UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
PROGRAMA DE MAESTRÍA Y DOCTORADO EN INGENIERÍA
INGENIERÍA ELÉCTRICA - PROCESAMIENTO DIGITAL DE SEÑALES

DETECCIÓN DE OBJETOS CON REDES NEURONALES PROFUNDAS PARA UN ROBOT DE SERVICIO

TESIS:
QUE PARA OPTAR POR EL GRADO DE:
MAESTRO EN INGENIERÍA

PRESENTA:
EDGAR ROBERTO SILVA GUZMÁN

TUTOR:
DR. JESÚS SAVAGE CARMONA
POSGRADO DE INGENIERÍA ELÉCTRICA

CIUDAD UNIVERSITARIA, CD. MX. NOVIEMBRE 2019

Jurado Asignado:

Presidente: Dr. Rascón Estebané Caleb

Secretario: Dr. Escalante Ramírez Boris

1er. Vocal: Dr. Savage Carmona Jesús

2do. Vocal: Dra. Olveres Montiel Jimena

3er. Vocal: Dr. Rivera Rivera Carlos

Lugar o lugares donde se realizó la tesis: Ciudad Universitaria

TUTOR DE TESIS:
Dr. Savage Carmona Jesús

Firma

Agradecimientos

- *A mi padre Francisco y mi madre Azucena, por su amor incondicional y por dedicar sus vidas a enseñarme a ser una mejor persona.*
- *A toda mi familia, Ángel, Alejandro, Ana, Leonel, Carmelita, Rubén, Blanca, Diana, Ricardo, Paola, Emi, Anabel, Frida, Camilo y Jorge, por enseñarme el gran valor de vivir al lado de tus seres queridos.*
- *A Ana, que compartió los mejores momentos conmigo y me motivó a seguir y vencer cada obstáculo de la vida.*
- *Al Dr. Jesús Savage, por introducirme a este mundo de la robótica y por darme la oportunidad de crecer junto a sus proyectos.*
- *A mi jurado, Caleb Rascón, Boris Escalante, Jimena Olveres y Carlos Rivera por su asesoría a este proyecto de tesis.*
- *A mis compañeros Manuel, Edgar, Adrián, Coyotzi, Hugo Sánchez, José Carlos, Reynaldo, Julio Cruz, Hugo Leon, Jesús Cruz, José Luis, Marco, Julio Martínez, Diego, Daniel, Angélica y a los demás integrantes del laboratorio de Bio-Robótica, por compartir sus conocimientos, darme buenos consejos, vivir experiencias inolvidables en las competencias de robótica y por formar parte de mi segunda familia.*
- *Al posgrado de Procesamiento de Señales, que me dio todos los elementos para mi formación profesional.*
- *Al CONACYT por los recursos otorgados para cursar este grado de maestría.*
- *Al proyecto PAPIIT IG100818.*

Resumen

Las Redes Neuronales Convolucionales son el estado del arte en la detección de objetos. Este tipo de redes neuronales se entrenan con grandes bases de datos de imágenes, como ImageNet o COCO, que incluyen más 330 000 imágenes etiquetadas. Sin embargo, el tratar de entrenar este tipo de redes con objetos propios, u objetos que no se encuentran en una base de datos pública, genera algunos inconvenientes: se requieren grandes cantidades de imágenes etiquetadas por cada clase u objeto y las imágenes deben ser etiquetadas por un software especializado, ya que el etiquetar a mano cada objeto podría tardar días de trabajo. En el presente proyecto de tesis, se plantea un sistema capaz de generar una base de datos de imágenes etiquetada de manera automatizada a partir de un archivo de video, que posteriormente se utiliza para el entrenamiento de una Red Neuronal Convolucional (YOLOv3). Además, se plantea un sistema de detección de objetos que utiliza esta Red Neuronal Convolucional en un robot de servicio.

El primer sistema está conformado por dos módulos principales desarrollados en ROS: el primer módulo segmenta un objeto de cada cuadro proveniente de un archivo de video. El segundo módulo crea la base de datos de imágenes sintéticas, es decir, escenas artificiales que contienen los objetos segmentados y etiquetados del módulo anterior, después, se aplican técnicas de aumentado de datos para generar un conjunto de datos más robusto. El sistema puede generar miles de imágenes etiquetadas con 30 objetos aproximadamente en cada una en minutos. Por último, se configuran los parámetros de entrenamiento para la Red Neuronal Convolucional YOLOv3.

En el segundo sistema, se propone una metodología para que, un robot de servicio, pueda manipular un objeto de manera autónoma. Las entradas del sistema son las imágenes adquiridas por una cámara RGB-D. Después, se detectan los objetos en un espacio bidimensional utilizando el modelo entrenado de la Red Neuronal Convolucional. Posteriormente, se hace el cálculo del centroide de cada objeto en un espacio tridimensional y se evalúa la mejor forma de manipular el objeto. Por último, se evalúa si el objeto ha sido manipulado.

El robot Takeshi y la robot Justina son los principales proyectos de robots de servicio desarrollados en el laboratorio de Bio-Robótica de la Facultad de Ingeniería de la UNAM. Por lo cual, uno de los objetivos principales de este proyecto de tesis es aplicar los sistemas propuestos a estos robots de servicio.

Abstract

Convolutional Neural Networks are the state of the art in object detection. These types of neural networks are trained with large image datasets, such as ImageNet or COCO, which include more than 330,000 tagged images. However, the training of this type of networks with own objects, or objects that are not in a public dataset, result in some problems: large amounts of labeled images are required by each class or object and the images must be labeled by specialized software, because labeling each object by hand could take a days of work. In this thesis project, we propose a system capable of generating a labeled image dataset automatically from video files, which is subsequently used for the training of a Convolutional Neural Network (YOLOv3). In addition, a system is proposed using the detection of objects with the Convolutional Neural Network in a service robot.

The first system is made up of two main modules developed in ROS: the first module segments an object from each frame from a video file. The second module creates the database of synthetic images, that is, artificial scenes containing the segmented and labeled objects of the previous module, then, data augmentation techniques are applied to generate a more robust set of data. The system can generate thousands of labeled images with approximately 30 objects in each one in minutes. Finally, the training parameters for the YOLOv3 Convolution Neural Network are configured.

In the second system, a methodology is proposed so that a service robot can manipulate an object autonomously. The system inputs are the images acquired by an RGB-D camera. Then, the objects are detected in a two-dimensional space using the trained model of the Convolutional Neural Network. Subsequently, the centroid calculation of each object in a three-dimensional space is made and the best way to manipulate the object is evaluated. Finally, it is evaluated if the object was manipulated.

The Takeshi and the Justina robots are the main service robot projects developed in the Bio-Robotics laboratory of the UNAM School of Engineering. Therefore, one of the main objectives of this thesis project is to apply the proposed systems to these service robots.

Índice general

1. Introducción **9**
1.1. Justificación 11
1.2. Hipótesis 13
1.3. Objetivos 13
1.4. Estructura de la tesis 14

2. Marco Teórico **17**
2.1. Procesamiento digital de imágenes 17
2.2. Visión computacional 18
2.3. Modelos de color 18
2.3.1. Modelo RGB 18
2.3.2. Modelo HSV 19
2.4. Segmentación de imágenes 21
2.5. Umbralización 22
2.6. Segmentación basada en regiones 24
2.6.1. Conectividad de pixeles 25
2.6.2. Crecimiento de regiones 25
2.6.3. División y fusión de regiones 26
2.7. Operaciones morfológicas 27
2.7.1. Teoría de conjuntos 27
2.7.2. Operaciones lógicas 29
2.7.3. Dilatación 30
2.7.4. Erosión 31
2.7.5. Apertura y Cerradura 31
2.8. Filtro Gaussiano 33
2.9. Detector de bordes de Canny 34

3. Aprendizaje Profundo **37**
3.1. Redes neuronales artificiales 38
3.2. Algoritmos de Aprendizaje Automatizado 40
3.3. Redes neuronales convolucionales 41

3.3.1. Capas convolucionales 42
3.3.2. Capas de reducción 44
3.3.3. Capas totalmente-conectadas 44
3.3.4. Capas residuales (ResNet) 45
3.4. Retro-propagación 46
3.4.1. Algoritmo del descenso del gradiente 47
3.4.2. Transferencia de conocimiento 47
3.4.3. Sobre-Ajuste y Sub-Ajuste 48
3.4.4. Algunas terminologías del aprendizaje profundo 49
3.5. YOLO 49
3.5.1. Arquitectura neuronal 49
3.5.2. Funcionamiento 52

4. Herramientas de Desarrollo **53**
4.1. Herramientas de software 53
4.1.1. OpenCV 53
4.1.2. ROS 54
4.2. Herramientas de hardware 55
4.2.1. Robots de servicio Takeshi y Justina 55
4.2.2. Adquisición de imágenes 57
4.2.3. Computadoras de desarrollo 57

5. Desarrollo **59**
5.1. Sistema de creación de un conjunto de escenas sintéticas . . . 59
5.1.1. Módulo de segmentación de objetos 59
5.1.2. Módulo de creación de escenas sintéticas 68
5.1.3. Entrenamiento de la red neuronal convolucional YOLOv3 81
5.2. Sistema de detección de objetos 83
5.2.1. DarkNet-ROS 83
5.2.2. Nodo de detección de objetos 83

6. Pruebas y Resultados **89**
6.1. Pruebas y Resultados del sistema de creación de un conjunto de escenas sintéticas 89
6.1.1. Pruebas del módulo de segmentación de objetos 89
6.1.2. Resultados del módulo de segmentación de objetos 91
6.1.3. Análisis de resultados 92
6.1.4. Pruebas del módulo de creación de imágenes sintéticas 93
6.1.5. Resultados del módulo de creación de imágenes sintéticas 93
6.1.6. Análisis de resultados 94

6.1.7. Entrenamiento de la red neuronal convolucional 95
6.1.8. Análisis de resultados 99
6.1.9. Conjunto de prueba para la evaluación de los modelos neuronales 99
6.1.10. Análisis de resultados 102
6.1.11. Pruebas del nodo de detección de objetos 104
6.1.12. HSR-Challenge y RoboCup@home 105

7. Conclusiones y Trabajo futuro **109**
7.1. Conclusiones 109
7.1.1. Módulo de segmentación de objetos 109
7.1.2. Módulo de creación de imágenes sintéticas 109
7.1.3. Nodo de detección de objetos 110
7.2. Trabajo futuro 110

Capítulo 1

Introducción

En la actualidad, la robótica ha tenido un avance significativo debido a los nuevos desarrollos tanto de software como de hardware. Debido a esto, gran cantidad de actividades cotidianas tratan de ser emuladas en un robot, ya sea para aumentar la producción en una fábrica, explorar lugares de difícil acceso o para interactuar a través de lenguaje natural con los seres humanos.

La robótica de servicio es una de las ramas de la robótica donde ha habido mayor avance. Estos robots están diseñados para hacer la vida más cómoda al ser humano, principalmente en el hogar o en una oficina, donde se encargan de realizar actividades sencillas, como barrer y trapear, tal es el ejemplo del robot Roomba [1]. Pero también existen robots de servicio de uso no comercial que hacen actividades complejas, como comunicarse a través de lenguaje natural con un humano, recibir una tarea específica a través de comandos de voz, desplazarse de una habitación a otra de manera autónoma, detectar y manipular objetos e interactuar con su entorno para poder hacer frente a situaciones imprevistas, como la evasión de obstáculos. En la mayoría de los casos, estos robots están diseñados específicamente para ser usados en investigaciones académicas.

El sistema de detección de objetos es una de las principales cualidades de un robot de servicio, de modo que, para manipular un objeto, el robot debe de conocer la posición exacta de éste en un entorno no controlado, es decir, el objeto puede aparecer en distintos lugares, posiciones y orientaciones. Una vez identificada la clasificación y la posición del objeto, el robot debe colocar su manipulador en un punto donde pueda tomarlo de manera correcta. Cabe destacar que un sistema de clasificación no es lo mismo que un sistema de detección. Un sistema de clasificación nos dice cuál es la probabilidad de que uno o varios objetos se encuentren en la escena, sin darnos más información. En cambio, un sistema de detección es más complejo, ya que además de decir cuál es la probabilidad de que uno o varios objetos se encuentren en la escena,

nos dice cuales son las coordenadas relativas del o de los objetos respecto a ésta.

Recientes resultados indican que los descriptores extraídos de las Redes Neuronales Convolucionales (CNN, por sus siglas en inglés) son muy poderosos. Razavian reporta en [2] una serie de experimentos aplicados a diversas tareas de reconocimiento donde utiliza el modelo *overfeat network*[3], el cuál ha sido pre-entrenado para el reconocimiento de objetos utilizando la base de datos de imágenes ILSVRC13[4]. En este artículo, se menciona que se puede sustituir el *pipeline* tradicional de los métodos *estado del arte* para las tareas reconocimiento por una **CNN** conectada a un clasificador, en este caso un SVM (Máquina de Soporte Vectorial, por su traducción al español) (figura 1.1 parte superior). Posteriormente, se comparan los resultados de los métodos siguientes para diversas tareas de reconocimiento: El mejor estado del arte, una Red Neuronal Convolucional pre-entrenada, una Red Neuronal Convolucional pre-entrenada con aumentado de datos y una Red Neuronal Convolucional especializada. Con estos resultados, se puede concluir que las Redes Neuronales Convolucionales superan en desempeño a la mayoría de los métodos *estado del arte*, por ejemplo: en clasificación de objetos, clasificación de escenas, detección de atributos humanos, sub-categorización de aves, entre otras (figura 1.1 parte inferior).

Con base en las conclusiones anteriores y en los buenos resultados que se reportan, este proyecto de tesis plantea hacer uso de las Redes Neuronales Convolucionales pre-entrenadas para solucionar la problemática de la detección de objetos en un robot de servicio.

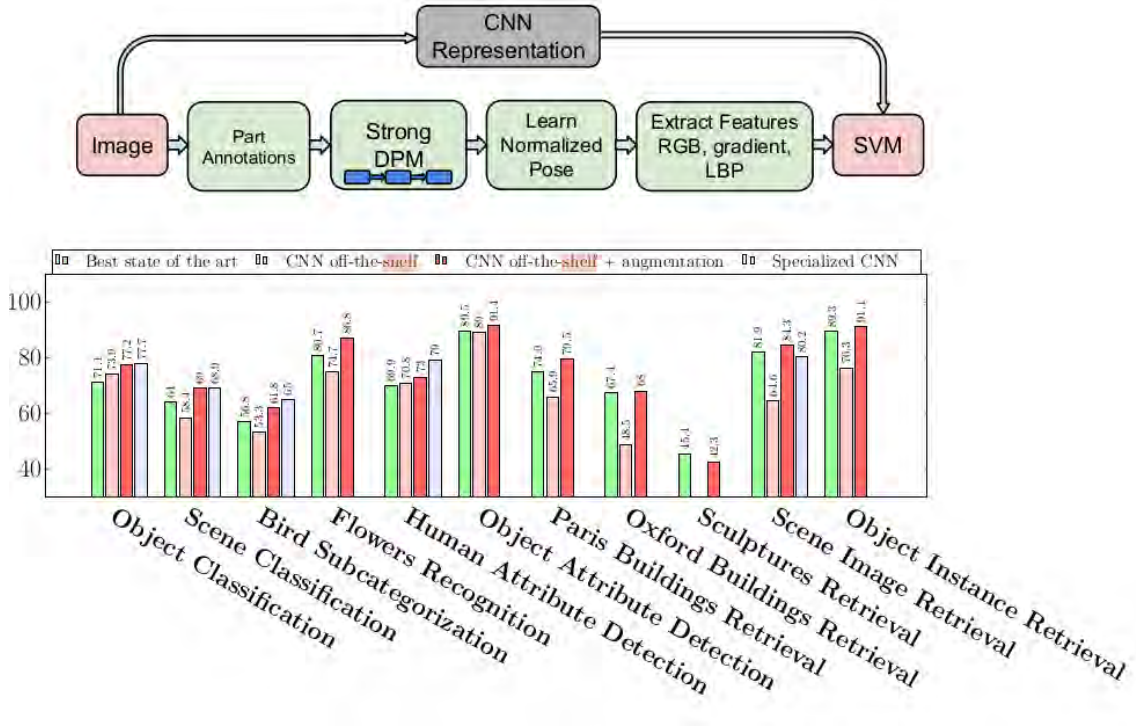


Figura 1.1: Parte Superior). La representación de una CNN reemplaza el *pipeline* tradicional de los métodos *estado del arte*.
Parte inferior). Una CNN con aumentado de datos y con un clasificador SVM supera el desempeño de los métodos *estado del arte* en múltiples tareas.
Figura tomada de [2].

1.1. Justificación

En 1997, la supercomputadora llamada *Deep Blue*, desarrollada por la empresa estadounidense IBM, logró vencer al campeón mundial de ajedrez, Gary Kaspárov, dando por terminado uno de los retos más importantes de de la comunidad de Inteligencia Artificial (AI). En ese mismo año, la misión de la NASA *Mars Pathfinder* logró un aterrizaje exitoso, y con el primer sistema robótico autónomo *Sojourner*, logro recopilar miles de datos sobre la superficie marciana. Ambos eventos causaron gran revolución en la comunidad de Inteligencia Artificial, y motivaron para seguir planteando nuevos y determinantes retos en este ámbito.

La RoboCup (Competencia Internacional de Robótica)[5] nació con el propósito de motivar el desarrollo de nuevos sistemas de Inteligencia Artificial

capaces de ser emulados en un robot autónomo. Un grupo de Investigadores Japoneses propusieron el juego de *Fotball Soccer* como marco de referencia para estos retos, ya que con este juego, se pueden hacer desarrollos importantes en varios temas, como la visión computacional, la navegación autónoma, la manipulación y la interacción humano-máquina. De esta manera, en el año de 1997, se convocó a la primera competencia oficial de robótica a nivel mundial, teniendo como principal objetivo para mediados del siglo XXI, desarrollar un equipo de robots autónomos capaces de competir con un equipo humano en un partido de Fooball Soccer[5].

A raíz del éxito que tuvieron las primeras competencias de RoboCup, se crearon varias ligas con diferentes objetivos en específico. En la actualidad existen las siguientes: RoboCupSoccer, RoboCupSoccerSimulation, RoboCupIndustrial, RoboCupRescue y RoboCup@Home[6]. La liga RoboCup@Home es de las más complejas, donde un robot de servicio semihumanoide, asiste a personas con capacidades diferentes o de edad avanzada a llevar a cabo las tareas de un hogar. En esta liga existen varias pruebas que evalúan la habilidades y el desempeño del robot en un ambiente real no estandarizado. Las actividades se centran en los siguientes dominios: Interacción humano-máquina, navegación y mapeo de ambientes dinámicos, manipulación de objetos, inteligencia artificial, visión computacional y detección de objetos. Posteriormente, los países participantes crearon sus propias competencias nacionales, con la finalidad de presentar pruebas similares a las existentes en RobobCup. En México existe el Torneo Mexicano de Robótica[7], en Japón la competencia World Robot Summit (WRS)[8], además de competencias vía remota a través de videoconferencias (HSR-Challenge[9]).

En la mayor parte de las pruebas de estas competencias se utilizan sistemas de detección de objetos. Estos sistemas deben ser suficientemente robustos para detectar gran cantidad de categorías de objetos en ambientes no controlados, es decir, con cambios de iluminación constantes, con objetos ocluidos, objetos rotados y con objetos deformables. El sistema debe de tener una respuesta precisa en tiempo real, ya que se establece una duración para cada prueba.

Debido a esta problemática, se plantea el presente proyecto de tesis, cuyo propósito es ser el sistema de detección de objetos para los robots de servicio Takeshi y Justina.

En el laboratorio de Bio-Robótica de la Facultad de Ingeniería de la UNAM[10] se tienen varios proyectos de robots de servicio liderados por el Dr. Jesús Savage Carmona:

- El proyecto Justina. Justina es un robot de servicio semihumaniode

desarrollado totalmente en la Facultad de Ingeniería, lleva 13 años participando en la RoboCup, destacando en los últimos dos años con un segundo lugar a nivel mundial.

- El proyecto Takeshi es un robot semihumanoide de mediano tamaño desarrollado por la empresa de Toyota, ha participado 2 años en la RoboCup, obteniendo un segundo lugar a nivel mundial, y una vez en el WRS obteniendo buenos resultados. Además, Takeshi participa 3 veces al año en demos vía remota transmitidos por videoconferencia.

1.2. Hipótesis

Al comienzo de este proyecto de tesis se generó la siguiente hipótesis:

"El entrenamiento de un modelo neuronal con imágenes sintéticas generadas de manera automática, tendrá altos índices de precisión en la detección de objetos en una escena real. La implementación de este modelo mejorará el sistema de detección de objetos en un robot de servicio, aumentando los índices de confidencia en la etapa de reconocimiento y mejorando la detección de objetos en una escena tridimensional."

Para comprobar la hipótesis, se desarrollarán varios sistemas computacionales y se realizarán pruebas experimentales en el robot de servicio HSR (Takeshi) del laboratorio de Bio-Robótica.

1.3. Objetivos

El objetivo principal para el presente proyecto de tesis es el siguiente:

- Crear un conjunto de escenas sintéticas con objetos de interés etiquetados para re-entrenar la red neuronal convolucional YOLOv3. El modelo neuronal obtenido será utilizado como sistema de detección de objetos en un robot de servicio.

Por lo cual, se plantean los siguientes objetivos específicos:

- Desarrollar un sistema de visión computacional que permita segmentar múltiples categorías de objetos utilizando la información de una imagen bidimensional de una cámara digital compacta, a fin de que se cree un conjunto de imágenes de objetos segmentados con distintas posiciones y orientaciones.

- Desarrollar un sistema de visión computacional que cree un conjunto de imágenes sintéticas, es decir, escenas artificiales que contengan los objetos segmentados del conjunto de imágenes del sistema anterior. Además, para robustecer el conjunto de imágenes se hace uso de técnicas de aumento de datos (del inglés, *data augmentation*). Este sistema debe generar un archivo de configuración que contenga las etiquetas de los objetos segmentados (posición relativa del objeto respecto a la imagen).
- Realizar el re-entrenamiento del modelo neuronal YOLOv3 con el conjunto de imágenes sintéticas generado.
- Desarrollar un módulo que utilice el modelo neuronal re-entrenado como un sistema de detección de objetos para un robot de servicio.

1.4. Estructura de la tesis

En el presente proyecto de tesis, se tratan diversos temas de visión computacional y de aprendizaje profundo, además se realizan distintas pruebas experimentales de los sistemas planteados y se obtienen resultados aceptables. Por lo cual, se plantea la siguiente estructura del proyecto de tesis:

- En el segundo capítulo, se abordan los fundamentos y la teoría que se utilizaron para el desarrollo del proyecto. Se da una descripción de los **modelos de color**, los métodos de **segmentación de imágenes**, las **operaciones morfológicas**, **teoría de conjuntos** y el algoritmo de detección de bordes de **Canny**.
- En el tercer capítulo, se expone la teoría relacionada con el aprendizaje profundo (en inglés, *deep learning*). Se da una explicación sobre las **redes neuronales profundas**, los tipos de **capas neuronales** que existen y los parámetros para su entrenamiento. Se explica el funcionamiento de la red neuronal convolucional **YOLOv3**.
- En el cuarto capítulo, se abordan los elementos de **hardware** y **software** que se utilizaron para el desarrollo de este proyecto de tesis.
- En el quinto capítulo, se expone de manera detallada cuál fue el procedimiento para desarrollar los siguientes sistemas: el **sistema de segmentación de imágenes** a partir de un archivo de video, el **sistema de creación de imágenes sintéticas** con aumento de datos, el **entrenamiento de la red neuronal convolucional** y el **sistema de detección de objetos en un robot de servicio**.

- En el sexto capítulo, se abordan las **pruebas** de los sistemas de visión computacional, el entrenamiento de la red neuronal convolucional y del sistema de detección de objetos en el robot de servicio Takeshi. De igual manera, se da un análisis a los **resultados** obtenidos.
- Por último, en el séptimo capítulo, se exponen las **conclusiones** del proyecto de tesis y se plantea el **trabajo a futuro**.

Capítulo 2

Marco Teórico

En este capítulo, se presentan los principales fundamentos teóricos sobre los cuales está planteado el presente proyecto de tesis. Se parte de la problemática de la segmentación (casi perfecta) de objetos en imágenes bidimensionales. Se utiliza el espacio de color HSV, los métodos de segmentación por regiones, el algoritmo de detección de bordes de Canny[11], y junto a una metodología propuesta, se obtiene una base de datos de imágenes de objetos correctamente segmentados.

2.1. Procesamiento digital de imágenes

El Procesamiento digital de imágenes es una disciplina en la que, tanto la entrada como la salida del sistema, es una imagen digital. Se enfoca en dos grandes objetivos: mejorar una imagen para su correcta interpretación humana y el procesamiento de datos de una imagen, ya sea para su almacenamiento, transmisión o representación, con la finalidad ser interpretados en maquinas autónomas.

Una imagen es definida por [12] como una función bidimensional $f(x, y)$, donde x y y son las coordenadas espaciales de un plano y la amplitud f de cada par de coordenadas (x, y), es llamado intensidad o nivel de gris en ese punto de la imagen. Cuando x, y y $f(x, y)$ son finitas, es decir, son cantidades discretas, la imagen es llamada **imagen digital**, la cual está compuesta por un número finito de elementos llamados **pixeles** (en singular **pixel**), cada uno con su particular localización y valor.

2.2. Visión computacional

A diferencia del Procesamiento digital de imágenes, la Visión computacional es una rama de la Inteligencia Artificial (IA) que tiene como objetivo utilizar imágenes digitales para emular el comportamiento de la visión humana, incluyendo métodos de aprendizaje, métodos para hacer inferencias y métodos para tomar decisiones basadas únicamente en las entradas visuales del sistema[12]. Dentro de la Visión computacional hay tres enfoques principales para los procesos computarizados: bajo, medio y alto nivel de procesamiento. En el bajo nivel se hacen las operaciones primitivas, como el preprocesamiento de imágenes para mejorar el contraste, la nitidez y reducir el ruido en la imagen. A la entrada y a la salida de este proceso se tienen imágenes. Los procesos de medio nivel involucran operaciones como la segmentación, ya sea por umbrales o por regiones (segmentación de objetos), o la clasificación (reconocimiento) de objetos individuales. A la entrada de este proceso se tienen imágenes, y a la salida atributos o características extraídas de los objetos, por ejemplo: bordes, esquinas o descriptores de un punto de interés. Finalmente, en el alto nivel de procesamiento se interpreta un conjunto de características o descriptores, se reconocen patrones o se interpretan escenas, infiriendo información acerca de los objetos contenidos en una imagen. En este proceso, se tratan de emular las funciones cognitivas relacionadas con la visión humana.[12]

2.3. Modelos de color

El uso del color en el procesamiento digital de imágenes se fundamenta principalmente por dos factores: Primero, el color es un poderoso descriptor que facilita el reconocimiento y la segmentación de objetos dentro de una escena. Segundo, el ojo humano puede distinguir miles de intensidades y tonos de color comparado con solo dos docenas de tonos en gris.

El objetivo de un modelo de color (también llamado *espacio de color*) es simplificar la representación de un color en forma numérica de manera estandarizada. De tal modo que, un modelo de color especifica un sistema de coordenadas y un subespacio en el cual cada color es representado por un punto[12].

