Diseño e implementación de un prototipo para detección de envases en residuos sólidos por medio de visión artificial

Deivith Enrique Amaya Lopez

Universidad Santo Tomás
Facultad de Ingeniería Electrónica
Bogotá, Colombia
2022

Diseño e implementación de un prototipo para detección de envases en residuos sólidos por medio de visión artificial

Deivith Enrique Amaya Lopez

Proyecto de grado presentado como requisito para optar al título de:

Ingeniero Electrónico

Director:
Dario Alejandro Segura Torres

Universidad Santo Tomás
Facultad de Ingeniería Electrónica
Bogotá, Colombia
2022

Autoridades de la universidad

RECTOR GENERAL

Fray José Gabriel Mesa Angulo, O.P.

VICERRECTOR ADMINISTRATIVO Y FINANCIERO GENERAL

Fray Rubén Darío LÓPEZ GARCÍA, O.P,

VICERRECTOR ACADÉMICO GENERAL

Fray Eduardo González Gil, O.P.

SECRETARIO GENERAL

Doctora Ingrid Lorena Campos Vargas

DECANO DIVISIÓN DE INGENIERÍAS

Fray Érico Juan Macchi Céspedes, O.P.

SECRETARIA DE DIVISIÓN

Doctora Luz Patricia Rocha Caicedo

DECANO FACULTAD DE INGENIERÍA ELECTRÓNICA

Ingeniero Carlos Enrique Montenegro Narváez

Nota de aceptación

Firma del tutor

Firma del jurado

Firma del jurado

BOGOTÁ D.C. _____ DE 2022

ADVERTENCIA

La Universidad Santo Tomás no se hace responsable de las opiniones y conceptos expresados en el trabajo de grado, solo velará por qué no se publique nada contrario al dogma ni a la moral católica y porque el trabajo no tenga ataques personales y únicamente se vea el anhelo de buscar la verdad científica.

Capítulo III –Art. 46 del Reglamento de la Universidad Santo Tomás.

Contenido

- Resumen **8**

- Abstract **9**

- Lista de Figuras **10**

- Lista de Tablas **12**

- Glosario **13**

1. Introducción **1**
1.1. Planteamiento del Problema 2
1.2. Objetivos 3
1.2.1. Objetivo General 3
1.2.2. Objetivos Específicos 3
1.3. Justificación 4
1.4. Impacto Social 5

2. Estado del Arte **6**

3. Marco Teórico **8**
3.1. Redes neuronales artificiales 8
3.2. Neurona artificial 8
3.3. Función de activación 9
3.4. Capas 9
3.5. Entrenamiento 10
3.6. Descenso de gradiente 10
3.7. Visión artificial 11
3.8. Digitalización de la imagen 11
3.9. Convolución 11
3.10. CNN (Convolutional Neural Network) 12
3.11. Bloque identidad y bloque convolucional 13
3.12. Modelos 13
3.13. SSD 14
3.14. YOLO 15
3.15. Software 16
3.16. Evaluación del modelo 16

4. Diseño Metodológico **18**
4.1. Fases y Metodología 18

4.1.1. Creación de la base de datos 18
4.1.2. Entrenamiento y elección del modelo 18
4.1.3. Creación de la banda e implementación del hardware 18
4.1.4. Diseño del algoritmo 19
4.1.5. Revisión del funcionamiento en el sistema embebido 19

5. Desarrollo Conceptual 20
5.1 Creación de la base de datos 20
5.2 Entrenamiento y elección del modelo 21
5.2.1 RCNN conjunto de datos combinados 21
5.2.2 RCNN conjunto de datos separados 27
5.2.3 SSD conjunto de datos combinados 29
5.2.4 SSD conjunto de datos separados 34
5.2.5 YOLO conjunto de datos combinados 35
5.2.6 YOLO conjunto de datos separados 39
5.3. Creación de la banda e implementación del hardware 41
5.4. Diseño del algoritmo 43
5.5. Revisión del funcionamiento en el sistema embebido 48

6. Resultados y Discusión 50

7. Conclusiones y Trabajos futuros 54

8. Referencias Bibliográficas 56

9. Anexos 59
Anexos 1. Código del sistema escrito en lenguaje de programación Python. 59
Anexos 2. Tablas de resultados obtenidos en las pruebas de selección del modelo 64

- Resumen

El presente proyecto de grado tiene como finalidad la creación de un sistema para detectar envases de tipo botella de plástico y vidrio en los residuos conglomerados, utilizando la visión computacional implementada en un sistema embebido que es capaz de: detectar una muestra de basura por medio de sensores colocados al inicio y al final de una banda transportadora, llevar los residuos desde el punto de origen hasta el punto de la adquisición de la imagen utilizando la banda transportadora, capturar una imagen y detectar los envases dentro de la muestra de residuos. Este proceso junto con consideraciones especiales conforman el algoritmo diseñado que explica el correcto funcionamiento del sistema.
El proyecto cumplió con los objetivos planteados logrando ejecutar correctamente el algoritmo y haciendo buenas inferencias sobre la muestra de basura. Sin embargo, se reconoció la limitación del sistema de detección de objetos al presentarse envases en posiciones en las que la inferencia no brinda la información necesaria para poder tomar el envase dentro de la muestra.

Palabras clave: visión computacional, sistema embebido, inferencia.

- Abstract

The purpose of this degree project is to create a system to detect plastic and glass bottle type containers in conglomerate waste, using computer vision implemented in an embedded system that is capable of: detecting a sample of waste by means of sensors placed at the beginning and end of a conveyor belt, carrying the waste from the point of origin to the point of image acquisition using the conveyor belt, capturing an image and detecting the containers within the waste sample. This process along with special considerations make up the designed algorithm that explains the correct operation of the system.
The project met the stated objectives by managing to correctly execute the algorithm and making good inferences about the waste sample. However, it was recognized the limitation of the object detection system by presenting containers in positions in which the inference does not provide the necessary information to be able to take the container within the sample.

Keywords: computer vision, embedded system, inference.

- Lista de Figuras

Figura 3.1. Estructura básica de una neurona artificial 9
Figura 3.2. Capas en una red neuronal 10
Figura 3.3. Proceso de convolución 12
Figura 3.4. CNN 13
Figura 3.5. Bloque identidad 14
Figura 3.6. Regiones de mapa de características 15
Figura 3.7. Estructura red SSD 15
Figura 3.8. Bounding Box de la estructura YOLO 16
Figura 3.9. Estructura de la red YOLO 16
Figura 5.2. Imágenes sobre la banda etiquetadas 21
Figura 5.3. Búsqueda selectiva 21
Figura 5.4. Comparación de coordenadas 22
Figura 5.5. Comparación de coordenadas para regiones sin botella 23
Figura 5.6. Importar y cargar el modelo 23
Figura 5.7. Cabecera de la red y armado de la red 24
Figura 5.8. Compilación del modelo 25
Figura 5.9. Entrenamiento de la red 25
Figura 5.10. Métricas del entrenamiento 26
Figura 5.11. Resultado de la red RCNN 27
Figura 5.12. Entrenamiento imágenes Google 27
Figura 5.13. Entrenamiento imágenes sobre banda 28
Figura 5.14. Resultado de la red RCNN modelo 2 29
Figura 5.15. Resultado de la red RCNN modelo 2 29
Figura 5.16. Instalación de dependencias y el API de Tenserflow 30
Figura 5.17. Modelos, velocidad y precisión 31
Figura 5.18 Descarga del modelo elegido 31
Figura 5.19. Configuración archivo pipeline.config 31
Figura 5.20. Entrenamiento de la red 32
Figura 5.21 Métrica de pérdida total 32
Figura 5.22. Resultado de la red SSD modelo 1 33
Figura 5.23. Resultado de la red SSD modelo 1 33
Figura 5.25. Reentrenamiento modelo SSD 34
Figura 5.26. Métrica de pérdida total 34
Figura 5.27. Resultado de la red SSD modelo 2 35
Figura 5.28. Descargar modelo YOLOv5 36
Figura 5.29. hyp.finetune configuración 36
Figura 5.30. Exportar etiquetas e imágenes 37
Figura 5.31. Configuración de la red 37
Figura 5.32. Entrenamiento de la red 37
Figura 5.33. Exportación del modelo 37
Figura 5.34. Pérdida de localización del objeto y creación de la caja 38
Figura 5.35. Resultado de la red YOLO modelo 1 39
Figura 5.36. Resultado de la red YOLO modelo 1 39
Figura 5.37. Segundo entrenamiento 40

Figura 5.38. Pérdida de localización del objeto y creación de la caja 40
Figura 5.39. Imágenes de prueba 41
Figura 5.40. Banda transportadora 43
Figura 5.41. DC motor driver 43
Figura 5.42. Esquemático de conexiones 44
Figura 5.43. Algoritmo de funcionamiento 45
Figura 5.44. Diagrama de árbol del fichero programa 46
Figura 5.45. Iniciar.py inicialización 47
Figura 5.46. Inicialización clase modelo 47
Figura 5.47. Bucle de inferencia 48
Figura 5.48. Bucle de funcionamiento 49
Figura 5.49. Prueba de funcionamiento 1 50
Figura 6.1. Pruebas de inferencias 54
Figura 8.1. Iniciar.py 58
Figura 8.2. Camara2.py 59
Figura 8.3. FIchero.py 60
Figura 8.4. Modelo.py 61
Figura 8.5. Motor.py 62
Figura 8.6. Pantalla.py 62
Figura 8.7. Sensor.py 63

- Lista de Tablas

Tabla 5.1. Resultados conjunto de datos unidos 26
Tabla 5.2. Resultados sobre banda 28
Tabla 5.3. Resultados conjunto de datos unidos 32
Tabla 5.4. Resultados datos separados 35
Tabla 5.5. Resultados datos combinados 38
Tabla 5.6. Resultados sobre banda 40
Tabla 5.7. Tabla de resultados 41
Tabla 6.1. Resultados de las pruebas de funcionamiento 51
Tabla 6.2. Resultados de las pruebas del modelo de visión artificial 52
Tabla 6.3. Resultados de las pruebas del modelo de visión artificial 53
Tabla 8.1. RCNN conjunto de datos combinados 63
Tabla 8.2. RCNN conjunto de datos separados 64
Tabla 8.3. SSD conjunto de datos juntos 65
Tabla 8.4. SSD conjunto de datos separados 65
Tabla 8.5. YOLOV5 conjunto de datos juntos 66
Tabla 8.6. YOLOv5 conjunto de datos separados 67

- Glosario

IOU: Es la abreviatura de intersection over union (intersección sobre unión) y es una métrica utilizada para evaluar la precisión de un detector de objetos.

CNN: Es la abreviatura de convolutional neural network que es una red neuronal que utiliza la convolución para hacer el procesamiento de imágenes.

RCNN: Son redes neuronales convolucionales basadas en regiones, la principal diferencia es que las CNN se utilizan para clasificación de imágenes y las RCNN para la detección de objetos asignando cuadros delimitadores. Esto se hace extrayendo propuestas de región, que son regiones de la imagen que potencialmente tienen los objetos que se buscan.

SSD: Es la abreviatura de single shot multibox (captura múltiple de un solo disparo) y es un modelo de red neuronal de detección de objetos.

Inferencia: Es el resultado que devuelve un modelo de detección de objetos al procesar una imagen de muestra, la inferencia contendrá la ubicación de los objetos detectados y la confianza de que sea de una clase específica.

Flatten: Función que transforma una matriz en un vector de 1 dimensión.

Dropout: Función que sirve para apagar secciones de la red durante el entrenamiento.

Data augmentation: Proceso que aumenta la cantidad de muestras de un conjunto de datos a partir de transformaciones a las muestras manteniendo la información útil que la red necesita aprender.

1. Introducción

La humanidad hoy en día se encuentra en una situación crítica, la gran cantidad de residuos que no son tratados debidamente para una correcta descomposición a lo largo de la historia, han dejado inmensos cúmulos de basura que generan una serie de complejos y muy variados problemas que afectan directamente al ambiente y por ende afectan directamente a las personas, aunque no se encuentren cerca de los sitios de almacenamiento de residuos. Desde la contaminación del suelo y subsuelo que termina contaminando las aguas subterráneas, hasta el más simple que es el uso cada vez mayor de espacio para su contención, incluso llegando a tener islas hechas de basura que afectan la vida marina impactando directamente a los animales que posteriormente serán consumidos por las personas.

Es un problema realmente grave y es donde nace la idea de este proyecto de grado, cualquier esfuerzo por ayudar a la situación es bien recibido y en este caso se centra en los sitios de disposición final de la basura. Se plantea un sistema que tenga la capacidad de detectar envases de tipo botella en la basura conglomerada, para ser separada y debidamente tratada. El objetivo de este proyecto de grado es solamente la detección del envase, para esto se utiliza una banda transportadora que lleva desde un punto de origen un cúmulo de basura y es transportada hasta el punto de la adquisición de la imagen, después se procesa la imagen y cómo resultado final la detección de los envases que puedan estar contenidos en el cúmulo de basura.

Para la creación del sistema de detección se utilizó la técnica de visión por computadora, esta técnica requiere una base de datos con la cual se entrenan los modelos de prueba. En este caso se crearon tres modelos: un modelo basado en la técnica de clasificador de imágenes con búsqueda selectiva, para la delimitación de las cajas contenedoras, y 2 de los mejores modelos en la actualidad los cuales son SSD y YOLOv5. Ya teniendo los modelos de detección se elige uno, seguido se pasa al diseño e implementación de un algoritmo que describe el funcionamiento del sistema y por último se pasa a hacer las pruebas finales del modelo sobre la banda transportadora.

1.1. Planteamiento del Problema

Actualmente en la sociedad es de suma importancia la manipulación de residuos y desperdicios humanos para la preservación del ecosistema terrestre [1], sin embargo, dicha manipulación no se ha realizado de forma adecuada a lo largo de la historia, y el manejo inadecuado que se ha dado a estos materiales ha ocasionado que se generen concentraciones de basura de gran tamaño tanto en el ambiente oceánico como en el terrestre, provocando un problema de talla mundial [2]. En estos tiempos, donde todas las decisiones que ha tomado el ser humano comienzan a repercutir contra el ambiente es menester tomar acción para la recuperación oportuna de los ecosistemas [3].

Según la AIDIS (Asociación Interamericana de Ingeniería Sanitaria y Ambiental), para el año 2016 los países latinoamericanos continúan implementando el esquema de disposición final de la basura, dejando a un lado el reciclaje y tratamiento de residuos [4]. En muchos países siguen utilizando vertederos o botaderos de basura a cielo abierto sin las especificaciones técnicas necesarias, siguiendo la práctica de recolección sin clasificación.

Debido a la problemática planteada, el dispositivo de identificación debe recolectar la información necesaria para la búsqueda de los objetos que cumplan con las características físicas deseadas, pero la toma de esta información presenta un problema: los residuos están apilados y presentan deformaciones de su estructura original. Esta situación implica que la base de datos con la que se entrenará el modelo de aprendizaje de máquina debe contener un gran número de ejemplares de muestra de los objetos a analizar, en su forma original y deformados, lo cual conlleva un trabajo arduo de recolección de datos.

¿Qué técnicas de visión artificial y aprendizaje de máquina, en conjunto, permitirían la clasificación de envases en la basura?

1.2. Objetivos

1.2.1. Objetivo General

Diseñar e implementar un sistema de detección de envases en grupos de residuos sólidos, a través de la adquisición y el procesamiento de imágenes por parte de un sistema embebido, con el fin de generar un aporte y un insumo en las tecnologías enfocadas al reciclaje.

1.2.2. Objetivos Específicos

- Hacer un registro de imágenes enfocado en envases de tipo botellas, botellones y frascos compuestos de vidrio y plástico en residuos sólidos para crear una base de datos.
- Comparar un mínimo de 3 métodos de visión artificial y clasificación de imágenes a partir del estado del arte, con el fin de seleccionar el qué mayor precisión ofrezca.
- Diseñar un algoritmo con base en el método seleccionado el cual basará sus parámetros en las imágenes de la base de datos.
- Implementar el método seleccionado un el sistema embebido, y evaluar su desempeño a través de la medición de la precisión en la detección de basura conglomerada y compuesta por diferentes residuos sólidos sobre la banda transportadora.

1.3. Justificación

El proyecto presenta la oportunidad de mejorar la calidad de vida de muchas personas en muchos lugares del país, que viven en situaciones no recomendadas, que son un lugar idóneo para la proliferación de distintas enfermedades u otros tipos de problemas. Además, los materiales reutilizables se podrían aplicar a esas mismas zonas dando algún beneficio o aporte social.

Para implementar un prototipo con las características deseadas, este proyecto plantea el uso de algoritmos de visión artificial y aprendizaje de máquina, con el fin de evaluar su eficiencia en el reconocimiento de los patrones visuales que caracterizan a los envases. Los métodos basados en visión artificial permiten medir las características de los elementos de una forma no invasiva, lo cual presenta además la facilidad de la implementación del prototipo, y su fácil uso en diferentes escenarios.

Además de lo expuesto anteriormente, los algoritmos de visión artificial utilizados en la actualidad, basados en redes neuronales, han demostrado presentar mejores resultados en tareas de clasificación que otros métodos [5]. Por esta razón, el autor ha optado por partir de este tipo de técnicas para la implementación del prototipo en cuestión.

La información será procesada buscando colores, bordes, formas y tamaño de los objetos que se encuentran agrupados y deformados. Debido a que en la industria la visión artificial está siendo utilizada para la revisión de calidad de los productos a una velocidad superior a la de los humanos [6], con lo que es el sistema ideal a implementar, ya que cuenta con la habilidad de caracterizar formas en grupos de objetos condensados y lo hace de una manera bastante eficaz.

