiza en la **GPU** y es que en este entrenamiento queremos que se realice desde la GPU del ordenador, en este caso es la Nvidia Geforce GTX 1050. Por ello, además de instalar *tensorflow-gpu* ha sido necesaria la instalación de versiones compatibles de otras herramientas, estas son **NVIDIA CUDA**, de la cual ya se ha hablado anteriormente y **cuDNN (CUDA Deep Neural Network);** esta es una biblioteca acelerada por GPU para redes neuronales profundas, esta se implementa en NVIDIA CUDA para que pueda ser compatible con nuestro sistema. Las versiones necesarias con las de las librerías que tenemos son **NVIDIA CUDA 10.0** y **cuDNN 7.6.0**.

Para este modelo la **versión de Pyhton** es la **3.7.13**, debido a que **TensorFlow 1.15.0** no es compatible con versiones posteriores de Python.

- **Instalar repositorio Mask RCNN y pesos preentrenados**: el sistema Mask RCNN personal necesita una serie de archivos que nos permiten realizar ciertas funciones. Estos archivos los podemos encontrar dentro del directorio de Github "*Mask R-CNN for object Detection and Segmentation*". Dentro de este directorio encontraremos una serie de ficheros de entre los cuales los que son de interés son:

- "***config.py***": en este archivo están todos los parámetros de rendimiento del entrenamiento:

```
Configurations:
BACKBONE                       resnet101
BACKBONE_STRIDES               [4, 8, 16, 32, 64]
BATCH_SIZE                     1
BBOX_STD_DEV                   [0.1 0.1 0.2 0.2]
COMPUTE_BACKBONE_SHAPE         None
DETECTION_MAX_INSTANCES        100
DETECTION_MIN_CONFIDENCE       0.8
DETECTION_NMS_THRESHOLD        0.3
FPN_CLASSIF_FC_LAYERS_SIZE     1024
GPU_COUNT                      1
GRADIENT_CLIP_NORM             5.0
IMAGES_PER_GPU                 1
IMAGE_CHANNEL_COUNT            3
IMAGE_MAX_DIM                  1024
IMAGE_META_SIZE                14
IMAGE_MIN_DIM                  800
IMAGE_MIN_SCALE                0
IMAGE_RESIZE_MODE              square
IMAGE_SHAPE                    [1024 1024    3]
LEARNING_MOMENTUM              0.9
LEARNING_RATE                  0.001
LOSS_WEIGHTS                   {'rpn_class_loss': 1.0, 'rpn_bbox_loss': 1.0, 'mrcnn_class_loss': 1.0, 'mrcnn_bbox_loss': 1.0,
'mrcnn_mask_loss': 1.0}
MASK_POOL_SIZE                 14
MASK_SHAPE                     [28, 28]
MAX_GT_INSTANCES               100
MEAN_PIXEL                     [123.7 116.8 103.9]
MINI_MASK_SHAPE                (56, 56)
NAME                           object
NUM_CLASSES                    2
POOL_SIZE                      7
POST_NMS_ROIS_INFERENCE        1000
POST_NMS_ROIS_TRAINING         2000
PRE_NMS_LIMIT                  6000
ROI_POSITIVE_RATIO             0.33
RPN_ANCHOR_RATIOS              [0.5, 1, 2]
RPN_ANCHOR_SCALES              (32, 64, 128, 256, 512)
RPN_ANCHOR_STRIDE              1
RPN_BBOX_STD_DEV               [0.1 0.1 0.2 0.2]
RPN_NMS_THRESHOLD              0.7
RPN_TRAIN_ANCHORS_PER_IMAGE    256
STEPS_PER_EPOCH                1000
TOP_DOWN_PYRAMID_SIZE          256
TRAIN_BN                       False
TRAIN_ROIS_PER_IMAGE           200
USE_MINI_MASK                  True
USE_RPN_ROIS                   True
VALIDATION_STEPS               50
WEIGHT_DECAY                   0.0001
```

Figura 3.30. Parámetros de entrenamiento de Mask RCNN personal

De entre todos estos, los más importantes los veremos más adelante en la implementación del archivo de configuración *persoalizado.py*.

- Podemos ver por la imagen anterior que hemos escogido como backbone la **Resnet101**.
- "***visualize.py***": gracias a este archivo podemos visualizar las pérdidas y los errores en todo el entrenamiento.

Aunque tengamos un gran conjunto de datos (225 imágenes de botellas PET), no es suficiente para un entrenamiento preciso, debido a esto se ha instalado un conjunto de pesos preentrenados de coco "***mask_rcnn_coco.h5***".

El conjunto de pesos preentrenados se guardarán dentro de la carpeta *Mask RCNN Personal* y el directorio de Github "*Mask R-CNN for object Detection and Segmentation"* en *Mask RCNN*.

- **Crear archivo para entrenar el sistema**: el archivo que ejecutaremos desde el terminal lo denominaremos "**personalizado**" y tendrá la extensión **.py**. El fichero programado está estructurado de la manera siguiente:

Figura 3.31. Diagrama de flujo del archivo de configuración *personalizado.py*

- **Librerías**: las librerías que importamos son las siguientes

```
import os
import sys
import json
import numpy as np
import skimage.draw
import cv2
from mrcnn.visualize import display_instances
import matplotlib.pyplot as plt
```

Figura 3.32. Librerías de MRCNN personal

En este caso, la única librería no comentada anteriormente es "***mrcnn.visualize***" y esta es el archivo "*visualize.py*" visto en el apartado anterior.

- **Parámetros del entrenamiento a configurar**: los parámetros más importantes para que el entrenamiento sea ligero y preciso se pueden ver y se definen en la siguiente imagen:

```
class CustomConfig(Config):


    # Etiqueta que le hemos puesto al objeto en VGG Image JSON
    NAME = "bottle"

    # Numero de imágenes por cada paso
    IMAGES_PER_GPU = 1

    # Número de clases (incluido el fondo)
    NUM_CLASSES = 1 + 1  # Fondo + objeto (botella PET)

    # Número de pasos por época
    STEPS_PER_EPOCH = 10

    # Rechazar detecciones con menos del 70% de confianza
    DETECTION_MIN_CONFIDENCE = 0.7
```

Figura 3.33. Parámetros del entrenamiento de MRCNN Personal

Una vez realizados todos los pasos previos ahora se puede llevar a cabo el entrenamiento ejecutando el archivo *personalizado.py*.