2.3.1. Modelo RGB

El modelo **RGB** es un modelo de color lineal que se basa en un sistema de coordenadas cartesianas, en el cual se describe a cada color a partir de

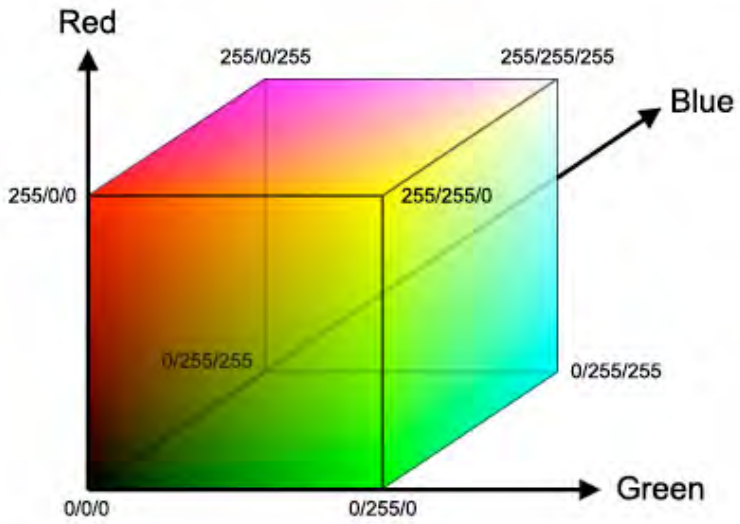


Figura 2.1: Espacio de color RGB.

una combinación de sus componentes espectrales primarias: **rojo, verde y azul**. El subespacio es conformado por un cubo unitario (figura 2.1) donde las componentes RGB se encuentran en los tres vértices principales. El color negro corresponde al origen del cubo unitario (donde cada componente tiene valor cero) y el color blanco será el vértice mas alejado al origen (donde cada componente tiene valor 1). De tal manera que la escala de grises es la diagonal que une al vértice negro con el vértice blanco. [12]. Por conveniencia, se asume que el valor de todas las componentes han sido normalizados.

La representación de las imágenes en el modelo RGB consiste en tres planos independientes, formados por las intensidades de cada una de las componentes RGB. La combinación de un punto en cada uno de los planos tiene como nombre *pixel*. Se le llama *profundidad de color* al número de bits utilizados para representar cada pixel. El término *color verdadero* se utiliza para las imágenes donde cada una de sus componentes es representada por 8 bits, es decir, 24 bits en tres diferentes planos.[13]

2.3.2. Modelo HSV

Los modelos de color lineales, como el modelo RGB, son idóneos para hacer implementaciones de hardware (como las pantallas LED), ya que coinciden con el hecho de que los colores primarios (rojo, verde y azul) son altamente perceptibles al ojo humano. Sin embargo, los modelos de color li-

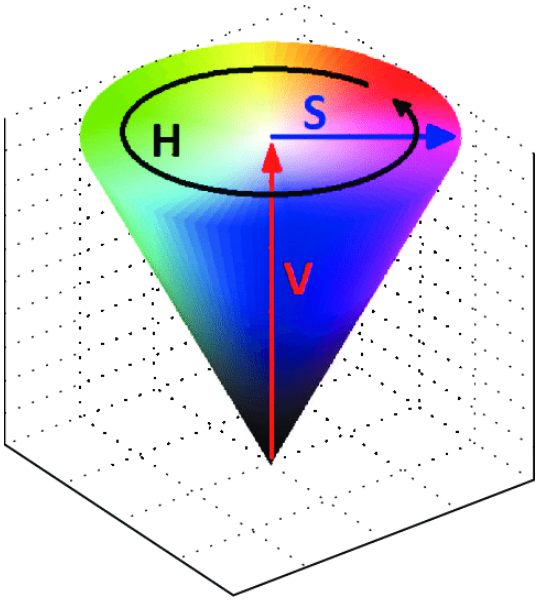


Figura 2.2: Espacio de color HSV. Figura tomada de [14].

neales son inadecuados para describir los colores en términos prácticos para la interpretación humana, por ejemplo, para describir el color de una computadora no se especifica el porcentaje de cada color primario que la compone. De igual manera, el ser humano no piensa que una imagen de color es la composición de tres imágenes que forman una imagen principal. Por estas razones surge el espacio de color **HSV** (*hue, saturation, value*), el cual se representa en un espacio tridimensional formado por un cono invertido (figura 2.2).

Los seres humanos describen un objeto de color en términos de su **tono** (en inglés *hue*), su **saturación** (en inglés *saturation*) y su **brillo** (en inglés *brightness*). El tono es una propiedad que describe un color puro (rojo, verde, azul, etc) y se representa a lo largo de la circunferencia del cono invertido (0°- 360°). La saturación representa el grado en que un color puro es diluido por la luz blanca. Un color muy saturado refleja colores vivos y un color poco saturado se decolora pasando por grises claros hasta llegar al color blanco. El valor o brillo describe que tanta luz refleja un color. Un color con poco brillo pasará a los grises oscuros hasta llegar al color negro.[12]

El modelo HSV surge de una transformación no lineal del modelo RGB en coordenadas cilíndricas, las siguientes ecuaciones se utilizan para dicha transformación:

Siendo R,G,B las compenentes rojo, verde y azul (red, green, blue) respectivamente

$$
\begin{aligned}
R' &= R/255 \\
G' &= G/255 \\
B' &= B/255 \\
Cmax &= max(R', G', B') \\
Cmin &= min(R', G', B') \\
\Delta &= Cmax - Cmin
\end{aligned}
$$

$$
H = \begin{cases}
indefinido & \Delta = 0 \\
60\left(\frac{G'-B'}{\Delta}\right) & Cmax = R' \\
60\left(\frac{B'-R'}{\Delta} + 2\right) & Cmax = G' \\
60\left(\frac{R'-G'}{\Delta} + 4\right) & Cmax = B'
\end{cases}
$$

$$
S = \begin{cases}
0 & Cmax = 0 \\
\frac{\Delta}{Cmax} & V \neq 0
\end{cases}
$$

$$
V = Cmax
$$

Uno de los objetivos principales de este proyecto de tesis es, segmentar objetos de una escena con un fondo controlado. En el modelo de HSV, la propiedad **tono** (*hue*) es robusta a condiciones no controladas del ambiente o de la escena de captura. Por esta razón, se eligió este modelo de color para llevar a cabo el presente proyecto de tesis.

2.4. Segmentación de imágenes

La segmentación de imágenes es el proceso por el cual se descompone una imagen en un conjunto de regiones, que ayudan a analizar de una forma más sencilla, las características más significativas de ésta. La finalidad de este proceso es definir regiones de interés, por ejemplo, en una imagen donde aparece una cabaña cerca de un bosque, el objetivo es extraer las regiones del bosque, del cielo, de la cabaña, del piso, etc., de esta forma se puede

llevar a cabo un procesamiento de alto nivel más acorde con los objetivos del sistema. De la misma forma, los bordes pueden delimitar varios objetos sobre una misma superficie, por ejemplo, al delimitar todas las monedas que hay sobre una mesa.

Por ende, las regiones se pueden definir como un conjunto de bordes de pixeles que tienen un forma particular o una característica en común, por ejemplo, una forma poligonal (circular, elíptica, cuadrada, etc.) o una distribución de color similar (el verde de las hojas de los árboles o el azul del mar)[15].

La segmentación es el paso fundamental del procesamiento de medio nivel. A la salida de este proceso, se obtiene una imagen que se utiliza para el procesamiento de alto nivel, como el reconocimiento de patrones o interpretación de escenas. El éxito en el procesamiento digital de imágenes depende de una buena segmentación, por lo cual, hacer una segmentación exacta aborda un problema complejo.

Dentro de los métodos de segmentación de imágenes existen dos enfoques principales: segmentación por discontinuidad, como la *umbralización*, y segmentación por similaridad, como la *segmentación basada en regiones*.

2.5. Umbralización

Debido a la facilidad de su implementación, la umbralización forma parte fundamental dentro las técnicas de segmentación por discontinuidad. La figura 2.3 muestra un histograma de los niveles de gris correspondientes a una imagen, $f(x, y)$, compuesta en su mayoría, por dos objetos luminosos dentro de un fondo oscuro, de tal manera que estos se tienen agrupados en dos modos dominantes.

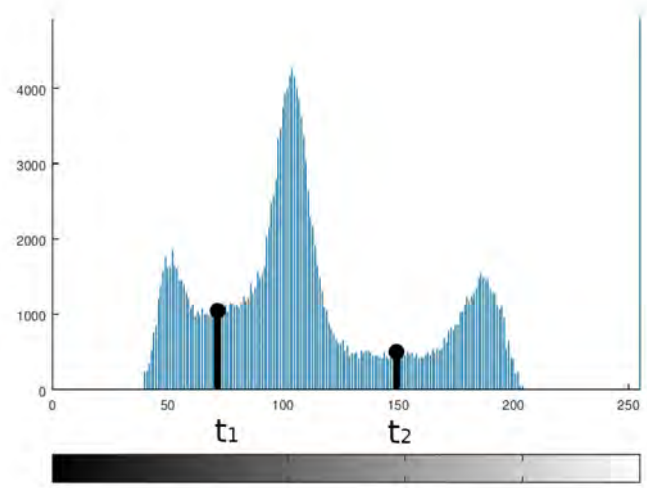


Figura 2.3: Histograma de niveles de gris

Un procedimiento sencillo para extraer los objetos luminosos del fondo oscuro, es seleccionar los umbrales t_1 y t_2 que delimitarán estas tres regiones. De modo que cualquier punto (x, y) para el cual $f(x, y) > t_2$ es llamado "punto del objeto 2", para $t_1 < f(x, y) < t_2$ es llamado "punto del objeto 1", y por último, $f(x, y) < t_1$ es llamado "punto del fondo".

Generalmente, los métodos de umbralización segmentan la imagen de salida en varia regiones, los pixeles del fondo y los pixeles de los objetos buscados, $f(x, y)$ representa el nivel de gris del pixel (x, y) y $p(x, y)$ denota alguna propiedad local de este punto, por ejemplo, el promedio del nivel de gris de los pixeles vecinos centrados en (x, y). Por lo cual, una imagen umbralizada es definida como:

$$T_{global_n}(f) = \begin{cases} 0 & \text{if } f(x,y) < t_1 \\ 1 & \text{if } t_1 \leq f(x,y) < t_2 \\ \vdots & \vdots \\ n & \text{if} f(x,y) \geq t_{n-1} \end{cases} \tag{2.1}$$

$$T_{local}(x, y) = \begin{cases} 0 & \text{if } g(x,y) \; \epsilon \; R_i < t_i \\ 1 & \text{if } g(x,y) \; \epsilon \; R_i \geq t_i \end{cases} \tag{2.2}$$

Cuando T depende solo de $f(x, y)$, el umbral es llamado "umbral global"(ecuación 2.1). Si T depende de $f(x, y)$ y de $p(x, y)$, se divide la imagen

en regiones R_i y se establece un umbral t_i para cada una de ellas, éste es llamado "umbral local"(ecuación 2.2).

2.6. Segmentación basada en regiones

La finalidad de un proceso de segmentación es dividir la imagen de entrada en un conjunto de regiones. Los métodos de segmentación por discontinuidad, como la umbralización, logran este objetivo buscando límites o bordes entre las regiones, en función de las discontinuidades obtenidas por las propiedades del nivel de gris o de color de los pixeles de la imagen. En cambio, la *segmentación basada en regiones* es un proceso en donde se obtiene la región directamente[12].

Existen dos enfoques principales: *crecimiento de regiones*, donde se tiene un punto inicial (*semilla*) y se expande a subregiones más grandes, y *división y fusión*, donde se tienen grandes subregiones y se dividen o fusionan dependiendo de sus similitudes.

Gonzalez y Woods[12] establecen las siguientes reglas fundamentales que deben cumplir los procesos de segmentación basados en regiones. Siendo R la región de toda la imagen y n subregiones $R_1, R_2, ..., R_n$ en las cuales se va a dividir R, tal que:

$$\bigcup_{i=1}^{n} R_i = R \tag{2.3}$$

$$R_i \text{ es una región conectada, } i = 1, 2, ..., n \tag{2.4}$$

$$R_i \bigcap R_j = \phi, i \neq j \tag{2.5}$$

$$P(R_i) = VERDADERO \text{ para } i = 1, 2, ..., n \tag{2.6}$$

$$P(R_i \bigcup R_j) = FALSO \text{ para cualquier región adyacente } R_i \text{ y } R_j \tag{2.7}$$

$P(R_i)$ es un predicado lógico definido sobre los pixeles de un conjunto R_i.

2.3) La segmentación debe de ser completa, es decir, todos los pixeles deben pertenecer a una región.

2.4) Los pixeles de una región deben de estar conectados por una propiedad predefinida.

2.5) Las regiones deben de ser disjuntas, es decir, las regiones no deben tener elementos en común.

2.6) Los pixeles de una región deben de cumplir con las mismas propiedades, por ejemplo, $P(R_i) = TRUE$ si todos los pixeles en R_i tienen el mismo nivel de gris.

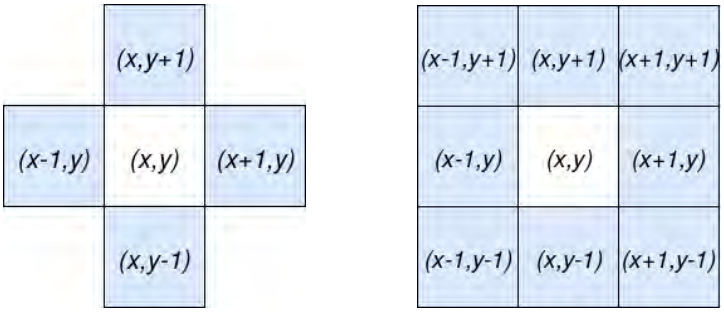
2.7) Los pixeles de diferentes regiones tienen propiedades diferentes.

2.6.1. Conectividad de pixeles

En el procesamiento digital de imágenes, la conectividad de pixeles es la forma en que un pixel se relaciona con sus pixeles vecinos. Existen diferentes tipos conectividad:

- Conectividad-4 (figura 2.4a). Los pixeles serán vecinos a cada pixel si son adyacentes a sus bordes, es decir, los pixeles vecinos están conectados vertical y horizontalmente al pixel de interés. Cada pixel que tenga las coordenadas $(x \pm 1, y)$ o $(x, y \pm 1)$ está conectado a (x, y).

- Conectividad-8 (figura 2.4b). Los pixeles serán vecinos a cada pixel si son adyacentes a sus bordes y a sus esquinas, es decir, serán pixeles vecinos los que están conectados vertical, horizontal y diagonalmente al pixel de interés. Cada pixel que tenga las coordenadas $(x \pm 1, y \pm 1)$ está conectado a (x, y).

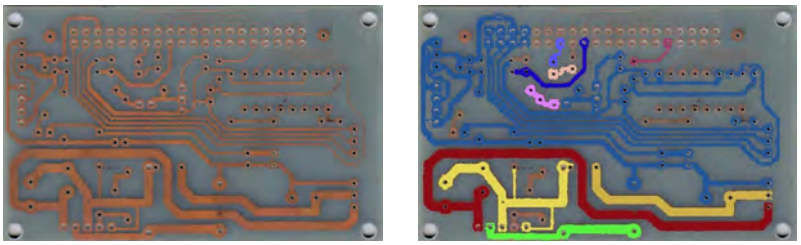


(a) (b)

Figura 2.4: Conectividad de Pixeles.

2.6.2. Crecimiento de regiones

El *crecimiento de regiones* es un tipo de segmentación donde se agrupan pixeles (o puntos) en grandes regiones, basados en un criterio de similaridad



(a) (b)

Figura 2.5: Crecimiento de Regiones aplicado a un circuito impreso.

predefinido. El procedimiento general es comenzar con una "semilla" (un pixel o un punto inicial), o con un conjunto de "semillas" que se van a convertir en regiones con base en las características en común con sus pixeles vecinos. Si los pixeles vecinos cumplen con el criterio de similaridad, serán agregados a la región formada por la semilla y, en una siguiente iteración del algoritmo, se comparan los pixeles vecinos de la región siguiendo el mismo criterio. El procedimiento finaliza cuando todos los pixeles vecinos a la región formada no cumplen con el criterio de similaridad.

En la figura 2.5b se muestra el procedimiento de segmentación basado en el crecimiento de regiones aplicado a la figura 2.5a. Cada región creada por una semilla se representa con un color distinto.

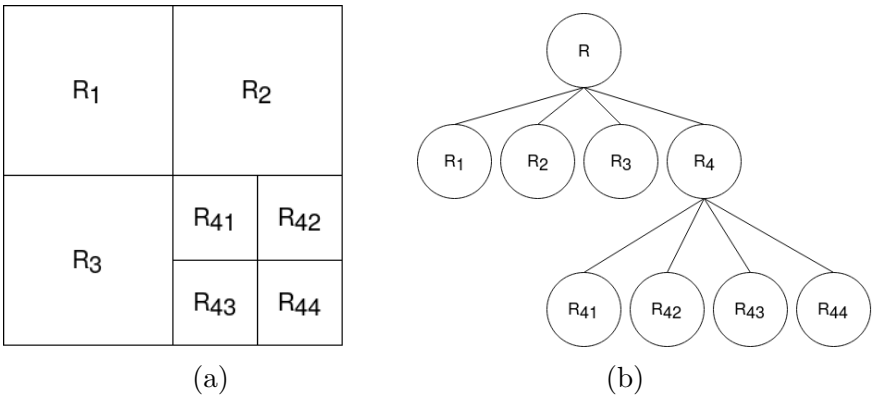
Las semillas son elegidas dependiendo del problema de la segmentación. Por ejemplo, si el objetivo es segmentar las pistas de cobre de una placa fenólica, las semillas serán un conjunto de puntos sobre las pistas. Si el sistema carece de información previa, se calcula el histograma de la imagen y se eligen los niveles de gris correspondientes a las pistas de cobre.

De igual manera, el criterio de similaridad se elige dependiendo del tipo de problema. Se pueden elegir los siguientes criterios: el nivel de gris, el color, la textura, forma o tamaño de una región, o información de profundidad proveniente de una cámara RGB-D.

2.6.3. División y fusión de regiones

La *división y fusión de regiones* es un tipo de segmentación en el cual se divide la imagen de entrada en grandes subregiones, a su vez, cada subregión se divide o se fusiona con su región vecina si se satisfacen o no las reglas fundamentales detalladas en la sección 2.6.

Gonzalez y Woods [12] describen el algoritmo en cuatro puntos importantes:



(a) (b)

Figura 2.6: Imagen dividida con su correspondiente *árbol cuaternario*.

- 1.Dividir la imagen de entrada en cuatro cuadrantes del mismo tamaño.
- 2.Para cada cuadrante, si $P(R_i) = FALSO$ subdividir en cuatro cuadrantes iguales (figura 2.6a). Esta técnica de división se representa en forma de un *árbol cuaternario* (del inglés *quadtree*)(figura 2.6b).
- 3.Fusionar regiones adyacentes R_j y R_k si $P(R_i \bigcup R_k) = TRUE$.
- 4.Repetir el punto 2 y el punto 3 hasta que cada cuadrante no cumpla con las condiciones establecidas para dividirse o fusionarse.

2.7. Operaciones morfológicas

La morfología es una rama de la biología que estudia la forma y estructura de las plantas o animales. De forma similar, la morfología matemática es un conjunto de operaciones de procesamiento digital de imágenes que tienen como finalidad, extraer los componentes principales de la imagen, estos son útiles para la representación y la descripción de una región, como sus límites y su forma.

Las operaciones morfológicas se fundamentan en la teoría de conjuntos. A continuación, se describen algunos conceptos básicos.

2.7.1. Teoría de conjuntos

Relación de pertenencia. Siendo X un conjunto de un espacio bidimensional Z^2 y $x = (x_1, x_2)$ un elemento que pertenece a X, esta relación se

puede expresar de la siguiente manera:

$$x \in X$$

Si x es un elemento que no pertenece a X, entonces se expresa:

$$x \notin X$$

Cuando un conjunto carece de elementos, se considera como un conjunto *vacío* o *nulo* y se denota por el símbolo ϕ.

En la morfología matemática, los elementos de los conjuntos representan las coordenadas de los pixeles de la imagen, estos pueden contener objetos u otros elementos de interés[12]:

$$C = \{x | x = -y, \ para \ y \in Y\} \tag{2.8}$$

La ecuación 2.8 se refiere a que el conjunto C tiene como elementos x tal que x es es el resultado de la multiplicación cada una de las coordenadas de todos los elementos de Y por -1.

Subconjunto (figura 2.7a). Si cada elemento del conjunto A también es un elemento del conjunto B, entonces el conjunto A es un subconjunto de B, se denota como:

$$A \subseteq B$$

Unión (figura 2.7b). La unión del conjunto A con el conjunto B es el conjunto C al cual pertenecen todos los elementos de A y todos los elementos de B:

$$C = A \cup B$$

Intersección (figura 2.7c). De manera similar, la intersección del conjunto A con el conjunto B es el conjunto C cuyos elementos pertenecen a ambos conjuntos:

$$C = A \cap B$$

Complemento (figura 2.7d). El complemento del conjunto B es el conjunto de elementos no contenidos en B:

$$B^c = \{x | x \notin B\}$$

Resta (figura 2.7e). La diferencia de dos conjuntos B y A es el conjunto de los elementos de B que no son elementos del conjunto A, se define como:

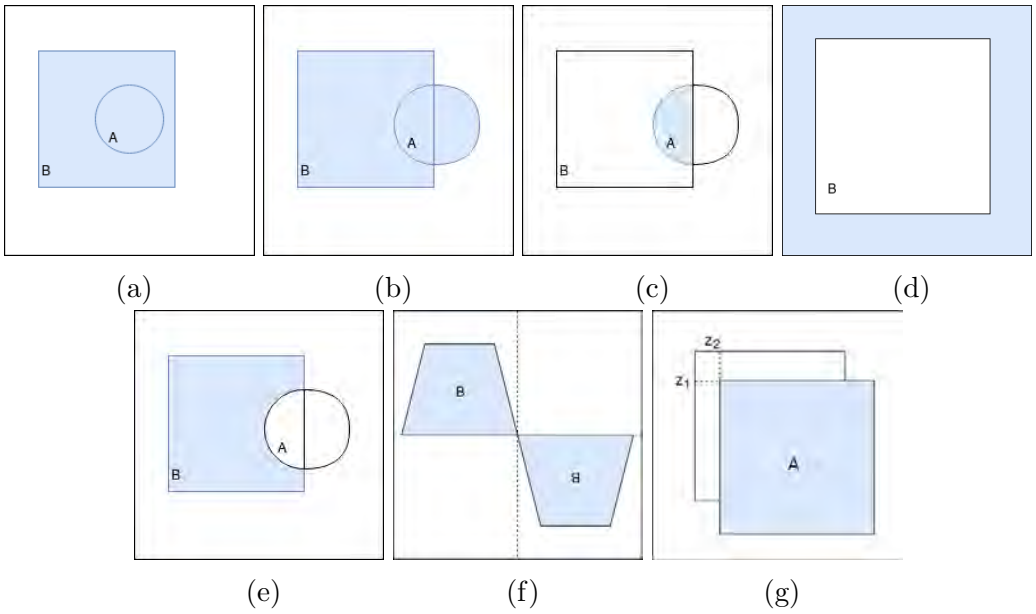


Figura 2.7: Teoría de conjuntos.

$$B - A = x|x \in B, x \notin A = B \cap A^c$$

Reflexión (figura 2.7f). Gonzalez y Woods[12] mencionan que la reflexión de un conjunto B, es el conjunto de elementos de B multiplicados por -1, se denota como $\widehat{B}$ y se define como:

$$\widehat{B} = \{x|x = -b, \ para \ b \in B\}$$

Traslación (figura 2.7g). Gonzalez y Woods[12] explican que la traslación de un conjunto A por un punto $z = \{z_1, z_2\}$, es denotado como $(A)_z$ y se define como:

$$(A)_z = \{x|x = a + z, \ para \ a \in A\}$$

2.7.2. Operaciones lógicas

La mayor parte del procesamiento basado en la morfología matemática se aplica a imágenes binarias, a razón de esto, se utilizan operaciones lógicas como un poderoso complemento para dicho procesamiento.

Las principales operaciones lógicas son AND, OR y NOT (complemento). Estas operaciones se realizan pixel a pixel entre los pixeles correspondientes de dos imágenes, con excepción de la operación NOT, donde se opera en los

pixeles de una única imagen. El resultado de la operación *AND* será 1 si el valor de ambos pixeles es 1, en el caso contrario será 0. El resultado de la operación *OR* es 0 si el valor de ambos pixeles es 0, en caso contrario es 1. Por último, el resultado de la operación *NOT* es el complemento del valor del pixel.

2.7.3. Dilatación

La dilatación y la erosión son las principales operaciones morfológicas del procesamiento morfológico matemático. La dilatación se utiliza para aumentar el área de las regiones de interés, uniendo regiones cercanas y cerrando pequeños huecos.

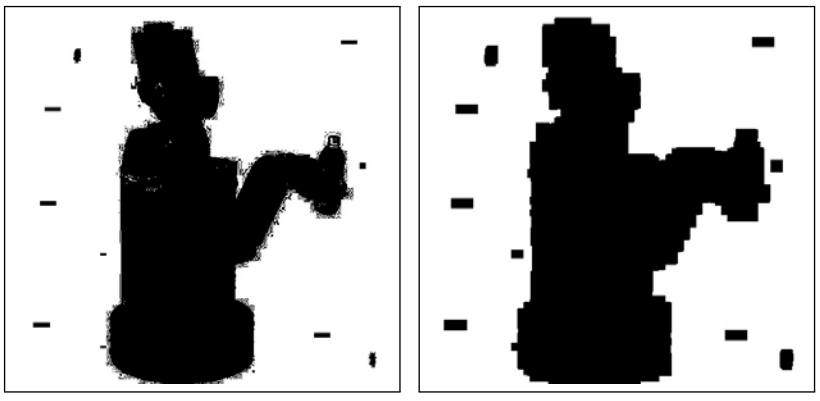
Siendo conjuntos A y B conjuntos en Z^2, la dilatación de A por B denotada como $A \oplus B$ se define en [12] como:

$$A \oplus B = \{z|(\widehat{B})_z \cap A \neq \phi\} \tag{2.9}$$

El proceso de dilatación de toda una imagen, es el resultado de trasladar la reflexión de B por z, es decir, la dilatación de A por B es el conjunto de todos los desplazamientos, z, tal que existe una intersección de B y A en al menos un elemento. Entonces, el proceso de dilatación de una imagen se puede reescribir como la ecuación:

$$A \oplus B = \{z|[(\widehat{B})_z \cap A] \subseteq A\}$$

Generalmente, el conjunto B es llamado *elemento estructurante*. En la figura 2.8b se muestra el proceso de dilatación aplicado a la figura 2.8a con un elemento estructurante en forma de recta.



(a) (b)

Figura 2.8: Dilatación.

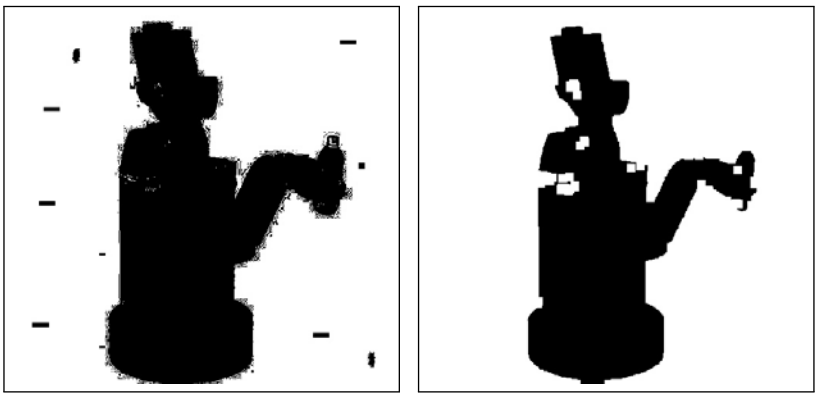
2.7.4. Erosión

La erosión es la operación contraria a la dilatación, ya que se utiliza para disminuir el área de una región de interés, eliminando pequeñas regiones o ruido en la imagen, y delimitando los bordes de una mejor manera.