El prototipo tendrá diferentes partes, cada una se encarga de un proceso específico, una banda transportadora que desplazará los residuos a través del campo de visión de una cámara, que se conectará a un sistema de cómputo que realizará el procesamiento respectivo.

1.4. Impacto Social

El área de interés al cual se pretende impactar directamente es el medio ambiente, ya que mejorando las condiciones ambientales de las zonas que han sido utilizadas como vertederos de basura se impacta indirectamente en las zonas aledañas. Según el artículo *Impacto de un botadero a cielo abierto en el desarrollo de síntomas respiratorios y en costos familiares de atención en salud de niños entre 1 y 5 años en Cali, Colombia*[7], las poblaciones que se encuentran viviendo en las zonas alrededor de los vertederos de basura tiene mayor probabilidades de sufrir enfermedades. mayormente respiratorias. y un aumento de "*los costos familiares relacionados con la atención de los síntomas*"[7]. La contaminación causada por el largo periodo de degradación de los plásticos y por el enfoque de este proyecto, la degradación de los envases plásticos, conlleva una contaminación debido a la liberación de sustancias tóxicas al suelo que acaban por filtrarse al subsuelo contaminando las aguas subterráneas y los nutrientes del mismo[8]. A diferencia del plástico el vidrio es completamente reutilizable pero su reutilización se ve obstruida por el peligro que representa recolectarlos, al ser residuos potencialmente cortopunzantes, un corte en estas condiciones de salubridad representa un gran riesgo para la salud del personal de la planta.
Un sistema de visión artificial completamente funcional integrado en un sistema automatizado de detección, localización y obtención de envases de plástico y vidrio, ubicado en el relleno sanitario de Doña Juana, tiene el potencial de reducir la contaminación del suelo ayudando a mitigar los problemas clínicos relacionados con los vertedores en las poblaciones aledañas. Esto se logra al poder juntar los envases de plástico para su correcta disposición. Además, la técnica de coprocesamiento nos brinda la posibilidad de recuperar las materias primas y también alimentar con energía alterna otros procesos productivos[9], utilizando las botellas de plástico que no pueden ser reutilizadas por su estado ya sea por estar dañadas o contaminadas. También ayuda a reducir el riesgo de sufrir lesiones relacionadas con la obtención de las botellas de vidrio al personal de la planta.

2. Estado del Arte

Diana Luz Pachón Espinel (2018) en su proyecto de grado *Prototipo de sistema automatizado con visión artificial para la selección de empaques de plástico, vidrio y lata en el proceso de reciclaje* [10], realiza un prototipo de clasificador de empaques por medio de visión artificial, comienza con un estudio de los casos y su fiabilidad, realiza la documentación completa sobre el sistema informático a implementar, realiza el entrenamiento de una de la inteligencia artificial basándose en el principio de clasificación y la data que utiliza son imágenes de envases sin basura u objetos diferentes a los que se quiere clasificar. Los componentes de hardware que utiliza en el proyecto es un Arduino para el control de una banda transportadora, una cámara web, un sensor de ultrasonido para detectar objetos y crea un dispositivo que utiliza un motor el cual según lo que indique el sistema girará dejando una abertura para separar los diferentes envases. En el apartado de Software utiliza la Matlab para el entrenamiento de la inteligencia y para crear la interfaz gráfica que se comunica con el Arduino para la toma de decisiones.

Daniel Estévez Trigo (2021) en *Detección de objetos en imágenes utilizando técnicas de aprendizaje profundo (Deep-Learning)* [11] realizó la detección y seguimiento de los jugadores de fútbol por medio de técnicas de Deep-Learning aplicadas a un video. Para este propósito utilizó la biblioteca de ImageIA la cual permite utilizar diferentes modelos de detección de objetos como RetinaNet, YOLOv3 y TinyYOLOv3 entrenados bajo los conjuntos de datos ImageNet-1000 que cuenta con 1000 categorías de objetos en la cual esta ‘persona’ y conjunto de datos COCO.
Comienza realizando pruebas con imágenes, estas pruebas se basan en variar el modelo utilizado y variar la velocidad de detección, concluye de estas pruebas que utilizando los parámetros de detección más lentos conseguía mejores resultados. En segundo lugar, hace pruebas en videos, para esto se apoya en las herramientas que proporciona ImageIA para la detección en video y lo que hace es realizar pruebas igualmente, estas pruebas se basan en cambiar parámetros al video como por ejemplo los fps o la propia resolución de este, concluye que los modelos funcionan mejor utilizando una mayor resolución y con menor fps, también concluye que para esta tarea que es procesar imágenes a alta velocidad lo mejor es usar YOLO y ResNet.

Natalia Bonifaz Oviedo y Laura Juliana Padilla Reyes (2021) en *Desarrollo de un sistema de recomendación inteligente para selección de prendas utilizando análisis de imágenes con redes neuronales* [12].Desarrollaron un sistema de visión artificial que recomienda prendas de vestir el cual se centra en la interacción hombre máquina para reconocer las preferencias del usuario, además de una constante actualización del mismo para proporcionar mejores recomendaciones y evolucionar junto con los gustos del usuario. El sistema de visión artificial se basa en la clasificación de imágenes por medio de una red CNN con una arquitectura RetNet50 para la clasificación de 7 tipos de clases entrenada con 14000 imágenes, cada clase cuenta con 2000 imágenes de acceso público. Por último el

sistema presenta las propuestas por medio de un tablero dividido en 3 regiones; parte alta para las prendas de torso, media para la zona de los piernas y baja para los zapatos y todas las prendas de los pies.

Alexander Bermudez Castañeda (2021) en *Implementación de un sistema para la detección de peligros potenciales para motociclistas basado en información visual* [13], Utilizó la técnica de detección de objetos para la detección de posibles riesgos al conducir una motocicleta, el sistema fue implementado en 2 tarjeta de desarrollo Dev Board y Jetson Nano. Además el sistema es capaz de obtener la distancia desde el sistema de captación de la imagen y el objeto detectado que junto al APi de google “Text to speech” le envían la información al usuario por medio de audio. La red entrenada fue la mobileNetV2 con datos tomados a partir de videos captados por una cámara colocada sobre el casco del usuario.

Jhonatan David Giraldo Quiñones (2022) *Clasificador de residuos sólidos haciendo uso de deep learning* [14], trabajó bajo la idea de un clasificador de residuo sin embargo el rango de objetos en clasificar es mayor, un total de 12 categorías. Pero este sistema no cuenta con un prototipo más allá de la inteligencia artificial y los sistemas embebidos y lo que se busca es comparar el rendimiento del clasificador en 3 diferentes sistemas embebidos NVIDIA jetson nano, Dev Board Coral y RASPBERRY PI 4B.
En el análisis de resultados se evidencia una diferencia en el rendimiento de los dispositivos y es de utilidad el notar que el rendimiento sobre el sistema embebido de una Raspberry Pi es sustancialmente menor, algo que es de esperar ya que a diferencia los otros sistemas tienen un hardware más potente y diseñado específicamente para el uso en machine learning, pero da luces de cómo esto afecta el funcionamiento del proyecto.

3. Marco Teórico

En este capítulo se expondrán las bases teóricas de la detección de objetos centrándose en la explicación de los procesos más que a la matemática detrás de estos, esto se hace para facilitar el entendimiento de la metodología utilizada.

3.1. Redes neuronales artificiales

Una red neuronal es un sistema de aprendizaje por refuerzo el cual se centra en el procesamiento de datos enfocado en aprender características útiles que sean necesarias para una toma de decisiones lo más cercanas posibles a los datos con los que se alimenta el sistema. La red neuronal está inspirada en las neuronas orgánicas y se conforma principalmente por nodos los cuales simulan las neuronas y que al conectarse entre ellas generan la red neuronal.

3.2. Neurona artificial

La neurona artificial es la unidad básica que conforma las redes neuronales artificiales y se componen de 4 elementos claves; las entradas que funcionan como mecanismo para enviar información entre neuronas y alimentar la red, los pesos que son multiplicadores asignados a cada entrada y que son el elemento en los cuales se hace principalmente el aprendizaje y define la importancia relativa de cada entrada [15], un peso sin entrada que funciona como un umbral de activación o acción y la función de activación la cual determina el valor de salida de la neurona.

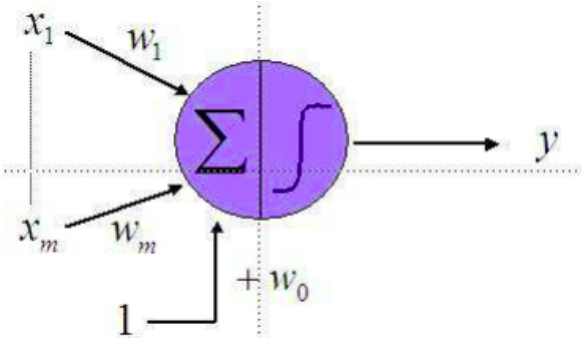

Figura 3.1. Estructura básica de una neurona artificial
Tomado de Miranda, R. M. (s.f.). Redes neuronales artificiales [Imagen]. Academia.edu - Share research. https://www.academia.edu/29894308/Redes_Neuronales_Artificiales

En la figura 3.1 se observa la estructura de la neurona artificial donde X_0 a X_n es el vector con las entradas a la neurona artificial, de W_0 a W_n son los pesos asociados a cada entrada, W_0 es el umbral de activación o sesgo y Y es la salida representada por la aplicación de la función de activación ϒ a la suma de todas la

entradas por su peso y el umbral de activación. La expresión que representa la salida de la neurona artificial se encuentra en la fórmula 3.2.1.

$$Y = \gamma((\sum_{i=1}^{m} Xi * W) + W0) \quad (3.2.1)$$

3.3. Función de activación

Son funciones matemáticas aplicadas al final de la neurona artificial y tiene como función permitir que la red neuronal resuelva problemas más complejos dando a la red la capacidad de resolver problemas no lineales. Dos de las funciones de activación más utilizadas son relu y sigmoid, la función relu da una salida 0 si la entrada es negativa y si la entrada es positiva la salida es la misma entrada. Sigmoid es otra función de activación muy utilizada mayormente cuando la salida que se desea represente un grado de probabilidad, la fórmula matemática 3.3.1 representa la función sigmoid.

$$f(x) = \frac{1}{1-e^{-x}} \quad (3.3.1)$$

3.4. Capas

Cuando las neuronas se conectan entre sí se forman capas, la capa inmediata a la entrada se conoce como capa de entrada, las capas posteriores antes de la salida se conocen como capas ocultas y la capa de salida varía dependiendo de la utilidad que se le está dando a la red neuronal. Como ejemplo aplicado al proyecto, se podrían tener 2 salidas con una función de activación sigmoid en la que una salida representaría la probabilidad de que sea una botella y la segunda salida la probabilidad de que no sea una botella, en la figura 3.2 se observan las diferentes capas de una red neuronal.

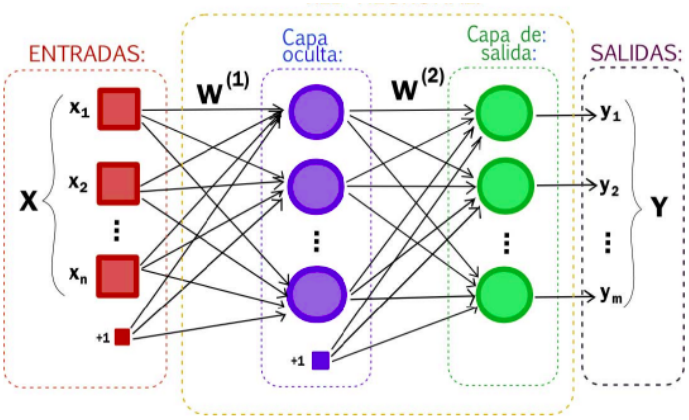

Figura 3.2. Capas en una red neuronal
Tomado de Bonilla Carrión, C. (2020). Redes convolucionales [Imagen]. https://idus.us.es/bitstream/handle/11441/115221/TFG%20DGMyE%20Bonilla%20Carrión,%20Carmelo.pdf?sequence=1&isAllowed=y

3.5. Entrenamiento

La red neuronal se puede ver como una red de capas donde cada capa consta de algunas variables internas y matemáticamente definidas. El proceso de entrenamiento se trata de ajustar las variables internas de las redes a los mejores valores posibles, esto se logra a través de un proceso de optimización llamado Gradient Descent(descenso de gradiente) que usa análisis numéricos para encontrar los mejores valores posibles para las N variables internas, que son los pesos explicados anteriormente W_0 y W_n, de cada capa del modelo.

El proceso es el siguiente; Se aplican los valores en las entradas de la red, estas van transformándose a medida que pasan a través de las capas, una vez se predice el valor de salida se realiza la diferencia entre el valor predicho y el valor real. Esta diferencia se conoce como pérdida, con la pérdida calculada se ajustan los valores internos (pesos y sesgos) de todas las capas de la red. El descenso de gradiente ajusta iterativamente los parámetros, llevándolos en la dirección correcta poco a poco hasta que alcance los mejores valores. La función que mide que tan bueno o malo es el modelo durante cada iteración se llama función de pérdida(loss function) y el objeto de cada iteración es disminuir la función de pérdida.

3.6. Descenso de gradiente

El descenso de gradiente es el proceso que optimiza los valores de los parámetros internos de la red, es un proceso básico de optimización en donde se hallan los mínimos en un hiperplano, como en la optimización normal esto se hace a partir de las derivadas parciales de la función.

$$\text{Función de pérdida=J(x)=}\frac{1}{m}\sum_{i=1}^{m} y^i Ln(hx(x^i)) + (1 - y^i)ln(1 - hx(x^i)) \qquad (3.6.1)$$

En esta función de pérdida, fórmula 3.6.1, hx(x^i) es el predicción de la muestra i y y^ies el valor verdadero de la muestra. Después de algunos análisis matemáticos el descenso de gradiente se representa como se ve en la fórmula 3.6.2.

$$\frac{dJ(w)}{dwi} = \frac{1}{m}\sum_{j=1}^{m}(hw(x^j) - y^j)x_i^j \qquad (3.6.2)$$

La fórmula que optimiza los pesos a partir de las derivadas parciales de la función de pérdida para los pesos y los sesgos son las fórmulas 3.6.3 y 3.6.4.

$$W0^{(k)} = W0^{(k-1)} - \alpha\frac{1}{m}\sum_{j=1}^{m}(hw(x^j) - y^j)x_0^j \qquad (3.6.3)$$

$$Wn^{(k)} = Wn^{(k-1)} - \alpha\frac{1}{m}\sum_{j=1}^{m}(hw(x^j) - y^j)x_n^j \qquad (3.6.4)$$

Este algoritmo es el que optimiza los pesos de la red neuronal, en este caso k representa el tiempo actual con los que se va a hacer la predicción inmediata y k-1 es el tiempo anterior y su predicción que se usa para optimizar el valor de W y realizar la predicción en k.

Un término sumamente importante es α que es un valor real positivo y se conoce como velocidad de aprendizaje o learning rate y determina qué tan rápido se hace el descenso de gradiente, este término debe ser ajustado a partir de observaciones de funcionamiento, ya que un learning rate muy grande hace que el valor del parámetro decrezca rápidamente pero como se buscan los mínimos puede que estos cambios sean muy grandes y no alcancen a tomar el valor mínimo. Por otro lado, si el valor es muy pequeño los valores decrecen muy lentamente y debido a que el hiperplano no es una curva perfecta los cambios pueden ser muy pequeños obteniendo un mínimo local, que no es del todo malo pero le puede quitar la posibilidad de salir de ese mínimo local para encontrar un mejor mínimo local o el mínimo absoluto [16].

3.7. Visión artificial

La visión artificial es la ciencia encargada de la programación de algoritmos y técnicas de procesamiento de datos obtenidos de imágenes y videos [17]. Hay 3 ramas principales de la visión artificial; clasificación, detección y segmentación. En la clasificación se busca clasificar la imagen en una o varias clases dependiendo del contenido de esta. En la detección se busca detectar objetos dentro de la imagen, se pueden tener varias clases o solo una y es el eje central de este proyecto. La segmentación busca igualmente detectar objetos dentro de la imagen, pero la diferencia es que en la detección la localización del objeto se hace en cajas rectangulares y en la segmentación se define el contorno exacto del objeto a buscar.

3.8. Digitalización de la imagen

Las imágenes se representan con matrices, dependiendo del tipo de imagen se tienen la dimensionalidad de la matriz, si la imagen es en blanco y negro solo se tiene una matriz y sus valores representan la intensidad lumínica, 0 si es el color es negro y 255 si es blanco. Si la imagen tiene colores se tienen más de una matriz, una imagen RBG tiene 3 matrices una para cada color: rojo (R), azul (B) y verde (G), sus valores representan la intensidad de color y el color del pixel que se en la pantalla es una combinación de las 3 intensidades.

3.9. Convolución

Los sistemas de procesamiento de imágenes se basan en la operación matemática de convolución, a partir de una matriz (3x3 o 5x5 píxeles en la mayoría de los casos) la cual es llamada kernel de convolución [18], en la figura 3.3. Se tiene una imagen que es la matriz de (5x5), el kernel de (3x3), el kernel realiza el recorrido por toda la imagen y el resultado de la convolución es la última matriz.