Los entrenamientos realizados han sido los siguientes.

3.5.2 ENTRENAMIENTO MASK RCNN PERSONAL CON 20 ÉPOCAS DE 10 PASOS Y 225 IMÁGENES

Como podemos visualizar en las siguientes gráficas, con este pequeño entrenamiento de tan solo 20 épocas obtenemos resultados bastante prometedores

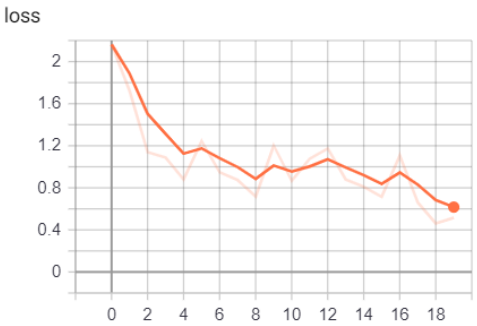

Figura 3.34. Gráfica del error total del ensayo

Figura 3.35. Gráfica del error de la caja delimitadora

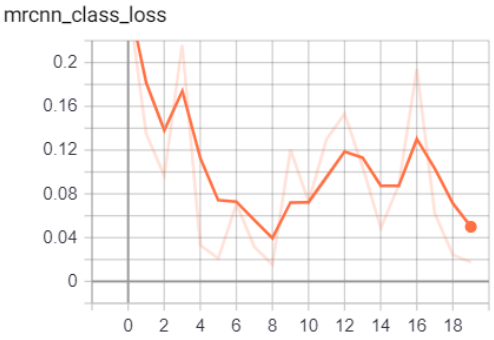

Figura 3.36. Gráfica del error de clasificación

Figura 3.37. Gráfica del error de segmentación

En la última época el **error total** es **0,6165**, el **error de caja delimitadora** es **0,2473**, el **error de clasificación 0,0499** y el **error de segmentación 0,1343**. Podemos ver que con esta pequeña cantidad de épocas ha resultado un entrenamiento con menos pérdidas que el de Faster RCNN de 2000 épocas. Además, con una **confianza** a partir del

70% el valor de la precisión media promedio **mAP** es de **0,1322**, lo cual para el número de épocas que tiene es una precisión bastante considerable.

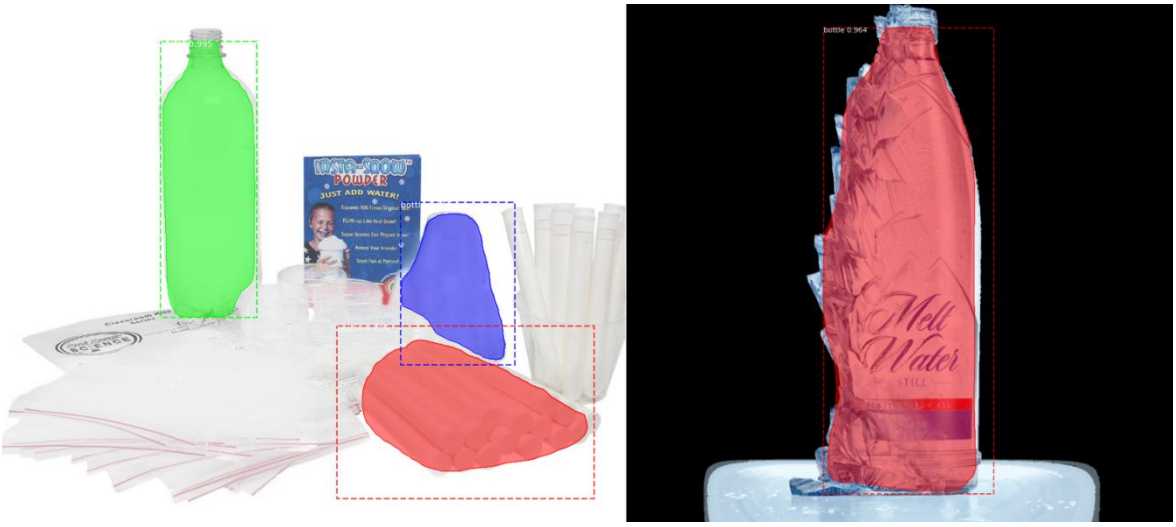

Figura 3.38. Ejemplo de detecciones correctas y erróneas de este modelo

3.5.3 ENTRENAMIENTO MASK RCNN PERSONAL CON 100 ÉPOCAS DE 10 PASOS Y GRAN CANTIDAD DE IMÁGENES

En este entrenamiento se ha implementado una nueva librería denominada "**imgaug**", gracias a esta se ha podido incrementar en gran medida el conjunto de imágenes de la misma manera que se hizo en los ensayos de Faster RCNN (volteando, cambiando el eje, difuminando la imagen, etc.).

Figura 3.39. Ejemplo de imagen volteada y difuminada con imgaug

Los resultados obtenidos en este entrenamiento están en las siguientes gráficas:

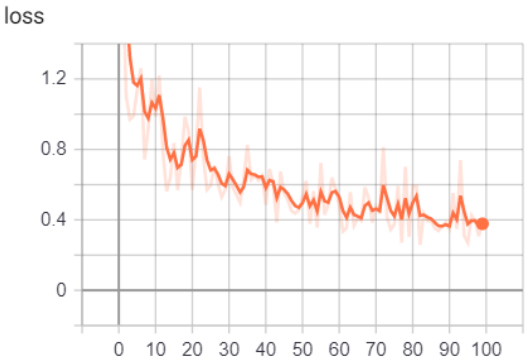

Figura 3.40. Gráfica del error total del ensayo

Figura 3.41. Gráfica del error de la caja delimitadora

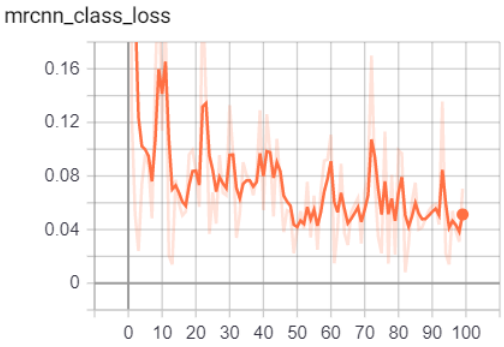

Figura 3.42. Gráfica del error de clasificación

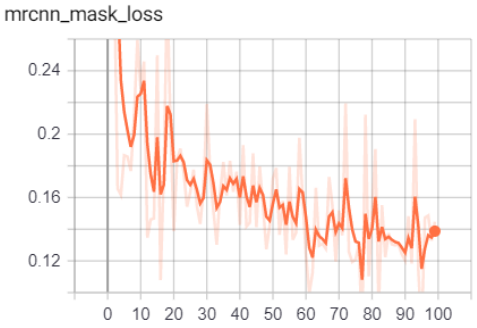

Figura 3.43. Gráfica del error de segmentación

En la última época el **error total** es **0,3777**, el error de **caja delimitadora** es **0,08146**, el **error de clasificación** es **0,0513** y el **error de la segmentación** es **0,1388**. La cantidad de errores es muy similar a la del entrenamiento de Faster RCNN preentrenada con 22000 épocas. Sin embargo, veremos a continuación que el valor del mAP cambia significativamente con respecto al resto de ensayos realizados.

En este caso el **mAP** con una confianza a partir del **70%** es de **0,4988** y con una confianza del **55%** es de **0,5339**, pero debido que a que con este grado de confianza el sistema tiene más posibilidades de detectar objetos inexistentes, mantenemos el grado de confianza al 70%.

3.6 COMPARACIÓN DE LOS DIFERENTES ENTRENAMIENTOS REALIZADOS

Podemos visualizar los resultados de los entrenamientos realizados en la siguiente tabla:

Entrenamientos	mAP	Error total	Error de clasificación	Error de caja delimitadora	Error de segmentación de objetos
Faster RCNN preentrenada con 2000 épocas	≈0%	0,6331	0,2412	0,308	Ø
Faster RCNN preentrenada con 22000 épocas	15,20%	0,2267	0,078	0,029	Ø
Mask RCNN personal con 20 épocas	13,22%	0,6165	0,0499	0,2473	0,1343
Mask RCNN personal con 100 épocas	53,39%	0,3777	0,0513	0,08146	0,1388

Tabla 3.1. Tabla comparativa de los modelos entrenados

Los entrenamientos Faster RCNN con 2000 épocas y Mask RCNN con 20 épocas y los entrenamientos Faster RCNN con 22000 épocas y Mask RCNN con 100 épocas tienen una gran similitud respecto a la cantidad de errores totales. Sin embargo, cuando probamos los entrenamientos vemos que la precisión de las Faster RCNN deja mucho que desear en comparación con las Mask RCNN, por ello el **mAP** de las Mask RCNN es mucho mayor. Esto se debe a que en el entrenamiento de las Faster RCNN el sistema ha memorizado los datos de entrenamiento y tiene un error muy bajo cuando compara su precisión de detección con los datos de entrenamiento, sin embargo, cuando el sistema tiene que detectar objetos nuevos es poco capaz de detectar las botellas PET, esto se debe a que las botellas son deformables y pueden ser de varias formas posibles, pero las cajas delimitadoras simplemente delimitan al objeto en un cuadrado. Con el entrenamiento de las Mask RCNN el resultado ha sido muy distinto ya que los objetos de entrada están delimitados por **polígonos** y por lo tanto al sistema ya le llega la información de la forma de la botella y como está esta deformada. Por el mAP de las **Mask RCNN** podemos comprobar que el sistema **ha aprendido lo que es una botella PET** y no simplemente ha memorizado los datos de entrenamiento.

Vamos a ver ahora la calidad con la que el sistema Mask RCNN personal es capaz de detectar las botellas PET.

3.7 PRUEBAS DE DETECCIÓN DE BOTELLAS PET DEL SISTEMA MASK RCNN PERSONAL CON 100 ÉPOCAS DE ENTRENAMIENTO

- **Detección de objetos en imágenes sacadas de internet**:

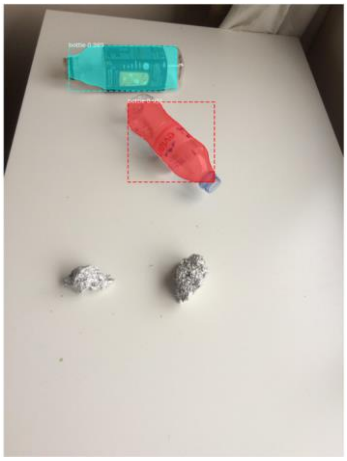

Figura 3.44 Detección de botellas PET en imagen

Figura 3.45 Detección de botellas PET en imagen

- **Detección a tiempo real en webcam**:

  Se ha programado el sistema para que se pueda ejecutar la cámara del ordenador en tiempo real y los resultados han sido los siguientes.

Figura 3.46 Detección de botella PET a tiempo real con otro residuo

Figura 3.47. Detección de botella PET a tiempo real con otro residuo

Figura 3.48. Detección de botella PET a tiempo real con otro residuo

Figura 3.49. Detección de botella PET a tiempo real con otro residuo

- **Análisis para la detección de varios elementos con confianza del 70%**:

Figura 3.50 Detección de botellas PET del sistema Mask RCNN con 70% de confianza

- **Detección de varios elementos con confianza del 55%**:

Figura 3.51. Detección de botellas PET del sistema Mask RCNN con 70% de confianza

3.7.1 PRUEBAS DE DETECCIÓN CON BOTELLAS DEFORMADAS Y OTRAS ORIENTACIONES DISTINTAS

Se han realizado pruebas con las botellas dobladas y enroscadas, así como imágenes del culo de la botella y del tapón y el sistema ha sido capaz de detectar la botella en su plenitud en la mayoría de los ensayos.