Para los conjuntos A y B en Z^2, la erosión de A y B denotada por $A \ominus B$ es definida en [12] como:

$$A \ominus B = \{z|(B)_z \subseteq A\} \tag{2.10}$$

El proceso de erosión de una imagen A por un elemento estructurante B es el conjunto de todos los puntos z, tales que existe una intersección del elemento B en A. En la figura 2.9b se muestra el proceso de erosión aplicado a la figura 2.9a con un elemento estructurante en forma de recta.



(a) (b)

Figura 2.9: Erosión.

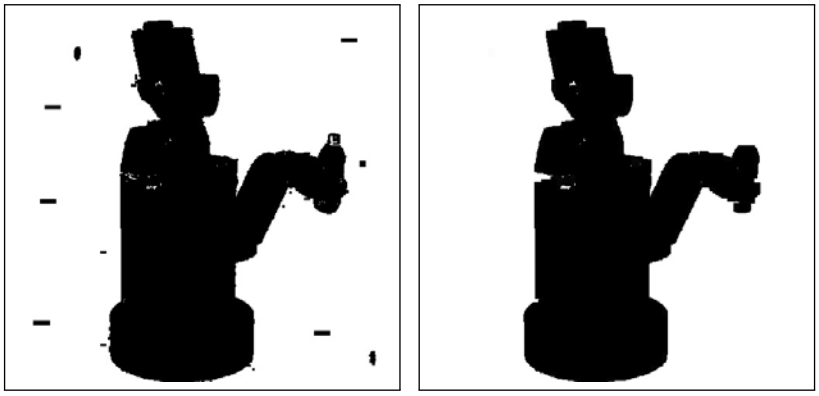
2.7.5. Apertura y Cerradura

La *apertura* y la *cerradura* son operaciones morfológicas conjuntas de la erosión y la dilatación. La *apertura* se utiliza generalmente para suavizar el contorno de una región y para eliminar pequeñas regiones y ligeras protuberancias. La *cerradura* une estrechas grietas, conecta pequeñas regiones y elimina pequeños huecos.

La apertura de un conjunto A por un elemento estructurante B, denotado $A \circ B$ es definida como:

$$A \circ B = (A \ominus B) \oplus B$$

Por lo cual, la apertura de A por B es la erosión del conjunto A por el elemento estructurante B, seguido por la dilatación del resultado de la primera operación. En la figura 2.10b se muestra el proceso de *apertura* aplicado a la imagen 2.10a.



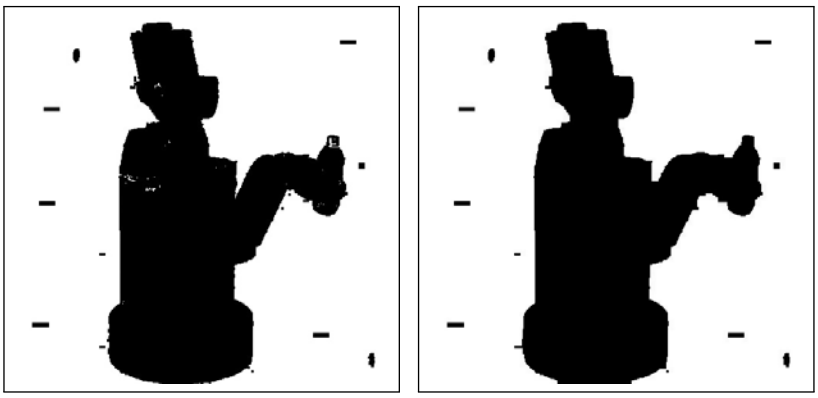
(a) (b)

Figura 2.10: Apertura.

La cerradura de un conjunto A por un elemento estructurante B, denotado por $A \bullet B$ es definido como:

$$A \bullet B = (A \oplus B) \ominus B$$

La cerradura de A por B es la dilatación del conjunto A por el elemento estructurante B, seguido de la erosión del resultado de la primera operación. En la figura 2.11a se muestra el proceso de *cerradura* aplicado a la imagen 2.11b.



(a) (b)

Figura 2.11: Cerradura.

2.8. Filtro Gaussiano

En procesamiento de señales, un filtro Gaussiano es un tipo de filtro lineal cuya respuesta al impulso es una aproximación a una función Gaussiana. Este tipo de filtros se usa comúnmente para eliminar parcialmente el ruido en una imagen, ya que el resultado reduce las variaciones de intensidad entre los pixeles vecinos. En esta técnica, la imagen de entrada debe convolucionarse con un núcleo Gaussiano y así producir una imagen suavizada.

Una función Gaussiana discreta de dos variables es definida en [16] como:

$$g(x, y) = e^{-\frac{x^2+y^2}{2\sigma^2}}$$

Donde σ determina el ancho de la campana de Gauss.

Una aproximación a una función Gaussiana es dada por los coeficientes de la expansión binomial:

$$(1+x)^n = \binom{n}{0} + \binom{n}{1}x + \binom{n}{2}x^2 + ... + \binom{n}{n}x^n$$

Por lo cual, se pueden usar las filas del triangulo de Pascal como una aproximación al filtro Gaussiano. Por ejemplo, la aproximación de cinco elementos correspondiente al triángulo de Pascal es:

1	4	6	4	1

Esta máscara se usa para suavizar una imagen en dirección horizontal. Un filtro Gaussiano bidimensional se puede implementar como dos convoluciones

sucesivas de dos máscaras Gaussianas unidimensionales, una en la dirección horizontal y otra en la dirección vertical.

Otro enfoque explicado en [16] para crear las máscaras Gaussianas, es obtener los elementos directamente de la expresión de la distribución Gaussiana discreta:

$$\frac{g[i,j]}{c} = e^{-\frac{x^2+y^2}{2\sigma^2}}$$

donde c es una constante de normalización.

2.9. Detector de bordes de Canny

La detección de bordes es una tarea fundamental del procesamiento de medio nivel de la Visión computacional, ya que los bordes delimitan, de una mejor manera, los contornos de un objeto o de una región de interés. El método de detección de bordes de Canny fue propuesto por John F. Canny en el año 1986, este método utiliza un algoritmo de múltiples etapas para detectar bordes con una alta confidencia, extrayendo la información estructural de la imagen con menor cantidad de datos procesados. Por estas razones, éste es uno de los métodos más utilizados en aplicaciones de Visión computacional.

Canny[11] se basa en tres criterios fundamentales para la detección de bordes:

- Buena detección. La probabilidad de no marcar puntos de bordes reales y de marcar puntos de bordes falsos debe ser baja.
- Buena localización. Los puntos marcados como puntos del borde deben de estar lo más cerca del centro del borde real.
- Solo una respuesta para cada borde. Cuando hay más de una detección para el mismo borde, solo una debe tomarse como verdadera.

Con base en estos criterios, Canny[11] divide su método de detección de bordes en cinco pasos principales:

1. Se aplica un filtro Gaussiano para suavizar la imagen. Los detectores de bordes son muy susceptibles al ruido de la imagen, ya que el ruido ocasiona falsas detecciones de bordes. Se aplica la convolución de un filtro Gaussiano con la imagen para disminuir el ruido en ésta. La ecuación para calcular el núcleo de un filtro Guassiano de tamaño $2k+1 * 2k+1$ viene dado por:

$$G_{i,j} = e^{-\frac{(i-(k+1))^2+(j-(k+1))^2}{2\sigma^2}}; 1 \leq i,j \leq (2k+1)$$

2. Cálculo del gradiente de la imagen. Se calcula la intensidad y la dirección del gradiente utilizando operadores de detección de bordes basados en la primera derivada (como *Prewitt*, *Sobel* o *Roberts*), tanto en dirección horizontal (G_x) como en dirección vertical (G_y). La magnitud G y la dirección del gradiente Θ en cada punto es calculado por las siguientes ecuaciones:

$$G = \sqrt{G_x^2 + G_y^2}$$

$$\Theta = atan2(G_x, G_y)$$

3. Supresión de no-máximos. La supresión de no-máximos se utiliza para encontrar el borde de mayor intensidad, suprimiendo todos los valores de los gradientes (configurándolos en cero), excepto los máximos locales, los pixeles con más en intensidad. El algoritmo para cada pixel es el siguiente:

   - Se compara la intensidad del pixel actual con la intensidad del pixel en la dirección del gradiente positivo y en la dirección del gradiente negativo.
   - Si la intensidad del pixel actual es mayor en comparación con los otros pixeles, se conserva su valor, de lo contrario el valor será suprimido.

4. Doble Umbral. Al aplicar la supresión de no-máximos se obtiene una representación más precisa de los bordes en la imagen. Sin embargo, existen pixeles marcados como falsos bordes a causa del ruido en la imagen. Para filtrar estos pixeles, se seleccionan dos umbrales, un umbral alto y un umbral bajo. Se hace lo siguiente:

   - Será un pixel de borde fuerte si el valor del gradiente del pixel es mayor que el valor del umbral alto.
   - Será un pixel de borde débil si el valor del gradiente del pixel es menor que el valor del umbral alto y mayor que el valor del umbral bajo.
   - Se suprime el pixel si el valor del gradiente del pixel es menor que el valor de umbral bajo.

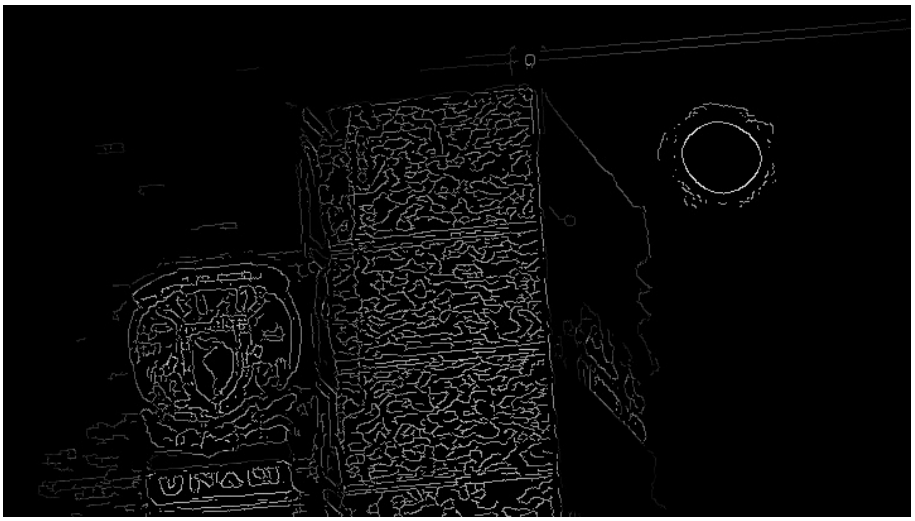
   Los umbrales se determinan empíricamente y dependerán del contenido de la imagen de entrada.

5. Seguimiento de bordes por histéresis. El proceso de histéresis tiene como objetivo transformar los pixeles de borde débiles en pixeles de borde

fuertes. Esto sucede si al menos uno de sus pixeles vecinos es pixel de borde fuerte, es decir, al analizar un pixel de borde débil, se busca si al menos uno de sus ocho pixeles vecinos es pixel de borde fuerte. Si es el caso, el pixel analizado se identifica como un pixel de borde que debe preservarse.

En la figura 2.12 se muestra el resultado del la detección de bordes con el método de Canny.

(a)



(b)

Figura 2.12: Detección de bordes de Canny.

Capítulo 3

Aprendizaje Profundo

En las últimas décadas, el *aprendizaje profundo* (del inglés, *deep learning*) ha tenido gran impacto en el desarrollo de diversos sistemas, como la clasificación de imágenes, segmentación de imágenes, reconocimiento de voz, la traducción automática, entre otros.

El *aprendizaje automatizado* o *aprendizaje máquina* (del inglés, *machine learning*) es una rama de la inteligencia artificial que tiene como objetivo, desarrollar algoritmos computacionales que doten la capacidad de "aprender" a las computadoras. Se dice que un sistema puede aprender si mejora su desempeño con base en experiencia, es decir, con base en ejemplos que se le presentan al sistema.

El *aprendizaje profundo* es un tipo de *aprendizaje automatizado*, en el cual se utilizan redes neuronales artificiales con una jerarquía establecida, donde en cada nivel se extraen las características significativas del nivel anterior. A esta jerarquía se le conoce como *modelo neuronal*. Como menciona Goodfellow en [17], es difícil para una computadora darle significado a un conjunto de datos sin procesar, por ejemplo, darle una descripción a un conjunto de pixeles de una imagen de entrada. El aprendizaje profundo resuelve este tipo de problemas, utilizando cada una de las capas del modelo neuronal para generar patrones representativos de los datos de entrada.

En la figura 3.1 se muestra un modelo neuronal que da la representación de una persona combinando simples conceptos, como esquinas, bordes y contornos. En la primera capa, la *capa visible* (nombrada así por que las variables que contiene son observables) se presentan los datos de entrada, en este caso es un conjunto de pixeles de una imagen. Después, se presenta una serie de *capas ocultas* que extraen características abstractas de la imagen. Reciben el nombre de *capas ocultas*, porque sus variables no son observables por el desarrollador. Goodfellow menciona en [17] que si se visualiza el contenido de las capas ocultas podemos encontrar distintos patrones. En la primera

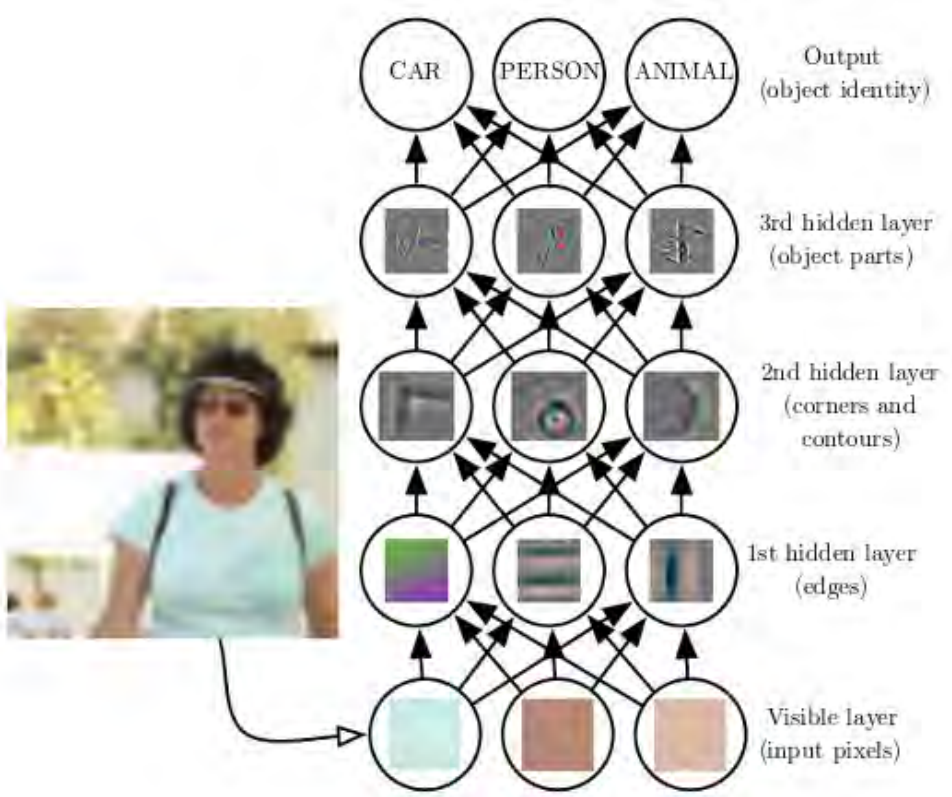


Figura 3.1: Modelo de aprendizaje profundo[17].

capa oculta se encuentran los bordes de la imagen, en la segunda capa oculta se encuentran esquinas y contornos, y en la tercera capa oculta se pueden encontrar objetos específicos, formados por conjuntos de bordes y esquinas representativos. Por último, en la *capa de salida* se obtiene una descripción del objeto en términos de los patrones que contiene.

3.1. Redes neuronales artificiales

Las redes neuronales artificiales son un modelo computacional que trata de emular los procesos biológicos del sistema nervioso, como la forma en la que el cerebro humano almacena información, aprende y reconoce patrones. Están formadas por un un conjunto de nodos conectados entre sí, cuyo principal objetivo es almacenar información, procesarla, y dependiendo de su estado de activación, transmitir o no señales a otros nodos.

La estructura general de una red neuronal artificial viene dada por con-

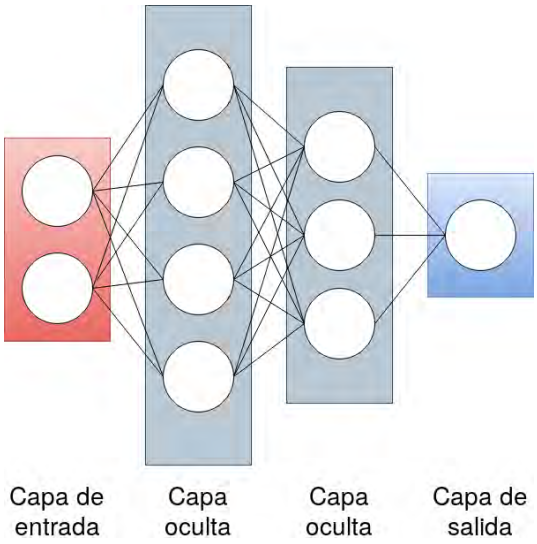


Figura 3.2: Estructura general de una red neuronal artificial.

juntos de capas de nodos, donde cada nodo representa una neurona artificial (figura 3.2). Cada capa tiene las siguientes características:

- En la capa de entrada, se adquieren las señales de entrada, es decir, la información proveniente de sensores o de algún sistema de procesamiento de bajo nivel.
- En las capas ocultas, se extraen características abstractas de la señal de entrada y se obtienen los descriptores o los patrones más significativos de ésta.
- En la capa de salida, se obtiene una clasificación, detección o interpretación de la señal de entrada.

En el año de 1957, Frank Rosenblatt, basado en la *teoría Hebbiana* de la plasticidad sináptica (el comportamiento de las neuronas biológicas durante el proceso de aprendizaje), crea el *perceptrón* como el modelo más simple de una neurona biológica. Como menciona Rojas en [18], el principal aporte de Rosenblatt fue la introducción de variables numéricas llamadas *pesos*, y una *función de activación* para cada neurona, la cual indica si la neurona transmitirá o no su información a la siguiente capa neuronal.

El modelo del perceptrón de Rosenblatt (figura 3.3) está compuesto por un vector de entradas $\vec{x} = x_1, x_2, x_3, ..., x_n$ asociado a un vector de pesos $\vec{w} = w_1, w_2, w_3, ..., w_n$, y produce una salida binaria, determinada por la suma ponderada $\sum w_j x_j$, un sesgo Θ (también llamado *bias*) y una función de

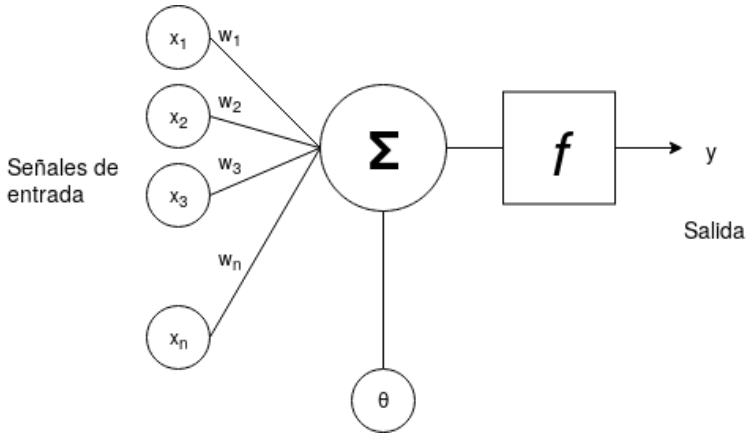


Figura 3.3: Modelo de un perceptrón.

activación, que en este caso es la función escalón de Heaviside. A continuación se muestra esta ecuación:

$$f(x) = \begin{cases} 1 & \text{si} \quad \sum_{i}^{n} w_i x_i + \Theta > 0 \\ 0 & \text{en otro caso} \end{cases}$$

En algunas aplicaciones del perceptrón, como la detección de bordes, los valores del vector de pesos $\vec{w}$ ya están previamente establecidos. Para encontrar los valores $\vec{w}$ de manera automática en otro tipo de aplicaciones, se hace uso de los *algoritmos de aprendizaje*.

3.2. Algoritmos de Aprendizaje Automatizado

Un algoritmo de aprendizaje automatizado es un algoritmo que puede aprender con base en los datos que se le presentan[17]. En otras palabras, es un algoritmo adaptativo mediante el cual, una red neuronal artificial modifica sus parámetros para lograr un comportamiento deseado. Esto se logra presentando una serie de ejemplos mapeados de entrada a salida, y ejecutando iterativamente un procedimiento de corrección de errores, hasta que la red neuronal dé la respuesta deseada[18].

La mayoría de los algoritmos de aprendizaje automatizado se pueden dividir en la categoría de *aprendizaje supervisado* y *aprendizaje no supervisado*.

Los algoritmos de **aprendizaje supervisado** utilizan un conjunto de datos de entrenamiento, donde cada ejemplo está asociado con una etiqueta u objetivo, el cual se espera como resultado del ejemplo dado. Los parámetros de la red se corrigen de acuerdo con la magnitud del error, definido por el algoritmo de aprendizaje. El aprendizaje supervisado se utiliza en sistemas de clasificación y sistemas de regresión (como la predicción de datos).

Los algoritmos de **aprendizaje no supervisado** utilizan un conjunto de datos de entrenamiento que, a diferencia del aprendizaje supervisado, no tienen una etiqueta u objetivo establecido. La finalidad de este tipo de aprendizaje, es encontrar un modelo de distribución de probabilidad de los ejemplos u observaciones dados[17]. Generalmente se utiliza en tareas de agrupamiento de datos, donde se divide la base de datos en conjuntos de datos con características en común.

3.3. Redes neuronales convolucionales

Las Redes neuronales convolucionales (en inglés, *Convolutional Neural Network*), son una versión *regularizada* del perceptrón multicapa, usadas principalmente en sistemas de detección y clasificación de objetos, segmentación de imágenes, procesamiento de lenguaje natural, entre otras disciplinas, debido a su buen desempeño y a que necesitan poco pre-procesamiento de datos. Las redes neuronales convolucionales son redes neuronales que utilizan la operación de *convolución* en lugar de una matriz de multiplicación en al menos en alguna de sus capas[17].

Una convolución es una operación matemática que transforma dos funciones en una tercera, la cual representa la magnitud de cuánto se superpone una función sobre la otra. Siendo f la función de entrada y g el núcleo, la convolución de f y g (el mapa de características), denotada como $f * g$, se define como la integral del producto de ambas funciones, después de desplazar una de ellas una distancia t:

$$s(t) = \int_{-\infty}^{\infty} f(\eta)g(t-\eta)d\eta$$

En el caso discreto, se utiliza una forma discreta de la convolución:

$$s(t) = (f * g)(t) = \sum_{n} f(n)g(t-n)$$

En las aplicaciones de aprendizaje profundo, la función de entrada es un arreglo multidimensional de datos y el núcleo es un arreglo multidimensional

de parámetros que se adaptan mediante un *algoritmo de aprendizaje*[17]. Por ejemplo, si se tiene una imagen bidimensional I como entrada, se tendrá un núcleo bidimensional K:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(m,n)K(i-m, j-n)$$

De igual forma, en muchas bibliotecas de redes neuronales se implementa una función similar llamada **correlación cruzada**, que actúa igual que la convolución, pero sin voltear la matriz del núcleo:

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n)$$

Cabe destacar que en algunos libros de aprendizaje profundo se implementa la función de correlación cruzada, pero la llaman convolución.

Las redes neuronales convolucionales están compuestas principalmente de capas convolucionales, capas de reducción (en inglés, *pooling layer*), y capas totalmente-conectadas (en inglés, *fully-conected*).

3.3.1. Capas convolucionales

La característica principal de una capa convolucional, es que puede extraer las características espaciales y temporales de una imagen a través de la aplicación de filtros (la matriz del núcleo) en cada una de sus capas. El objetivo es reducir el tamaño de las imágenes sin perder sus características más significativas. Entonces, el resultado de la convolución es una matriz de menor tamaño definida por el parámetro *paso* (en inglés, *stride*), que indica cómo el filtro convoluciona a través de la imagen. Por último, la salida de cada neurona es definida por una *función de activación*. Generalmente en las capas convolucionales, se utiliza la función de activación **ReLU** (siglas del inglés, *Rectified Linear Unit*).

En la figura 3.4a, se muestra como ejemplo una imagen de una sola capa, de dimensiones 5x5x1 y un núcleo de 3x3x1. El resultado de la operación convolución con *paso=1* en la primera iteración se muestra en la figura 3.4b. El resultado de la siguiente iteración se muestra en la figura 3.4c. Y el resultado de la última iteración se muestra en la figura 3.4d. En la figura 3.4 se aprecia cómo el núcleo se recorre a través de la imagen para hacer la operación de convolución.

Funciones de activación

La función de activación, es una transformación no lineal que define la salida de una neurona artificial en términos de la entrada dada. La salida

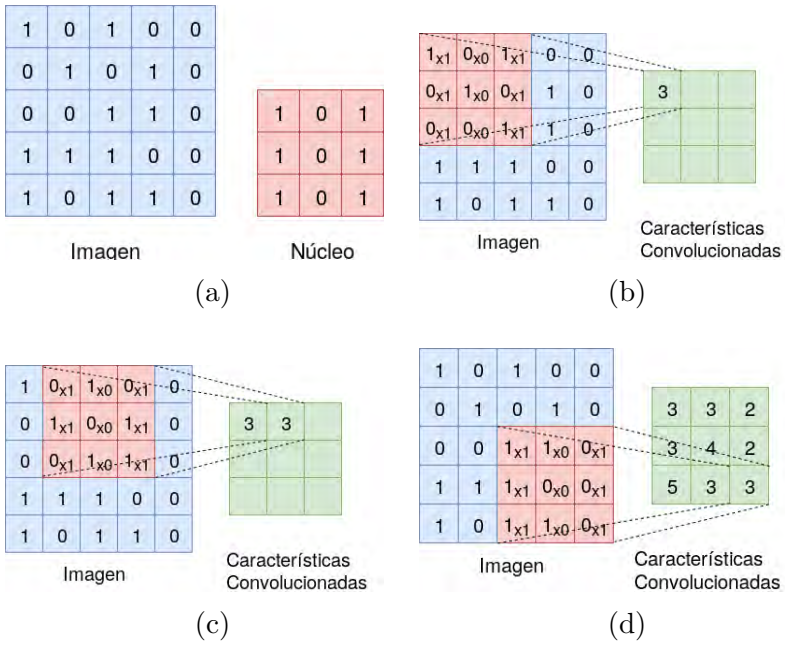


Figura 3.4: Operación Convolución.

será enviada a la entrada de la siguiente capa neuronal.

Existen diferentes tipos de funciones de activación, siendo la más usada la función ReLU. En la figura 3.5, se muestran las funciones de activación más comunes.

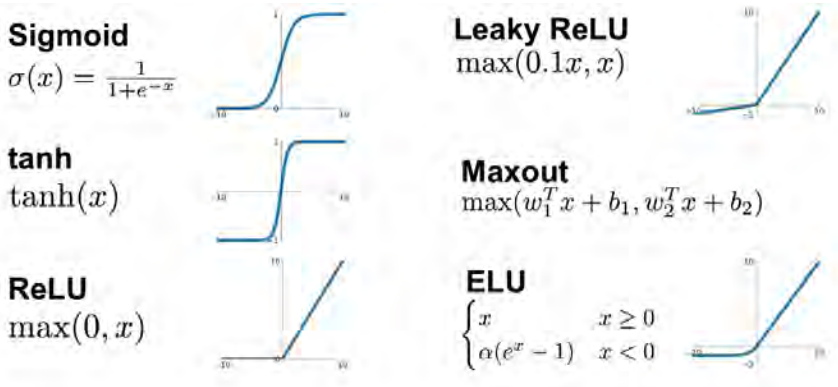


Figura 3.5: Funciones de activación[19].