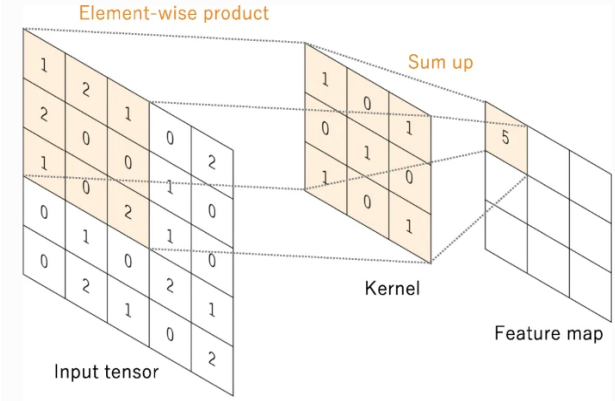

Figura 3.3. Proceso de convolución
Tomado de Convolutional neural networks: An overview and application in radiology - Insights into Imaging [Imagen]. (s.f.). SpringerOpen. https://insightsimaging.springeropen.com/articles/10.1007/s13244-018-0639-9

3.10. CNN (Convolutional Neural Network)

Es una red neuronal que utiliza la convolución para hacer el procesamiento de imágenes y extracción de las características útiles que deberá aprender la red para cumplir con el objetivo asignado, los pesos a entrenar se encuentran en los kernel o filtros y es en estos en los que se tienen los parámetros a optimizar. Los filtros utilizan el proceso de descenso de gradiente para minimizar la pérdida que junto con las técnicas de pooling y normalización amplifican las características que ayudan a mejorar la red, a medida que la imagen va avanzando por la red y es procesada se va reduciendo su tamaño pero las características que se buscan se van haciendo más grandes en proporción al tamaño de la imagen. En la figura 3.4 se tiene una red CNN completamente estructurada en la se observa cómo se va reduciendo el tamaño de la imagen y la extracción de las características.

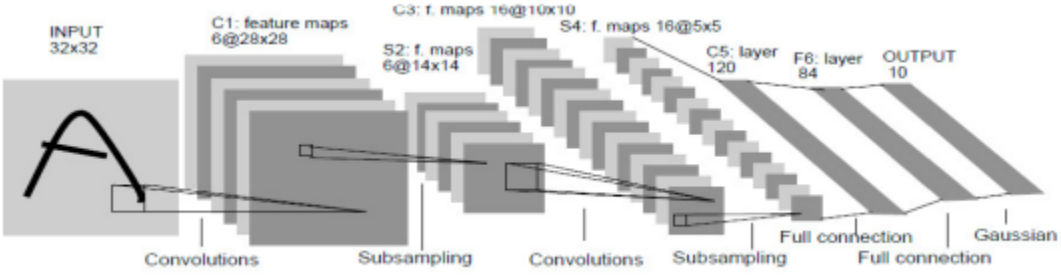

Figura 3.4. CNN
Tomado de Indolia, S., Kumar Goswami, A., & Mishrab, S. P. (s.f.). Conceptual understanding of convolutional neural network- A deep learning approach | semantic scholar [Imagen]. Semantic Scholar | AI-Powered Research Tool. https://www.semanticscholar.org/paper/Conceptual-Understanding-of-Convolutional-Neural-A-Indolia-Goswami/f1e5a1f1334a447a105ee8befba6d0dbea9c73ea

Los filtros se agrupan en capas y la estructura que se utiliza es aplicar una capa de filtros, seguido de una capa de agrupación (pooling), la función de activación y por último una capa de normalización, esto se repite hasta terminar con el tamaño de imagen deseado. Por último normalmente se hace el aplanamiento de la última capa de filtros utilizando la función flatten seguido de un n número de capas completamente conectadas (fully connected), las salidas dependen del objetivo de la red, si es clasificación tienen el mismo número de clases en la salida y si es detección se pueden tener 4 salidas que forman el rectángulo de localización del objeto.

3.11. Bloque identidad y bloque convolucional

Son bloques que tienen una estructura de capas definida, normalmente las capas se agrupan de la siguiente manera; capa convolucional, capa de normalización, función de activación, esto se puede repetir pero termina con una capa de convolución y una capa de normalización. Lo importante de estos bloques es que tiene un puente (shortcut), el cual se puede ver en la figura 3.5, desde el inicio del bloque hasta el final del bloque. La salida del bloque y el puente se conectan al final por medio de una suma. Si el puente es simple el bloque es de identidad pero si el puente tiene una capa convolucional es un bloque convolucional. Estos bloques son útiles para llevar información poco procesada a través de la red.

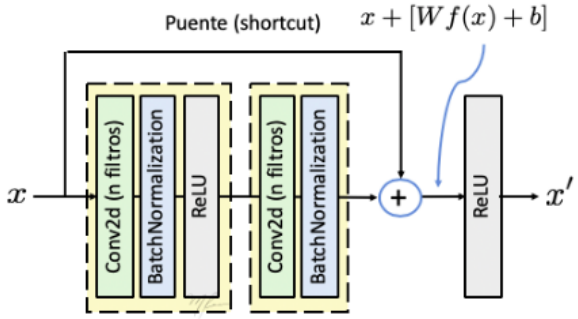

Figura 3.5. Bloque identidad
Tomado de Resnet.md [Imagen]. (s.f.). http://personal.cimat.mx:8181/~mrivera/cursos/aprendizaje_profundo/resnet/resnet.html

3.12. Modelos

Una característica importante de deep learning es que se pueden utilizar las estructuras de red creadas por profesionales del campo, además no solo la estructura se puede utilizar, con la técnica de transfer learning se descargan modelos completamente entrenados con bases de datos extensas de hasta mil clases.
Hay 3 formas de utilizar la técnica de transfer learning; la primera es usar los modelos con las clases que se entrenaron, la segunda es utilizar las capas convolucionales y los bloques quitando la cabecera final y agregando una cabecera personalizada, la cabecera personalizada será la que se entrena, de

esta manera se mantienen los mapas de características aprendidas por la red, y por último se entrena toda la red completa sobre los pesos y sesgos anteriores, esta última forma tiene el problema de que el error se va a propagar por todo la red perdiendo los mapas de características anteriores.

3.13. SSD

En una red convolucional que produce una serie de cuadros delimitadores de tamaño fijo para la presencia de instancias para realizar inferencias [19]. En este modelo parte de la propia red hace inferencias de la posición de los objetos a buscar, además la red crea regiones en donde se dan más detecciones de los objetos, en la figura 3.6 se observan 2 de estas regiones de inferencia. En este modelo se utiliza una red VGG16, que tiene una estructura de capas definida, seguida de capas convolucionales pero estas capas convolucionales tienen las 2 funcionalidades antes dichas, generan regiones de interés y hacen detección en cada capa. Al final todas las predicciones se unen en la última capa y se hace supresión no máxima (NMS) que es una función para suprimir todas las detecciones que tengan un IOU, que se define como el área de intersección entre 2 cuadros delimitadores sobre el área total de los cuadros delimitadores, superior a un valor dado dejando solo la caja con mayor probabilidad.

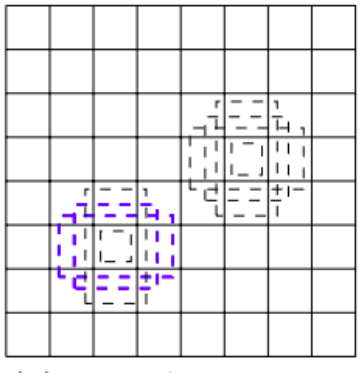

Figura 3.6. Regiones de mapa de características
Tomado de Wei, L., & Dragomir, A. (s.f.). SSD: Single shot multibox detector [Imagen]. arXiv.org. https://arxiv.org/abs/1512.02325

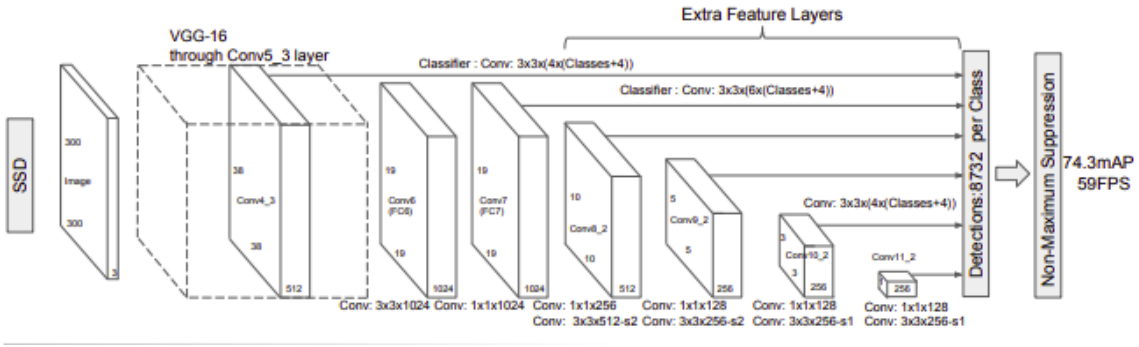

Figura 3.7. Estructura red SSD
Tomado de Wei, L., & Dragomir, A. (s.f.). SSD: Single shot multibox detector [Imagen]. arXiv.org. https://arxiv.org/abs/1512.02325

En la figura 3.7 se encuentra la estructura de la red SSD, en esta se puede ver cómo se conectan todas las capas intermedias de la red al final de esta, estas conexiones hacen referencia a las inferencias que se realizan en cada capa en la que al final se hace NMS para obtener los cuadros delimitadores finales.

3.14. YOLO

Es una red neuronal convolucional que divide la imagen en una serie de cuadros que realizan la detección y al mismo tiempo proporcionan la probabilidad individualmente del tipo de clase detectada, en la figura 3.8 se observan los resultados de la red. La respuesta es un vector en el que cada sector tiene una finalidad, el primer término indica si hay un objeto dentro de la región, se sigue con 4 términos que representan la posición central de la caja en X , Y , el ancho y alto, y termina con las probabilidades de que el objeto pertenezca a alguna clase.

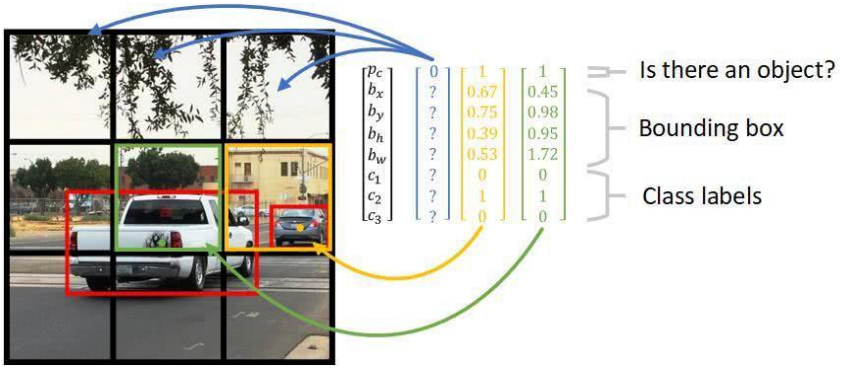

Figura 3.8. Bounding Box de la estructura YOLO
Tomado de Redmon, J., & Divvala, S. (s.f.). You only look once: Unified, real-time object detection [Imagen]. Page Redirection. https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Redmon_You_Only_Look_CVPR_2016_paper.pdf

La arquitectura de la red YOLOV5 está formada por 24 capas convolucionales y 2 capas completamente conectadas. También se utilizan las capas convolucionales de 1x1 las cuales permiten reducir la profundidad de los mapas de características

y se finaliza con un tensor de salida con dimensiones (7,7,30) [20]. En la figura 3.9 se puede ver la estructura de la red YOLO junto a los cuadros de inferencia.

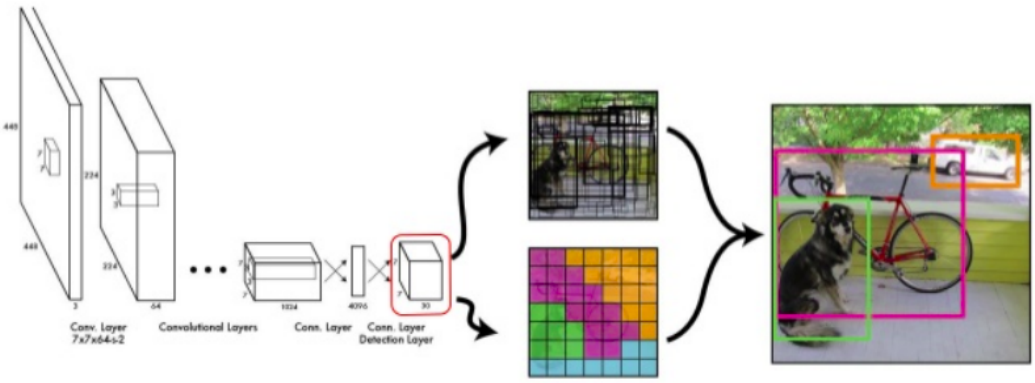

Figura 3.9. Estructura de la red YOLO
Tomado de Redmon, J., & Divvala, S. (s.f.). You only look once: Unified, real-time object detection [Imagen]. Page Redirection. https://www.cv-foundation.org/openaccess/content_cvpr_2016/papers/Redmon_You_Only_Look_CVPR_2016_paper.pdf

3.15. Software

Para hacer el entrenamiento de los modelos se utilizó Google Colab una aplicación web que brinda acceso limitado a tarjetas gráficas que son esenciales para hacer machine learning por la cantidad de datos a procesar y de cálculos a realizar. El lenguaje de programación utilizado fue Python el cual brinda una larga lista de herramientas y librerías completamente dispuestas para el aprendizaje de máquina como Tenserflow, Keras y Pytorch.

3.16. Evaluación del modelo

Para hacer el análisis de resultados de los modelos de detección se utilizan 4 etiquetas otorgadas a las detecciones realizadas por el modelo.

TP = Verdadero positivo , detecciones realizadas que corresponden correctamente al objeto buscado, la detección es verdadera cuando su IOU >= 0.5.
FP = Falso positivo, cuando no se realiza una detección del objeto buscado.
TN = Verdadero negativo, realiza una detección falsa que no corresponde al objeto buscado
FN = Falso negativo, relacionada a la clasificación binaria.

$$precisión \ = \frac{TP}{TP + Fp} \tag{3.16.1}$$

La fórmula 3.16.1 relaciona el número de detección falsas con el número de detecciones correctas.

$$exactitud = \frac{TP+FN}{TP+FN+FP+TN} \quad (3.16.2)$$

La fórmula 3.16.2 relaciona todos los aspectos de la detección tanto las detecciones falsas como las no detecciones de los objetos que sí están en la imagen. Al ser la mejor manera de evaluar el rendimiento del modelo se optó por tomar esta métrica a la hora de la elección del modelo.

4. Diseño Metodológico

Se plantea una metodología lineal centrada en cumplir los objetivos específicos ya que estos están en el orden correcto en el que se deben cumplir las tareas para lograr la culminación de estos.
En la figura 4.1 se muestran las 5 etapas elegidas para el proyecto; creación de la base de datos, entrenamiento y elección del modelo, creación del hardware, diseño del algoritmo y por último pruebas sobre la banda transportadora utilizando el sistema embebido.

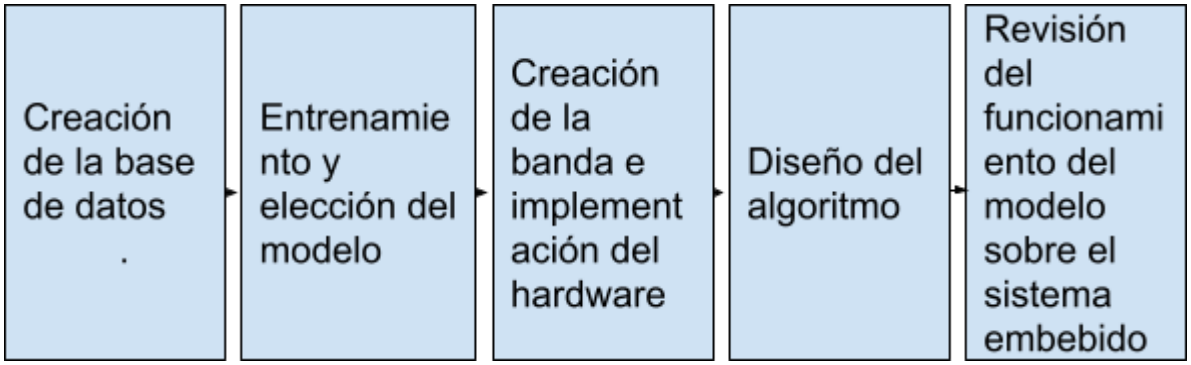

Figura 4.1 Etapas del proyector

4.1. Fases y Metodología

4.1.1. Creación de la base de datos

En este apartado se crean las bases de datos con las imágenes que se van a utilizar para el entrenamiento de los modelos. Partiendo de la recolección de las imágenes, la clasificación y por último la división de estas para formar los conjuntos de imágenes de entrenamiento, validación y prueba.

4.1.2. Entrenamiento y elección del modelo

Se entrenan los modelos utilizando las imágenes de las bases de datos creadas y se revisa su comportamiento. Para esto se utilizan los modelos haciendo predicciones en imágenes que el modelo no ha visto, es decir las imágenes del conjunto de prueba, y se mide su exactitud. Después de esto se elige el modelo con mayor exactitud.

4.1.3. Creación de la banda e implementación del hardware

Se crea la banda transportadora implementando los diferentes dispositivos electrónicos que permitan controlarla, los sensores que se utilizaran y además de la implementación del equipo para la captación de la imagen, es decir una cámara.

4.1.4. Diseño del algoritmo

Creación del algoritmo definiendo el comportamiento del sistema.

4.1.5. Revisión del funcionamiento en el sistema embebido

Por último el modelo seleccionado se ejecuta sobre el sistema embebido para revisar su funcionamiento, además utilizando el algoritmo creado se implementa la funcionalidad base planteada para sobre la banda transportadora. En este paso se realizan los cambios necesarios al modelo partiendo de las pruebas realizadas sobre el sistema embebido.