Figura 3.52. Detección de la botella enroscada

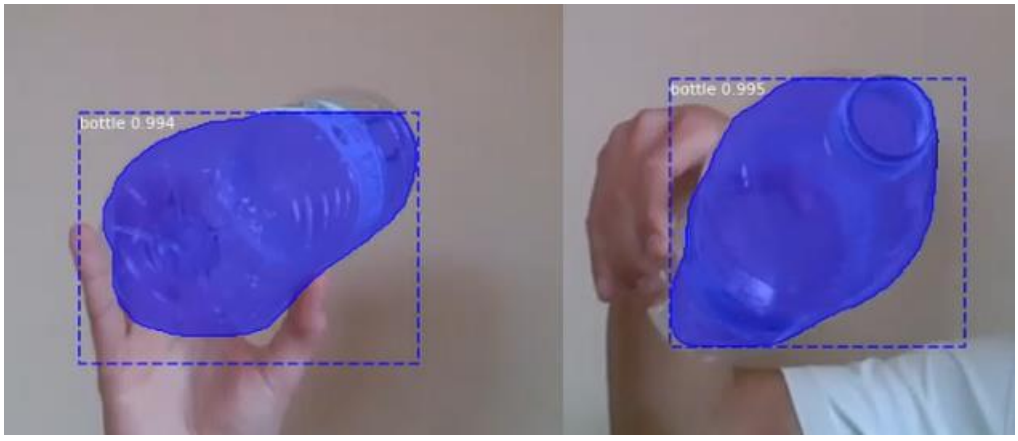

Figura 3.53. Detección del culo y la cabeza de la botella

3.7.2 **PROBLEMAS DEL SISTEMA**

Al realizar estos últimos ensayos se ha comprobado que cuando la botella está invertida o inclinada o deformada la detección del objeto es peor. Esto se debe a la falta de inputs en el entrenamiento con botellas PET inclinadas o boca abajo. Aunque mediante la librería *imgaug* se ha incrementado el número de imágenes de entrenamiento y con distintas inclinaciones e incluso invertidas, estas no han sido suficientes para tener una precisión correcta en estos casos. Sin embargo, la clasificación de la botella funciona estupendamente, debido a que, aunque la máscara no sea demasiado precisa en las imágenes de botellas inclinadas o boca abajo, estas son detectadas igualmente, por tanto, este problema no es de mayor interés debido a que la detección del objeto se realiza igual y con respecto a las botellas deformadas, el sistema es capaz de detectar el cuello de la botella o su cilindro o la rosca del tapón y por tanto detectar la botella de igual manera.

Figura 3.54. Detección del cuello y la rosca de la botella en la botella enroscada

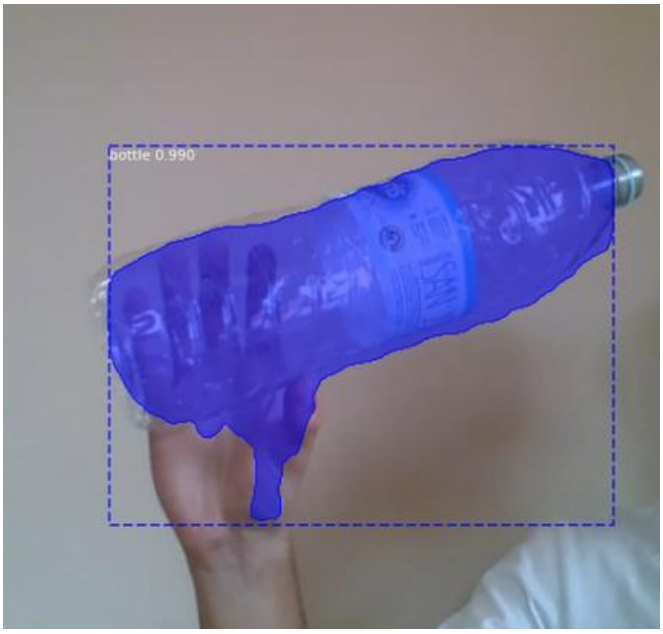

Figura 3.55. Detección de la botella inclinada con cierta imprecisión de la máscara

Con respecto a la detección de botellas que no son del residuo PET se ha obtenido el siguiente resultado:

Figura 3.56. Detección del sistema en una imagen con botellas PET y otro tipo de botellas

Podemos ver en la imagen que el sistema ha detectado, además de las botellas de PET, también una botella de HDPE debido a su alta similitud en la forma con las botellas de PET. La mayoría de estas botellas son de colores claros (blanco y amarillo) mientras que las botellas de PET son transparentes o azules la mayoría. Existe un sistema denominado **método de detección de colores** con el cual, mediante **lógica difusa**, podríamos capturar los tonos blancos y amarillos y eliminar estos elementos posteriormente de la recogida de las botellas. Como podemos comprobar, es sistema no tiene problema para no detectar esta botella de cristal y esta garrafa debido a que sus formas difieren de las de las botellas PET (la botella de cristal tiene un cuello alargado y la garrafa de aceite de coche tiene forma cuadrada). Aun así, algunas botellas de cristal con un cuello no tan alargado podrían ser detectadas, para evitar este problema, se podría implementar posterior a la selección del material PET un **separador por densidad** que fuese capaz de extraer el vidrio restante. Estas implementaciones del método de detección de colores o del separador de densidad sería para el máximo rendimiento del sistema, pero los resultados obtenidos han sido bastante favorables y aun con estos pequeños inconvenientes el sistema sigue siendo bastante eficiente y capaz de detectar la mayoría de las botellas con un error de clasificación mínimo de 5,13% y un mAP que ronda el 50% de precisión.

3.7.3 **DETECCIÓNES EN FONDO NEGRO**

Las detecciones en un fondo negro opaco con buena iluminación han sido las más exitosas, detectando el objeto perfectamente en todos los ángulos y con gran exactidud (mAP muy elevado)

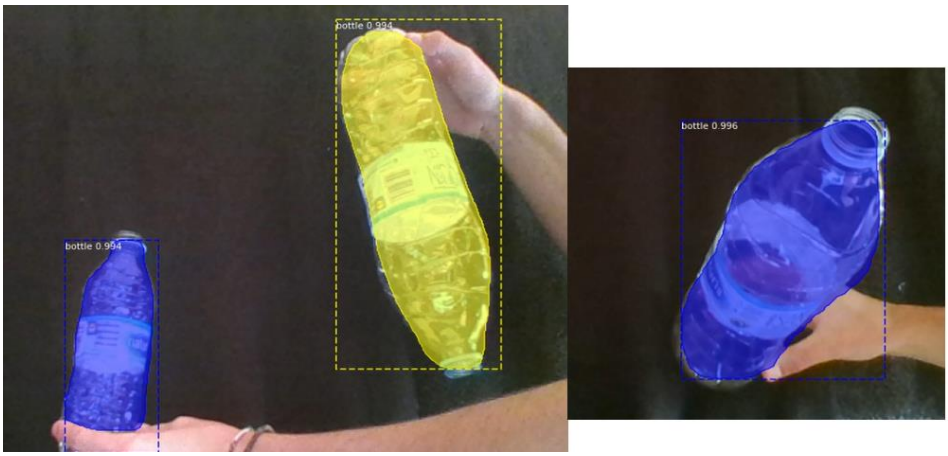

Figura 3.57. Detección de botellas en fondo oscuro casi en su totalidad con botellas en distintas orientaciones

Esto se debe a que al ser un fondo negro y opaco no se van a reflejar colores ni se van a apreciar deformidades en el fondo.