3.3.2. Capas de reducción

La capa de reducción (en inglés, *pooling layer*) tiene como objetivo, reducir el tamaño espacial de las características convolucionadas para, de esta manera, reducir también el poder de cómputo requerido para el procesamiento de los datos. Además, se extraen características dominantes que son invariantes a rotaciones y traslaciones[20].

Hay dos tipos de capas de reducción, la capa de máxima-reducción (en inglés, *max-pooling*) y la capa de promedio-reducción (en inglés, *average-pooling*). La capa de máxima-reducción regresa el valor máximo de la parte de la imagen cubierta por el núcleo. La capa de promedio-reducción regresa el promedio de todos los valores de la parte de la imagen cubierta por el núcleo. En la figura 3.6 se muestra el resultado de la operación en ambas capas.

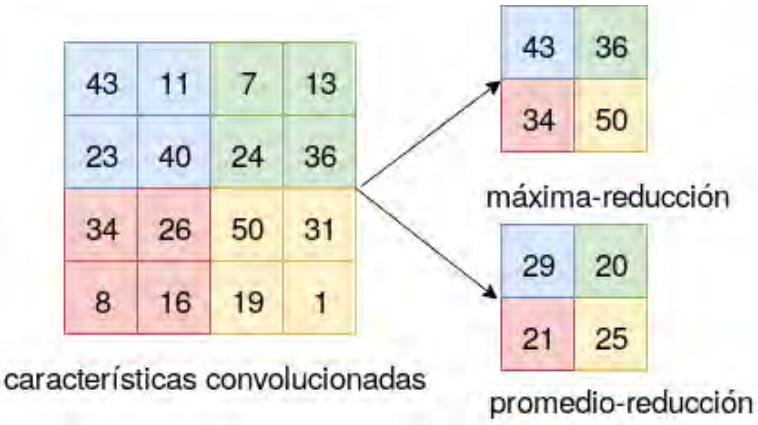


Figura 3.6: Capas de Reducción con *paso*=2.

3.3.3. Capas totalmente-conectadas

Generalmente en las arquitecturas de redes neuronales artificiales, se utilizan una o más capas totalmente-conectadas para la etapa de clasificación. A la salida de las múltiples capas convolucionales y capas de reducción, se tienen las características de alto nivel, que pueden ser fácilmente clasificadas por un perceptrón multicapa. La clasificación se logra transformando las matrices multidimensionales en un vector columna (que es la entrada del perceptrón multicapa) para, posteriormente utilizar el método de *retropropagación* (en inglés, *backpropagation*) y así ajustar los parámetros de la red neuronal artificial (hacer el entrenamiento de la red). Por último se tienen diferentes técnicas de clasificación, como la función *softmax*.

La función Softmax es la función más utilizada a la salida de las capas de clasificación. Ésta representa la distribución de probabilidad sobre n diferentes clases[17], en otras palabras, la función regresa un vector de n probabilidades, dónde la suma de todos sus elementos es 1.

La función Softmax viene dada por la siguiente ecuación:

$$S(y_i) = \frac{e^{y_i}}{\sum_j e^{y_j}}$$

3.3.4. Capas residuales (ResNet)

En la mayor parte de las arquitecturas neuronales, cada capa alimenta a la siguiente capa, es decir, cada una de las capas se conecta únicamente con la siguiente capa neuronal. A diferencia de esto, en una red residual (en inglés, *Residual Network* o *ResNet*), en cada bloque residual, cada capa alimenta a la siguiente capa y a una capa que está de dos a tres capas de distancia. En la figura 3.7 se muestra el diagrama de un bloque residual.

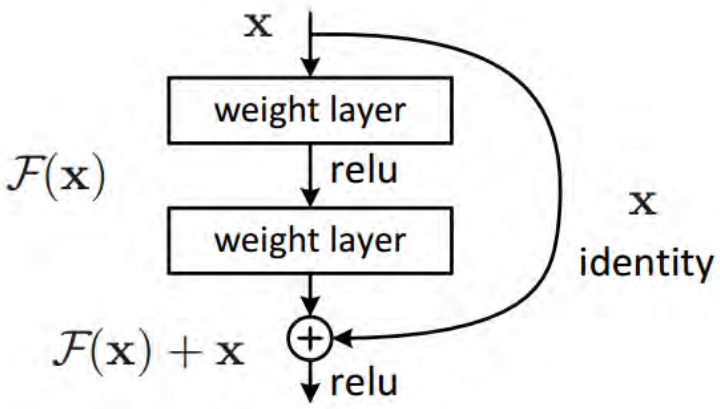


Figura 3.7: Capa Residual[21].

La razón principal de utilizar bloques residuales, es evitar el problema del desvanecimiento del gradiente (en inglés, *vanishing gradient*), lo que significa que no puede encontrar un gradiente óptimo al agregar demasiadas capas a la arquitectura neuronal. Con los bloques residuales, se pueden propagar los gradientes más grandes a las capas iniciales, para que de esta manera, las capas iniciales también puedan aprender de manera rápida al igual que las capas finales[21].

3.4. Retro-propagación

Cuando se utiliza una red pre-alimentada (en inglés, *feedforward network*) se recibe una entrada x y se produce una salida y. Para esto, la información fluye hacia adelante, propagándose por todas las capas ocultas a través de la red para, finalmente, producir una salida. Este procedimiento es llamado *propagación hacia adelante* (del inglés, *fordward propagation*). Una vez calculada la salida de la red, es posible calcular el error de la propagación hacia adelante respecto a los *datos reales* (en inglés, *ground truth*), es decir, la información de la base de datos del entrenamiento.

El algoritmo de retro-propagación (en inglés, *back-propagation*), es un algoritmo de aprendizaje supervisado que permite que la información del error se propague hacia atrás en la red. Su objetivo es minimizar la *función de costo* ajustando los parámetros de pesos y sesgos de la red, utilizando el *algoritmo del descenso del gradiente*.

Una función de costo, (en inglés, *cost function*) es una medida del desempeño de un modelo de aprendizaje máquina, que cuantifica el error entre las predicciones hechas y los datos reales del modelo. El valor obtenido por esta función es llamado *costo*, *error* o *pérdida*. La finalidad de este método, es encontrar los parámetros para los cuales el valor de costo es muy pequeño[22].

Existen diversas funciones de costo, cada una se utiliza dependiendo del problema de aprendizaje:
Error absoluto medio:

$$MAE = \frac{1}{n}\sum_{i=1}^{n}|\widehat{y_i} - x_i|$$

Error Cuadrático Medio:

$$MSE = \frac{1}{2}\sum_{i=1}^{n}(\widehat{y_i} - x_i)^2$$

Entropía Cruzada Binaria:

$$CE = -\frac{1}{n}\sum_{i=1}^{n}(x * log(\widehat{y_i}) + (1-x) * log(1-\widehat{y_i}))$$

Entropía Cruzada Categórica:

$$CE = -\frac{1}{n}\sum_{i=1}^{n} x_i * log(\widehat{y_i})$$

Donde:

- i es el índice de la muestra.
- $\widehat{y}$ es valor predicho.
- x es el valor esperado.
- n es el número de clases en el conjunto de datos.

3.4.1. Algoritmo del descenso del gradiente

El descenso del gradiente es un algoritmo de optimización utilizado para minimizar el valor de una función de costo. Se calcula iterativamente en dirección del menor valor en cada paso, definido por el negativo del gradiente ($-\nabla J$). Para definir la relación entre cualquier peso w_{ij} y la función de costo J, se hace uso de las derivadas parciales y la *regla de la cadena*:

$$\frac{\partial J}{\partial w_{ij}} = \frac{\partial J}{\partial a_j}\frac{\partial a_j}{\partial w_{ij}} = \frac{\partial J}{\partial a_j}\frac{\partial a_j}{\partial z_j}\frac{\partial z_j}{\partial w_{ij}}$$

Dónde

- J es la función de costo.
- w_{ij} es el peso i de la capa j.
- a_j es la salida de la neurona después de la función de activación.
- z_j es la combinación lineal resultado de $\sum \vec{x}\vec{w}$.

De esta manera, el nuevo valor para el peso w_{ij} viene dado por:

$$w_{ij} = w_{ij} - \alpha\frac{\partial J}{\partial w_{ij}}$$

Donde α es la tasa de aprendizaje (en inglés, *learning rate*).

3.4.2. Transferencia de conocimiento

La transferencia de conocimiento (en inglés, *Transfer Learning*) es una técnica del aprendizaje máquina que utiliza, como punto de partida, un modelo preentrenado de una red neuronal artificial, en los casos donde se tiene como objetivo, entrenar una conjunto de datos de imágenes pequeño con muchas características similares al modelo anterior.

En esta metodología se utiliza un modelo neuronal entrenado para un conjunto de datos robusto, como ImageNet[23] o COCO[24], y se hace el reentranamiento o *ajuste fino* (en inglés, *fine-tuning*) de las últimas capas, las capas de clasificación. De esta manera, se tiene un nuevo modelo que aprovecha las características extraídas del modelo anterior, para clasificar unicamente las nuevas clases de interés para el sistema.

3.4.3. Sobre-Ajuste y Sub-Ajuste

Uno de los desafíos más importantes de un modelo neuronal, es que debe funcionar con datos de entrada nuevos, es decir, datos que no sean parte del conjunto de datos de entrenamiento de la red. A esta capacidad del modelo se le llama *generalización*.

Los términos sobre-ajuste (del inglés, *overfitting*) y sub-ajuste (del inglés, *underfitting*) hacen referencia a las deficiencias de un modelo para poder desempeñarse de manera correcta en presencia de datos nuevos.

Cuando en el conjunto de datos de entrenamiento se introducen datos irrelevantes, con ruido, distorsiones y muestras que no son suficientemente representativas para el entrenamiento, el modelo los considerará como ejemplos válidos, por lo cual, la red sufrirá de un sobre-ajuste. Esto quiere decir que el modelo predecirá correctamente solo los datos idénticos al conjunto de datos de entrenamiento, y los datos que salen de los rangos establecidos serán descartados.

Por el contrario, el sub-ajuste ocurre cuando el modelo no tiene suficientes datos de entrenamiento, lo cual produce una baja generalización y el modelo será incapaz de hacer predicciones correctas.

En la figura 3.8 se muestra la curva de desempeño de un modelo entrenado con sobre-ajuste, sub-ajuste y un modelo correctamente entrenado.

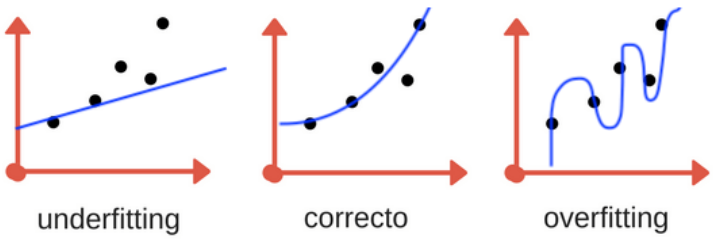


Figura 3.8: Sub-Ajuste y Sobre-Ajuste[25].

Para evitar el sobre-ajuste y el sub-ajuste, se deben tener en cuenta los siguientes puntos:

- Hacer uso de técnicas de aumento de datos (en inglés, *data augmentation*)
- Tener una cantidad suficiente de muestras por cada clase de los datos de entrenamiento.
- Las clases deben ser variadas y equilibradas en cantidad.

- Dividir los datos en un conjunto de entrenamiento y un conjunto de validación.
- Evitar la cantidad excesiva de capas neuronales en la arquitectura neuronal.

3.4.4. Algunas terminologías del aprendizaje profundo

Época. Se llama época (del inglés, *epoch*) al procedimiento donde todo el conjunto de entrenamiento se propaga hacia adelante y se retro-propaga por toda la arquitectura de la red neuronal una sola vez.

Tamaño de lote. El tamaño de lote (del inglés, *batch size*) es el número de ejemplos que se le presentan a la red para estimar el error del gradiente.

Iteraciones. Las iteraciones (del inglés, *iterations*), es el número de lotes (del inglés, *batches*) que se procesan durante el entrenamiento de una red neuronal.

3.5. YOLO

YOLO (por sus siglas en inglés, *You Only Look Once*) es un sistema *estado del arte*, que utiliza una red neuronal convolucional para la detección de objetos en tiempo real[26]. El sistema aplica una única red neuronal a la imagen completa, razón por la cual es muy rápido. Esta red divide la imagen en regiones y predice múltiples cajas delimitadoras (en inglés, *bounding box*), y la probabilidad de detección de las clases de entrenamiento para cada caja delimitadora. Como resultado, el sistema imprime los objetos que detectó, su confidencia y el tiempo de ejecución. En la figura 3.9 se muestra una ejecución del sistema sobre una imagen de entrada.

En el presente proyecto de tesis, se utilizó la versión 3 de YOLO (YOLOv3). Para explicar el funcionamiento del YOLOv3, se abordarán brevemente las partes que componen el sistema.

3.5.1. Arquitectura neuronal

La arquitectura de YOLOv3 está compuesta por 53 capas convolucionales, razón por la cual recibe el nombre de *Darknet-53* (figura 3.10). Cada capa convolucional es seguida de una normalización de lote (en inglés, *batch normalization*) y la función de activación **Leaky ReLU**. No se utiliza ninguna capa de reducción, en su lugar, se utilizan capas convolucionales con

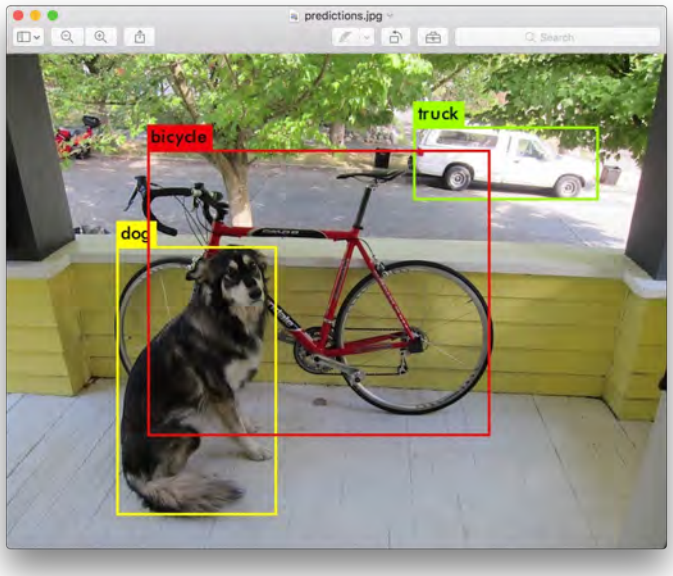


Figura 3.9: Detección de objetos con YOLO[26].

paso=2. Con esto, se reduce la dimensionalidad del mapa de características, evitando perder las características de bajo nivel.

Por otro lado, se plantea otra arquitectura sin las capas residuales y con menos capas convolucionales llamada *YOLOv3_tiny*. Como resultado se tiene una red neuronal más rápida, pero con menos confidencia en la detección y clasificación. En la figura 3.11 se muestra la arquitectura de *YOLOv3_tiny*

	Type	Filters	Size	Output
	Convolutional	32	3 × 3	256 × 256
	Convolutional	64	3 × 3 / 2	128 × 128
1×	Convolutional	32	1 × 1	
	Convolutional	64	3 × 3	
	Residual			128 × 128
	Convolutional	128	3 × 3 / 2	64 × 64
2×	Convolutional	64	1 × 1	
	Convolutional	128	3 × 3	
	Residual			64 × 64
	Convolutional	256	3 × 3 / 2	32 × 32
8×	Convolutional	128	1 × 1	
	Convolutional	256	3 × 3	
	Residual			32 × 32
	Convolutional	512	3 × 3 / 2	16 × 16
8×	Convolutional	256	1 × 1	
	Convolutional	512	3 × 3	
	Residual			16 × 16
	Convolutional	1024	3 × 3 / 2	8 × 8
4×	Convolutional	512	1 × 1	
	Convolutional	1024	3 × 3	
	Residual			8 × 8
	Avgpool		Global	
	Connected		1000	
	Softmax			

Figura 3.10: Arquitectura de Darknet-53[26].

Layer	Type	Filters	Size/Stride	Input	Output
0	Convolutional	16	3 × 3/1	416 × 416 × 3	416 × 416 × 16
1	Maxpool		2 × 2/2	416 × 416 × 16	208 × 208 × 16
2	Convolutional	32	3 × 3/1	208 × 208 × 16	208 × 208 × 32
3	Maxpool		2 × 2/2	208 × 208 × 32	104 × 104 × 32
4	Convolutional	64	3 × 3/1	104 × 104 × 32	104 × 104 × 64
5	Maxpool		2 × 2/2	104 × 104 × 64	52 × 52 × 64
6	Convolutional	128	3 × 3/1	52 × 52 × 64	52 × 52 × 128
7	Maxpool		2 × 2/2	52 × 52 × 128	26 × 26 × 128
8	Convolutional	256	3 × 3/1	26 × 26 × 128	26 × 26 × 256
9	Maxpool		2 × 2/2	26 × 26 × 256	13 × 13 × 256
10	Convolutional	512	3 × 3/1	13 × 13 × 256	13 × 13 × 512
11	Maxpool		2 × 2/1	13 × 13 × 512	13 × 13 × 512
12	Convolutional	1024	3 × 3/1	13 × 13 × 512	13 × 13 × 1024
13	Convolutional	256	1 × 1/1	13 × 13 × 1024	13 × 13 × 256
14	Convolutional	512	3 × 3/1	13 × 13 × 256	13 × 13 × 512
15	Convolutional	255	1 × 1/1	13 × 13 × 512	13 × 13 × 255
16	YOLO				
17	**Route 13**				
18	Convolutional	128	1 × 1/1	13 × 13 × 256	13 × 13 × 128
19	Up-sampling		2 × 2/1	13 × 13 × 128	26 × 26 × 128
20	**Route 19 8**				
21	Convolutional	256	3 × 3/1	13 × 13 × 384	13 × 13 × 256
22	Convolutional	255	1 × 1/1	13 × 13 × 256	13 × 13 × 256
23	YOLO				

Figura 3.11: Arquitectura de YOLOv3-tiny.

3.5.2. Funcionamiento

La imagen de entrada del sistema YOLOv3 se puede configurar de diferentes dimensiones en cada proceso de entrenamiento, como único requisito es que sean imágenes cuadradas de un tamaño múltiplo de 32, por ejemplo 608x608 ó 416x416. De lo contrario, *Darknet* redimensiona las imágenes al tamaño establecido en el archivo de configuración.

La entrada viene dada por:

$$Entrada(m, h, w, d)$$

Donde:

- m es el tamaño del lote de entrenamiento (en inglés, *batch*).
- h es la altura de la imagen (en inglés, *height*).
- w es el ancho de la imagen (en inglés, *width*).
- d son los canales de la imagen de entrada (en inglés, *depth*).

Dependiendo de la tarjeta gráfica, el tamaño del lote de entrenamiento puede ser subdividido. De esta manera, si se tiene un lote de tamaño 64 (64 imágenes) y subdivisiones de 16, entonces se procesarán 4 imágenes en paralelo en cada iteración de la red neuronal convolucional. Siguiendo este ejemplo y utilizando una imagen de 3 canales (RGB) de 416x416, a la entrada de YOLOv3 tendremos:

$$Entrada(64, 416, 416, 3)$$

YOLOv3 hace el proceso de detección en tres diferentes escalas, redimensionando la imagen de entrada por un factor de 32, 16 y 8 respectivamente. Por ejemplo, si se tiene como entrada una imagen de 416x416 y reduce por un factor de 32, el tamaño del mapa de características será de de 13x13. A cada elemento del mapa se le conoce como *celda*[27]. En el mapa de características, cada celda predice un número fijo de cajas delimitadoras.

Por ejemplo, en un mapa de características se tiene: B*(5+C). B representa el número de cuadros delimitadores que cada celda puede predecir. Según [26], cada uno de estos cuadros delimitadores B puede especializarse en la detección de cierto tipo de objeto. Cada uno de los cuadros delimitadores tiene atributos 5 + C, que describen las coordenadas del centro, las dimensiones, la puntuación de objetividad y las confidencias de clase C para cada cuadro delimitador. YOLO v3 predice 3 cuadros delimitadores para cada celda.

Capítulo 4

Herramientas de Desarrollo

Con la finalidad de cumplir con los objetivos establecidos en el presente proyecto de tesis y de tener un buen desempeño en los sistemas propuestos, se hace uso de diversas herramientas de software y de hardware, las cuales se describen brevemente a continuación.

La adquisición de los archivos de video para el sistema de segmentación de objetos, se hace a través de la cámara compacta *Sony DSC-RX100M5*. Cada cuadro del video es procesado utilizando múltiples métodos y algoritmos programados y optimizados por la librería de código abierto *OpenCV*, y de la misma manera, se procesa el sistema de creación de imágenes sintéticas. El sistema *ROS* funge como el módulo central que intercambia información entre el software presente en los robots de servicio y el módulo de detección de objetos propuesto.

El robot Takeshi tiene integrado un sensor RGB-D *Xtion Pro Live*, por medio del cual se capturan las imágenes que sirven como entrada al sistema de detección de objetos propuesto.

4.1. Herramientas de software

Se utilizó el lenguaje C++ como lenguaje de programación para los sistemas de segmentación y creación de imágenes sintéticas, al igual que para el módulo de detección de objetos.

4.1.1. OpenCV

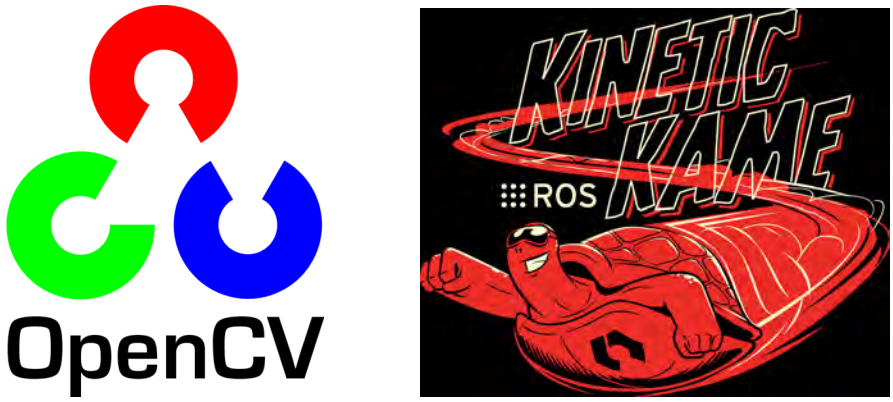
OpenCV, (siglas del inglés, Open Source Computer Vision Library) cuyo logo se muestra en la figura 4.1a, es una biblioteca de código abierto que proporciona múltiples métodos y algoritmos dedicados a sistemas de visión

computacional y aprendizaje automático[28]. Es compatible con los lenguajes de programación C++, Python, Java y MATLAB, y tiene soporte en los sistemas operativos Windows, Linux, Android y Mac OS[28]. Esta biblioteca contiene más de 2500 algoritmos optimizados, de entre los cuales se hace uso principalmente de los siguientes:

- Algoritmos de filtrado de imágenes, como el filtro Gaussiano.
- Operaciones morfológicas, como la dilatación y la erosión.
- Métodos de detección de bordes, como el método de detección de bordes de Canny.
- Técnicas de segmentación de regiones, como el crecimiento de regiones.
- Conversión de espacios de color, como el espacio HSV.

4.1.2. ROS

El Sistema Operativo Robótico (ROS), (siglas del inglés, Robot Operating System) cuyo logo se muestra en la figura 4.1b, es un marco de trabajo para el desarrollo de software integrado a robots. Básicamente se trata de un conjunto de herramientas, bibliotecas y algoritmos que tienen como objetivo, implementar un comportamiento robusto y complejo de una manera sencilla en una gran variedad de plataformas robóticas[29]. Este sistema da un manejo centralizado de la información, encargándose del intercambio de datos de todos los módulos que componen el sistema de software de un robot.



(a) Logo de OpenCV [28] (b) Logo de ROS Kinetic [29]

Figura 4.1: Herramientas de Software.

El módulo propuesto de detección de objetos se adaptó a la distribución Kinetic de ROS, con la finalidad de intercambiar información con el sistema de los robots de servicio, Takeshi y Justina, de una forma optimizada y modular.

4.2. Herramientas de hardware

El propósito principal de este proyecto de tesis es, implementar un sistema de detección de objetos basado en redes neuronales convolucionales en un robot de servicio. En el laboratorio de Bio-Robótica de la Facultad de Ingeniería de la UNAM, se cuenta con dos robots de servicio, Takeshi y Justina, para los cuales se desarrolló el sistema propuesto.

4.2.1. Robots de servicio Takeshi y Justina

El robot HSR (siglas del inglés, Human Support Robot), también llamado Takeshi por el laboratorio de Bio-Robótica (figura 4.2a), es un robot de servicio semi-humanoide diseñado y desarrollado por la empresa Toyota, cuyo objetivo es asistir a personas con capacidades limitadas o de edad avanzada en sus actividades diarias, ya sea en una casa o en una oficina. Su propósito principal es servir como una plataforma de desarrollo académico para diversos algoritmos y sistemas autónomos.

Al comienzo del proyecto del HSR, se adaptó el software del robot Justina en el robot Takeshi, con el objetivo de demostrar la modularidad del sistema y los algoritmos implementados en otra plataforma de desarrollo. Las principales áreas de desarrollo son los siguientes:

- Detección de objetos con una cámara RGB-D.
- Manipulación de objetos con un brazo robótico con cuatro grados de libertad.
- Navegación autónoma con evasión de obstáculos en ambientes dinámicos.
- Reconocimiento de lenguaje natural, como voz y gestos.
- Planeador de acciones.

Takeshi forma parte de un convenio de desarrollo de software entre la UNAM y Toyota-Japón, producto de los buenos resultados obtenidos en la competencia mundial RoboCup[6] en el año de 2017 con el robot Justina. De la misma manera, Takeshi tiene grandes logros participado en la RoboCup, ha

(a) Robot Takeshi[30]. (b) Robot Justina[10].

Figura 4.2: Robots de servicio de la Facultad de Ingeniería.

obtenido un segundo lugar en el año 2018 y un cuarto lugar en el año 2019 en la categoría RoboCup@Home DSPL.

Justina es un robot de servicio semi-humanoide diseñado y desarrollado totalmente en el laboratorio de Bio-Robótica de la facultad de Ingeniería de la UNAM (figura 4.2b). Desde el año del 2007, participa cada año en la competencia mundial de robótica RoboCup[6] logrando buenos resultados, entre los cuales resaltan un segundo lugar en el año 2018 y un segundo lugar en el año 2019 en la categoría RoboCup@Home OPL. El objetivo principal del proyecto Justina, es promover el desarrollo académico en diversas áreas, como inteligencia artificial, procesamiento digital de imágenes, procesamiento de voz, navegación robótica, entre otros, en la comunidad estudiantil de la Facultad de Ingeniería.

4.2.2. Adquisición de imágenes

El sistema de segmentación de objetos tiene como entrada, un archivo de video con capturas del objeto de interés, dentro (total o parcialmente) de un fondo controlado. Para esta tarea, se utilizó la cámara *Sony DSC-RX100M5* debido a que captura imágenes y videos en alta resolución y además es compacta (figura 4.3a). Sus características principales son:

- Auto enfoque automático de 0.05s.
- Sensor CMOS tipo 1.0 de 20.1 megapixeles efectivos.
- Lente ZEISS Vario-Sonnar.
- Grabación de video en 4k a 30fps.

El sensor *Xtion-PRO LIVE* es una cámara RGB-D desarrollada por la empresa ASUS, cuyo propósito es crear videojuegos y aplicaciones donde se involucre la detección o movimiento del cuerpo humano (figura 4.3b). Esta cámara está integrada al robot Takeshi con el objetivo de identificar personas, objetos y obstáculos con gran precisión.