5. Desarrollo Conceptual

5.1 Creación de la base de datos

Para el entrenamiento de los modelos de visión artificial se requiere tener un conjunto de datos que los modelos utilizan para crear los mapas de características. Para el correcto funcionamiento de los modelos los datos deben ser tomados en el ambiente en donde se desempeñará el modelo o en lo posible de datos lo más parecidos a estos, con esto en mente lo idóneo hubiese sido tomar los datos en un botadero o relleno sanitario pero por motivos de seguridad no pudo ser obtenido el permiso para ingresar en las instalaciones del relleno sanitario de Doña Juana, entonces se optó por tomar fotos sobre la banda con basura cotidiana intentando simular un estado de aglomeración, se terminó con un total de 198 imágenes y para reforzar se creó una segunda base de datos. Esta segunda base de datos fue creada utilizando el buscador de Google y los códigos de uso libre proporcionados por PYIMAGESEARCH en el artículo "*How to create a deep learning dataset using Google Images*" [21], los cuales descargan imágenes directo del buscador y eliminar las duplicadas, esto es esencial para que el modelo no aprenda características repetidas, posteriormente son etiquetadas, que esencialmente es ubicar por medio de cajas delimitadoras la ubicación del objeto a detectar. La plataforma de Roboflow permite el etiquetado gratis y la exportación a diferentes formatos sólo dejando que los datos sean de acceso libre. En las figuras 5.1 y 5.2 se tienen las imágenes etiquetadas obtenidas de Google y tomadas sobre la banda respectivamente.

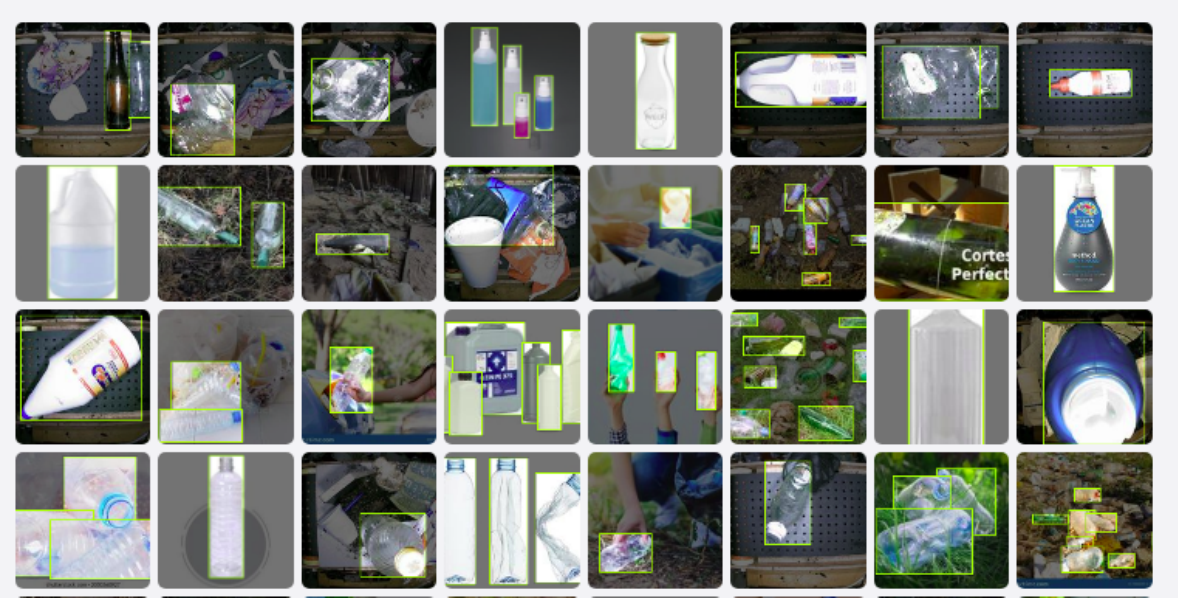

Figura 5.1. Imágenes Google etiquetadas en Roboflow

Figura 5.2. Imágenes sobre la banda etiquetadas

5.2 Entrenamiento y elección del modelo

Se trabajó con 3 modelos; RCNN clasificador con búsqueda selectiva, SSD y YOLOv5. El patrón de entrenamiento para los modelos fue el mismo; 2 modelos entrenados, el primero con las 2 bases de datos juntas y el segundo con un entrenamiento con la base de datos de las imágenes extraídas de Google y después reentrenar con las imágenes que se tomaron sobre la banda transportadora.

5.2.1 RCNN conjunto de datos combinados

El modelo de RCNN clasificador con búsqueda selectiva fue creado en su totalidad siguiendo códigos de acceso libre. El procedimiento para su entrenamiento fue el siguiente.

1. Las imágenes de la base de datos se dividen en dos ficheros, botella y sin botella, el primero contiene las partes de las imágenes originales que contienen el objeto que se busca y el segundo fichero contiene imágenes con las partes de las imágenes originales que no contienen los objetos que se buscan, esto se hace con la función de búsqueda selectiva que busca objetos dentro de la imagen que se le envía, devolviendo un conjunto de puntos que representan las regiones propuestas.

```
ss = cv2.ximgproc.segmentation.createSelectiveSearchSegmentation()
ss.setBaseImage(img)
#ss.switchToSelectiveSearchFast()
ss.switchToSelectiveSearchQuality()
rects = ss.process()
```

Figura 5.3. Búsqueda selectiva
Figura: Autor

En la figura 5.3 se realiza el proceso de búsqueda selectiva. Hay 2 formas de hacer búsqueda selectiva, la rápida y la de calidad, en este caso se optó por realizar la búsqueda de calidad la cual da mejores resultados y un mayor número de propuestas. Esto se hace creando un objeto, ss, de búsqueda selectiva proporcionado por la librería opencv(cv2), se carga la imagen en el objeto ss, se selecciona el tipo de búsqueda, en este caso Quality para la búsqueda de calidad, y se guarda la inferencia de las propuestas llamando al método process() del objeto ss. Lo que sigue es buscar en estas regiones propuestas las imágenes que tienen las características necesarias para entrar en el conjunto de imágenes con botella o sin botella comparándolas con las regiones de los cuadros delimitadores(Bounding Box) que contienen las imágenes etiquetadas. Las características para entrar en la base de datos de botella son: la región debe tener un IOU > 1.3 y IOU < 0.7 y que el cociente de las coordenadas sea > 0.9 y < 1.1, con esta última comparación lo que se busca es que la región propuesta está en la misma zona y sea casi igual de grande que la etiqueta, este proceso se ve en la figura 5.4.

```
if(con[i]==0):
  (gtstartx,gtstarty,gtendx,gtendy) = gtbox
  if((propstartx/gtstartx >= 0.9 and propstartx/gtstartx <= 1.1)
    sirve = True
    con[i]==1
```

Figura 5.4. Comparación de coordenadas

Los cuadros delimitadores de la etiqueta se encuentran en gtbox y los cuadros delimitadores de las propuestas son prop y su respectiva coordenada, para el caso de la primera coordenada en X y Y se tiene propstartx y propstarty respectivamente. Se realiza la comparación entre las cajas y si cumple con los requerimientos la variable sirve se hace true y se aumenta el contador, el contador sirve para tener un control sobre el número de propuestas por etiqueta, esto se hace para que no haya varias propuestas que tengan la misma imagen de la etiqueta evitando tener datos repetidos, en este caso se optó por tener una propuesta por etiqueta. A las regiones que sirven se les asigna un nombre, se hace el recorte de la región propuesta en la imagen original y se guarda en el fichero de botella.
Las características para entrar en la base de datos de sin botella son; tener un IOU < 0.1 y una serie de comparaciones para dejar fuera las regiones propuestas que contengan dentro un poco de los cuadros delimitadores de las etiquetas como se ve en la figura 5.5.

```
vale = True
for gtboxx in gtboxes:
  (gtstartx2,gtstarty2,gtendx2,gtendy2) = gtboxx
  if((((gtstartx2>= propstartx) and (gtstartx2<=propendx)) or((gtendx2>=propstartx)
    vale = False
    break

if iou < 0.1 and negativeROIs<config.MAX_NEGATIVE and ara>0.1 and ara< 4 and vale:
```

Figura 5.5. Comparación de coordenadas para regiones sin botella

Se realiza la comparación de todas las etiquetas gtbox con las regiones propuestas, lo que se busca es que ningún cuadro delimitador de la etiqueta esté dentro de la propuesta, para esto se hacen una serie de comparaciones y si alguna comparación resulta verdadera la propuesta es marcada utilizando la variable vale como false y se rompe el ciclo for, si la propuesta es marcada como false no se prosigue a darle un nombre ni a hacer su respectivo recorte en la imagen original. A las regiones con las características necesarias se les asigna un nombre, se hace el recorte de la región propuesta en la imagen original y se guarda en el fichero de sin botella.

2. Compilación del modelo y entrenamiento. En esta parte se utiliza uno de los conceptos más importante del aprendizaje de máquina; transferencia de aprendizaje o Transfer learning. Se elige la red pre entrenada, en este caso se optó por utilizar MobilenetV2 la cual consta de una profundidad de 50 capas ocultas de bloques convolucionales.

```
from tensorflow.keras.applications import MobileNetV2

basemodel = MobileNetV2(weights="imagenet",include_top=False,
  input_tensor=Input(shape=(640,640,3)))
```

Figura 5.6. Importar y cargar el modelo

En la figura 5.6 se importa y se carga el modelo, al cargar el modelo se le asignan los pesos obtenidos con el entrenamiento de las imágenes y clases dentro del conjunto de datos de “imagenet”, se le retira la cabecera, que será la parte de la red a entrenar y se le especifica el tamaño de la imagen. Posteriormente se crea la cabecera personalizada como se puede ver en la figura 5.7. Se optó por crear 2 bloques convolucionales seguido de la capa de aplanamiento, flatte n, después 2 capas densas o capas completamente conectadas, una capa de Dropout que sirve para apagar secciones de la red durante el entrenamiento para que los mapas de características que se van creando se amplíen a toda la red, lo que es útil para que la red no tenga secciones con mapas de características específicos para detectar ciertos patrones sino que toda la red tenga esas cualidades en mayor o menor medida. Esta

cabecera fue estructurada de esta manera porque la red MobileNet utiliza varias capas de bloques convolucionales, entonces se optó por no dejar una cabecera simple sino crear una con bloques convolucionales.

```
headmodel = AveragePooling2D(pool_size=(7,7))(headmodel)
x = Conv2D(64,(3,3),padding='same')(headmodel)
x = BatchNormalization()(x)
x = Activation('relu')(x)

x = Conv2D(32,(3,3),padding='same')(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)

x = Conv2D(16,(3,3),padding='same')(x)
x = BatchNormalization()(x)

line = Conv2D(16,(1,1),strides=2)(headmodel)
line = BatchNormalization()(line)

xx = add([x,line])
xx = Activation('relu')(xx)
##
x = Conv2D(16,(3,3),padding='same')(xx)
x = BatchNormalization()(x)
x = Activation('relu')(x)

x = Conv2D(16,(3,3),padding='same')(x)
x = BatchNormalization()(x)
x = Activation('relu')(x)

x = Conv2D(8,(3,3),padding='same')(x)
x = BatchNormalization()(x)


line = Conv2D(8,(1,1),strides=2)(xx)
line = BatchNormalization()(line)

xx = add([x,line])
xx = Activation('relu')(xx)
```

```
model = Model(inputs=basemodel.input,outputs=headmodel2)

for layer in basemodel.layers:
  layer.trainable = False
```

Figura 5.7. Cabecera de la red y armado de la red

3. Después de tener la red armada se prosigue a crear el optimizador y compilar el modelo. El optimizador elegido fue SGD, Stochastic Gradient Descent, que es el descenso de gradiente explicado en el marco teórico, la diferencia es que se le asigna un parámetro llamado momentum. Para este parámetro solo se revisaron redes parecidas y se asignó el mismo parámetro de momentum. En la función de pérdida se eligió entropía cruzada binaria (binary_crossentropy) porque la salida es binaria entre 0 y 1, y se

elige la métrica de precisión (accuracy) con una tasa de decaimiento de la tasa de aprendizaje (learning rate) igual a la tasa de aprendizaje dividida en el número de épocas entrenadas, de esta forma a medida que la red se entrena su tasa de aprendizaje decrece, como se aprecia en la figura 5.8 donde init_lr es la tasa de aprendizaje y epochs el número de épocas seleccionadas.

```
decay_rate = init_lr/epochs
momentum =0.5
opt = SGD(lr=init_lr,momentum=momentum,decay=decay_rate,nesterov=False)
model.compile(loss="binary_crossentropy",optimizer=opt,
  metrics=["accuracy"])
```

Figura 5.8. Compilación del modelo

El método compile() compila el modelo, a este se asignan los parámetros: función de pérdida binary_crossentropy , optimizador opt y las métricas que serán guardadas, en este caso precisión.

4. Separar en 4 arreglos las imágenes; trainx que son las imágenes con las que se entrenará la red, trainy que son las etiquetas de las imágenes de entrenamiento, testx imágenes para probar la red en el entrenamiento y testy etiquetas de las imágenes con las que se prueba la red.
5. Por último solo se entrena el modelo y se guardan las métricas obtenidas.

```
h = model.fit(
  aug.flow(trainx,trainy,batch_size=bs),
  steps_per_epoch=len(trainx) //bs,
  validation_data=(testx,testy),
  validation_steps=len(testx) //bs,
  callbacks=[checkpoint,early],
  epochs=epochs
)
```

Figura 5.9. Entrenamiento de la red

En la figura 5.9 se hace el entrenamiento del modelo, se utiliza el método fit del objeto model que fue anteriormente compilado, a esta función se le pasan los arreglos con las imágenes de entrenamiento y de validación, también se le pasa el número de épocas y el tamaño del lote de entrenamiento que representa cuántas imágenes se entrenan por iteración. Además, se le pueden pasar funciones llamadas callbacks las cuales cumplen la función de tomar decisiones al finalizar cada época en base al rendimiento del modelo, en este caso se utilizaron dos callbacks; checkpoint que guardó el modelo cuando tiene mejoras en su rendimiento y early que detiene el entrenamiento cuando el modelo deja de mejorar. Este último es muy importante ya que sirve para evitar el sobreajuste de la red, lo cual empeora el rendimiento de esta, debido a que la red aprende

muy bien a detectar sobre las imágenes de entrenamiento pero al momento de hacer la inferencia con las imágenes de prueba el modelo no es capaz de generalizar lo aprendido.

Al final el mejor resultado dejó un valor de precisión en el entrenamiento superior al 0.9 y un valor de precisión de prueba superior al 0.85 con una tasa de entrenamiento inicial de 1e-3. Algo muy importante es que el modelo debe ser probado con imágenes que nunca haya visto para revisar que efectivamente el modelo generaliza lo aprendido y pueda hacer buenas predicciones.
En la figura 5.10 se tienen las métricas correspondientes a la pérdida y la precisión a lo largo del entrenamiento.

Figura 5.10. Métricas del entrenamiento

Los resultados del primer modelo de RCNN no fueron del todo malos, ya que en la mayoría de las muestras si se detectaba el objeto deseado, el problema radica en la cantidad de detecciones incorrectas que hacen que su exactitud esté por debajo del 50%. En la tabla 5.1 se evidencia que es mayor el número de detecciones falsas que el número de positivos-falsos y en la figura 5.11 la inferencia realizada por el modelo, este resultado no es bueno para implementarlo con el sistema embebido.

Tabla 5.1. Resultados conjunto de datos unidos

Verdadera \ Inferencia	Positivo	Negativo
Positivo	18	19
Negativo	6	0
Exactitud	0,4186046512	

Figura 5.11. Resultado de la red RCNN

5.2.2 RCNN conjunto de datos separados

Ahora, según lo establecido se realizó el entrenamiento de un segundo modelo pero utilizando la base de datos separada. El proceso es el mismo solo que al entrenar el modelo solo se hace con las imágenes obtenidas de Google, el modelo se guarda y posteriormente el modelo se vuelve a entrenar pero con las imágenes sobre la banda y una tasa de aprendizaje menor. Esto hace que los mapas de características obtenidos no sufran cambios muy rápidamente que la hagan perder información y que se aprendan características útiles de la segunda base de imágenes. Las métricas obtenidas del segundo entrenamiento son las figuras 5.12, base de imágenes de Google y 5.13, base de imágenes sobre la banda.

Figura 5.12. Entrenamiento imágenes Google

Figura 5.13. Entrenamiento imágenes sobre banda

Estos resultados fueron obtenidos con una tasa de aprendizaje de 1e-3 para el entrenamiento de imágenes de Google y una tasa de aprendizaje de 1e-4 para el entrenamiento de imágenes sobre banda. Igualmente se tienen que probar los resultados con imágenes que nunca haya visto. En las figuras 5.12 y 5.13 se puede observar que el reentrenamiento comienza con una precisión de 0.5 lo que indica que el modelo si obtuvo características útiles del primer entrenamiento, pero algo importante a señalar es que el valor de pérdida al comienzo del re-entrenamiento, figura 5.13, es alto y que decrece a medida que el entrenamiento continúa pero el valor de la precisión no aumenta de la misma manera lo que puede indicar que el modelo no puede extraer más características importantes y de continuar con el entrenamiento de esta manera se obtendrá un modelo con sobreajuste.