Por lo tanto, cabe destacar que, **la implementación de este sistema** en una planta de reciclaje debería ir acompañada de estos factores. Para ello sería importante instalar un **buen sistema de iluminación** y el **fondo** donde se analicen las imágenes debería ser **limpio y que absorba el resto de luz restante y no se creen reflejos o se muestren irregularidades innecesarias (fondo oscuro o prácticamente negro y opaco)**.

Una buena iluminación se puede conseguir con focos de luz que emitan todos los colores del espectro visible

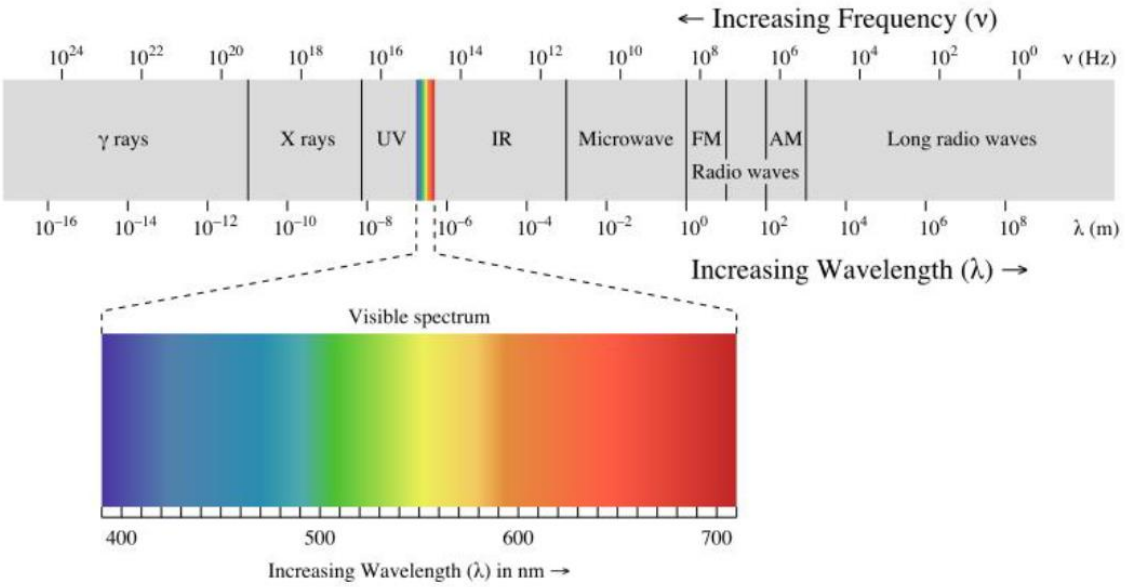

Figura 3.58. Espectro total de la luz(Novak 2014)

Hoy en día las únicas fuentes de energía que emiten todo el espectro visible son el sol y las luces incandescentes. Otra fuente de luz artificial que produce la mayoría de los colores del espectro visible serían los halógenos, pero estos están prohibidos desde 2018(SMARTLIGHTING 2018). Hoy en día con la combinación de luces **LED** y **luminóforos de fósforo** podemos conseguir un espectro muy grande del espectro visible. Estos no son muy costosos y con ciertas herramientas como un foco y un fondo oscuro podemos hacer que se incremente en gran cantidad la detección de botellas PET.

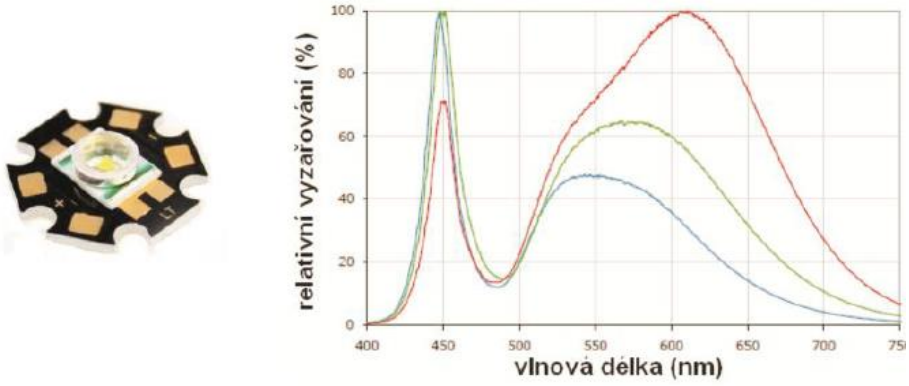

Figura 3.59. Espectros visibles de los LED con luminóforo de fósforo (Novak 2014)

4 Conclusiones

A la vista de los ensayos realizados, el método con mayor calidad de detección es el sistema de **Mask RCNN personal con 100 épocas**. Aunque su precisión no sea perfecta puede detectar la mayoría de las botellas con un **mAP** que ronda el 50% y un error de **clasificación** mínimo del **5,13%**. La falta de precisión que pueda tener el sistema se debe en gran medida en la calidad de las imágenes del entrenamiento debido a que, en algunas de ellas, las botellas eran casi imperceptibles para el ojo humano y también se debe a que en la mayoría de las imágenes las botellas están derechas y cuando la imagen aparece al revés el sistema tiene mayor dificultad de detectar los datos.

El sistema se podría mejorar con la implementación de una gran cantidad de imágenes de botellas PET en las que se detecte bien las posiciones de las botellas. Esto no se ha podido realizar debido a la falta de conjuntos de estas imágenes con segmentación de objetos de calidad.