El sistema de detección de objetos toma la información RGB-D proveniente de la cámara *Xtion-PRO LIVE* para hacer la detección de objetos en un espacio bidimensional, posteriormente calcula un punto en el espacio tridimensional donde colocar el manipulador del robot para, de esta manera, poder tomar el objeto. Las características principales de este sensor son:

- Sensor RGB-D y dos micrófonos.
- Imagen de profundidad VGA(640x480): 30fps; QVGA(320x240): 60fps.
- Distancia de uso: 0.4m - 3.5m.
- Resolución SXGA (1280*1024).

4.2.3. Computadoras de desarrollo

En el presente proyecto de tesis se utilizaron dos computadoras: una computadora de escritorio y una computadora portátil. La computadora de escritorio se utilizó para el sistema de segmentación de objetos, el sistema de creación de escenas sintéticas y para el entrenamiento de la red neuronal convolucional YOLOv3. Sus características principales son:

- Procesador Intel Xeon(R) CPU E5-2650 2.20GHz.
- Tarjeta gráfica Nvidia Quadro P4000.

(a) Sony DSC-RX100M5 [31] (b) Xtion-PRO LIVE [32]

Figura 4.3: Adquisición de Imágenes.

- RAM: 16Gb
- Disco Duro: 1 Tb.

La computadora portátil se utiliza en el robot Takeshi como soporte para el procesamiento de algoritmos que demanden mucho poder de cómputo. En este proyecto de tesis, se utilizó para cargar la arquitectura de la red neuronal convolucional YOLOv3 y así poder hacer la detección de objetos en cada cuadro proveniente de la cámara *Xtion PRO LIVE*. Sus características principales son:

- Alienware 15R3.
- Intel Core i7 7820HK.
- NVIDIA GTX 1080.
- 16 GB RAM.
- 128 SSD + 1TB HDD.

Capítulo 5

Desarrollo

El objetivo principal de este proyecto de tesis es: *crear un conjunto de escenas sintéticas con objetos etiquetados para re-entrenar la red neuronal convolucional YOLOv3, y de esta manera, utilizarla como sistema de detección de objetos propios para un robot de servicio.* Para lograr este objetivo, se proponen dos sistemas de visión computacional: el primer sistema genera un conjunto de escenas sintéticas etiquetadas para el reentrenamiento de YOLOv3. El segundo sistema plantea una metodología para que un robot de servicio, pueda detectar y manipular los objetos entrenados autónomamente.

5.1. Sistema de creación de un conjunto de escenas sintéticas

El primer sistema está compuesto por dos módulos desarrollados en el lenguaje de programación C++. En el primer módulo, se hace el procesamiento de bajo y medio nivel de una secuencia de imágenes provenientes de un archivo de video. En el segundo módulo, se crea un conjunto de escenas sintéticas ya etiquetado para el reentrenamiento de la red neuronal convolucional YOLOv3.

5.1.1. Módulo de segmentación de objetos

El propósito del módulo de segmentación de objetos, es crear un conjunto de imágenes de objetos segmentados, teniendo como entrada, un conjunto de archivos de video de los objetos de interés. En la figura 5.1 se muestra a detalle, el diagrama de bloques del funcionamiento del sistema.

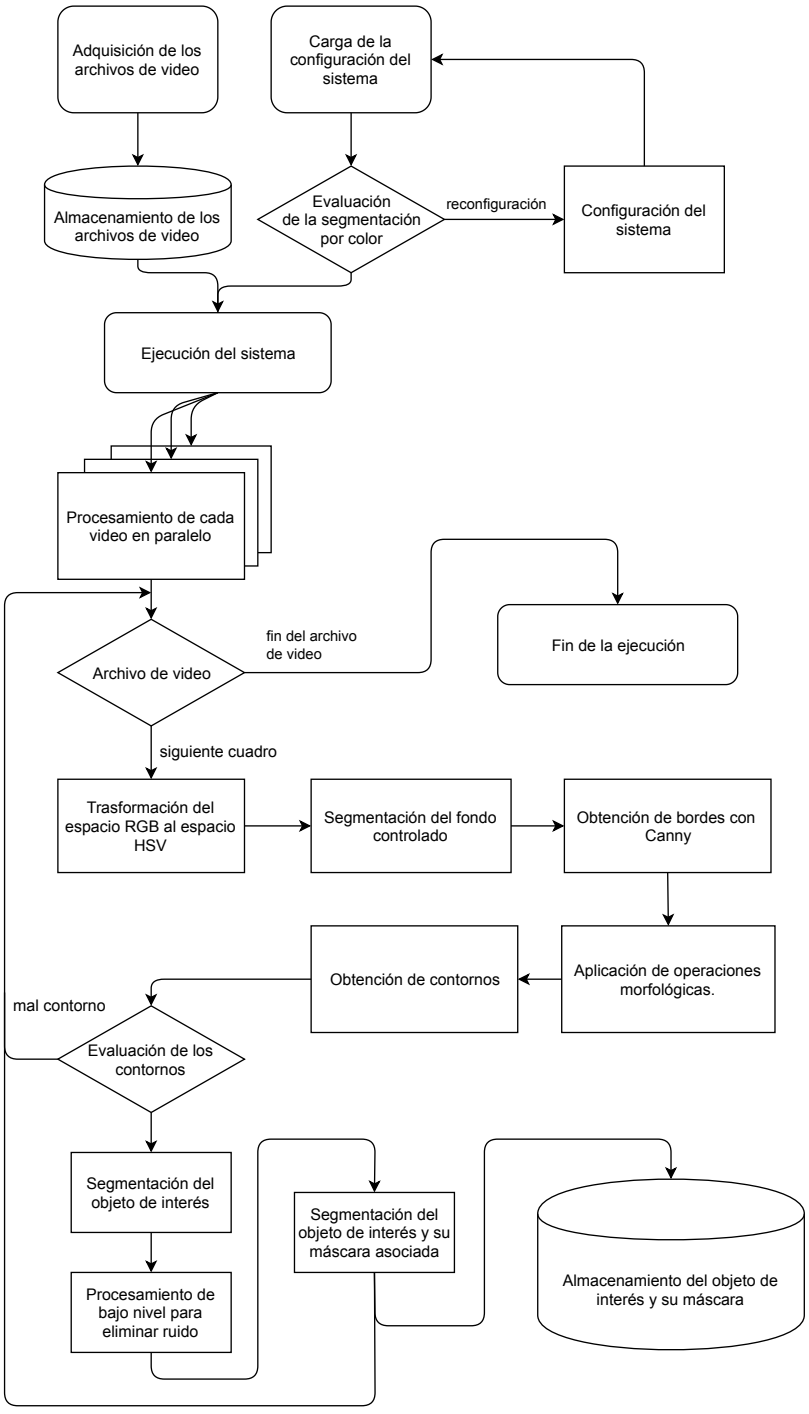


Figura 5.1: Diagrama de bloques del sistema de segmentación de objetos.

Adquisición de los archivos de video

Para el funcionamiento del sistema, es necesario capturar un archivo de video del objeto de interés, éste debe estar colocado sobre un fondo controlado. El fondo controlado es de un color uniforme, fácil de segmentar en el espacio de color HSV. Cabe resaltar que el objeto no necesariamente debe estar inmerso dentro del fondo controlado, siendo suficiente el colocar la base de éste dentro del fondo. La captura debe ser de tal manera que se abarquen los 360°alrededor del objeto y de 0 a 90°en el eje vertical del objeto.

Los archivos de video son capturados por la cámara compacta descrita en la sección 4.2.2, con dimensiones de 1920x1080 pixeles a 60 cuadros por segundo y con una duración aproximada de 45 segundos. Todos los archivos son almacenados en una ruta específica configurada en el sistema.

Configuración del sistema

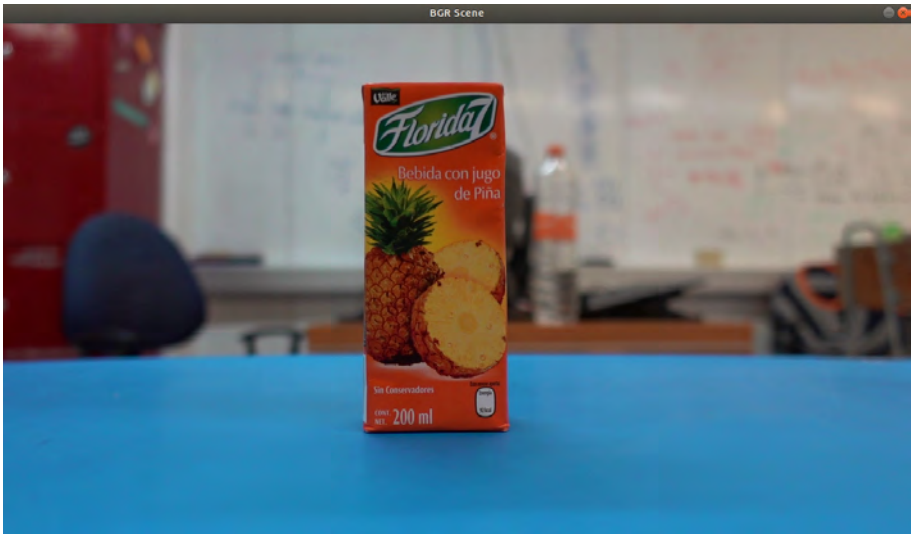
El primer paso antes de la ejecución, es cargar la configuración del sistema desde dos archivos de texto. Un archivo tiene la configuración de las banderas de depuración del sistema, cada una activa el despliegue de distintas imágenes que muestran el desempeño de diversos algoritmos, como los algoritmos de segmentación, los algoritmos de bordes o las operaciones morfológicas. El otro archivo de texto, carga la configuración de los parámetros de color (en el espacio HSV), de los fondos controlados entrenados previamente. El sistema elegirá los parámetros de color dependiendo del archivo de video de entrada.

Como segundo paso se evalúa si la segmentación del fondo controlado es precisa, de lo contrario, se depura el color mediante un método del sistema. Para realizar la depuración de manera sencilla y rápida, el sistema despliega la imagen correspondiente a un cuadro del archivo de video de entrada, y en la parte superior, se despliegan 6 barras de seguimiento (en inglés, *trackbar*), correspondientes al máximo y al mínimo de cada uno de los parámetros HSV (figura 5.2b). Adicionalmente, se tiene un evento relacionado al clic izquierdo del ratón, que captura los valores HSV de un pixel de la imagen mostrada. De esta manera, se puede depurar de forma precisa la segmentación del fondo controlado, ya sea moviendo las barras de seguimiento, mediante el clic del ratón o por ambos métodos.

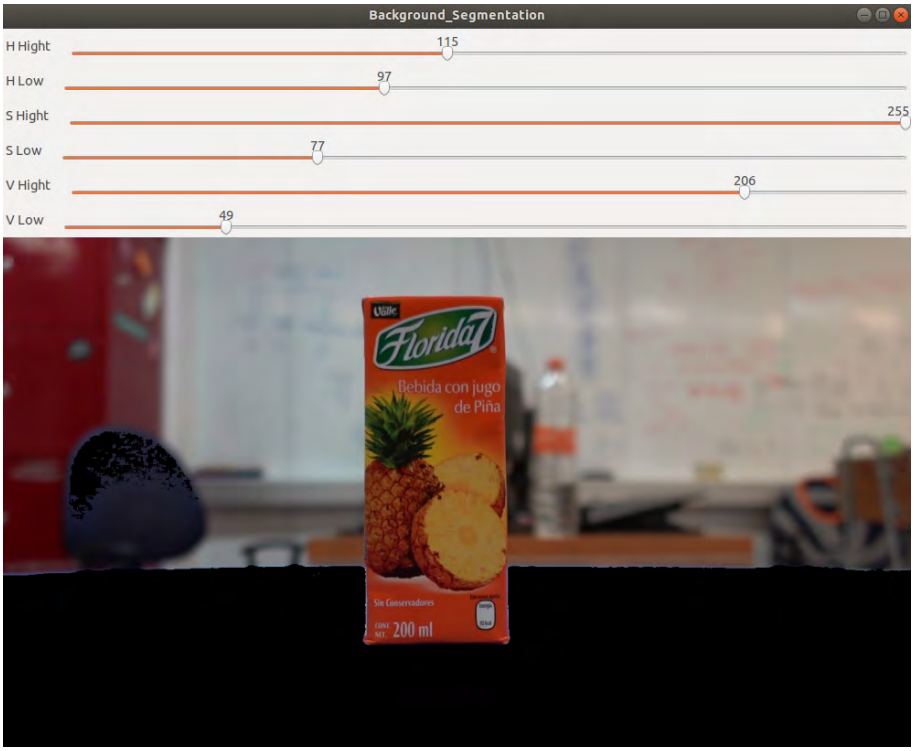
En la figura 5.3, se muestra el funcionamiento del método de depuración de la segmentación del fondo controlado.

Ejecución del sistema

El sistema revisa la carpeta de videos y, con base en el nombre del archivo, elige los parámetros HSV y el nombre de la clase del objeto que segmentará.



(a) Imagen RGB



(b) Segmentación del fondo controlado

Figura 5.2: Método de depuración para la segmentación del fondo controlado.

Cada archivo de video está nombrado de la siguiente manera:

nombre del archivo = color del fondo controlado _ clase del objeto _ tipo de video

Por ejemplo:

nombre del archivo = blueBackground_orangeJuice_.mp4

Cabe mencionar que es conveniente elegir el fondo controlado dependiendo del color del objeto que se va a segmentar, ya que si ambos tienen colores similares, en el proceso de segmentación se quitarán pixeles que pueden ser característicos del objeto.

Para optimizar el funcionamiento del sistema, cada video será procesado en paralelo, y en cada video, los cuadros serán procesados secuencialmente.

Segmentación del fondo controlado

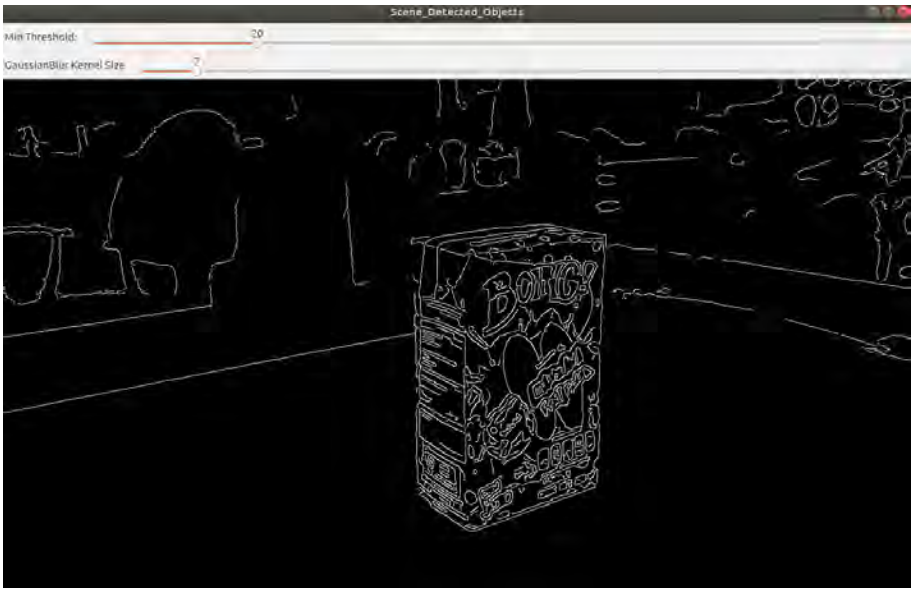
Se transforma la imagen de entrada del espacio de color RGB al espacio de color HSV. Después, se utiliza la operación de umbralización **inRange()** de OpenCV, con los parámetros HSV configurados previamente, para segmentar el fondo controlado. El resultado es una imagen binaria con el fondo controlado en color blanco. A partir de esta imagen, podemos visualizar la escena sin el fondo controlado (figura 5.2b).

Detección de bordes

Como entrada al algoritmo de detección de bordes, se utiliza la imagen de la escena en el espacio RGB transformada a una imagen en escala de grises, después, como se describe en el algoritmo propuesto por Canny[11], se aplica un filtro Gaussiano para disminuir los cambios bruscos de intensidades entre pixeles vecinos, y así evitar que se detecten bordes falsos. Para el proceso de detección de bordes, se utilizó la función **Canny()** de OpenCV, la cual tiene como entrada la imagen de la escena en escala de grises, y como resultado, una imagen en escala de grises con los bordes detectados.

De forma similar que el método de depuración de color del fondo controlado, se tiene un método para depurar el algoritmo de detección de bordes. Se tienen dos barras de seguimiento, la primera cambia el umbral mínimo del algoritmo de Canny, necesario para la supresión de no-máximos (sección 2.9). La segunda barra de seguimiento, modifica el tamaño del núcleo del filtro Gaussiano. En la figura 5.3 se muestra el resultado del algoritmo de detección de bordes.

(a) Imagen RGB

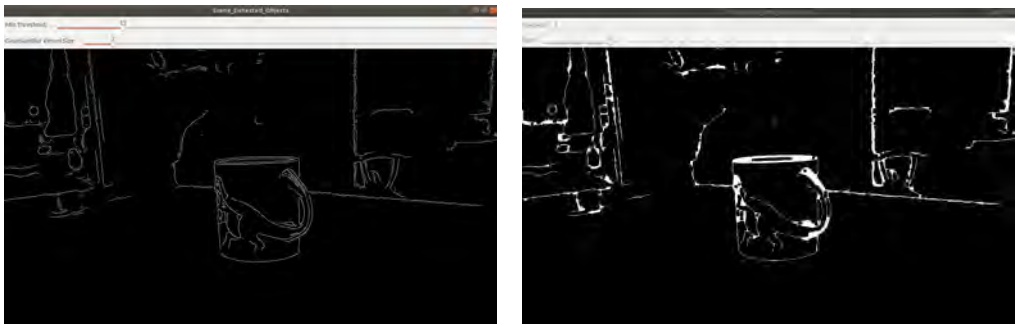


(b) Detección de bordes en la escena

Figura 5.3: Algoritmo de detección de Bordes.

Operaciones Morfológicas

Al utilizar el algoritmo de detección de bordes de Canny, los bordes del contorno del objeto no están totalmente delimitados. Para solucionar este problema, se aplica la operación morfológica de cerradura para unir los bordes del contorno y tener bien delimitado al objeto. También de utilizan barras de seguimiento para poder depurar estas operaciones. En la figura 5.4b se muestra el resultado de la operación morfológica de cerradura en una imagen con bordes detectados.



(a) Detección de bordes (b) Operaciones morfológicas

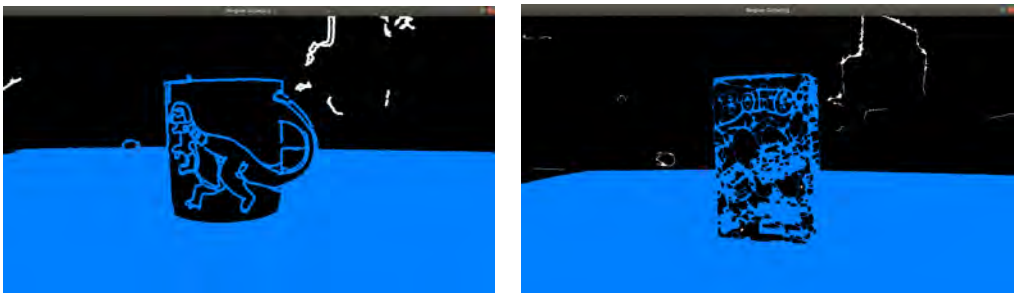
Figura 5.4: Resultado de aplicar las operaciones morfológicas a los bordes detectados.

Crecimiento de regiones

El siguiente paso es evaluar si los bordes de la escena pertenecen al objeto de interés, es decir, solo los bordes detectados que estén inmersos en el fondo controlado serán considerados parte del objeto de interés. Para llevar a cabo dicha tarea, se aplica el algoritmo de OpenCV de segmentación por crecimiento de regiones **floodFill**. Se toma como única semilla el primer pixel encontrado del fondo controlado. Después, la región crece desde este pixel a los bordes detectados inmersos en el fondo controlado. En la figura 5.5 se tienen dos imágenes, cada una es el resultado de aplicar el algoritmo de crecimiento de regiones.

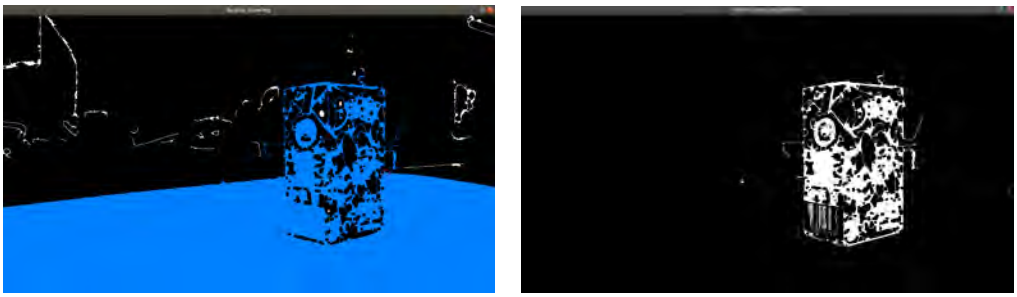
Segmentación de los bordes de interés

Hasta el momento se tienen los bordes que están inmersos en el fondo controlado, el siguiente paso es segmentarlos de la escena. Para llevar a cabo esta tarea, se toma la escena resultado del algoritmo de crecimiento de regiones y la escena del fondo controlado segmentado. Ambas imágenes se



(a) (b)

Figura 5.5: Resultado de aplicar algoritmos de crecimiento de regiones a los bordes detectados.



(a) (b)

Figura 5.6: Resultado de segmentar los bordes de interés en escena.

binarizan y se les aplica la operación lógica **AND**. El resultado es una escena donde solo se encuentran los bordes de interés. En la figura 5.6b se muestra el resultado de la segmentación de bordes de interés de la figura 5.6a.

Eliminación de ruido en la escena

El resultado de la operación anterior es una escena donde en su mayoría aparecen los bordes del objeto, sin embargo, también aparecen otros bordes producto de la escena externa, los cuales representan ruido para el sistema propuesto. Para eliminar este ruido, primero se rellenan las regiones entre los bordes y se descartan regiones pequeñas (figura 5.7a). Después se aplica la operación morfológica **apertura** para eliminar protuberancias, eliminar pequeñas regiones y formar un objeto más sólido. La imagen resultante es una escena binaria con el objeto de interés más definido (figura 5.7b). En esta figura, se depura el tamaño del núcleo del elemento estructurante aplicado

(a) (b)

Figura 5.7: Se aplican operaciones morfológicas para eliminar pequeñas regiones y bordes no deseados.

en la operación morfológica.

Segmentación del objeto de interés

Como último paso, se recorta de la escena original la parte donde se encuentra el objeto de interés. Para realizar este procedimiento, se aplica la operación lógica **AND** entre la escena original y la imagen binaria del objeto segmentado. De la imagen resultante, se recorta el área donde se encuentra el objeto segmentado.

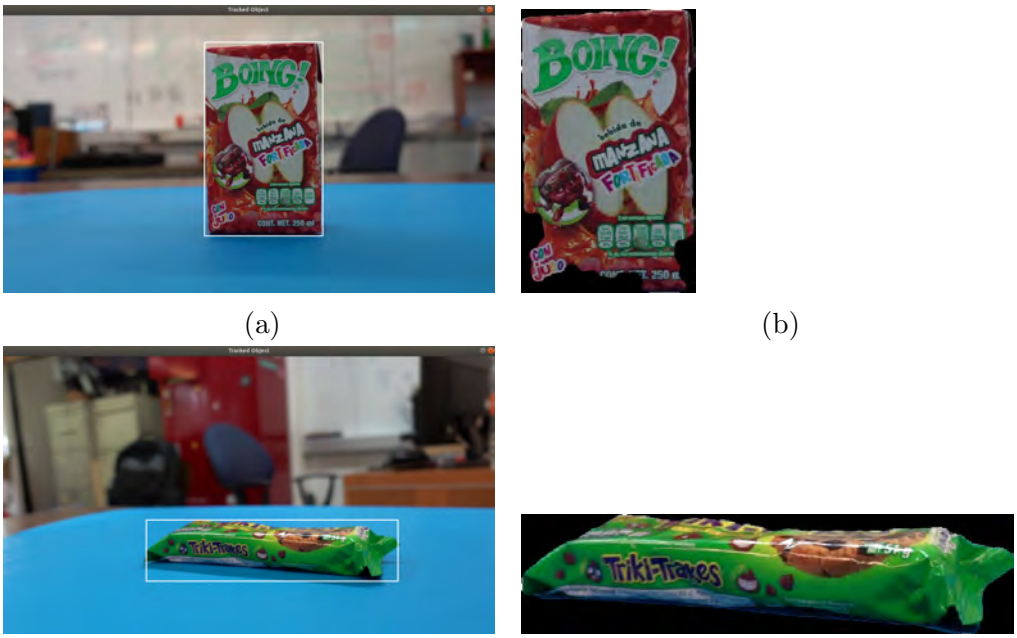
Como resultado, se tiene una imagen que únicamente contiene el objeto de interés. Después, se aplica una vez más la técnica de segmentación por color para eliminar parte del fondo controlado que pudiera aparecer en objetos que tengan agujeros en su interior, como el asa de las tazas de café. La imagen final se guarda en el directorio configurado al inicio del programa.

En las figuras 5.8a y 5.8c se muestra la escena original y se dibuja una caja delimitadora de color blanco, que especifica qué área se va segmentar. En la figura 5.8d y la figura 5.8b se muestran los objetos segmentados y recortados de la figura 5.8c y la figura 5.8a respectivamente, estos son guardados en la carpeta de imágenes de objetos segmentados.

Al finalizar la ejecución del sistema, se genera un conjunto de imágenes de objetos segmentados con su respectiva máscara. La ruta donde se guardan las imágenes es la siguiente:

data/images/class_name/

Donde *class_name* es el nombre de la carpeta para clase de objeto segmentado. También se generó un archivo de texto con el nombre de todas las clases



(a) (b)

(c) (d)

Figura 5.8: Segmentación y recorte de la escena del objeto de interés.

de objetos, colocados renglón por renglón, que posteriormente servirá como información necesaria para el siguiente módulo del sistema.

El sistema, con un correcto funcionamiento, genera al rededor de 2500 imágenes de objetos segmentados diferentes por cada archivo de video, el cual tiene una duración aproximada de 45 segundos.

5.1.2. Módulo de creación de escenas sintéticas

En este módulo se crea el conjunto de escenas sintéticas, es decir, el conjunto de imágenes artificiales para el entrenamiento de la red neuronal convolucional YOLOv3.

Como entrada se tiene el conjunto de imágenes de objetos segmentados y un conjunto de imágenes de escenas reales, éstas se dividen para el conjunto de entrenamiento y el conjunto de validación. Como primer paso, se aplican técnicas de aumento de datos a las imágenes de los objetos segmentados, después, éstos son colocados dinámicamente en todas las escenas reales mediante algoritmos de visión computacional. Una vez colocados los objetos, se aplican otras técnicas de aumento de datos a toda la escena para robustecer el conjunto de entrenamiento de la red. En la figura 5.9 se muestra a detalle,

el diagrama de bloques del funcionamiento del sistema.

Conjunto de imágenes de escenas reales

El conjunto de imágenes de escenas reales se adquirió mediante la cámara compacta descrita en la sección 4.2.2.

Para realizar el entrenamiento de la red neuronal convolucional, se necesita un archivo asociado a cada escena sintética con los datos del etiquetado de cada objeto que aparezca en ésta. Este etiquetado no es preciso, ya que el fondo incluido en las etiquetas formará parte de las características del objeto. Por esta razón, en el conjunto de entrenamiento y validación se incluyen escenas reales con características similares a las escenas donde se detectarán los objetos en las etapas de prueba.