Tabla 5.2. Resultados sobre banda

Verdadera \ Inferencia	Positivo	Negativo
Positivo	1	19
Negativo	9	0
Exactitud	0,03448275862	

Para realizar la tabla 5.2 sólo se tomaron los primero 10 resultados de las imágenes ya que se vio que las detecciones no eran buenas y la única imagen que logró una detección con un IOU mayor a 0.5 fue la figura 5.14.

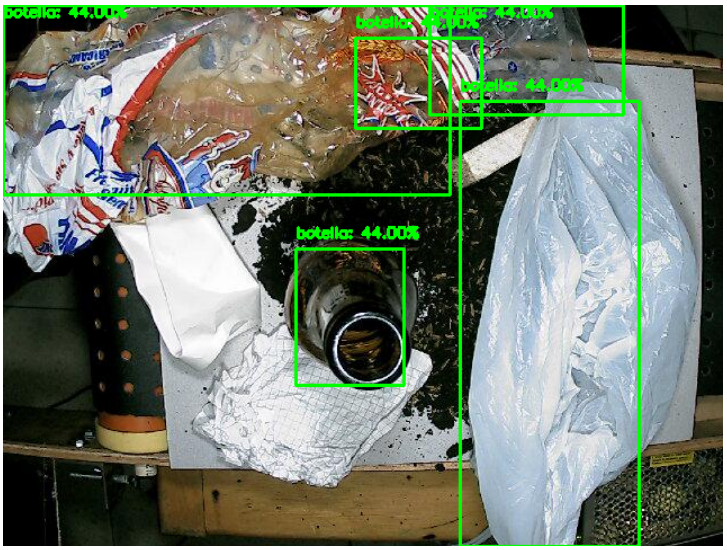

Figura 5.14. Resultado de la red RCNN modelo 2

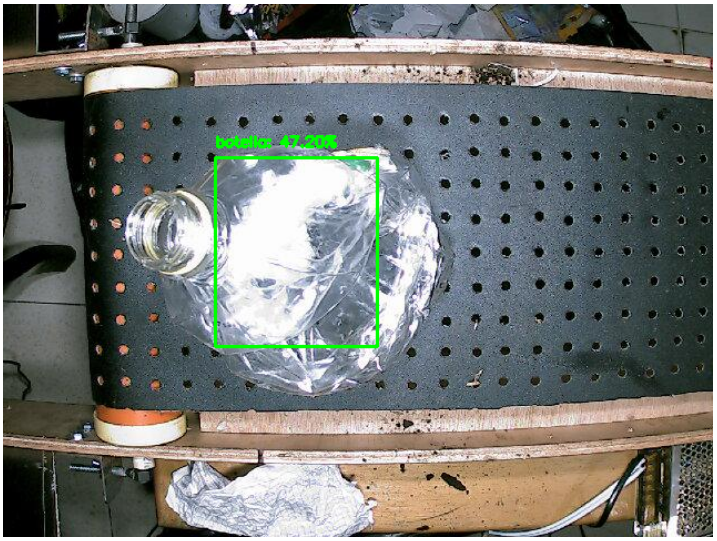

Figura 5.15. Resultado de la red RCNN modelo 2

La primera imagen de muestra, figura 5.14, tiene la única detección realizada que cumple con el parámetro IOU > 0.5, pero cuenta con una amplia cantidad de detecciones falsas. La segunda imagen de muestra, figura 5.15, cuenta con la detección pero no cumple con el parámetro IOU establecido. Este modelo está completamente fuera de entre los modelos que se consideraron para implementarlo por el resultado mostrado en la tabla 5.2 donde su exactitud es exageradamente bajo: 0.03.

5.2.3 SSD conjunto de datos combinados

Se continua con el entrenamiento del siguiente modelo y con la base de datos combinada, en este caso se utiliza el APi de Tensorflow. Para este

modelo se tiene que instalar una serie de dependencia y una de las más importantes es la versión 2.7.0 de Tensorflow.

```
!pip install tensorflow==2.7.0
```

```
import os
%cd /content
!git clone --quiet https://github.com/tensorflow/models.git
%cd /content/models/
#!git checkout 58d19c67e1d30d905dd5c6e5092348658fed80af
!apt-get update && apt-get install -y -qq protobuf-compiler python-pil python-lxml python-tk
!pip install -q Cython contextlib2 pillow lxml matplotlib
!pip install -q pycocotools
%cd /content/models/research
!protoc object_detection/protos/*.proto --python_out=.
os.environ['PYTHONPATH'] += ':/content/models/research/:/content/models/research/slim/'
!python object_detection/builders/model_builder_test.py
```

Figura 5.16. Instalación de dependencias y el API de Tenserflow

En la figura 5.16 se instalan las dependencias y el API de Tensorflow, esta API tiene todos los archivos para entrenar, inferir y exportar el modelo. Para hacer el entrenamiento se tienen que preparar los datos, las imágenes y sus etiquetas se pasan al formato tfrecord el cual en un sólo archivo guarda toda la información de las imágenes y las etiquetas por lo que se tiene 2 archivos tfrecord, train.tfrecord y test.tfrecord para entrenamiento y testeo respectivamente.
Después de tener los archivos y las dependencias instaladas se prosigue con la configuración y el entrenamiento.

1.Volviendo a utilizar el concepto de transferencia de aprendizaje, que quiero resaltar como uno de los conceptos más importantes del aprendizaje de máquina. Se elige un modelo entre los modelos pre entrenados que ofrece el API de Tensorflow , figura 5.17. Por la profundidad de la red, 152 capas, además por ser la que mayor precisión ofrece y ser entrenada con la base de imágenes COCO 2017 dataset que cuenta con 330.000 imágenes y 80 clases, se opta por el modelo SSD ResNet152 V1 FPN 1024x1024 (RetinaNet152).

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

Figura 5.17. Modelos, velocidad y precisión

```
!wget --no-check-certificate http://download.tensorflow.org/models/object_detection/tf2/20200711/ssd_resnet152_v1_fpn_1024x1024_coco17_tpu-8.tar.gz \
-O /content/ssd_resnet152_v1_fpn_1024x1024_coco17_tpu-8.tar.gz
```

Figura 5.18 Descarga del modelo elegido

En la figura 5.18 se descarga el fichero que contiene todos los archivos que se necesitan para hacer el entrenamiento del modelo.

2. Configuración del entrenamiento de la red, esta configuración se hace sobre el archivo pipeline.config, figura 5.19.

```
label_map_pbtxt_fname= "/content/labelmap.pbtxt"
train_record_fname = "/content/train.tfrecord"
test_record_fname = "/content/test.tfrecord"
```

```
pipeline_config.model.ssd.num_classes = 1
pipeline_config.train_config.batch_size=8
pipeline_config.train_config.fine_tune_checkpoint = "{}/checkpoint/ckpt-0".format(output_path)
pipeline_config.train_config.fine_tune_checkpoint_type = "detection"
pipeline_config.train_input_reader.label_map_path = label_map_pbtxt_fname
pipeline_config.train_input_reader.tf_record_input_reader.input_path[0]= train_record_fname
pipeline_config.eval_input_reader[0].label_map_path = label_map_pbtxt_fname
pipeline_config.eval_input_reader[0].tf_record_input_reader.input_path[0]= test_record_fname
pipeline_config.train_config.optimizer.momentum_optimizer.learning_rate.cosine_decay_learning_rate.learning_rate_base=4e-3
pipeline_config.train_config.optimizer.momentum_optimizer.learning_rate.cosine_decay_learning_rate.warmup_learning_rate=0.0001
```

Figura 5.19. Configuración archivo pipeline.config

Primero se asignan en variables las ubicaciones de los archivos tfrecord y el archivo labelmap.pbtxt, este archivo contiene el número de las clases y sus etiquetas, en este caso es 1 clase y su etiqueta es ‘botella’. En el archivo de configuración también se colocan el número de clases, seguido del tamaño del lote, en este caso son 8 imágenes por iteración. Se especifica el nombre con que serán guardados los checkpoints y el tipo de entrenamiento, se especifican las localizaciones de los archivos tfrecord para entrenamiento y testeo, y por último la tasa de aprendizaje. En el modelo anterior se tenía una tasa de aprendizaje inicial que descendía a medida que el entrenamiento avanzaba, en este caso se tiene un valor inicial

warmup_learning_rate = 0.0001 que va aumentando a medida que avanza el entrenamiento hasta llegar a la tasa de aprendizaje especificada, learning_rate_base = 4e-3. Cuando llega al valor final se mantiene constante hasta finalizar el entrenamiento.

3. Por último se entrena la red y se exportan los archivos de entrenamiento con los archivos que brinda el API de Tensorflow. El archivo para entrenamiento es model_main_tf2.py el cual tiene de parámetros el archivo pipeline, la dirección de salida del modelo y el número de épocas o steps, figura 5.20.

```
num_steps = 20000
model_dir = "/content/ssd_resnet/4e0_00001x20"
!python /content/models/research/object_detection/model_main_tf2.py  --pipeline_config_path={target_config} --model_dir={model_dir} --num_train_steps={num_steps}
```

Figura 5.20. Entrenamiento de la red

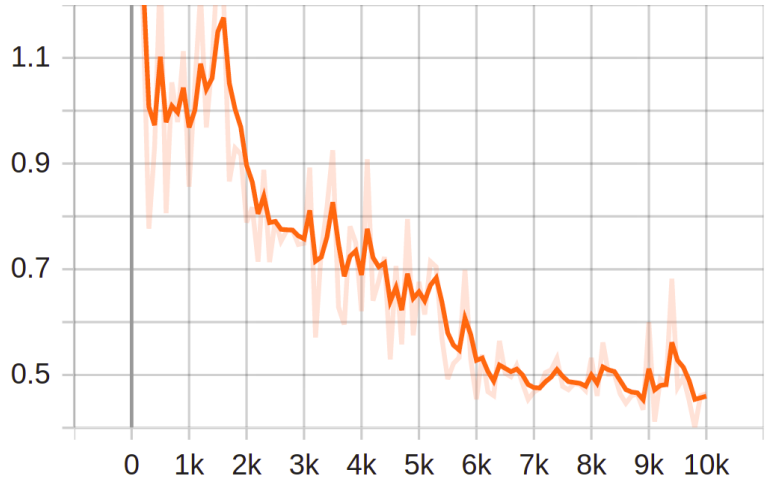

Figura 5.21 Métrica de pérdida total

En la figura 5.21 se puede observar el decrecimiento de la función de pérdida del modelo y en la tabla 5.3 los resultados sobre la base de datos de prueba. Su exactitud obtenida fue del 65%. En la figura 5.22 se puede ver una doble detección de la botella pero este problema se puede solucionar filtrando por un nivel de confianza superior.

Tabla 5.3. Resultados conjunto de datos unidos

Verdadera \ Inferencia	Positivo	Negativo
Positivo	21	8
Negativo	3	0
Exactitud	0,65625	

Figura 5.22. Resultado de la red SSD modelo 1

Figura 5.23. Resultado de la red SSD modelo 1

En la figura 5.23 se puede ver el mayor número de detecciones falsas que realizó el modelo. En este caso ajustar el nivel de confianza no ayuda al modelo ya que si bien se pueden quitar detecciones incorrectas, varias de estas pasarían y si el filtro de confianza es alto la detección correcta no aparece.

Por último se exporta el modelo utilizando el archivo exporter_main_v2.py como se puede ver en la figura 5.24. Se especifican el tipo de dato a exportar, en este caso un tensor que puede ser leído con las bibliotecas de Tensorflow, el archivo de configuración, los checkpoints y el fichero de salida.

```
output_directory = "/content/4e0_00001"
!python /content/models/research/object_detection/exporter_main_v2.py \
--input_type image_tensor \
--pipeline_config_path {target_config} \
--trained_checkpoint_dir {model_dir} \
--output_directory {output_directory}
```

Figura 5.24. Exportación del modelo

5.2.4 SSD conjunto de datos separados

Ahora se realiza el segundo modelo SSD pero con los datos separados. Los únicos cambios con respecto a los pasos anteriores son; cambiar el learning_rate_base a 0.00008 y el warmup_learning_rate a 0.00001 y en la sección de entrenamiento no se deja una carpeta vacía sino que se asigna la ruta del modelo entrenado con la data de Google, más exactamente en la carpeta de los checkpoints para que la red comience el entrenamiento desde el último punto de guardado como se muestra en la figura 5.25.

```
num_steps = 10000
model_dir = "/content/8e-3_0_0001de0guar/checkpoint"
!python /content/models/research/object_detection/model_main_tf2.py
--pipeline_config_path={target_config} --model_dir={model_dir} --num_train_steps={num_steps}
```

Figura 5.25. Reentrenamiento modelo SSD

y los resultados están en la figura 5.26.

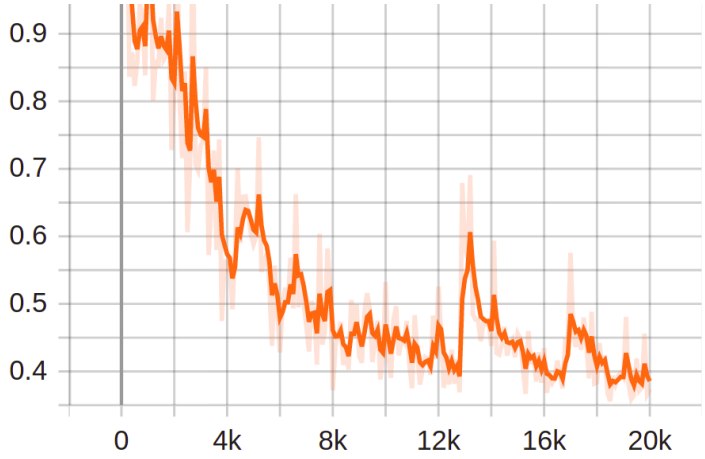

Figura 5.26. Métrica de pérdida total

En la figura 5.26 se puede ver como la pérdida del modelo decrece hasta alcanzar un valor de 0.4 que es un mejor resultado que el obtenido con el entrenamiento con las bases de datos juntas, esto se puede ver mucho más claro en la tabla de la tabla 5.4 donde se evidencia que su exactitud es mejor, 0.8, lo que es un valor aceptable para un modelo de detección de objetos.

Tabla 5.4. Resultados datos separados

Verdadera \ Inferencia	Positivo	Negativo
Positivo	20	4
Negativo	1	0
Exactitud	0,8	

Figura 5.27. Resultado de la red SSD modelo 2

En la figura 5.27 se puede ver una predicción realizada con el modelo la cual da un nivel de confianza del 100%. Este resultado junto con la exactitud obtenida representan que el modelo funciona adecuadamente para el sistema de detección planteado en este proyecto.

5.2.5 YOLO conjunto de datos combinados

YOLO es el más reciente de los 3 modelos y en particular YOLOv5 que es una de las últimas versiones de YOLO, este modelo divide la imagen de entrada en pequeños cuadros los cuales cada uno extrae su propio mapa de característica a lo largo del avance de la imagen por la red convolucional, al tener cada uno de estos cuadros un propio mapa de características les permite realizar individualmente la detección del objeto. YOLOv5 cuenta con 5 infraestructuras, desde la más simple hasta la más compleja y profunda YOLOv5x, que fue la elegida para ser entrenada por ser la que mejor precisión ofrece. Al igual que con SSD YOLOv5 cuenta con una serie de archivos donde se configuran sus parámetros de entrenamiento, cómo la tasa de aprendizaje y el número de épocas. Es importante resaltar que YOLO fue escrito con la librería para machine learning PyTorch porque ofrece: “f*lexibilidad, capacidades de depuración y una formación de corta duración*”[22].

1.Se clona el repositorio de YOLOv5 en github. Este repositorio cuenta con un archivo requirements.txt que indica las librerías que se necesitan y su versión y con pip hacemos la instalación de estas dependencias como se puede ver en la figura 5.28.

```
!git clone https://github.com/ultralytics/yolov5  # clone repo
%cd yolov5
%pip install -qr requirements.txt # install dependencies
%pip install -q roboflow

import torch
import os
from IPython.display import Image, clear_output  # to display images
```

Figura 5.28. Descargar modelo YOLOv5

2. En el archivo hyp.finetune.yaml se especifican los valores de la tasa de aprendizaje, figura 5.29. Al igual que en las anteriores redes se estuvo probando los modelos y sus resultados variando la tasa de aprendizaje y en este caso los mejores resultados fueron obtenidos con los parámetros de la figura 5.29.

```
lr0: 1e-4
lrf: 1e-5
```

Figura 5.29. hyp.finetune configuración

El resto de parámetros se dejan igual y en este modelo, al igual que en el modelo RCNN, se utiliza el decaimiento de la tasa de aprendizaje, comienza con 1e-4 y termina con una tasa de aprendizaje de 1e-5, como se puede observar en la figura 5.29.

3. Se preparan los datos. YOLO recibe las imágenes y las etiquetas por separado y las etiquetas deben estar en un formato .txt. Con la ayuda de Roboflow se exportó la base de datos y sus etiquetas en un formato .txt preparado para YOLOv5 como se ve en la figura 5.30.