Este proyecto se ha realizado tan solo para un tipo de residuo (PET), pero se podría entrenar el mismo sistema con distintos residuos como latas, otros tipos de plásticos, etc. o incluso varios de estos a la vez y de esta manera el programa sería capaz de detectar una gran cantidad de desechos para reciclar en una planta de residuos. Sin embargo, la obtención de datos de calidad para residuos es difícil y se ha decidido implementar un sistema que solo detecte material PET.

Durante la realización de este proyecto se han adquirido conocimientos en programación con Python, visión artificial, inteligencia artificial, programación de RCNN y diagramas de flujo de los cuales prácticamente el autor desconocía su funcionamiento. Además, también se han obtenido conocimientos en la elaboración de un proyecto, los pasos a seguir, el cumplimiento del calendario, la investigación online y la realización de proyectos que empiezan siendo una simple idea en la cabeza.

En conclusión, se han alcanzado los objetivos deseados ya que se han **comparado e implementado diferentes modelos de detección con inteligencia artificial** y **el sistema Mask RCNN personal se podría integrar en una planta de residuos de manera económica** (solo es necesario una cámara para captar las imágenes y un ordenador con el

sistema Mask RCNN incorporado para detectar las botellas) y sería capaz de detectar la gran mayoría de las botellas. Además, **en lo que respecta a materiales deformables** como estas botellas de PET podemos decir que, dependiendo de la calidad de las imágenes de entrada, el sistema de **Mask RCNN puede llegar a detectar sin ningún problema un material en la mayoría de sus formas** (doblado, volteado, etc.)

5 Glosario

AI: Artificial Intelligence

AlexNet: Alex Network

AP: Average Precision

API: Application Programming Interface

CMU: Carnegie Mellon University

CNN: Convolutional Neural Network

COCO: Common Objects in Context

CPU: Central Processing Unit

CUDA: Compute Unified Device Architecture

cuDNN: CUDA Deep Neural Network

Dendral: sistema con IA para ayudar a determinar la estructura de un compuesto.

Fast RCNN: Fast Region Based Convolutional Neural Network

Faster RCNN: Faster Region Based Convolutional Neural Network

GoogleNet: Google Network

GPS: General Problem Solver

GPU: Graphics Processing Unit

HDF5: Hierarchical Data Format version 5

HDPE: High Density Polyethylene

HMM: Hidden Markov Model

IA: Inteligencia artificial

IBM: International Business Machines

IDE: Integrated Development Environment

ImgAug: Image Augmentation

IoU: Intersection over Union

JSON: JavaScript Object Notation

LDPE: Low Density Polyethylene

LeNet: LeCun Network

LT: Logic Theorist

mAP: Mean Average Precision

Mask RCNN ó MRCNN: Mask Region Based Convolutional Neural Network

MIT: Massachusetts Institute of Technology

Mycin: sistema con IA para diagnosticar enfermedades infecciosas de sangre.

OOP: Object-Oriented Programming

OpenCV: Open Source Computer Vision Library

OS: Operating System

PET: Polyethylene Terephthalate

PP: Polypropylene

PS: Polystyrene

PyPI: Python Package Index

PyTorch: Python Torch

R1: programa para la configuración de sistemas informáticos

RCNN: Region Based Convolutional Neural Network

ReLU: Rectified Linear Unit

RoI: Region of Interest

RPN: Region Proposal Network

SHRDLU: programa con IA para la comprensión del lenguaje. Proviene de la frase sin sentido "*etaoin shrdlu"*

Sputnik: primer satélite artificial de la historia

SSD: Solid State Drive

SVM: Support Vector Machine

TACO: Trash Annotations in Context

TFRecord: TensorFlow Record

Trash-ICRA19: Trash Infection Control Risk Assessment 2019

UAVVaste: Unmanned Aerial Vehicle Vaste

UUID: Universally Unique Identifier

VGG: Visual Geometry Group Network

VGGNet: Visual Geometry Group Network

XML: Extensible Markup Language

ZF Net: Zeiler and Fergus Network

6 Agradecimientos

Agradecimientos a la tutora Angela Díez Díez.

Agradecimientos a mis padres, a mi hermana, a mi novia y amigos por aguantarme las veces que las cosas no salían como debían, las frustraciones y a su vez animarme a continuar con este proyecto y en general con este grado, no lo hubiese conseguido sin vosotros.

7 Bibliografía

Bai, T. et al. 2020. remote sensing An Optimized Faster R-CNN Method Based on DRNet and RoI Align for Building Detection in Remote Sensing Images. Disponible en: www.mdpi.com/journal/remotesensing.

Birodkar, V. 2021. models/tf2_detection_zoo.md at master · tensorflow/models · GitHub. Disponible en: https://github.com/tensorflow/models/blob/master/research/object_detection/g3doc/tf2_detection_zoo.md [Accedido: 31 agosto 2022].

Brownlee, J. 2019. A Gentle Introduction to Object Recognition With Deep Learning. Disponible en: https://machinelearningmastery.com/object-recognition-with-deep-learning/ [Accedido: 19 junio 2022].

Cabrera, J.M., Becerra Sánchez, A., Valles, G., Guerrero, S., Ramírez, U. y Correa, G. 2016. *SISTEMA DE RECONOCIMIENTO DE DÍGITOS MANUSCRITOS UTILIZANDO REDES NEURONALES*.

Caponetti, Laura. 2017. *Fuzzy Logic for Image Processing A Gentle Introduction Using Java*. 1st ed. 2017. Castellano, Giovanna. ed. Cham: Springer International Publishing. Disponible en: https://sci-hub.hkvisa.net/10.1007/978-3-319-44130-6 [Accedido: 10 abril 2022].

Casas Roma, Jordi., Bosch Rué, Anna. y Lozano Bagén, Toni. 2019. *Deep learning : principios y fundamentos*. Disponible en: https://elibro-net.unileon.idm.oclc.org/es/ereader/unileon [Accedido: 18 abril 2022].

Ecoembes 2020. Ecoembes Informe Anual 2020 | El reciclaje en 2020. Disponible en: https://ecoembes.com/landing/informe-anual-2020/valor-compartido/el-reciclaje-en-2020/ [Accedido: 4 septiembre 2022].