Para el conjunto de entrenamiento, se capturan escenas similares a escenas donde pueden aparecer los objetos a detectar, por ejemplo, escenas con escritorios, mesas, parte del piso, sillones, muebles de cocina, etc. Para el conjunto de validación, es recomendable capturar imágenes donde sí van a aparecer los objetos a detectar, por ejemplo, en la competencia de RoboCup se crea un departamento completo, necesario para llevar a cabo todas las pruebas establecidas en las reglas de la competencia. Para este caso, es recomendable tener capturas de los sitios del departamento en los cuales van a estar colocados los objetos.

En la figura 5.10 se muestran escenas reales utilizadas para crear el conjunto de entrenamiento y el conjunto de validación.

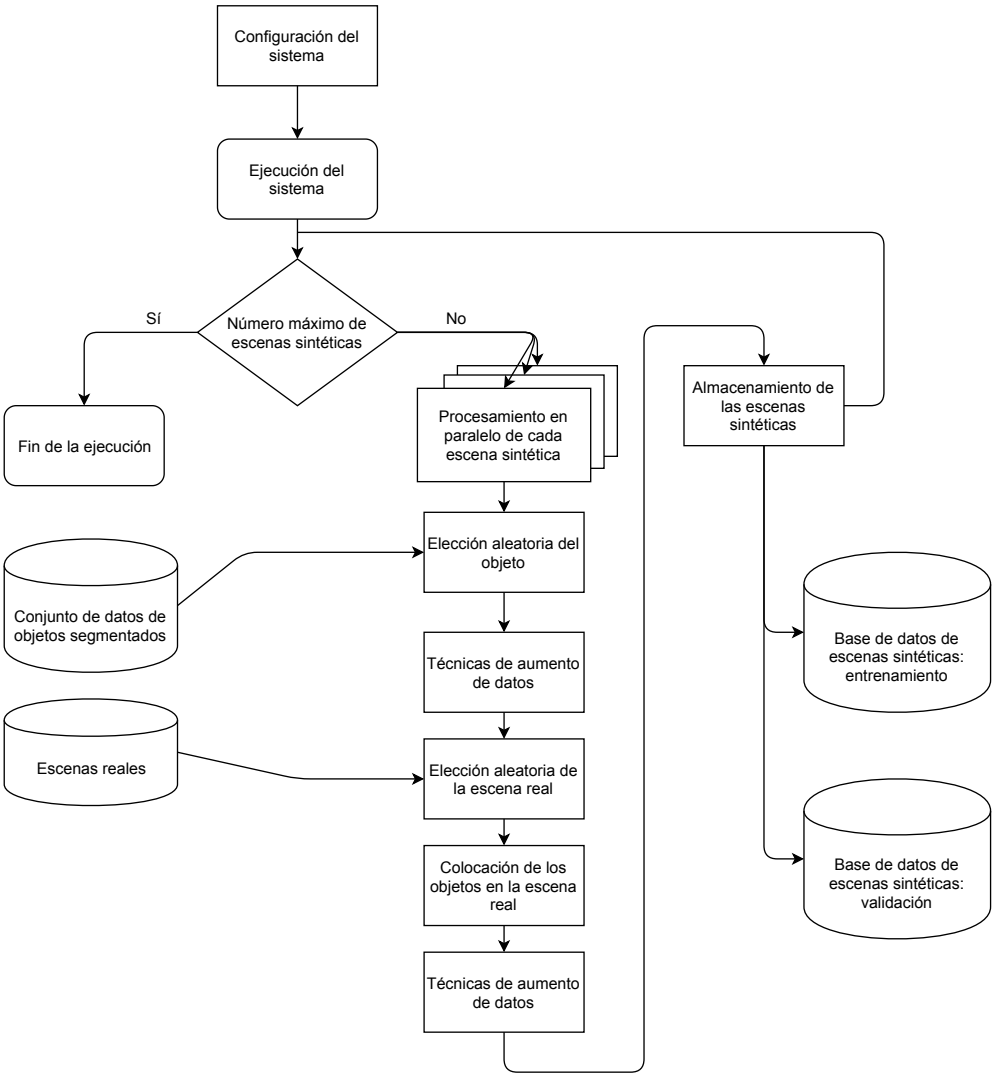
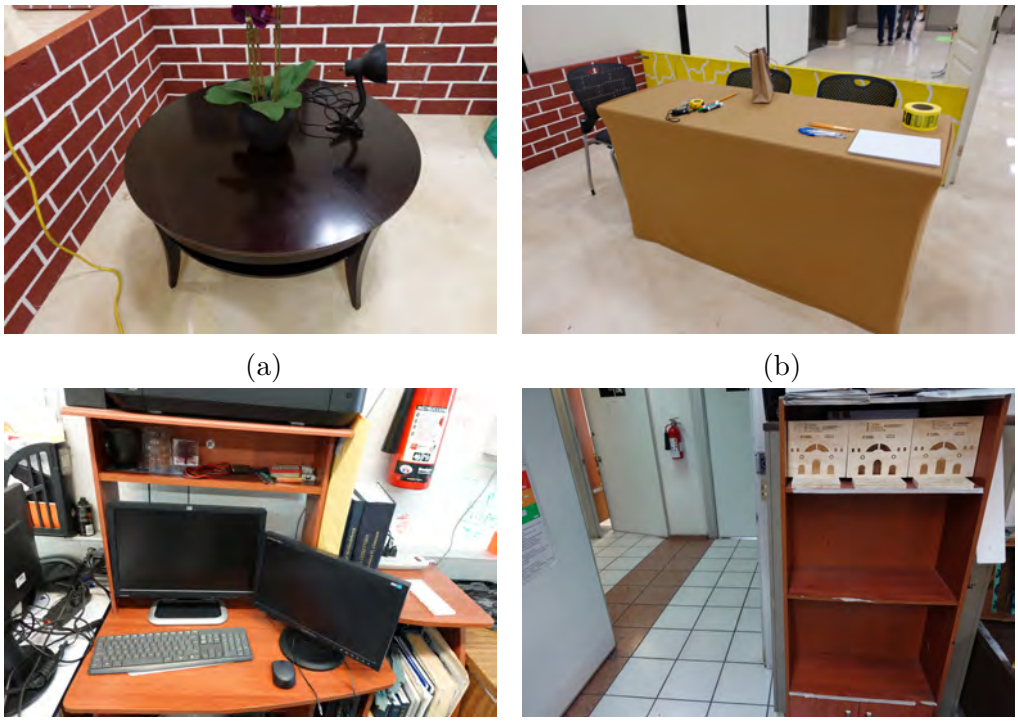


Figura 5.9: Diagrama de bloques del sistema de creación de escenas sintéticas.



(a) (b)

(c) (d)

Figura 5.10: Imágenes reales para el conjunto de entrenamiento y el conjunto de validación.

Configuración del sistema

Antes de la ejecución del programa, se cargan los datos de configuración de tres archivos de texto. Se tiene un archivo que configura las banderas de depuración del sistema, donde en cada método de depuración, se muestra la imagen resultado de aplicar las técnicas de aumento de datos, o se muestran las escenas sintéticas generadas. En el segundo archivo se configura el tamaño de la imagen de salida. Y en el tercer archivo, se configura el número de imágenes sintéticas que se crearán para el conjunto de entrenamiento y para el conjunto de validación.

Ejecución del sistema

Como primer paso, el sistema carga el nombre de las clases de los objetos segmentados del archivo de texto creado por el módulo anterior. Posteriormente, se revisa cuántas imágenes se tienen de cada clase y cuántas escenas

reales se tienen para el conjunto de entrenamiento y el conjunto de validación. La ruta de cada imagen se almacena como una constante en un vector de tipo *string*.

Cada escena sintética se procesa en paralelo. Primero se aplican técnicas de aumento de datos a los objetos segmentados, como desplazamientos y cambio de tamaño. Estos objetos son colocados de manera dinámica en la escena real, sin sobreponerse entre ellos. Después, se aplican técnicas de aumento de datos a toda la imagen, como cambio de contraste, cambio de brillo y difuminación a la escena. Por último, el sistema genera las etiquetas (o cajas delimitadoras) para cada clase de objeto presente en la escena sintética.

Aumento de datos en los objetos segmentados

Para evitar la repetitividad en la creación de escenas sintéticas, se utiliza la siguiente metodología. Se eligen de manera aleatoria los siguientes elementos de cada conjunto de imágenes:

- Una clase de las clases de objetos segmentados del módulo anterior.
- Un objeto dentro de la clase elegida.
- Una escena real, ya sea para el conjunto de entrenamiento o para el conjunto de validación.

Una vez elegido el objeto segmentado, se aplican técnicas de aumento de datos para robustecer el conjunto de imágenes que se le presentan a la red neuronal convolucional.

Desplazamientos. Se utiliza este tipo de aumento de datos para simular las oclusiones que se le pueden presentar a un objeto en una escena real. Al 50 % de los objetos segmentados se le aplican desplazamientos, que pueden ser de cuatro tipos, desplazamientos a la izquierda, desplazamientos a la derecha, desplazamientos hacia arriba y desplazamientos hacia abajo. El porcentaje de pixeles que se desplazan se elige de manera aleatoria dentro el rango de 5 % al 40 % del total de la imagen del objeto.

La imagen resultante es un recorte de la imagen original dado por las siguientes ecuaciones:

Desplazamiento a la izquierda:
$P_1(x, y) = P(object_w * r, 0)$
$P_2(x, y) = P(object_w - (object_w * r), object_h)$

Desplazamiento a la derecha:

$P_1(x, y) = P(0, 0)$
$P_2(x, y) = P(object_w - (object_w * r), object_h)$

Desplazamiento hacia arriba:
$P_1(x, y) = P(0, object_h * r)$
$P_2(x, y) = P(object_w, object_h - (object_h * r))$

Desplazamiento hacia abajo:
$P_1(x, y) = P(0, 0)$
$P_2(x, y) = P(object_w, object_h - (object_h * r))$

Donde:

- P_1 es el punto superior izquierdo.
- P_2 es el punto inferior derecho
- r es el porcentaje de pixeles que serán desplazados.
- $object_w$ es el ancho del objeto segmentado.
- $object_h$ es el alto del objeto segmentado.

En la figura 5.11 se muestra un ejemplo de los cuatro tipos de desplazamientos aplicados. La figura 5.11b muestra un desplazamiento hacia abajo del 25 % de pixeles de la figura 5.11a. La figura 5.11d muestra un desplazamiento hacia arriba del 20 % de pixeles de la figura 5.11c. La figura 5.11f muestra un desplazamiento a la derecha del 35 % de pixeles de la figura 5.11e. La figura 5.11h muestra un desplazamiento a la izquierda del 20 % de pixeles de la figura 5.11g.

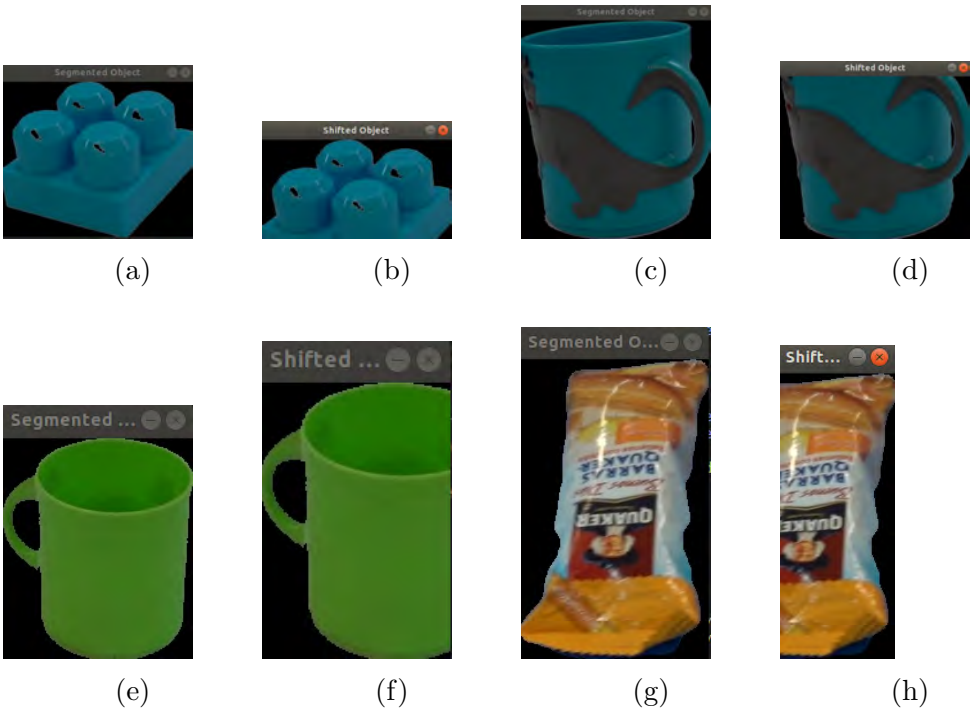


Figura 5.11: Desplazamientos en los objetos segmentados.

Cambio de tamaño. Para esta técnica de aumento de datos, se aplica el método **resize()** de OpenCV para disminuir el tamaño de la imagen del objeto segmentado, en proporción a un factor elegido aleatoriamente. El 50 % de los objetos segmentados disminuirán su tamaño en un rango de 1.3 a 3 veces su tamaño original.

En la figura 5.12b se muestra la reducción de tamaño de la imagen 5.12a por un factor de 1.666.

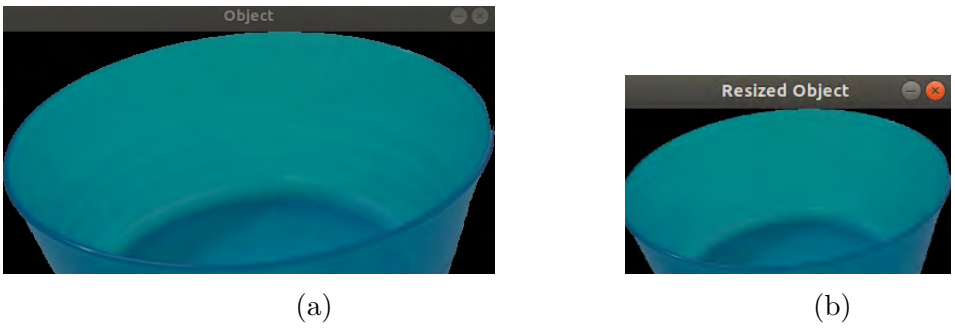


Figura 5.12: Cambio de tamaño en los objetos segmentados.

Proceso de colocación de los objectos segmentados en la escena real

La metodología para colocar los objetos segmentados en la escena real se dividió en dos enfoques diferentes:

Módulo de configuración de la escena real. Se creó un módulo donde se configura el pixel central en el cual se colocarán los objetos segmentados. Se utiliza un clic izquierdo del ratón para elegir varios puntos en distintos lugares de la escena, en los cuales podrían aparecer los objetos, por ejemplo, en mesas, en el piso, en sillas o en muebles de la cocina. Todos los puntos se guardan en un archivo de texto.

El módulo de creación de escenas sintéticas lee el archivo de texto y coloca el objeto segmentado en el pixel especificado. En la figura 5.13 se muestra una escena real con los pixeles configurados donde se colocarán los objetos segmentados.

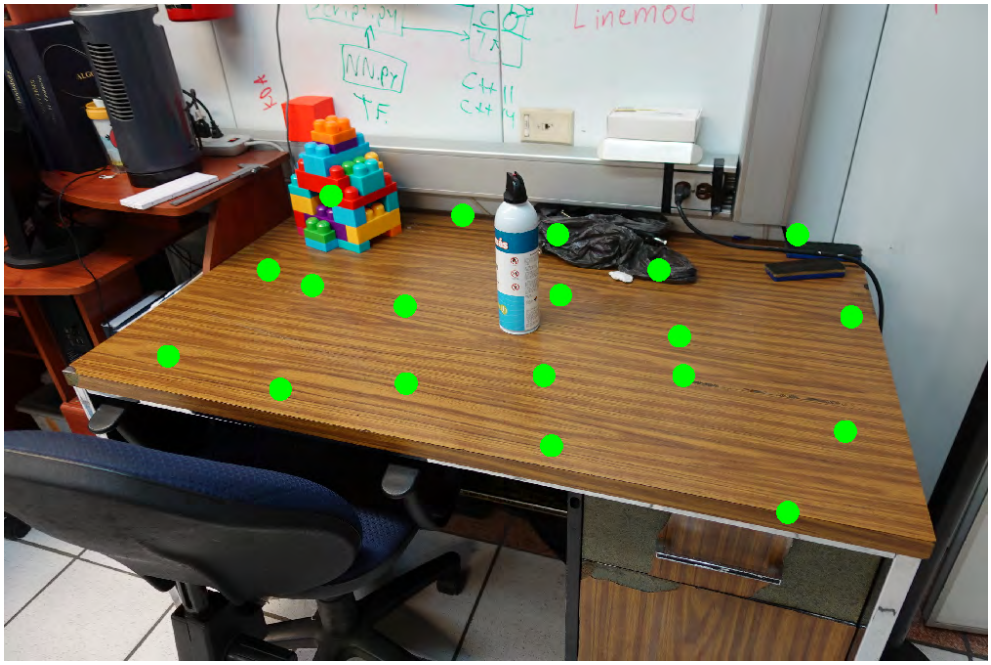


Figura 5.13: Módulo de configuración de la Escena Real.

Sin embargo, este enfoque tiene algunas desventajas:

- El usuario tiene que configurar todos los pixeles de todas las escenas donde se colocarán los objetos.

- En pruebas del sistema, algunos objetos se sobreponen a otros, llegando a ocluirlos totalmente. Esto provoca que la red sufra de sobre-ajuste, ya que se le están presentando ejemplos irrelevantes para la clase que tiene el objeto ocluido. Para resolver este problema, se deben configurar menos pixeles y separarlos una distancia considerable, lo cual provoca que se le presenten a la red neuronal menos objetos de ejemplo por cada escena sintética generada.

- En pruebas de reconocimiento se comprobó que la red neuronal sufre de sobre-ajuste. La red neuronal aprende a relacionar el fondo donde está colocado el objeto (como las mesas, sillas o el piso) con el objeto entrenado. Por lo cual, en la etapa de reconocimiento, el objeto se reconoce con alta confidencia solo si se encuentra sobre muebles similares a los que fue entrenado.

Debido a estas desventajas, se decidió utilizar otro enfoque para crear las escenas sintéticas.

Módulo de colocación de objetos segmentados. Para evitar el sobre-ajuste y la sobreposición entre objetos del *módulo de configuración de la escena real*, se decidió crear la escena sintética con la técnica de **collage**. Para llevar a cabo esta técnica, se elige el espacio en pixeles entre cada objeto segmentado en cuatro direcciones (izquierda, derecha, arriba y abajo). El primer objeto se coloca en la parte superior izquierda, y los demás objetos se colocan de izquierda a derecha hasta el límite derecho de la imagen. De la misma manera se llena la siguiente fila. Se repite el procedimiento hasta llegar al límite inferior de la imagen. Como resultado, se tiene la escena real llena de objetos segmentados.

En este módulo, se tiene un método que calcula el siguiente punto donde se colocará el objeto segmentado con base en sus dimensiones. Para llevar a cabo este procedimiento, se plantea el siguiente algoritmo:

Algorithm 1 Cálculo del pixel central donde se colocará el objeto

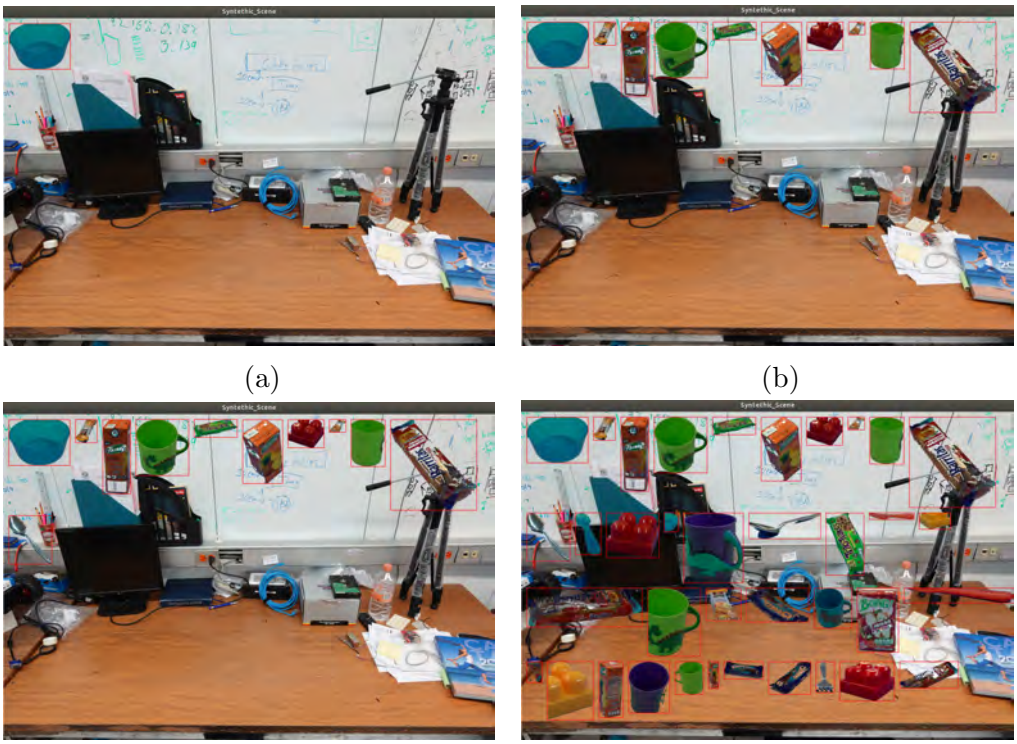
1: **procedure** $P_c(x, y)$
2: $P_n.x = S_i.x + P_s + O.x$
3: **if** $(P_n.x > S.x)$ **then**
4: $S_i.x = 0$
5: $S_i.y = S_n.y$
6: $S_n = P_s + O.x$
7: **end if**
8: $P_c.x = S_i.x + P_s + O.x/2$
9: $P_c.y = S_i.y + P_s + O.y/2$
10: $S_i.x = S_n.x$
11: **if** $(O.y + S_i.y + P_s > S_n.y)$ **then**
12: $S_n = O.y + S_i + P_s$
13: **end if**
14: **if** $(S_i.y > S.y)$ **then**
15: $return - false$
16: **end if**
17: $return\_true$
18: **end procedure**

Donde:

- P_c es el pixel central donde se colocará el objeto.
- P_n es el siguiente pixel central calculado.
- S_i es el pixel de referencia para colocar el siguiente objeto segmentado sin sobreponerlo a otro objeto.
- P_s es el espacio entre objetos dado en pixeles.
- O es la dimensión del objeto segmentado.

Al colocar cada objeto segmentado en la escena real, se crea un archivo de texto con los datos del etiquetado de cada objeto, compatible con el marco de trabajo de *darknet*. El cálculo se hace con respecto a las dimensiones de la escena real y las dimensiones del objeto segmentado. Los datos son relativos al tamaño de la escena sintética y se colocan de la siguiente manera:
(*número de clase*) (*pixel central en x*) (*pixel central en y*) (*ancho*) (*alto*)

En la figura 5.14 se muestra el procedimiento para la colocación de los objetos segmentados en la escena real. Se dibuja la caja delimitadora (en color rojo) de cada objeto para corroborar que sus etiquetas, necesarias para



(a) (b)

(c) (d)

Figura 5.14: Procedimiento para la creación de la escena sintética con la técnica de collage.

el entrenamiento de la red neuronal convolucional, estén colocadas de manera correcta. En la figura 5.14a se coloca el primer objeto en la parte superior izquieda. Los objetos se colocan de izquierda a derecha hasta llenar la primera fila (figura 5.14b). Después se coloca el siguiente objeto en la siguiente fila y se repite el procedimiento (figura 5.14c). El procedimiento acaba cuando ya no hay espacio para colocar más objetos (figura 5.14d).

Con este enfoque se tienen las siguientes ventajas:

- La colocación de los objetos la hace el sistema de manera eficiente.
- Ningún objeto se sobrepone a otro.
- Toda la imagen se utiliza para colocar los objetos, por lo cual se le presentan más ejemplos a la red en una misma imagen.
- Al colocar los objetos de manera dinámica en la escena, la red generaliza de una mejor manera, haciendo que las características que se aprendan

sean únicamente del objeto, diferenciándolas de las características de la escena.

Aumento de datos en la escena sintética

Una vez generada la escena sintética, al 50 % de escenas sintéticas se les aplican otras técnicas de aumento de datos para robustecer el conjunto de entrenamiento y el conjunto de validación.

Cambio de brillo y de contraste. El brillo hace referencia a la claridad u oscuridad de los pixeles de una imagen. Entre más brillo tenga una imagen, los pixeles tenderán al blanco, por el contrario, los pixeles tenderán al color negro. El contraste es la diferencia de intensidad entre los pixeles de una imagen. Al aumentar el contraste en una imagen, las áreas claras aumentarán su intensidad y las áreas oscuras disminuirán su intensidad.

Se puede utilizar el brillo y el contraste como técnica de aumento de datos y así aportar más ejemplos al conjunto de imágenes. Para ésto, se aplica la siguiente ecuación:

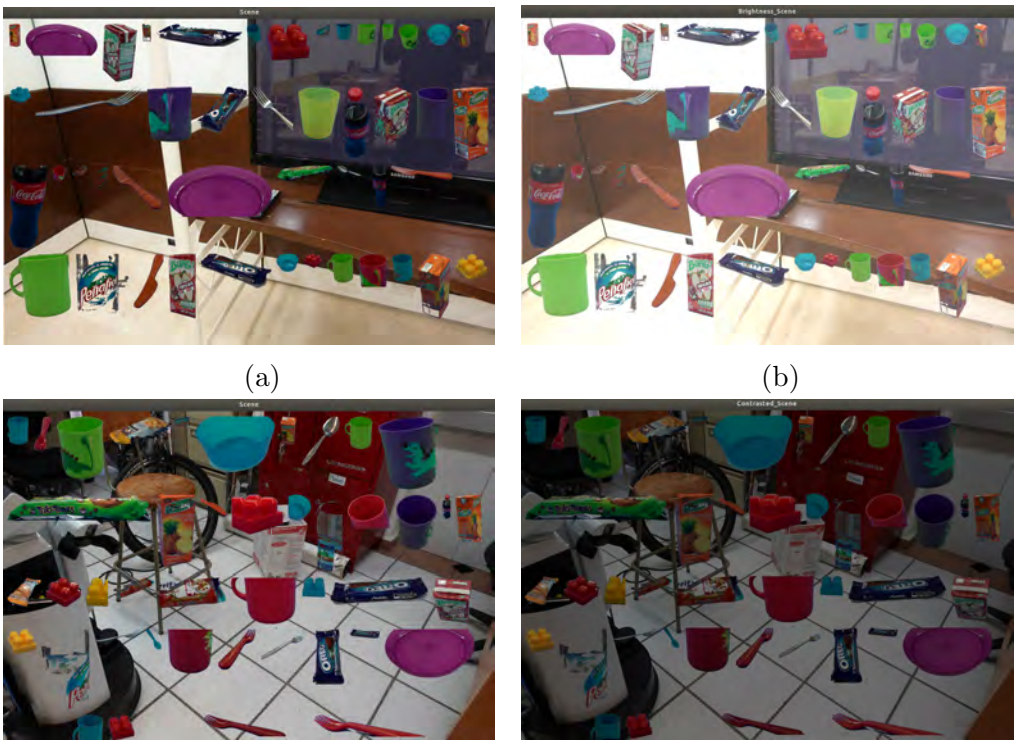
$$g(i,j) = \alpha * f(i,j) + \beta$$

Donde:

- $g(i,j)$ es la imagen resultado.
- $f(i,j)$ es la escena sintética.
- α o *ganancia*, es el parámetro que controla el contraste en la imagen.
- β o *bias*, es el parámetro que controla el brillo en la imagen.