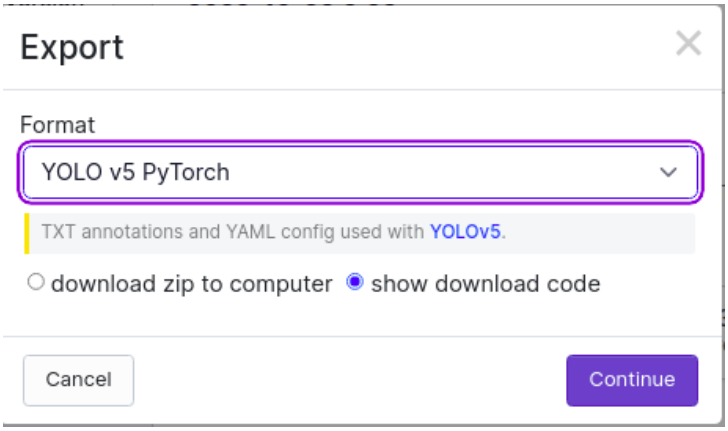

Figura 5.30. Exportar etiquetas e imágenes

Por medio del archivo de configuración customer.yaml se especifican los ficheros de entrenamiento y validación, figura 5.31, además el número de clases y la etiqueta, botella.

```
train: /content/yolo_reentrenamiento/images/train
val: /content/yolo_reentrenamiento/images/val

# Classes
nc: 1  # number of classes
names: ['botella']  # class names
```

Figura 5.31. Configuración de la red

4. Se entrena el modelo con el archivo train.py como se puede observar en la figura 5.32, se especifican los parámetros de tamaño de la imagen 640, número de épocas 300, un lote de 4 imágenes, la localización del archivo customer.yaml y los pesos con lo que se entrenara. Como es el entrenamiento de la data combinada se entrenará sobre el modelo base aplicando nuevamente transferencia de aprendizaje, esto se hace por el parámetro de weigths en el cual se especifica el modelo a entrenar.

```
!python /content/yolov5/train.py --img 640 --batch 4 --epochs 300 --data /content/yolov5/data/customer.yaml --weights yolov5x.pt --cache
```

Figura 5.32. Entrenamiento de la red

5. Por último, se exporta el modelo con el archivo export.py como se ve en la figura 5.33. Se eligió exportar a Tensorflow y Tensorflow lite. El archivo que guarda el modelo es best.pt.

```
!python export.py --weights /content/yolov5/runs/train/exp2/weights/best.pt --include saved_model tflite
```

Figura 5.33. Exportación del modelo

Las métricas obtenidas para el primer modelo están en la figura 5.34 donde se ve la pérdida de la ubicación de las cajas delimitadoras y la pérdida de la detección del objeto respectivamente.

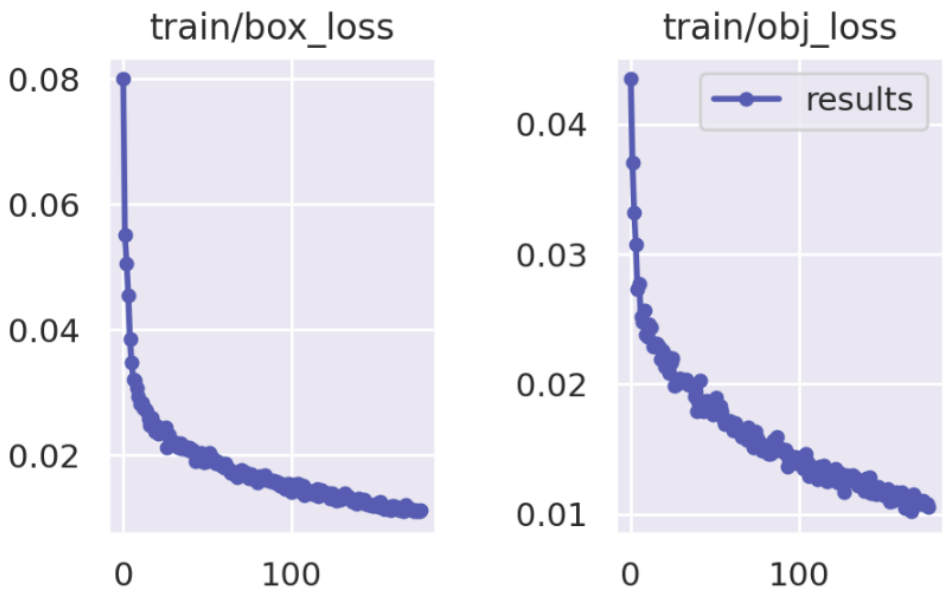

Figura 5.34. Pérdida de localización del objeto y creación de la caja

Este modelo es el que mayor exactitud presenta como se puede ver en la tabla 5.5. Su exactitud llega al 95% con las imágenes de prueba. Esto es reforzado por la baja función de pérdida, decreciendo hasta valores de 0.02 en la formación de la caja delimitadora y la detección como se ven en la figura 5.34.

Tabla 5.5. Resultados datos combinados

Verdadera \ Inferencia	Positivo	Negativo
Positivo	23	0
Negativo	1	0
Exactitud	0,9583333333	

Figura 5.35. Resultado de la red YOLO modelo 1

Figura 5.36. Resultado de la red YOLO modelo 1

En la figura 5.35 se ve una inferencia del modelo con una confianza mayor a 0.9 y en la figura 5.36 se tiene la única imagen de prueba que no logró tener una detección.

5.2.6 YOLO conjunto de datos separados

Ahora se pasa a entrenar con las bases de datos separadas, al igual que con los anteriores modelos se entrena la red con los datos de Google y después con las de la banda.

1. Se crean 2 archivos de customer.yaml, esto se hace para tener las rutas de los conjuntos de imágenes separadas.

2. Primero se hace el entrenamiento sobre la base de datos de Google, el modelo se guarda como best.py y es con este que se entrena el modelo nuevamente. Para especificar que se va a trabajar sobre el modelo entrenado con las imágenes de Google en el parámetro weights no se le pasa el modelo YOLOv5x sino se le envía el archivo best.pt, como se ve en la figura 5.37, y además en el parámetro data se le pasa el segundo archivo customer.yaml, el cual tiene las rutas de las imágenes sobre la banda transportadora.

```
!python /content/yolov5/train.py --img 640 --batch 4 --epochs 300 --data /content/yolov5/data/customer2.yaml
--weights /content/yolov5/runs/train/exp3/weights/best.pt --cache
```

Figura 5.37. Segundo entrenamiento

3. Se exporta el modelo igualmente a Tensorflow y Tensorflow lite, esto es exactamente igual al modelo YOLOv5 anteriormente explicado.

Por último los resultados obtenidos se ven en la figura 5.38.

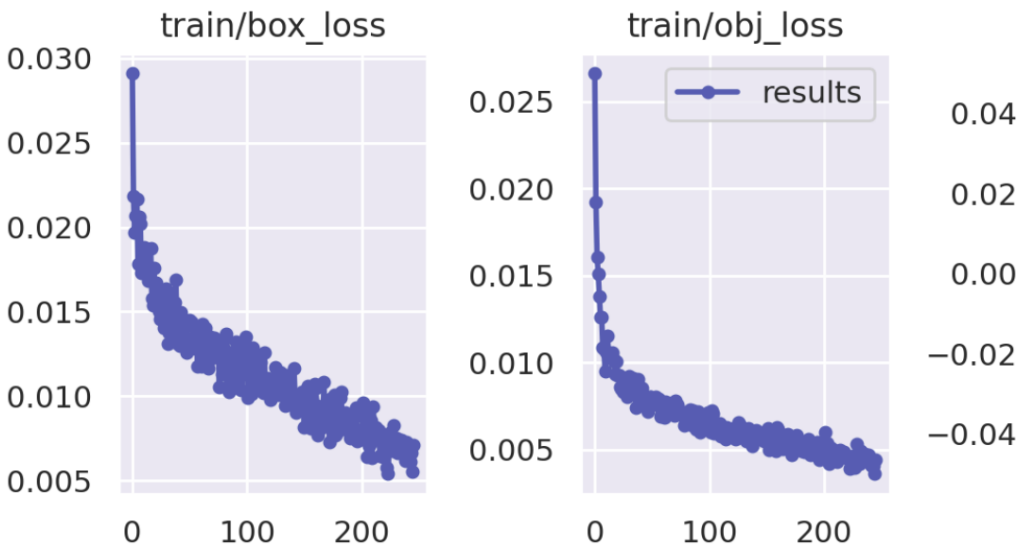

Figura 5.38. Pérdida de localización del objeto y creación de la caja

La función de pérdida de este modelo llegó a niveles más bajos que los niveles del modelo con los datos juntos, figura 5.38. Pero su exactitud es menor al resultado de 0.9 obtenido con el modelo YOLOv5 entrenado con las bases de datos juntas, tabla 5.6. Sin embargo, el valor de exactitud sigue siendo bueno y se considera para realizar la implementación.

Tabla 5.6. Resultados datos separados

Verdadera \ Inferencia	Positivo	Negativo
Positivo	20	1
Negativo	4	0
Exactitud	0,8	

Lo que se busca en esta parte es elegir el modelo con mayor exactitud. Como se había explicado anteriormente la manera correcta de saber si la red funciona es hacerla inferir sobre muestras que nunca antes haya visto.

Figura 5.39. Imágenes de prueba

En la figura 5.39 se tienen las imágenes utilizadas para hacer las pruebas de los modelos, los resultados obtenidos de las inferencias realizadas sobre estas imágenes están en las tablas que se presentaron en cada modelo. Los resultados obtenidos se agruparon en una sola tabla, tabla 5.7, para facilitar la comparación.

Tabla 5.7. Tabla de resultados

Modelo	Exactitud
RCNN 1	0,4186046512
RCNN 2	0,03448275862
SSD 1	0,65625
SSD 2	0,8
YOLOv5 1	0,9583333333
YOLOv5 2	0,8

El modelo elegido es YOLOv5 entrenado con las bases de datos juntas, sin embargo este modelo requiere de un sistema con especificaciones altas para un sistema embebido, un procesador de 64 bits y minimo 2 gb de Ram, por eso el modelo SSD entrenada con base de datos separada se tuvo como modelo suplente.

5.3. Creación de la banda e implementación del hardware

En el proyecto se propuso la creación de una banda transportadora la cual lleva los residuos desde un punto de inicio hasta un punto final en el que se encuentra la cámara, que cumple la función de obtener la imagen que se le pasa al modelo para realizar la inferencia. La infraestructura de la banda cuenta con 2 rodillos, 1 banda de plástico, 1 soporte para la cámara, 2 bombillos led, un motor elevavidrios, una fuente conmutada y 4 led infrarrojos, 2 emisores y 2 receptores. En la figura 5.40 se tiene todos los elementos antes mencionados, en la parte

superior izquierda la banda transportadora, en la parte superior derecha la cámara de 1080P y en la parte baja la fuente conmutada de 100-240 V a 12V 10A. Los leds infrarrojos, ubicados al inicio y final de la banda, tienen la función de indicar al sistema la posición de los residuos, los leds del inicio informan cuando hay basura sobre la banda y los leds del final le indican al sistema que la basura está debajo de la cámara.

Figura 5.40. Banda transportadora

El sistema embebido elegido fue la Raspberry pi 4 de 4gb de Ram, la elección fue tomada partiendo del hecho que Pytorch funciona en dispositivos con una arquitectura de 64 bits.
El motor es controlado por medio del driver DRI0018 de DFROBOT, figura 5.41, que tiene 2 canales para controlar 2 motores, su voltaje de entrada está entre los 4.8V y 35V con una corriente máxima de 15A, para el proyecto se utilizaron 4 pines del driver, 2 de alimentación y 2 de control, los pines de control son: M2_PWM con el que se controla la velocidad del motor por medio de una señal pwm y M2_EN que controla la dirección de giro del motor. Además, cuenta con 4 pines para verificar el estado del driver pero en el proyecto no fueron utilizados.

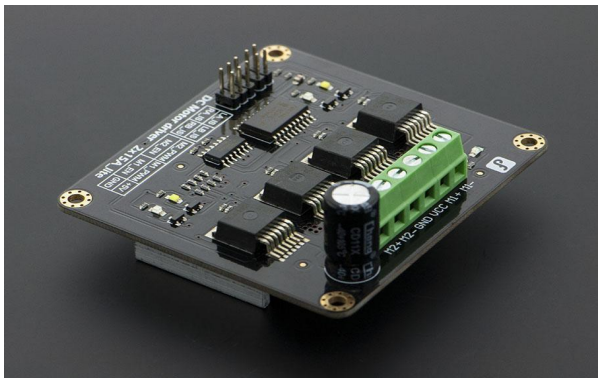

Figura 5.41. DC motor driver
Tomado de 2×15A DC motor driver [Imagen]. (s.f.). DFRobot Electronics, open-source hardware electronics and kits. https://www.dfrobot.com/product-796.html

El esquemático que representa las conexiones de la raspberry con el driver y los leds infrarrojos es la figura 5.42.

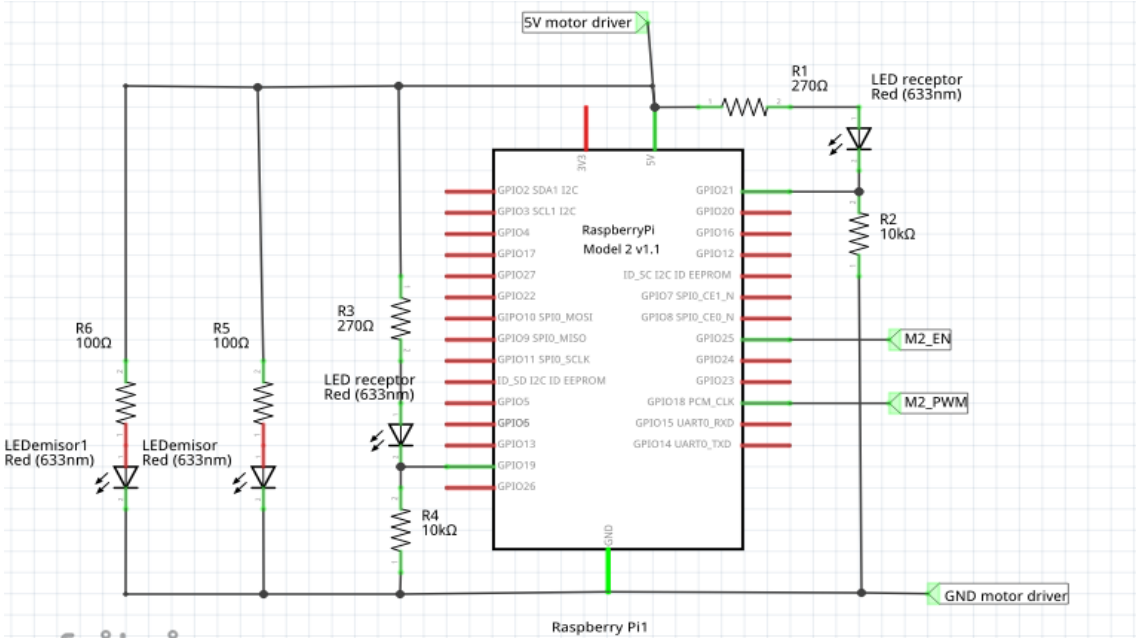

Figura 5.42. Esquemático de conexiones

Los leds que aparecen en el esquemático representan los leds infrarrojos utilizados. Los pines 19 y 21 son los utilizados para realizar la detección con los leds y los pines 18 y 25 son utilizados para la conexión con el driver.

5.4. Diseño del algoritmo

En esta sección se especifica el correcto funcionamiento del sistema. Se parte considerando la totalidad de este proyecto como la etapa de detección de un proyecto más grande que logra sacar los envases dentro de la basura. La representación más simple del algoritmo se puede observar en la figura 5.43. Este

diagrama no tiene en cuenta diversas funciones que hacen que el sistema funcione mejor, como el guardado de las imágenes de las inferencias o una función que hace avanzar la banda después de la inferencia por un tiempo establecido.

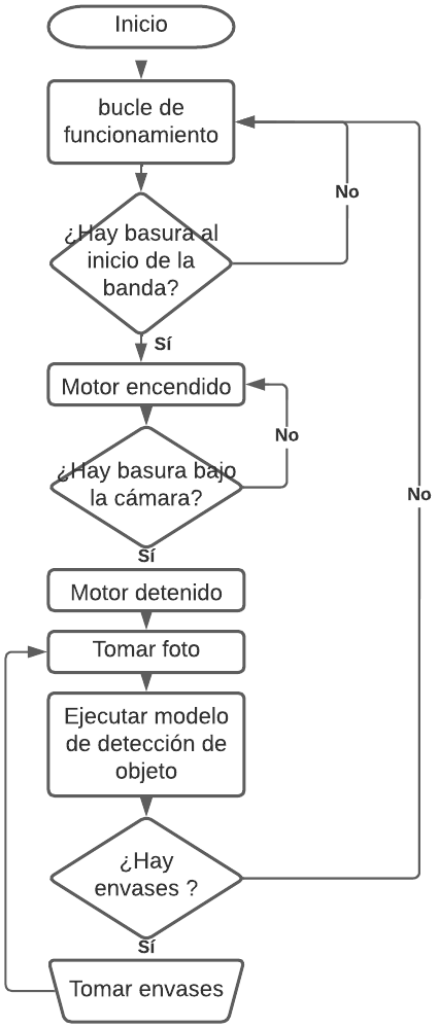

Figura 5.43. Algoritmo de funcionamiento

El diagrama de árbol del fichero que contiene el programa que ejecuta el código del sistema se ve en la figura 5.44.

```
programa
├── clases
│   ├── camara2.py
│   ├── fichero.py
│   ├── modelo.py
│   ├── motor.py
│   └── pantalla.py
```

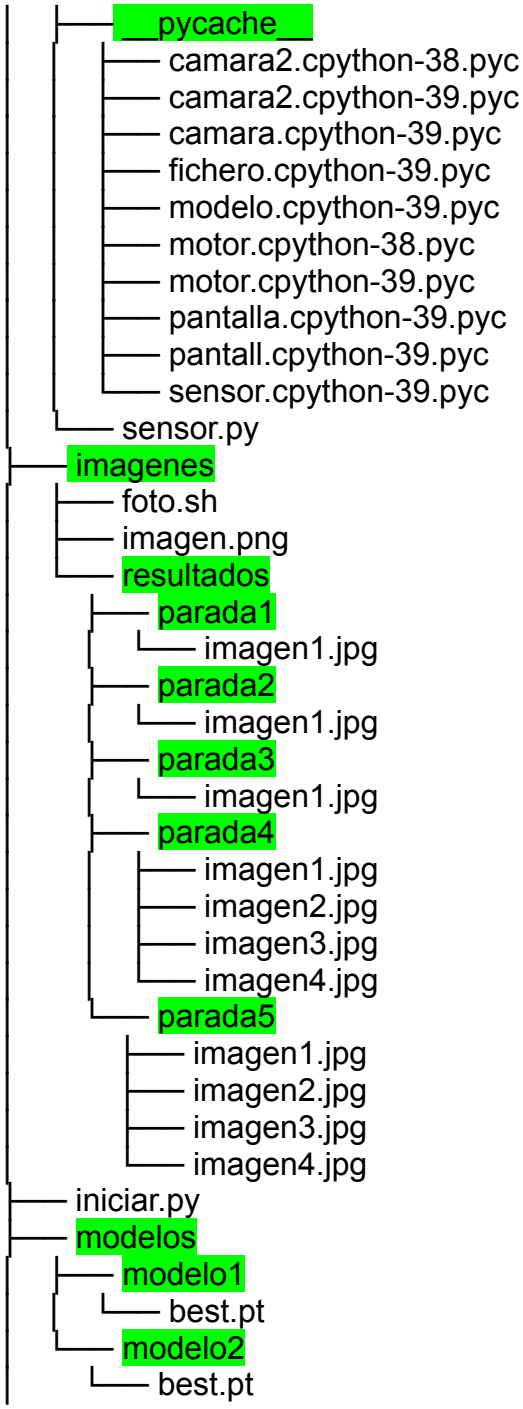

Figura 5.44. Diagrama de árbol del fichero programa

En esta sección se explicarán las funciones más básicas del programa y la totalidad de código se puede encontrar en los anexos. En la figura 5.44 los nombres que se encuentran resaltados en verde son ficheros y el resto son archivos. La carpeta raíz se llama programa y el archivo main principal del sistema se llama iniciar.py.