Feigenbaum, E., Buchanan, B. y Lederberg, J. 1970. *On Generality and Problem Solving: A Case Study Using the Dendral Program*. Stanford. Disponible en:

https://ntrs.nasa.gov/api/citations/19710028679/downloads/19710028679.pdf [Accedido: 24 marzo 2022].

Girshick, R., Donahue, J., Darrell, T. y Malik, J. 2014. Rich feature hierarchies for accurate object detection and semantic segmentation Tech report (v5). Disponible en: http://www.cs.berkeley.edu/˜rbg/rcnn. [Accedido: 18 agosto 2022].

Google's protocol buffers 2022. GitHub - tensorflow/models: modelos y ejemplos creados con TensorFlow. Disponible en: https://github.com/tensorflow/models [Accedido: 31 agosto 2022].

He, K., Gkioxari, G., Dollár, P. y Girshick, R. 2018. Mask R-CNN.

Kemal, E. 2020. Understanding Region of Interest - Part 2 (RoI Align) - Blog by Kemal Erdem. Disponible en: https://erdem.pl/2020/02/understanding-region-of-interest-part-2-ro-i-align [Accedido: 20 agosto 2022].

Kim, N. 2022. GitHub - protocolbuffers/protobuf: Protocol Buffers - Google's data interchange format. Disponible en: https://github.com/protocolbuffers/protobuf [Accedido: 31 agosto 2022].

Kirillov, A., He, K., Girshick, R., Rother, C. y Dollár, P. 2021. Panoptic Segmentation. Disponible en: https://arxiv.org/abs/1801.00868. [Accedido: 20 agosto 2022].

Lai, S.-H. y Lepetit, V. 2016. *Computer Vision ACCV 2016*. Disponible en: https://link.springer.com/content/pdf/10.1007/978-3-319-54193-8.pdf [Accedido: 6 septiembre 2022].

M. Reza, F. 1961. *An Introduction to Information Theory*. New York: Dover Publications, INC. Disponible en: https://books.google.cz/books?hl=es&lr=&id=RtzpRAiX6OgC&oi=fnd&pg=PA1&dq=information+theory+what+is+it&ots=JXxbn2yXz4&sig=wFqKkU8Q08Ub1s9LnxhGmgLf8Gg&redir_esc=y#v=onepage&q=information%20theory%20what%20is%20it&f=false [Accedido: 28 marzo 2022].

Majchrowska, S., Mikołajczyk, A., Ferlin, M., Klawikowska, Z., Plantykow, M.A., Kwasigroch, A. y Majek, K. 2022. Deep learning-based waste detection in natural and urban environments. *Waste Management* 138, pp. 274–284. Disponible en: https://github.com/AgaMiko/waste-datasets-review [Accedido: 28 agosto 2022].

Matlab 2022. Support Vector Machine (SVM) - MATLAB & Simulink. Disponible en: https://es.mathworks.com/discovery/support-vector-machine.html [Accedido: 18 agosto 2022].

Max AI 2022. Max-AI® VIS (for Visual Identification System.

MEYER Company 2022. OPTICAL SORTING PROCESS - MEYER Europe. Disponible en: https://meyer-corp.eu/optical-sorting-process/ [Accedido: 27 agosto 2022].

Nielsen. Michael 2019. *Neural networks and deep learning*. Disponible en: http://neuralnetworksanddeeplearning.com/ [Accedido: 12 abril 2022].

Novak, T. 2014. Basics of interior lighting systems design.

Parlamento Europeo 2018. Directiva (UE) 2018/ del Parlamento Europeo y del Consejo, de 30 de mayo de 2018, por la que se modifica la Directiva 2008/98/CE sobre los residuos.

Ren, S., He, K., Girshick, R. y Sun, J. 2016. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. Disponible en: http://image-net.org/challenges/LSVRC/2015/results [Accedido: 20 agosto 2022].

Rezatofighi, H., Tsoi, N., Gwak, J., Sadeghian, A., Reid, I. y Savarese, S. 2019. Generalized Intersection over Union: A Metric and A Loss for Bounding Box Regression.

Rich, E. y Knight, K. 1991. *ARTIFICIAL INTELLIGENCE*. 3ª ed. Disponible en: https://www.vssut.ac.in/lecture_notes/lecture1423725949.pdf [Accedido: 16 marzo 2022].

Russell, S. y Norvig, P. 2010. *Artificial Intelligence A Modern Approach*. 3ª ed. Disponible en: https://zoo.cs.yale.edu/classes/cs470/materials/aima2010.pdf [Accedido: 16 marzo 2022].

Santarcangelo, J. 2020. Python para Data Science y AI | Coursera. Disponible en: https://www.coursera.org/learn/python-para-data-science-y-ai [Accedido: 29 agosto 2022].

Scikit-learn 2022. Precision-Recall — scikit-learn 1.1.2 documentation. Disponible en: https://scikit-learn.org/stable/auto_examples/model_selection/plot_precision_recall.html [Accedido: 18 agosto 2022].

SMARTLIGHTING 2018. ¿Por qué se prohíben las bombillas halógenas? Disponible en: https://smart-lighting.es/se-prohiben-las-bombillas-halogenas-septiembre-2018/ [Accedido: 5 septiembre 2022].

TACO trash annotations 2022. TACO Trash Annotations in Context. Disponible en: http://tacodataset.org/ [Accedido: 29 agosto 2022].

Traore, B.B., Kamsu-Foguem, B. y Tangara, F. 2018. Deep convolution neural network for image recognition. *Ecological Informatics* 48, pp. 257–268. doi: 10.1016/J.ECOINF.2018.10.002.

Verma, Y. 2021. R-CNN vs Fast R-CNN vs Faster R-CNN - A Comparative Guide. Disponible en: https://analyticsindiamag.com/r-cnn-vs-fast-r-cnn-vs-faster-r-cnn-a-comparative-guide/ [Accedido: 20 agosto 2022].

VGG Image Annotator 2019. The VIA annotation software for images, audio and video. *MM 2019 - Proceedings of the 27th ACM International Conference on Multimedia* , pp. 2276–2279. doi: 10.1145/3343031.3350535.

Yadav, Neha. 2015. *An Introduction to Neural Network Methods for Differential Equations*. 1st ed. 2015. Yadav, Anupam. y Kumar, Manoj. eds. Dordrecht: Springer Netherlands. doi: 10.1007/978-94-017-9816-7.