El 18 % de las escenas sintéticas cambiarán de contraste. De igual manera, el 18 % de las escenas sintéticas cambiarán de brillo.

En la figura 5.15 se muestra el cambio de brillo y contraste en las escenas sintéticas. La figura 5.15b muestra un aumento de brillo aplicado a la figura 5.15a. La figura 5.15d muestra una disminución de contraste aplicado a la figura 5.15c.



(a) (b)

(c) (d)

Figura 5.15: Escenas sintéticas con cambio de brillo y contraste.

Difuminación de la escena sintética. El sistema de detección propuesto debe funcionar en un robot de servicio, por lo cual, algunas imágenes pueden ser capturadas durante algún movimiento del robot, ya sea un movimiento de la base o un movimiento de la cabeza (sitio donde se encuentra la cámara RGB-D). En este caso, la imagen adquirida podría estar difuminada. Para simular el ruido causado por una mala adquisición de la imagen, se utiliza el aumento de datos dado por una difuminación de la escena sintética. Lo anterior se logra aplicando un filtro Gaussiano. Se aplica este tipo de aumento de datos al 18 % de escenas sintéticas.

En la figura 5.16 se muestra el método de difuminación resultado de aplicar un filtro Gaussiano. El sistema elige aleatoriamente el tamaño del núcleo del filtro, de un rango de 3 a 15 pixeles. La imagen 5.16b es el resultado de aplicar un filtro Gaussiano con un núcleo de 15 pixeles de tamaño a la imagen 5.16a.

Este sistema puede generar miles imágenes sintéticas en pocos minutos de ejecución. Para cada una se crea un archivo de etiquetas, donde vienen las

(a) (b)

Figura 5.16: Difuminación de la escena sintética.

etiquetas para cada uno de los objetos que aparecen en las escenas sintéticas. Con este conjunto de imágenes, se puede re-entrenar la red neuronal convolucional YOLOv3 con resultados aceptables.

5.1.3. Entrenamiento de la red neuronal convolucional YOLOv3

El marco de trabajo *darknet*[33] hace la *transferencia de conocimiento* del modelo neuronal de YOLOv3, preentrenado para el conjunto de imágenes ImageNet[23], al nuevo modelo que será reentrenado.

Para llevar a cabo el reentrenamiento de la red, se necesitan los siguientes elementos:

- Un conjunto de imágenes etiquetadas.
- Descargar los pesos convolucionales preentrenados para ImageNet (*darknet53.conv.74*) de [26].
- El archivo *train.txt*, que contiene la ruta de todas las imágenes del lote de entrenamiento.
- El archivo *valid.txt*, que contiene la ruta de todas las imágenes del lote de validación.
- Configurar el archivo *yolov3.cfg*.
- Configurar el archivo *yolov3.names*.
- Configurar el archivo *yolov3.data*.

Conjunto de datos de imágenes etiquetado. El conjunto de imágenes etiquetadas son las escenas sintéticas generadas por el sistema anterior, y se dividen en el conjunto de entrenamiento y el conjunto de validación.

Configuración del archivo *yolov3.cfg*. Este archivo de configuración contiene la arquitectura neuronal de YOLOv3: las capas convolucionales, las capas de reducción, las capas residuales y las capas de clasificación. También contiene los parámetros necesarios para hacer el reentrenamiento:

- *batch*. Tamaño del lote de entrenamiento para actualizar los parámetros internos de la red.
- *subdivisions*. El número de subdivisiones que tendrá el parámetro *batch*. Se utiliza para no agotar los recursos de la tarjeta de video.
- *width* y *height*. El tamaño de la imagen de entrada.
- *channels*. El número de canales de la imagen de entrada.
- *learning_rate*. La taza de aprendizaje.
- *max_batches*. Número máximo de lotes que procesa la red neuronal en el proceso de entrenamiento.

Configuración del archivo *yolov3.names*. Este archivo contiene el nombre de las clases de objetos presentes en el conjunto de imágenes sintéticas.

Configuración del archivo *yolov3.data*. En este archivo se configuran los siguientes parámetros:

- *classes*. Número de clases presentes en el conjunto de datos.
- *train*. Ruta del archivo *train.txt*.
- *valid*. Ruta del archivo *valid.txt*.
- *names*. Ruta del archivo *yolov3.names*.
- *backup*. Ruta donde se almacenarán las copias de seguridad de los pesos generados.

Después de configurar los parámetros del marco de trabajo DarkNet, la red comienza el entrenamiento con el siguiente comando:

```
./darknet detector train yolov3.data yolov3.cfg
darknet53.conv.74
```

El entrenamiento de la red termina cuando se cumple el número máximo de lotes configurados con el parámetro *max_batches*, o se detiene cuando el promedio de error es muy bajo.

5.2. Sistema de detección de objetos

El sistema de detección de objetos propuesto tiene como objetivo detectar objetos de interés para un robot de servicio, y dependiendo del caso, dar la posición del objeto para una consecuente manipulación. Este sistema intercambia información con el módulo DarkNet-ROS[34] para hacer una detección precisa de múltiples objetos en una misma escena. El sistema se desarrolló en el lenguaje de programación C++ en el marco de trabajo ROS. En la figura 5.17 se muestra el diagrama de bloques a detalle del funcionamiento del sistema propuesto.

5.2.1. DarkNet-ROS

DarkNet-ROS es un módulo desarrollado en ROS que utiliza el marco de trabajo DarkNet para sistemas de detección de objetos. Como primer paso, DarkNet-ROS carga su configuración interna, la arquitectura de la red y los pesos entrenados. Se utiliza un archivo propio de ROS del tipo *launch* para ejecutar el nodo de detección y configurar la ruta de los siguientes archivos:

- El archivo *ros.yaml*. Este archivo configura el nombre de los tópicos[29] de los publicadores[29] (en inglés, *publishers*) y los subscriptores[29] (en inglés, *subscribers*) del nodo.
- El archivo *yolov3.weights*. Este archivo contiene los pesos re-entrenados de los objetos de interés.
- El archivo *yolov3.cfg*. Este archivo de configuración contiene la arquitectura neuronal de YOLOv3. Es el mismo archivo que se utilizó en la etapa de entrenamiento.
- El archivo *yolov3.yaml*. En este archivo se tienen configurados los nombres de las clases de los objetos entrenados.

Una vez ejecutado, el sistema espera a que otro nodo publique una imagen en el tópico configurado para comenzar el proceso de detección de objetos. Como resultado, DarkNet-ROS publica en un tópico la *clase del objeto*, la *confidencia* y las *coordenadas relativas a la imagen*(la caja delimitadora) de cada objeto detectado.

5.2.2. Nodo de detección de objetos

Como parte del sistema de un robot de servicio, se desarrolló un nodo que se encarga de intercambiar información con el nodo de DarkNet-ROS.

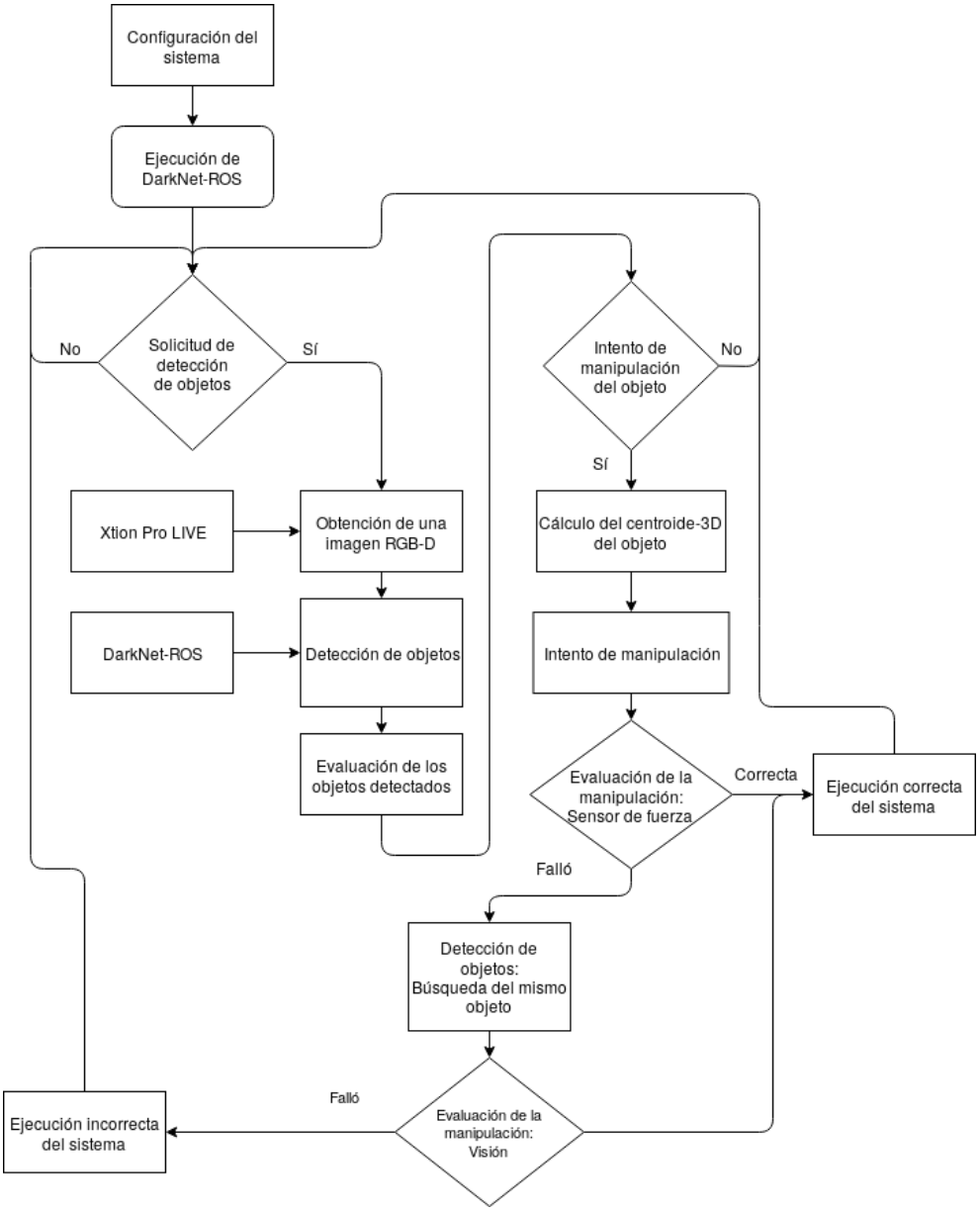


Figura 5.17: Diagrama de bloques del sistema de detección de objetos con DarkNet-ROS.

Este nodo captura una imagen RGB-D proveniente de la cámara del robot de servicio y publica la imagen RGB al tópico configurado en el nodo de DarkNet-ROS. Después, espera el resultado del nodo detección y evalúa los objetos detectados. Por último, si es el caso, el nodo da las coordenadas de la posición del objeto al nodo de manipulación para intentar tomarlo.

Este nodo tiene tres métodos principales, *detect_all_objects* (detección de todos los objetos), *detect_specific_object* (detección de objetos específicos) y *grasp_and_verify_object* (manipulación de un objeto con verificación).

El algoritmo propuesto para el método *detect_all_objects* es el siguiente:

Algorithm 2 Algoritmo *detect_all_objects*.

```
1: procedure detect_all_objects(objects)
2:     image_RGBD = capture_image_RGBD(Xtion_PRO_LIVE)
3:     image_RGB = get_image_RGB(image_RGBD)
4:     publish(image_RGB)
5:     if DarkNet_ROS_response then
6:         for all detected_objects do
7:             evaluate_object_position(objects)
8:             get_object_centroid3D()
9:             sort_objects()
10:            save_detected_object_data(objects)
11:        end for
12:    end if
13: end procedure
```

Donde:

- *evaluate_objects_position*(*objects*) es un método que evalúa la posición de los objetos detectados en un espacio tridimensional, y solo elige los que están dentro de un rango predefinido. Este método se utiliza para poder manipular solo los objetos cercanos al robot.
- *get_object_centroid*3*D*() es un método que calcula el centroide de los objetos detectados en un espacio tridimensional. En una tarea de manipulación, el robot colocará su manipulador en este punto. Siendo $f(x, y, z)$ un punto en el espacio tridimensional perteneciente a la caja delimitadora predicha $b(x, y)$, el centroide $C(x, y, z)$ es:

$$C(x, y, z) = \frac{1}{n} (\Sigma x, \Sigma y, \Sigma z) \ \ \forall f(x, y, z) \in b(x, y)$$

- *sort_objects*() es un método que ordena los objetos detectados por su

cercanía al robot de servicio. Siendo C el centroide calculado anteriormente y R las coordenadas relativas del robot respecto al mundo:

$$d = \sqrt{(C_x - R_x)^2 + (C_y - R_y)^2 + (C_z - R_z)^2}$$

- *save_detected_object_data*(*objects*) es un método que guarda el resultado de la detección y los datos calculados anteriormente en un tipo de mensaje predefinido.

El algoritmo *detect_specific_object* evalúa si la clase del objeto detectado es de interés para la tarea que se está ejecutando.

Algorithm 3 Algoritmo *detect_specific_objects*.

```
1: procedure detect_specific_objects(specific_object_classes)
2:     objects = detect_all_objects(objects)
3:     for all objects do
4:         if (object.class == specific_object_class) then
5:             save_object(specific_objects)
6:         end if
7:     end for
8: end procedure
```

Por último, se tiene el algoritmo *grasp_and_verify_objects*. En este algoritmo se intenta manipular un objeto, después se verifica de dos maneras si el objeto fue tomado. La primera verificación viene dada por el sensor de fuerza del manipulador del robot de servicio, si el objeto supera un umbral de peso se considera que la manipulación fue exitosa. En caso contrario, la manipulación pudo haber fallado o el objeto manipulado podría ser muy liviano, por lo cual, se hace una segunda detección del objeto que se intentó manipular. Si el objeto se detecta y se encuentra cerca de donde se detectó la primera vez, se considera como una manipulación fallida, y si la tarea lo permite, se intentará una segunda manipulación.

Algorithm 4 Algoritmo *grasp_and_verify_objects*.

```
1: procedure grasp_and_verify_objects(detected_objects)
2:     nearest_object = detected_objects[0]
3:     grasp_object(nearest_object)
4:     if (verify_with_force_sensor == false) then
5:         detect_specific_objects(nearest_object)
6:         if verify_with_vision == false then
7:             grasp_object_failed
8:         end if
9:     end if
10:    grasp_object_successful
11: end procedure
```

Capítulo 6

Pruebas y Resultados

En este capítulo se abordan las pruebas y resultados de los tres sistemas desarrollados a lo largo del presente proyecto de tesis. Se utilizaron 15 objetos de diferentes clases para medir el desempeño de los sistemas propuestos.

En la figura 6.1 se muestran las imágenes de los 15 objetos utilizados.

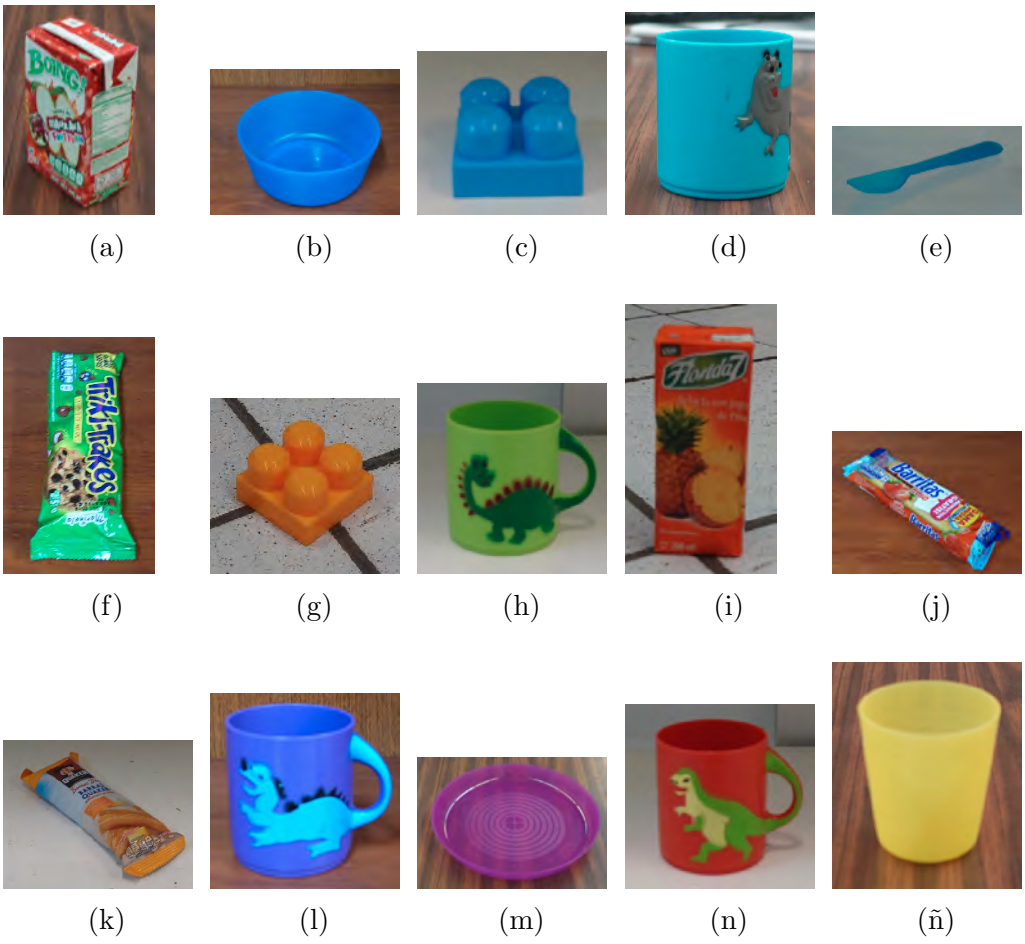
6.1. Pruebas y Resultados del sistema de creación de un conjunto de escenas sintéticas

Este sistema se divide en dos módulos, el módulo de segmentación de objetos y el módulo de creación de imágenes sintéticas. Para cada módulo se presentan las pruebas de su funcionamiento y los resultados obtenidos.

6.1.1. Pruebas del módulo de segmentación de objetos

Se capturaron 15 archivos de video (un archivo para clase de objeto) con una duración aproximada de 45 segundos y dimensiones de 1920x1080 pixeles. Cada archivo fue renombrado conforme a los requisitos del sistema y se colocó en la ruta configurada del sistema.

El sistema se ejecutó en la computadora de escritorio, descrita en la sección 4.2.3. Como primer paso, se configuraron los parámetros de color HSV para los diferentes colores de los fondos controlados (azul, blanco y verde). Después, el sistema se ejecutó y cada archivo de video fue procesado en paralelo de manera correcta.



(a) (b) (c) (d) (e)

(f) (g) (h) (i) (j)

(k) (l) (m) (n) (ñ)

Figura 6.1: Objetos elegidos para las pruebas del sistema. 6.1a Jugo de manzana. 6.1b Tazón azul. 6.1c Lego azul. 6.1d Taza azul. 6.1e Cuchara azul. 6.1f Galletas de chocolate. 6.1g Lego amarillo. 6.1h Taza verde. 6.1i Jugo de piña. 6.1k Galletas de piña. 6.1l Taza morada. 6.1m Plato morado. 6.1n Taza Roja. 6.1ñ Vaso amarillo

6.1.2. Resultados del módulo de segmentación de objetos

La ejecución del sistema fue satisfactoria y se procesaron todos los videos contenidos en la carpeta configurada. Como resultado se tienen los siguientes datos:

- El sistema tuvo un tiempo de ejecución de **5 min. 57 s.** para el procesamiento de los **15** archivos de video.
- El sistema segmentó **38,029** objetos de un total de **40,200** cuadros de video. Cabe mencionar que cada objeto tiene asociada una imagen que representa su máscara binaria.

Mediante la inspección visual de un usuario, se tomaron los siguientes criterios para evaluar el resultado de la segmentación de cada objeto.

- El objeto segmentado debe tener más del **80 %** del área total del objeto real.
- El objeto segmentado no debe contener partes de la escena superiores al **10 %** de su tamaño real.

De acuerdo con estos criterios, la tabla 6.1 muestra los resultados detallados del sistema de segmentación de objetos para cada archivo de video.

Nombre del objeto contenido en el archivo de video	**Duración (s.)**	**Número de cuadros**	**Número de objetos correctamente segmentados**	**Porcentaje de objetos correctamente segmentados**
1.Jugo de manzana	47	2,820	2,779	98.54
2.Galletas de chocolate	45	2,700	2,700	100.00
3.Jugo de piña	48	2,910	2,729	93.78
4.Galletas de piña	50	3,000	2,917	97.23
5.Taza roja	47	2,820	2,374	84.18
6.Vaso amarillo	42	2,520	2,473	98.13
7.Lego amarillo	41	2,490	2,367	95.06
8.Taza morada	43	2,610	2,172	83.21
9.Tazón azul	37	2,250	2,190	97.33
10.Lego azul	44	2,640	2,604	98.63
11.Taza azul	47	2,820	2,735	96.98
12.Cuchara azul	41	2,490	2,476	99.43
13.Plato morado	38	2,280	2,007	88.02
14.Taza verde	48	2,880	2,539	88.15
15.Galletas de fresa	49	2,970	2,967	99.89
Total	**667**	**40,200**	**38,029**	**94.59**

Tabla 6.1: Resultados del módulo de segmentación de objetos para cada archivo de video.

Para resaltar estos resultados, en la figura 6.2 se muestra el porcentaje de objetos segmentados de acuerdo al número de cuadros que se capturaron en cada archivo de video.

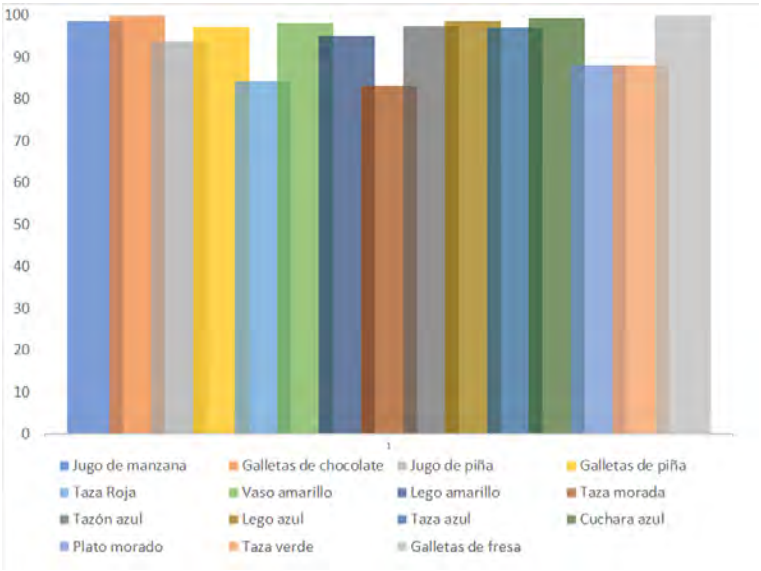


Figura 6.2: Porcentaje de objetos correctamente segmentados en relación a los cuadros capturados en cada archivo de video.

6.1.3. Análisis de resultados

En la mayoría de los archivos de video se obtuvieron resultados superiores al 98 % de objetos correctamente segmentados. Sin embargo, objetos como la taza roja y la taza verde tuvieron porcentajes de segmentación menores al 90 %, debido a que el sistema falló al detectar los bordes de la base de estos objetos, por lo cual, en varias imágenes se segmentó únicamente la figura que tienen en una de sus caras (el dinosaurio).

En promedio se tienen **2,500** objetos segmentados por cada clase, cada uno de ellos varía del otro en perspectiva y en rotación. Por esta razón, se considera que la ejecución del sistema de segmentación de objetos fue exitosa y generó un buen conjunto de imágenes de objetos segmentados para crear las escenas sintéticas del siguiente módulo.

6.1.4. Pruebas del módulo de creación de imágenes sintéticas

La ejecución de este módulo se llevó a cabo en la computadora de escritorio descrita en la sección 4.2.3. Antes de ejecutar el módulo, se configuraron los siguientes parámetros iniciales:

- Adquisición de las escenas reales. Para las pruebas se utilizaron 80 escenas reales, 90 % para el conjunto de entrenamiento y 10 % para el conjunto de validación.
- Número de escenas sintéticas para el conjunto de entrenamiento.
- Número de escenas sintéticas para el conjunto de validación.
- Tamaño de salida de la escena sintética: **1280x720** pixeles.

El sistema se ejecutó correctamente y generó cuatro conjuntos diferentes de escenas sintéticas.

6.1.5. Resultados del módulo de creación de imágenes sintéticas

Para evaluar el desempeño de este módulo, se generaron cuatro conjuntos de imágenes sintéticas con una cantidad diferente de imágenes, cada uno se utilizará para re-entrenar un modelo diferente de la red neuronal convolucional:

- **2,500** imágenes sintéticas. 2,250 para el conjunto de entrenamiento y 250 para el conjunto de validación. Tiempo de ejecución del sistema: **0 min 40 s.**.
- **5,000** imágenes sintéticas. 4,500 para el conjunto de entrenamiento y 500 para el conjunto de validación. Tiempo de ejecución del sistema: **1 min 18 s.**.
- **10,000** imágenes sintéticas. 9,000 para el conjunto de entrenamiento y 1,000 para el conjunto de validación. Tiempo de ejecución del sistema: **2 min 26 s.**.
- **20,000** imágenes sintéticas. 18,000 para el conjunto de entrenamiento y 2,000 para el conjunto de validación. Tiempo de ejecución del sistema: **4 min 54 s.**.

A continuación, se muestran a detalle los resultados de la ejecución del sistema en dos tablas. En la tabla 6.2 se muestra el número de objetos segmentados que se colocaron en cada conjunto de escenas sintéticas (C.E.S).

Nombre del Objeto	Número de objetos en el C.E.S. de 2,500 imágenes	Número de objetos en el C.E.S. de 5,000 imágenes	Número de objetos en el C.E.S. de 10,000 imágenes	Número de objetos en el C.E.S de 20,000 imágenes
1.Jugo de manzana	8,439	16,758	33,433	66,651
2.Galletas de chocolate	8,252	16,572	33,272	66,356
3.Jugo de piña	8,384	16,612	33,416	66,689
4.Galletas de piña	8,351	16,640	33,090	66,991
5.Taza roja	8,262	16,578	33,171	66,791
6.Vaso amarillo	8,368	16,480	33,255	66,726
7.Lego amarillo	8,418	16,673	33,184	66,382
8.Taza morada	8,457	16,449	33,071	66,786
9.Tazón azul	8,283	16,666	33,440	67,040
10.Lego azul	8,313	16,765	33,644	66,739
11.Taza azul	8,129	16,879	33,485	67,128
12.Cuchara azul	8,302	16,727	33,499	66,922
13.Plato morado	8,371	16,648	33,567	67,048
14.Taza verde	8,400	16,784	33,571	66,731
15.Galletas de fresa	8,242	16,862	33,030	66,426
Total de Objetos	**124,971**	**250,093**	**500,128**	**1,001,406**

Tabla 6.2: Número de objetos segmentados que se colocaron en cada conjunto de escenas sintéticas (C.E.S.).

En la tabla 6.3 se muestran el número de escenas reales y el número de objetos segmentados a los cuales se les aplicaron técnicas de aumento de datos.

Tipo de Aumento de datos	C.E.S. de 2,500 imágenes	C.E.S. de 5,000 imágenes	C.E.S. de 10,000 imágenes	C.E.S. de 20,000 imágenes
Número de objetos con desplazamientos	62,382	124,685	249,404	499,899
Número de objetos con cambios de tamaño	62,273	124,811	249,987	499,608
Número de escenas con cambios de brillo	480	949	1,796	3,706
Número de escenas con cambios de contraste	475	902	1,838	3,673
Número de escenas con difuminación	481	889	1,889	3,602

Tabla 6.3: Número de escenas reales y número de objetos segmentados a los cuales se les aplicó técnicas de aumento de datos.