Los programas fueron escritos utilizando el lenguaje de programación Python, en iniciar.py se encuentran las inicializaciones de las instancias de las clases y las rutas de los modelos, figura 5.45.

```
1 from clases.sensor import sensor
2 from clases.motor import motor
3 from clases.camara2 import camara2
4 from clases.modelo import modelo
5 from clases.pantalla import pantalla
6 from clases.fichero import carpetas
7 from time import sleep
8
9 pin = 19
10 pin2 = 21
11 pwm = 12
12 dire = 18
13 model = '/home/buho/Documents/proyecto/programa/modelos/modelo1/-
   best.pt'
14 model2 = '/home/buho/Documents/proyecto/programa/modelos/modelo2/-
   best.pt'
15 imagen = '/home/buho/Documents/proyecto/programa/imagenes/imagen.png'
16
17 sensor_entrada = sensor(pin)
18 sensor_salida = sensor(pin2)
19 monitor = pantalla()
20 motorBanda = motor(pwm,dire,sensor_salida)
21 camara = camara2()
22 fichero = carpetas()
23 yolo = modelo(model2,camara,fichero,monitor)
```

Figura 5.45. Iniciar.py inicialización

En la figura 5.45 desde la línea uno a la siete se importan las clases creadas y las librerías que se usaron, de la línea nueve a quince se especifican los pines de la Raspberry Pi utilizados y las rutas de los modelos, y para finalizar se crean los objetos utilizados en el proyecto instanciando las clases creadas para abstraer partes esenciales del funcionamiento de la banda, las clases creadas se encuentran en el fichero clases.
Cuando se crea el objeto yolo por medio de la inicialización de la instancia se carga el modelo que se utilizó, la clase del objeto yolo, línea 23, tiene la ruta relativa a la carpeta /programa de la siguiente manera: /programa/clases/modelo.py.

```
1 import torch
2 import cv2
3 from matplotlib import pyplot as plt
4 import numpy as np
5 from time import sleep
6
7 class modelo:
8         def __init__(self,modelo,camara,carpeta,monitor):
9                 self.modelo = torch.hub.load('ultralytics/yolov5',
   'custom', path=modelo)
10                self.modelo.conf = 0.5
11                self.camara = camara
12                self.carpeta = carpeta
13                self.monitor = monitor
14                self.monitor.imprimir("MODELO CARGADO")
```

Figura 5.46. Inicialización clase modelo

Al instanciar la clase modelo se le pasan los parámetros que necesitará para realizar las diferentes funciones creadas. En la figura 5.46 se comienza con la

importación de las librerías utilizadas, en este caso las dos librerías esenciales para realizar la inferencia son torch, encargada de cargar el modelo y cv2 librería que utiliza el modelo para leer la imagen y mostrarla. En la línea nueve por medio de la función hub.load de torch se carga el modelo, se le pasan tres parámetros los cuales son: la clase a modelo a cargar, ultralytics/yolov5, si es un modelo personalizado, custom y por último la ruta donde se encuentra el archivo .pt, que es el archivo que contiene el modelo y después se asignan los atributos de la clase.

```
26 def bucleDeInferencia():
27         inferencia,valor = yolo.inferir(imagen,1)
28         while(valor!=False):
29                 monitor.imprimir("TOMANDO BOTELLA")
30                 sleep(5)
31                 monitor.cerrar_ventana()
32                 camara.tomar()
33                 inferencia,valor = yolo.inferir(imagen,2)
```

Figura 5.47. Bucle de inferencia

En la figura 5.47 se ve la definición del bucle de inferencia dentro de iniciar.py, este bucle cumple de función de tomar la foto sobre la banda y utilizar la función inferir del objeto yolo para hacer la inferencia de la imagen, a la función inferir se le pasa la imagen tomada y un valor, este valor si es uno representa que la imagen tomada es una nueva muestra de basura y si es dos que es la misma muestra de basura y que se están tomando las botellas una por una. Esto se realizó de esta manera teniendo en cuenta que las muestras son de basura conglomerada y que al tomar una botella la posición de las demás botellas puede cambiar, por eso es que se toma una botella y se toma una nueva foto para hacer la inferencia nuevamente. La función inferir devuelve las respuestas de la inferencia y además devuelve un valor el cual representa si dentro de la muestra se detectó alguna botella, si el valor es negativo se sale del bucle y se continúa con los siguientes procesos para traer abajo de la cámara la nueva muestra de basura, y si el valor es positivo se entra al siguiente bucle en el cual se toma la botella y se realiza la nueva inferencia y así sucesivamente hasta que el valor sea negativo lo que indica que sobre esa muestra ya no hay más botellas para tomar, se sale del bucle de inferencia y se prosigue con los procesos para traer una nueva muestra de basura bajo la cámara.

```
34 while True:
35         while sensor_entrada.detectar():
36                 sleep(0.2)
37         motorBanda.avanzar()
38         camara.tomar()
39         bucleDeInferencia()
40         motorBanda.avanzar_10()
```

Figura 5.48. Bucle de funcionamiento

En la figura 5.48 se ve el bucle de funcionamiento dentro de iniciar.py, este bucle representa el algoritmo de presentado en el figura 5.43, inicia esperando que se presente una muestra de basura en el sensor del inicio de la banda, línea 35 y 36, si hay muestra de basura se sale del ciclo while de la línea 35, posteriormente la banda transportadora lleva la basura hasta la cámara y se detiene cuando el sensor se salida detecta la muestra, línea 37, dentro del método avanzar de la clase motor se hace recoge la información del sensor de salida para detener la banda transportadora en el lugar adecuado. Por último se toma la foto de la muestra, línea 38, y se realiza la inferencia con el bucle de inferencia presentado anteriormente. El metdos avanzar_10 del objeto motorBanda, línea 40, se encarga de avanzar la banda cuando no se detectan botellas y llevar una muestra de basura nueva bajo la cámara para realizar el proceso nuevamente sin tener que detectar la nueva muestra con el sensor de inicio, al pasar un tiempo determinado y si no se detectó una nueva muestra de basura con el sensor de salida la banda se detiene y se regresa al ciclo while de la línea 35 para esperar una nueva muestra de basura.

5.5. Revisión del funcionamiento en el sistema embebido

Se realizó una primera revisión del funcionamiento del sistema con el modelo de visión artificial YOLOv5 entrenado y elegido, el sistema funciona correctamente pero el modelo no hace inferencias muy buenas aparte de no lograba detectar todas las botellas.

Figura 5.49. Prueba de funcionamiento 1

En la figura 5.49 se pueden observar varias imágenes de diferentes muestras de basura. En la imagen de la esquina superior izquierda y esquina inferior izquierda se hizo una inferencia correcta pero hay una botella que no fue detectada, en la imagen de la esquina superior derecha se realizó la inferencia correctamente pero es una muestra simple y la botella es fácil de detectar, y por último en la esquina inferior derecha se hace una inferencia incorrecta, ya que aunque sí hay parte de la botella dentro del cuadro delimitador la inferencia es más hacia la cartuchera de metal.

Debido a estos malos resultados se optó por tomar más imágenes para entrenar nuevamente el modelo elegido, YOLOv5. El anterior modelo fue entrenado con una base de datos de 689 imágenes las cuales se dividieron en 2 grupos, imágenes de Google, 491 imágenes, e imágenes tomadas sobre la banda, 198 imágenes. Para el nuevo entrenamiento se tomaron 417 imágenes sobre la banda transportadora, además, a las imágenes viejas sobre la banda se les hizo un re-etiquetado para dejar los cuadros delimitadores más cerca a los objetos. En total la base de datos con la que se entrenó nuevamente el modelo terminó con 1101 imágenes, constituida con 491 imágenes tomadas de Google y 610 imágenes sobre la banda transportadora. Por último se aplicó data augmentation sólo en las imágenes tomadas sobre la banda. La base de datos final para entrenar el modelo quedó con 1916 imágenes. Los cambios en las imágenes sobre la banda fueron: rotación de 90 grados, inversión en horizontal y zoom de 40%.

6. Resultados y Discusión

La funcionalidad de sistema fue puesto a prueba utilizando diferentes muestras de basura, los resultados de las pruebas de funcionamiento paras las diferentes funciones del sistema se presentan en la tabla 6.1.

Tabla 6.1. Resultados de las pruebas de funcionamiento

	Cumplimiento	
Funcionalidad	**Si**	**No**
Detectar la muestra de basura al inicio de la banda	x	
Arrancar el motor para mover la banda	x	
Detectar la muestra de basura al final de la banda	x	
Detener el motor para detener la banda	x	
Tomar la foto	x	
Realizar la inferencia	x	
Seguir los pasos establecidos para la toma del envase y tomar la nueva foto para realizar la inferencia	x	

Como se puede ver en los resultados presentados en la tabla 6.1 las funcionalidades básicas contempladas para el correcto funcionamiento de la banda transportadora fueron cumplidos. Las pruebas siguientes se centraron en revisar el funcionamiento del modelo de inteligencia artificial, YOLOv5. Para hacer las pruebas del modelo se dispusieron de 20 muestras, dentro de las cuales habían 3 muestras con 3 envases, 7 con 2 envases y por último 10 con 1 envase.

Tabla 6.2. Resultados de las pruebas del modelo de visión artificial

Muestra	**Positivo/Positivo**	**Positivo/Falso**	**Falso/Positivo**	**Falso/Falso**	**Número verdadero**
1	2	0	0	0	2
1	1	0	0	0	1
1	0	1	0	0	0
2	1	0	0	0	1

3	1	0	1	0	2
4	1	0	0	0	1
5	3	0	0	0	3
5	2	0	0	0	2
5	1	0	0	0	1
6	0	1	0	0	0
7	0	1	2	0	2
7	1	0	0	0	1
8	1	1	1	0	2
8	1	0	0	0	1
9	2	0	0	0	2
9	1	0	0	0	1
10	1	0	0	0	1
11	1	0	0	0	1
11	0	1	0	0	0
12	1	0	1	0	2
13	1	0	0	0	1
14	2	0	0	0	2
14	1	0	0	0	1
15	2	0	0	0	2
15	1	0	0	0	1
16	2	0	0	0	2
16	2	0	0	0	2
16	1	0	0	0	1
17	1	0	0	0	1
18	2	1	0	0	2
18	1	0	0	0	1
19	1	0	0	0	1
20	1	0	1	0	2
20	1	0	0	0	1

En la tabla 6.2 se presentan los resultados obtenidos a partir de las inferencias en las 20 muestras sobre la banda transportadora. Como se puede ver en los

resultados en general el modelo realizó buenas inferencias y fueron pocas las inferencias falsas y las no inferencias de envases sobre las muestras.

Tabla 6.3. Resultados de las pruebas del modelo de visión artificial

Verdadera \ Inferencia	Positivo	Negativo
Positivo	40	6
Negativo	6	0
Exactitud	0,7692307692	
Precisión	0,8695652174	

En la tabla 6.3 se tienen los resultados de la precisión y exactitud del modelo de visión artificial. Estos resultados de precisión y exactitud indican que el modelo hace más no inferencias de la muestra que inferencias incorrectas, por una parte esto es bueno ya que no se detecta mucha basura como si fuera un envase, pero hace que hayan envases en la basura que no se detectan. Sin embargo, los resultados no son nada malos, considerando el valor de exactitud se detectan mínimo 7 envases de 10.

En la figura 6.1 se presentan dos muestras de las pruebas realizadas, el ciclo en el que se toma el envase y se realiza nuevamente la inferencia sobre una nueva foto.

Figura 6.1. Pruebas de inferencias

En la figura 6.1. lado izquierdo se tiene primeramente una muestra con 3 envases dentro, el modelo detecta correctamente los 3 envases, siguiendo con el algoritmo se realiza una nueva captura a la muestra para volver a realizar la inferencia que sería la imagen del lado izquierdo central y por último repitiendo el ciclo la imagen izquierda inferior donde el último envase. En las imágenes del lado derecho de la figura 6.1 se tiene primeramente una muestra con 2 envases y se continúa con el ciclo antes explicado hasta terminar con la imagen derecha central donde solo hay 1 envase.

7. Conclusiones y Trabajos futuros

- Las imágenes tomadas de bases de datos o buscadores sirven para reforzar datasets con pocos datos, solo si son cuidadosamente seleccionadas cumpliendo con la necesidad de agregar información útil a la red. Sin embargo, para un mejor funcionamiento del modelo es recomendable que sea mayor el número de datos específicos del ambiente en el cual se va a desempeñar el modelo.
- Realizando pruebas de funcionamiento se concluyó que el modelo YOLOV5 al utilizar regiones para hacer inferencias separadas es susceptible a que algunas de estas regiones no tengan la información necesaria para realizar correctamente la inferencia, ya que durante las pruebas en la zona baja de la banda se hacían detecciones falsas confundiendo cajas con botellas lo cual no pasaba en la zona superior de la banda. Esto solo es un inconveniente si como en este proyecto se desea detectar un objeto específico en cualquier parte de la imagen, pero si el contenido de las imágenes tiene una estructura específica, como por ejemplo una carretera, esta cualidad del modelo sería de utilidad creando zonas de interés, potencialmente mejorando el rendimiento del sistema al no tener que detectar otro tipo de objetos en zonas específicas.
- El análisis de los posibles escenarios que se pueden presentar en el ambiente en el cual se va a desempeñar el modelo es esencial para prevenir posibles errores o diseñar correctamente la funcionalidad del sistema. Se puede ver cuando al considerar que: al tomar un envase de la muestra el resto de envases podían cambiar de posición o alterar la visibilidad de los mismos, esto se puede apreciar claramente en las imágenes de la derecha de la figura 6.1, cuando al tomar el primer envase la muestra cambió y se pudo hacer una mejor detección del segundo envase en la muestra.
- El sistema creado es completamente funcional, al no haber especificado una precisión o exactitud mínima de funcionamiento este aspecto del sistema quedaba muy a merced de los resultados, sin embargo, de los resultados obtenidos se concluye que el modelo detecta correctamente envases en la basura con una exactitud de 0.769..
- La rama de detección de objetos tiene limitaciones en el escenario propuesto para el sistema planteado, si los objetos a detectar estas en vertical u horizontal el sistema dará la información correcta a un posterior sistema que se encargue de tomar el envase, pero si el envase está dispuesto diagonalmente la información suministrada no será suficiente para poder tomar el objeto. Por este motivo el modelo de visión artificial que es más adecuado para este sistema debe aplicar la técnica de segmentación, ya que este modelo sí proporciona la información necesaria para la toma del envase sin importar la forma en la que esté dispuesto el envase en la muestra.

- Independientemente de la técnica utilizada en el sistema, un prototipo completamente funcional deberá transformar las coordenadas de la detección sobre la imagen a las coordenadas físicas correspondientes sobre la banda transportadora para localizar efectivamente el objeto y ser tomado.

8. Referencias Bibliográficas

[1] Felipe Antón, Mª del Rosario, "Interacciones microorganismos-suelos-planta en la preservación del Medio Ambiente y la Salud," Discursos, 2003.

[2] J. A. Martínez Sepúlveda et al, "Residuos en Hispanoamérica de lo ambiental a lo social," 2016.

[3]P. Tello and P. Mijailova and G. Ruiz, “Temas de Ingeniería Sanitaria y Ambiental,"2014.

[4] Sáez et al, "Omnia ISSN: 1315-8856 revistaomnia@gmail.com Universidad del Zulia Venezuela," vol. 20, 2014. Available: http://www.redalyc.org/resumen.oa?id=73737091009.