8 Anexos

ANEXO A. NORMATIVA EUROPEA SOBRE LA GESTIÓN DE RESIDUOS

Este proyecto se ha realizado cumpliendo todas las normas europeas sobre la gestión residuos y el sistema realizado para la detección de residuos es favorable con la "*Normativa europea sobre la gestión de residuos*" ('Directiva (UE) 2018/ del Parlamento Europeo y del Consejo, de 30 de mayo de 2018, por la que se modifica la Directiva 2008/98/CE sobre los residuos', 2018). Las ideas generales de esta normativa son las siguientes:

- La gestión de residuos de la UE debe ser sostenible con el medio ambiente y proteger la salud humana.
- El uso de la materia prima debe ser eficiente y los residuos deben reciclarse para evitar la extracción de recursos.
- Hay que promover el uso de las energías renovables
- Se debe promover el reciclaje y reutilización de residuos
- Toda implementación sobre la gestión de residuos debe de cumplir con el marco legal
- Las sustancias explosivas o toxicas deben llevar una gestión especial a las demás
- No se deben juntar los residuos tóxicos o explosivos con el resto de los residuos
- Hay que potenciar la economía circular evitando la importación de materias primas foráneas
- La gestión de residuos municipales debe comprometer un régimen eficiente de recogida, depósito y separación de residuos.
- La gestión de residuos debe llegar a todas partes incluidas las zonas más desfavorecidas
- Todo el mundo tiene derecho a reciclar
- Las plantas de residuos deben estar localizadas en zonas que no afecten demasiado al entorno, la naturaleza o a la población.
- El procesamiento de los residuos debe ser sostenible y crear el menor impacto posible en el medioambiente.

- El personal ocupado de la gestión de los residuos debe ir debidamente protegido y las instalaciones deben garantizar la protección de estos.

ANEXO B. VÍDEO DE ANÁLISIS DEL SISTEMA MASK RCNN PERSONAL A TIEMPO REAL

El video a tiempo real de detección de botellas PET con otros residuos realizado por el sistema Mask RCNN a tiempo real está en el siguiente enlace:

https://drive.google.com/file/d/1EBh4VAfru071xHxhJWLfyGn8W6plNtOG/view?usp=sharing

ANEXO C. REPOSITORIO GITHUB CON LA PROGRAMACIÓN REALIZADA

El repositorio está en el siguiente enlace:

https://github.com/DanielComesana/TESIS-FINAL-GRADO.git

ANEXO D. CRONOGRAMA

El diagrama de Gantt que se ha seguido para la realización de este proyecto es el siguiente:

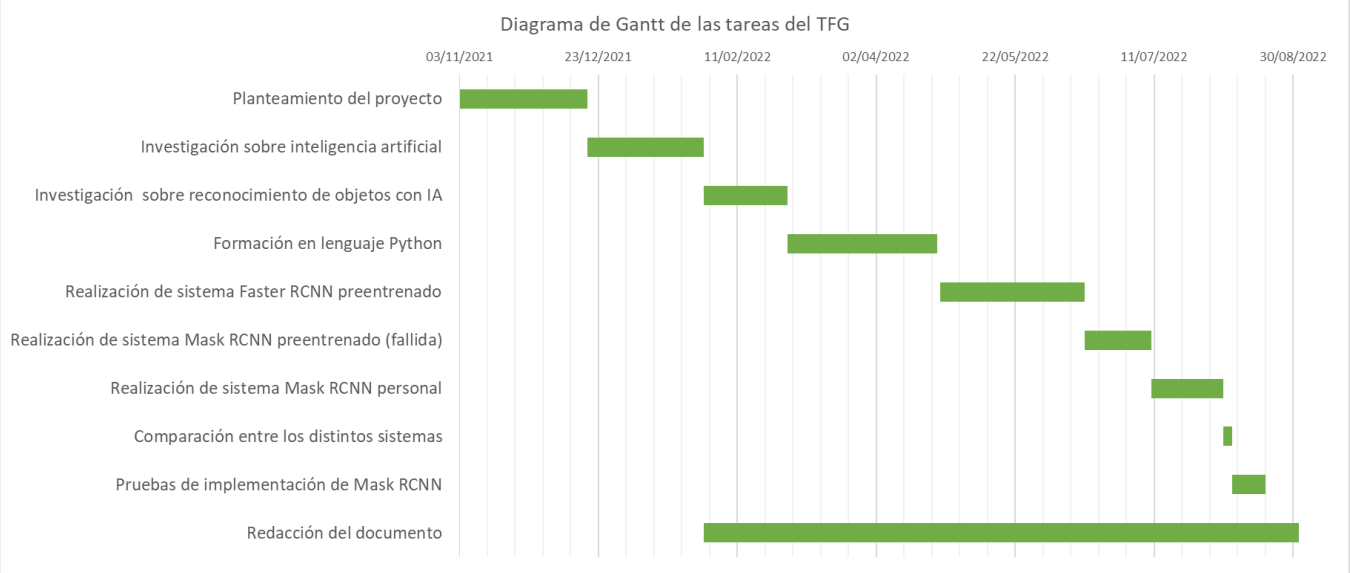

Figura 8.1. Diagrama de Gantt de este proyecto

ANEXO E. MATERIALES Y PERSONAL AL CARGO DE ESTE PROYECTO

Este proyecto ha sido realizado por Daniel Comesaña Pereira, alumno de la Universidad de León con la tutoría de Ángela Díez Díez.

El material utilizado para la elaboración de este proyecto ha sido un ordenador MSI GL62M 7RDX con Intel i7 y una tarjeta gráfica Nvidia GTX 1050. Los programas implementados han sido Python, Anaconda, Jupyter, Flowchart, Word, Visual Studio, Excel y Mendeley.

La cámara para las pruebas del sistema ha sido una MSI webcam **BisonCam nb pro** HD type (30fps@720p); la cual funciona con una **resolución HD 720** (1280 x 720 píxeles), tiene una **velocidad** de 30 imágenes por segundo (**30fps**), una cantidad de **0,92 MP, definición HD**, **sensor RGB** con un total de **76932 colores**. Los atributos tomados en las pruebas han sido luminosidad del 43,43%, brillo de 42,75% y saturación del 2,75%.