[5] H. Ponce, E. Moya-Albor and J. Brieva, "A novel artificial organic control system for mobile robot navigation in assisted living using vision- and neural-based strategies," Comput Intell Neurosci, vol. 2018, 2018. Available: https://search.ebscohost.com/login.aspx?direct=true&db=psyh&AN=2018-63858-001&lang=es&site=ehost-live. DOI: 10.1155/2018/4189150.

[6] W. B. Rauch-Hindin, Aplicaciones De La Inteligencia Artificial En La Actividad Empresarial, La Ciencia Y La Industria. 1989.

[7] Girón, S. L., Mateus, J. C., & Méndez, F. (2009). Impacto de un botadero a cielo abierto en el desarrollo de síntomas respiratorios y en costos familiares de atención en salud de niños entre 1 y 5 años en Cali, Colombia. Biomédica, 29(3), 392–402. https://doi.org/10.7705/biomedica.v29i3.11

[8] "Impacto Medioambiental Del Plástico: Causas Y Efectos". Roll'eat®. https://rolleat.com/es/impacto-ambiental-del-plastico/ (accedido el 28 de noviembre de 2022).

[9] "Disposición de residuos plásticos".https://coprocesamiento.org. https://coprocesamiento.org/disposicion-de-residuos-plasticos/.

[10] D. L. PACHÓNESPINEL, "PROTOTIPO DE SISTEMA AUTOMATIZADO CON VISIÓN ARTIFICIAL PARA LA SELECCIÓN DE EMPAQUES DE PLÁSTICO, VIDRIO Y LATA EN EL PROCESO DE RECICLAJE", UNIVERSIDAD DE CUNDINAMARCA EXTENSIÓN CHÍA, 2018. [En línea]. Disponible: https://repositorio.ucundinamarca.edu.co/bitstream/handle/20.500.12558/2497/Trabajo%20de%20grado.pdf?sequence=3&isAllowed=y

[11] D. Estévez Trigo, "Detección de objetos en imágenes utilizando técnicas de aprendizaje profundo (Deep-Learning)", Universidad de Sevilla, 2021. [En línea]. Disponible:

https://biblus.us.es/bibing/proyectos/abreproy/93729/fichero/TFG-3729+ESTÉVEZ+TRIGO,+DANIEL.pdf

[12] N. B. Bonifaz Oviedo, "DESARROLLO DE UN SISTEMA DE RECOMENDACIÓN INTELIGENTE PARA SELECCIÓN DE PRENDAS UTILIZANDO ANÁLISIS DE IMÁGENES CON REDES NEURONALES.", UNIVERSIDAD SANTO TOMÁS, 2021. [En línea]. Disponible: http://hdl.handle.net/11634/34732

[13] A. Bermudez Castañeda, "Implementación de un sistema para la detección de peligros potenciales para motociclistas basado en información visual.", UNIVERSIDAD SANTO TOMÁS, 2021. [En línea]. Disponible: https://repository.usta.edu.co/bitstream/handle/11634/34736/2021alexanderbermudez.pdf?sequence=1&isAllowed=y

[14] J. D. Giraldo Quiñones, "Clasificador de residuos sólidos haciendo uso de deep learning", Universidad Autónoma de occidente. [En línea]. Disponible: https://red.uao.edu.co/bitstream/handle/10614/14209/T10373_Clasificador%20de%20residuos%20sólidos%20haciendo%20uso%20de%20deep%20learning?sequence=1

[15] "Redes Neuronales artificiales: Qué son y cómo se entrenan ". Xeridia. https://www.xeridia.com/blog/redes-neuronales-artificiales-que-son-y-como-se-entrenan-parte-i.

[16] “Supervised Machine Learning: Regression and Classification”notas de clase, Supervised Machine Learning: Regression and Classification by deeplearning.ai and Universidad de Stanford, Coursera, Enero 2022.

[17] V. Alvear-Puertas, P. Rosero-Montalvo, D. Peluffo-Ordóñez y J. Pijal-Rojas, "Internet de las Cosas y Visión Artificial, Funcionamiento y Aplicaciones: Revisión de Literatura", Enfoque UTE, vol. 8, n.º 1, pp. 244–256, febrero de 2017. Accedido el 28 de noviembre de 2022. [En línea]. Disponible: https://doi.org/10.29019/enfoqueute.v8n1.121.

[18] W. Ejsmont, "Https://www.dbc.wroc.pl/dlibra/publication/44195/edition/39674", Didactics of Mathematics, vol. 13, n.º 17, pp. 13–16, 2016. Accedido el 28 de noviembre de 2022. [En línea]. Disponible: https://doi.org/10.15611/dm.2016.13.02.

[19] K. S. Kumaran y S. Balaji, "SSD: Single shot multibox detector", Acta Scientific Neurology, pp. 03–09, agosto de 2022. [En línea]. Disponible: https://doi.org/10.31080/asne.2022.05.0523

[20] T. Ahmad, Y. Ma, M. Yahya, B. Ahmad, S. Nazir y A. U. Haq, "Object detection through modified YOLO neural network", Scientific Programming, vol. 2020, pp. 1–10, junio de 2020. [En línea]. Disponible: https://doi.org/10.1155/2020/8403262.

[21] "How to create a deep learning dataset using Google Images - PyImageSearch".PyImageSearch.https://pyimagesearch.com/2017/12/04/how-to-create-a-deep-learning-dataset-using-google-images/.

[22] J. Terra. "Pytorch vs tensorflow vs keras: Here are the difference you should know". Simplilearn.com. https://www.simplilearn.com/keras-vs-tensorflow-vs-pytorch-article#:~:text=TensorFlow%20offers%20better%20visualization,%20which,to%20the%20TensorFlow%20Serving%20framework.

9. Anexos

Anexos 1. Código del sistema escrito en lenguaje de programación Python.

```
from clases.sensor import sensor
from clases.motor import motor
from clases.camara2 import camara2
from clases.modelo import modelo
from clases.pantalla import pantalla
from clases.fichero import carpetas
from time import sleep

pin = 19
pin2 = 21
pwm = 12
dire = 18
model = '/home/buho/Documents/proyecto/programa/modelos/modelo1/best.pt'
model2 = '/home/buho/Documents/proyecto/programa/modelos/modelo2/best.pt'
model3 = '/home/buho/Documents/proyecto/programa/modelos/modelo3/best.pt'
imagen = '/home/buho/Documents/proyecto/programa/imagenes/imagen.png'

sensor_entrada = sensor(pin)
sensor_salida = sensor(pin2)
monitor = pantalla()
motorBanda = motor(pwm,dire,sensor_salida)
camara = camara2()
fichero = carpetas()
yolo = modelo(model3,camara,fichero,monitor)


def bucleDeInferencia():
        inferencia,valor = yolo.inferir(imagen,1)
        while(valor!=False):
                print("inferencia",inferencia)
                monitor.imprimir("TOMANDO BOTELLA")
                sleep(5)
                monitor.cerrar_ventana()
                fichero.moverImagenMuestra()
                camara.tomar()
                camara.resize()
                inferencia,valor = yolo.inferir(imagen,2)
        fichero.moverImagenMuestra()
while True:
        monitor.imprimir("ESPERANDO MUESTRA")
        while sensor_entrada.detectar():
                sleep(0.2)
```

```
        motorBanda.avanzar()
        camara.tomar()
        camara.resize()
        bucleDeInferencia()
        motorBanda.avanzar_10()
        while not sensor_salida.detectar():
                camara.tomar()
                bucleDeInferencia()
                motorBanda.avanzar_10()
```

Figura 9.1. Iniciar.py

```
from os import system
import cv2
class camara2:
        def tomar(self):
                system("sudo fswebcam -r 1280x720 --no-banner --png 9
/home/buho/Documents/proyecto/programa/imagenes/imagen1280.png")
        def resize(self):
                dire =
"/home/buho/Documents/proyecto/programa/imagenes/imagen1280.png"
                img = cv2.imread(dire)
                h,w,x = img.shape
                clone  = img[40:650,280:1200]
                res = cv2.resize(clone,(640,480))
                name =
"/home/buho/Documents/proyecto/programa/imagenes/imagen.png"
                cv2.imwrite(name,res)

        def eliminar(self):
                system("sudo rm -r
/home/buho/Documents/proyecto/pythonpru/imagenes/imagen.jpg")
```

Figura 9.2. Camara2.py

```
from os import system
import os
class carpetas:
        def __init__(self):
                self.direccion                                          =
'/home/buho/Documents/proyecto/programa/imagenes/resultados/'
                self.actual = ''
                self.direccion2                                         =
'/home/buho/Documents/proyecto/programa/imagenes/muestras'
                contenido = os.listdir(self.direccion)
```

```
            contenido2 = os.listdir(self.direccion2)
            self.contador = len(contenido)
            self.contador2 = 0
            self.contador3 = len(contenido2)
        def dirCarpeta(self,estado):

            if estado == 1:
                self.contador += 1
                self.contador2 = 1
                self.actual = self.direccion + "parada{}".format(self.contador)
                system("sudo mkdir {}".format(self.actual))
                system("sudo chmod a+w {}".format(self.actual))

            elif estado ==2:
                self.contador2 += 1
            return self.actual

        def moverImagenYEliminarCarpeta(self):
            system("sudo mv {}/resultado/imagen.jpg
{}/imagen{}.jpg".format(self.actual,self.actual,self.contador2))
            system("sudo rm -r {}/resultado".format(self.actual))

        def moverImagenMuestra(self):
            self.contador3 += 1
            system("sudo mv
/home/buho/Documents/proyecto/programa/imagenes/imagen.png
{}/imagen{}.png".format(self.direccion2,self.contador3))
```

Figura 9.3. FIchero.py

```
import torch
import cv2
from matplotlib import pyplot as plt
import numpy as np
from time import sleep

class modelo:
        def __init__(self,modelo,camara,carpeta,monitor):
            self.modelo = torch.hub.load('ultralytics/yolov5', 'custom',
path=modelo)
            self.modelo.conf = 0.48
            self.modelo.iou = 0.4
            self.camara = camara
            self.carpeta = carpeta
            self.monitor = monitor
            self.monitor.imprimir("MODELO CARGADO")
```

```
    def inferir(self,imagen,estado):
            print("realizando inferencia")
            self.resul = self.modelo(imagen)
            print("inferencia realizada")
            if (len(self.resul.pandas().xyxy[0])):
                    self.monitor.imprimir("HAY BOTELLAS")
                    dire = self.carpeta.dirCarpeta(estado)
                    resulDire = dire+"/resultado"
                    self.resul.save(save_dir=resulDire)
                    self.carpeta.moverImagenYEliminarCarpeta()
                    self.resul.show()
                    return self.resul.pandas().xyxy[0],True
            else:
             self.monitor.imprimir("NO HAY BOTELLAS")
       return self.resul.pandas().xyxy[0],False
```

Figura 9.4. Modelo.py

```
import RPi.GPIO as GPIO
from time import sleep

GPIO.setmode(GPIO.BOARD)


class motor:
    def __init__(self,PPwm,PDir,Sensor):
            self.pwm = PPwm
            self.dir = PDir
            self.sensor = Sensor
            GPIO.setup(self.pwm,GPIO.OUT)
            GPIO.setup(self.dir,GPIO.OUT)
            self.pulse = GPIO.PWM(self.pwm,100)
            self.pulse.start(0)

    def avanzar(self):
            self.pulse.ChangeDutyCycle(50)
            while self.sensor.detectar():
                    sleep(0.1)
            self.pulse.ChangeDutyCycle(0)

    def avanzar_10(self):
            contador = 0
            self.pulse.ChangeDutyCycle(50)
            while self.sensor.detectar() ==False:
                    sleep(0.1)
```

```
            while self.sensor.detectar() and contador <20 :
                sleep(0.5)
                contador += 1
            self.pulse.ChangeDutyCycle(0)
```

Figura 9.5. Motor.py

```
from os import system

class pantalla:
    def __init__(self):
        system("export DISPLAY=:0")

    def cerrar_ventana(self):
        system("pkill gpicview")

    def imprimir(self,mensaje):
        print("xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx")
        print("          {}          ".format(mensaje))
        print("xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx")
```

Figura 9.6. Pantalla.py

```
import RPi.GPIO as GPIO
from time import sleep

GPIO.setmode(GPIO.BOARD)

class sensor:
    def __init__(self,pin):
        self.pin = pin
        GPIO.setup(pin,GPIO.IN,pull_up_down = GPIO.PUD_UP)

    def detectar(self) :
        while True:
            input = GPIO.input(self.pin)
            if input == False:
                break
            return True
            sleep(0.2)
        return False
```

Figura 9.7. Sensor.py

Anexos 2. Tablas de resultados obtenidos en las pruebas de selección del modelo

Tabla 9.1. RCNN conjunto de datos combinados

Muestra	Positivo/Positivo	Positivo/Falso	Falso/Positivo	Falso/Falso	Número verdadero
1	1	1	0	0	1
2	1	2	0	0	1
3	1	1	0	0	1
4	1	1	1	0	1
5	1	0	0	0	1
6	1	0	0	0	1
7	0	1	1	0	1
8	1	2	1	0	2
9	1	1	0	0	1
10	1	1	0	0	1
11	1	0	0	0	1
12	1	0	0	0	1
13	0	1	1	0	1
14	0	1	1	0	1
15	0	1	1	0	1
16	1	1	0	0	1
17	1	0	0	0	1
18	1	2	0	0	1
19	1	2	0	0	1
20	0	1	1	0	1
21	1	0	0	0	1
22	1	0	0	0	1
23	1	0	0	0	1

Tabla 9.2. RCNN conjunto de datos separados

Muestra	Positivo/Positivo	Positivo/Falso	Falso/Positivo	Falso/Falso	Número verdadero
1	1	4	0	0	1
2	0	5	1	0	1

3	0	1	1	0	1
4	0	1	1	0	1
5	0	2	1	0	1
6	0	2	1	0	1
7	0	1	1	0	1
8	0	1	1	0	2
9	0	1	1	0	1
10	0	1	1	0	1
11	0	0	0	0	1
12	0	0	0	0	1
13	0	0	0	0	1
14	0	0	0	0	1
15	0	0	0	0	1
16	0	0	0	0	1
17	0	0	0	0	1
18	0	0	0	0	1
19	0	0	0	0	1
20	0	0	0	0	1
21	0	0	0	0	1
22	0	0	0	0	1
23	0	0	0	0	1

Tabla 9.3. SSD conjunto de datos juntos

Muestra	Positivo/Positivo	Positivo/Falso	Falso/Positivo	Falso/Falso	Número verdadero
1	1	0	0	0	1
2	1	0	0	0	1
3	1	0	0	0	1
4	1	1	0	0	1
5	1	0	0	0	1
6	1	0	0	0	1
7	1	0	0	0	1
8	2	0	0	0	2
9	1	0	0	0	1

10	1	0	0	0	1
11	1	0	0	0	1
12	1	0	0	0	1
13	1	0	0	0	1
14	1	1	0	0	1
15	1	0	0	0	1
16	1	0	0	0	1
17	0	1	1	0	1
18	1	0	0	0	1
19	0	1	1	0	1
20	1	1	0	0	1
21	1	3	0	0	1
22	0	0	1	0	1
23	1	0	0	0	1

Tabla 9.4. SSD conjunto de datos separados

Muestra	Positivo/Positivo	Positivo/Falso	Falso/Positivo	Falso/Falso	Número verdadero
1	1	0	0	0	1
2	1	0	0	0	1
3	1	0	0	0	1
4	1	0	0	0	1
5	1	0	0	0	1
6	0	1	0	0	1
7	1	0	0	0	1
8	2	0	0	0	2
9	1	0	0	0	1
10	1	0	0	0	1
11	1	0	0	0	1
12	1	0	0	0	1
13	0	1	0	0	1
14	0	1	0	0	1
15	1	0	0	0	1
16	1	0	0	0	1

17	1	0	0	0	1
18	1	0	0	0	1
19	1	0	0	0	1
20	1	0	0	0	1
21	0	1	1	0	1
22	1	0	0	0	1
23	1	0	0	0	1

Tabla 9.5. YOLOV5 conjunto de datos juntos

Muestra	Positivo/Positivo	Positivo/Falso	Falso/Positivo	Falso/Falso	Número verdadero
1	1	0	0	0	1
2	1	0	0	0	1
3	1	0	0	0	1
4	1	0	0	0	1
5	1	0	0	0	1
6	1	0	0	0	1
7	1	0	0	0	1
8	2	0	0	0	2
9	1	0	0	0	1
10	1	0	0	0	1
11	1	0	0	0	1
12	1	0	0	0	1
13	1	0	0	0	1
14	1	0	0	0	1
15	1	0	0	0	1
16	1	0	0	0	1
17	1	0	0	0	1
18	1	0	0	0	1
19	0	0	1	0	1
20	1	0	0	0	1
21	1	0	0	0	1
22	1	0	0	0	1

23	1	0	0	0	1

Tabla 9.6. YOLOv5 conjunto de datos separados

Muestra	Positivo/Positivo	Positivo/Falso	Falso/Positivo	Falso/Falso	Número verdadero
1	1	0	0	0	1
2	1	0	0	0	1
3	1	0	0	0	1
4	1	0	0	0	1
5	1	0	0	0	1
6	0	0	1	0	1
7	1	0	0	0	1
8	2	0	0	0	2
9	1	0	0	0	1
10	1	0	0	0	1
11	1	0	0	0	1
12	1	0	0	0	1
13	0	1	1	0	1
14	0	0	1	0	1
15	1	0	0	0	1
16	1	0	0	0	1
17	1	0	0	0	1
18	1	0	0	0	1
19	1	0	0	0	1
20	1	0	0	0	1
21	0	0	1	0	1
22	1	0	0	0	1
23	1	0	0	0	1