6.1.6. Análisis de resultados

Todas las imágenes sintéticas se generaron de manera correcta, cada una con sus correspondientes etiquetas.

Se hicieron cuatro ejecuciones del sistema, donde se se generaron cuatro conjuntos diferentes de escenas sintéticas etiquetadas para el entrenamiento de cuatro modelos diferentes de la red neuronal convolucional. En promedio se tienen 40 objetos en cada escena, por lo cual la red recibe más ejemplos en cada imagen y el aumento de datos hace más robusto el conjunto de entrenamiento y el conjunto de validación. En todos los conjuntos de imágenes sintéticas, se tiene la cantidad suficiente de objetos para que la red neuronal pueda re-entrenarse de manera correcta. En la figura 6.3 se muestran ejemplos de las escenas sintéticas generadas.

Figura 6.3: Escenas sintéticas del sistema.

6.1.7. Entrenamiento de la red neuronal convolucional

El entrenamiento de los modelos de la red neuronal convolucional YOLOv3 se hicieron en la computadora de escritorio descrita en la sección 4.2.3, utilizando una tarjeta de video Nvidia Quadro P4000. Para cada modelo se eligieron los siguientes parámetros:

- La arquitectura *YOLOv3_tiny*.
- *batch*=64.
- *subdivisions*=4.
- *width*=416.
- *height*=416.

- *channels*=3.
- *learning_rate*=0.001.
- *max_batches*=500200.

Se fijó un tiempo máximo de entrenamiento de 8 horas, tiempo establecido para una noche de entrenamiento antes de una competencia, por ejemplo la competencia de RoboCup[6].

La evaluación de la etapa de entrenamiento se lleva acabo utilizando las métricas mAP(ecuación 6.1) (siglas en inglés de *Mean Average Precision*) e IoU(ecuación 6.2) (siglas en inglés de *Intersection Over Union*), las cuales evalúan la clasificación y la detección de objetos respectivamente.

Siendo TP los verdaderos positivos y FP los positivos, mAP se define como:

$$mAP = \frac{1}{|clases|} \sum_{C \in clases} \frac{\#TP(C)}{\#TP(C) + \#FP(C)} \tag{6.1}$$

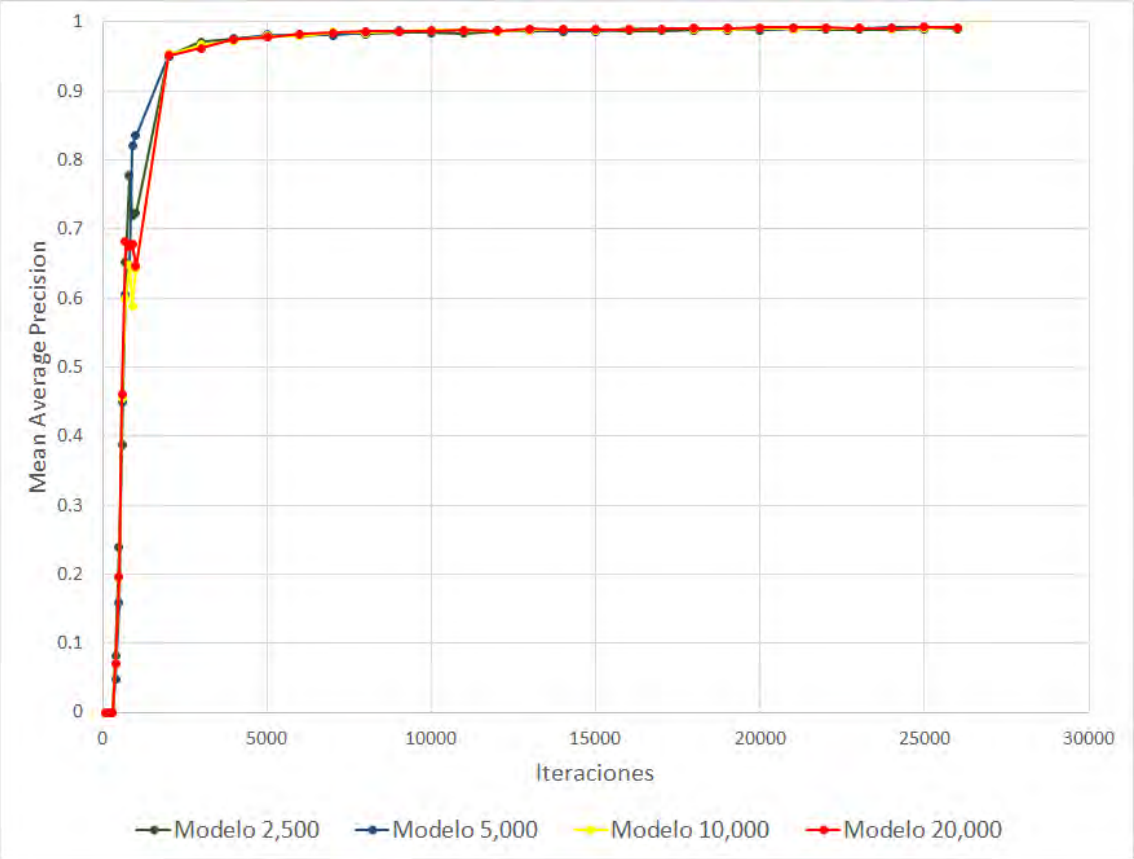
Siendo A el conjunto de pixeles predichos por la Red Neuronal y B el conjunto de pixeles del objeto del conjunto de entrenamiento, IoU se define como:

$$IoU(A, B) = \frac{A \cap B}{A \cup B} \tag{6.2}$$

Durante el proceso de entrenamiento de los modelos neuronales, se hizo el respaldo de los archivos de pesos entrenados cada 100 iteraciones hasta llegar a la iteración 1,000. Después se hizo el respaldo cada 1,000 iteraciones. Para cada uno de los archivos respaldados se hace la evaluación de las métricas establecidas. A continuación se muestran los resultados de ambas métricas para los cuatro modelos re-entrenados de la red neuronal convolucional (el modelo re-entrenado con 2,500 imágenes sintéticas, el modelo re-entrenado con 5,000 imágenes sintéticas, el modelo re-entrenado con 10,000 imágenes sintéticas y el modelo re-entrenado con 20,000 imágenes sintéticas).

En la figura 6.4 se muestran los resultados de las métricas **mAP** para los cuatro modelos entrenados.

En la figura 6.5 se muestran los resultados de las métricas **IoU** para los cuatro modelos entrenados.



(a)

Figura 6.4: Resultados de las métricas **mAP** para los cuatro modelos entrenados.

(a)

Figura 6.5: Resultados de las métricas **IoU** para los cuatro modelos entrenados.

6.1.8. Análisis de resultados

Las figuras 6.4 y 6.5 muestran que los cuatro modelos re-entrenados tienen la cantidad de ejemplos necesarios para que la red neuronal convolucional se entrene con altos índices en ambas métricas establecidas. Cabe resaltar que los resultados obtenidos son con base en el conjunto de validación, el cual también se creó con escenas sintéticas en el módulo de creación de escenas sintéticas. De igual manera, cabe resaltar que para la creación del conjunto de entrenamiento y el conjunto de validación se utilizaron diferentes escenas reales (ver sección 5.1.2).

Respecto a la métrica **mAP**, el entrenamiento rápidamente converge en la iteración 5,000 en todos los modelos neuronales, llegando a **99 %** de **mAP**. En las siguientes iteraciones el resultado se mantiene constante.

Respecto a la métrica **IoU**, el índice llega rápidamente al 80 % en la iteración 5,000. Después aumenta hasta llegar al **85 %** de **IoU** en la iteración 15,000.

Se concluye que a partir de la iteración **15,000** (**4.5** horas de entrenamiento) no se tienen mejoras significativas en los modelos neuronales.

Como último paso para esta evaluación, se creó un conjunto de prueba, es decir, un conjunto de imágenes etiquetadas a mano con los objetos utilizados para el entrenamiento de la red neuronal convolucional. El procedimiento se describe en la siguiente sección.

6.1.9. Conjunto de prueba para la evaluación de los modelos neuronales

Para la creación del conjunto de prueba, se tienen lugares específicos para colocar los 15 objetos elegidos para este proyecto de tesis: una mesa, un refrigerador, un estante, un casillero y el piso. Para cada lugar y para cada objeto, se capturaron 16 escenas con la cámara compacta descrita en la sección 4.2.2. En la tabla 6.4 se muestra el procedimiento de captura de cada imagen. En cada toma, el objetó se rotó 45°sobre su eje. Dos imágenes de cada clase de objetos fueron capturadas con desenfoque.

Distancia del objeto al foco de la cámara (m.)	Inclinación de 0° respecto al plano horizontal	Inclinación de 30° respecto al plano horizontal	Inclinación de 45° respecto al plano horizontal	Inclinación de 60° respecto al plano horizontal
0.3	✓	✓	✓	✓
0.5	✓	✓	✓	✓
1.0	✓	✓	✓	✓
1.5	✓	✓	✓	✓

Tabla 6.4: Procedimiento para la adquisición del conjunto de imágenes de prueba para cada objeto.

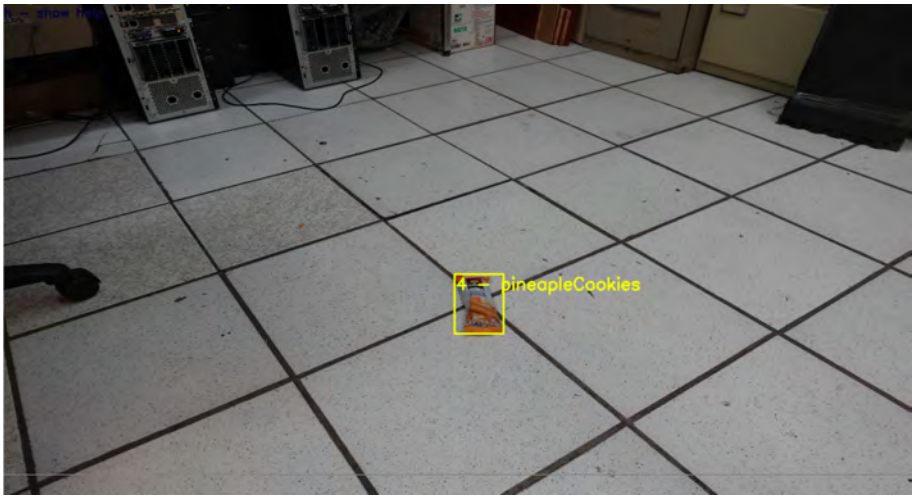
Después de la adquisición de todas las imágenes (un total de **1200** imágenes), se utilizó el sistema *Yolo_Mark*[35] para el etiquetado de los objetos. La ventaja de *Yolo_Mark* es que genera un archivo de etiquetas específico para el entrenamiento de la red neuronal convolucional, además de proveer herramientas para etiquetar los objetos de manera sencilla. Para poder hacer una correcta evaluación de los modelos neuronales entrenados, el etiquetado de cada objeto debe ser preciso. Cada objetó se etiquetó en 10 s., por lo consiguiente, el etiquetar todo el conjunto de prueba tardó aproximadamente **3 horas, 20 minutos**.

En las figuras 6.6, 6.7 y 6.8 se muestran ejemplos del conjunto de prueba etiquetado.

(a)

Figura 6.6: Objetos etiquetados del conjunto de prueba.

(a)



(b)

(c)

Figura 6.7: Objetos etiquetados del conjunto de prueba.

(a)

Figura 6.8: Objetos etiquetados del conjunto de prueba.

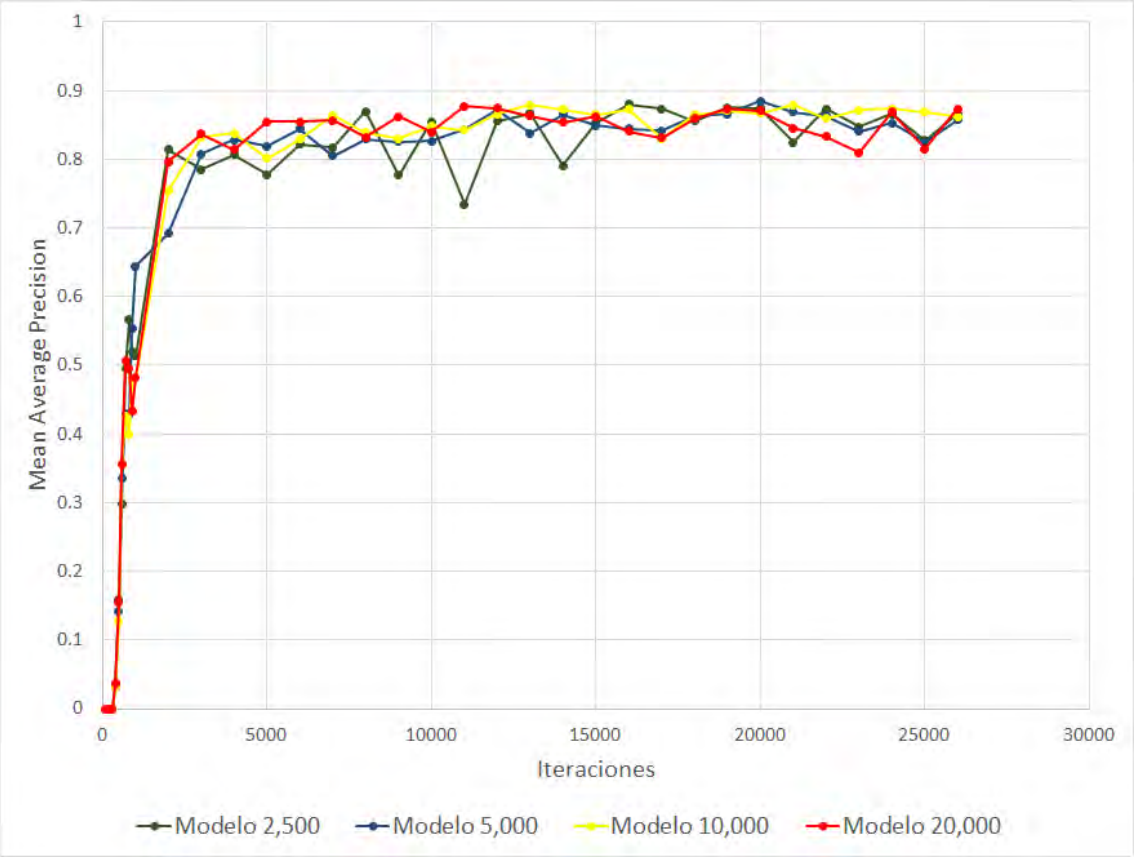
Una vez elaborado el conjunto de prueba, se evaluaron los cuatro modelos neuronales re-entrenados anteriormente, de la misma forma que la evaluación del conjunto de validación (sección 6.1.8).

En la figura 6.9 se muestran los resultados de la métrica **mAP** para los cuatro modelos neuronales.

6.1.10. Análisis de resultados

El índice de mAP de los cuatro modelos neuronales rápidamente llega al 80 % de mAP en la iteración 5,000. Entre más grande sea el conjunto de entrenamiento, el índice llegará a su máximo más rápido. Los modelos neuronales de 5,000, 10,000 y 20,000 imágenes oscilan entre el **83 %** y el **88 %**de **mAP** a partir de la iteración 10,000 y de ahí no se tienen mejoras significativas. El modelo neuronal de 2,500 imágenes oscila entre en 72 % y el 88 % a partir de la iteración 10,000. Por lo cual se concluye que este modelo neuronal necesita más ejemplos para poder generalizar de manera correcta con menos iteraciones.

En comparación con una base de imágenes etiquetada real, como COCO [24], el modelo neuronal entrenado de YOLOv3 tiene resultados de **57.9 %** de **mAP** para 80 clases de objetos. La mayoría de modelos neuronales entrenados para este proyecto de tesis tienen resultados del **85 %** de **mAP** para 15 clases de objetos.



(a)

Figura 6.9: Resultado de las métricas **mAP** para los cuatro modelos neuronales entrenados.

6.1.11. Pruebas del nodo de detección de objetos

Con base en los resultados obtenidos en los modelos neuronales entrenados anteriormente, el nodo de detección de objetos propuesto se tomó como el sistema de detección de objetos predefinido dentro del sistema del robot Takeshi[30]. Los resultados de las métricas de la figura 6.9a, muestran que se tienen buenos resultados en el reconocimiento de objetos en tan solo tres horas de entrenamiento (10,000 iteraciones) con **85 % de mAP**.

En la figura 6.10 se muestra una imagen adquirida por la cámara RGB-D del robot Takeshi, la cual fue procesada por la red neuronal convolucional con un modelo entrenado en tres horas. La mayoría de objetos que aparecen en la figura 6.10 son utilizados como los objetos de análisis de este proyecto de tesis. Otros objetos, como la cuerda, el billete, el trapo rojo o la envoltura de chocolate, fueron objetos predefinidos para la competencia **HSR-Challenge-4**[9], cuya fecha fue el 18 de Agosto del 2019.

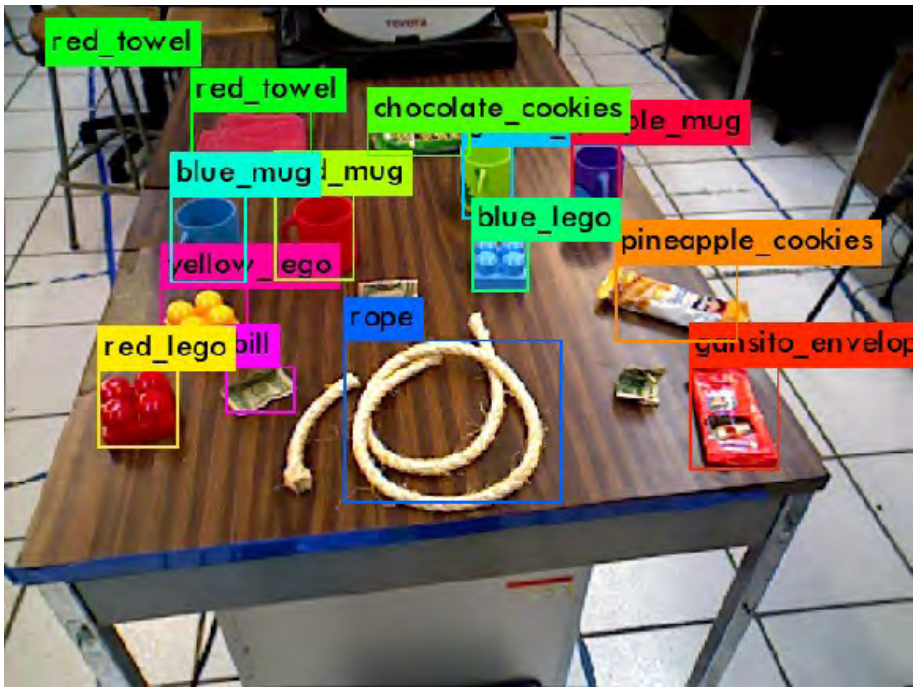


Figura 6.10: Nodo de Detección de Objetos. Detección con un modelo neuronal de tres horas de entrenamiento.

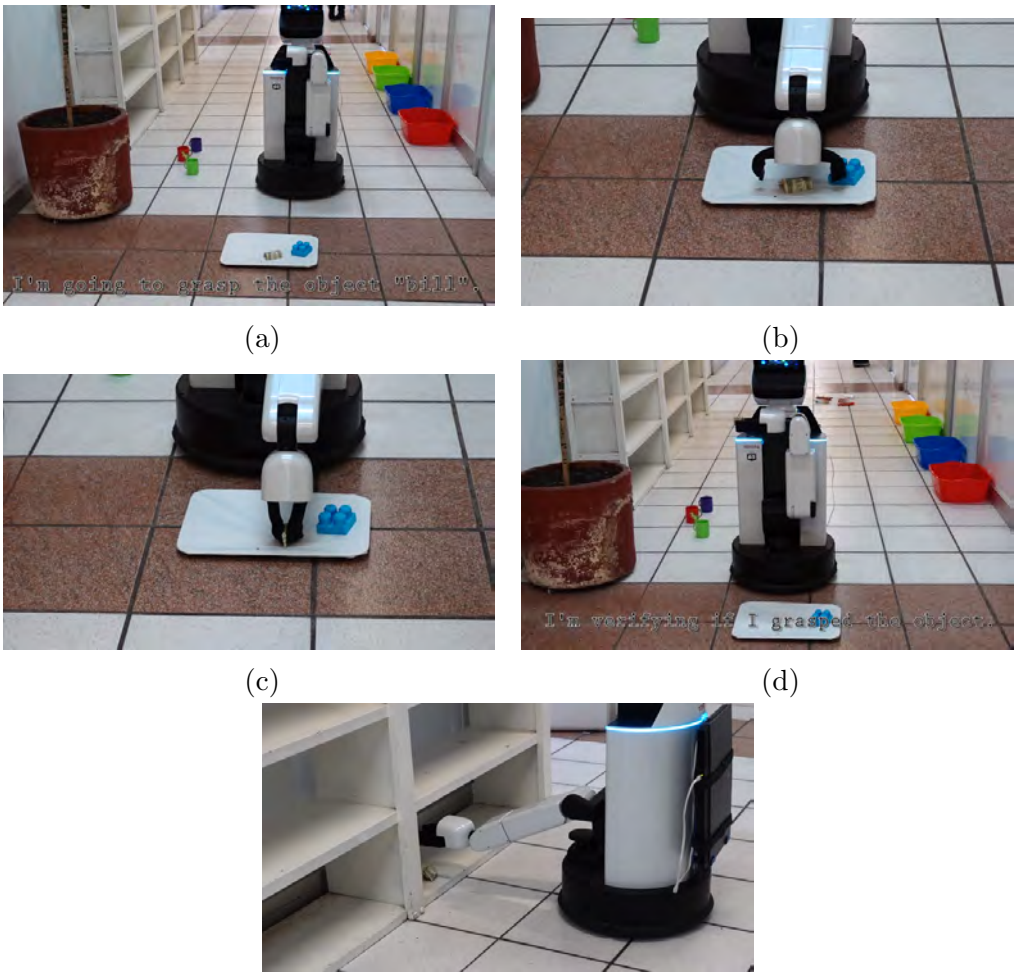
6.1.12. HSR-Challenge y RoboCup@home

Los sistemas propuestos en este proyecto de tesis se utilizaron dentro del sistema del robot Takeshi en varias competencias a nivel mundial, como **RoboCup@home 2019**[6] y los **HSR Challenge**[9] propuestos por la comunidad de desarrollo del robot HSR de Toyota. En ambas competencias hay tareas específicas que incluyen la detección y manipulación de objetos.

El objetivo de la competencia *HSR Challenge* es detectar un objeto, manipularlo y dejarlo en un lugar predefinido. En la figura 6.11a, el robot utiliza el nodo de detección propuesto para detectar el objeto *bill* (del inglés, *billete*). Posteriormente se calcula la posición a la cual debe de colocarse el manipulador del robot (figura 6.11b) para cerrar su pinza (figura 6.11c). Por último, se hace una verificación por una segunda detección, ya que el objeto *bill* es muy liviano (figura 6.11d). Para finalizar, el robot coloca el objeto en su sitio predefinido. (figura 6.11e).

Para la competencia HSR-Challenge-3, celebrada el 13 de abril del 2019, se entrenaron **12** objetos específicos. El modelo utilizado en esta competencia tuvo un tiempo de entrenamiento de **8 horas** y se utilizaron **9,000** imágenes sintéticas para el conjunto de entrenamiento y **1,000** para el conjunto de validación. En la tabla 6.5 se muestra la matriz de confusión para el sistema de detección presentado en la prueba HSR-Challenge-3. La prueba completa se puede ver en el canal de YouTube del robot Takeshi en [9].

Dados los resultados de la tabla 6.5 se puede concluir que el sistema pudo clasificar correctamente todos los objetos que se le presentaron. Además, se detectaron correctamente todos los objetos, razón por la cual la manipulación fue precisa.



(a) (b) (c) (d) (e)

Figura 6.11: Funcionamiento de los sistemas propuestos en la competencia HSR-Challenge[9].

	a	b	c	d	e	f	g	h	i	j	k	l
a	1	-	-	-	-	-	-	-	-	-	-	-
b	-	1	-	-	-	-	-	-	-	-	-	-
c	-	-	1	-	-	-	-	-	-	-	-	-
d	-	-	-	1	-	-	-	-	-	-	-	-
e	-	-	-	-	1	-	-	-	-	-	-	-
f	-	-	-	-	-	2	-	-	-	-	-	-
g	-	-	-	-	-	-	1	-	-	-	-	-
h	-	-	-	-	-	-	-	1	-	-	-	-
i	-	-	-	-	-	-	-	-	1	-	-	-
j	-	-	-	-	-	-	-	-	-	1	-	-
k	-	-	-	-	-	-	-	-	-	-	1	-
l	-	-	-	-	-	-	-	-	-	-	-	1

Tabla 6.5: Matriz de Confusión para la prueba del HSR-Challenge-3. a) Lego rojo. b) Lego azul. c) Lego verde. d) Billete. e) Taza roja. f) Taza verde. g) Taza morada. h) Galletas de cajeta. i)Galletas de fresa. j) Jabón. k) Galletas de piña. l) Jugo de manzana.

Capítulo 7

Conclusiones y Trabajo futuro

7.1. Conclusiones

Con base en los resultados obtenidos en el capítulo 6, se tienen las siguientes conclusiones para cada uno de los sistemas propuestos.

7.1.1. Módulo de segmentación de objetos

El módulo de segmentación de objetos es capaz segmentar el **94.59 %** de los cuadros totales de 15 archivos de video (de dimensiones de 1920x1080 pixeles), donde en promedio se tienen **2,500** objetos segmentados por cada clase, con un tiempo de ejecución de **5 minutos 57 segundos**. Por esta razón, para cada clase de objetos se tienen ejemplos con tomas de 360°al rededor del objeto y de 0°a 90°en el eje vertical, es decir, un conjunto de datos robusto para hacer el re-entrenamiento de un modelo neuronal con objetos propios.

Se concluye que el módulo de segmentación cumple con el objetivo: *desarrollar un sistema de visión que permita segmentar múltiples categorías de objetos utilizando información de una imagen bidimensional adquirida por una cámara digital compacta.*

7.1.2. Módulo de creación de imágenes sintéticas

El módulo de creación de imágenes sintéticas es capaz de crear grandes cantidades de imágenes en poco tiempo. Por ejemplo, para el conjunto de escenas sintéticas de 5,000 imágenes, el sistema tuvo un tiempo de ejecución de **1 minuto 58 segundos**. En promedio se tienen **16,000** objetos por cada clase, donde para su creación, se utilizaron técnicas de aumento de datos

para que, de esta manera, el conjunto de entrenamiento y validación sea muy robusto a las condiciones dinámicas de una escena real.

Se concluye que el módulo de creación de imágenes sintéticas cumple con el objetivo: *desarrollar un sistema de visión computacional que cree un conjunto de imágenes sintéticas robusto y etiquetado, para hacer el re-entrenamiento de una red neuronal convolucional.*

7.1.3. Nodo de detección de objetos

Con base en los resultados obtenidos en la sección 6.1.10, se desarrolló un nodo que intercambia información con el modelo neuronal YOLOv3 re-entrenado con el conjunto de imágenes sintéticas generado. Se tienen buenos resultados en la evaluación del modelo neuronal con un conjunto de prueba de escenas reales, con un índice de reconocimiento **mAP** del **85 %** para 15 clases de objetos propios.

Además, todos los sistemas se probaron en la competencia *HSR-Challenge-3*, donde se obtuvo el **100 %** de reconocimiento de 12 objetos utilizados en la prueba.

Por lo cual, se concluye que el nodo de detección de objetos propuesto cumple con el objetivo: *desarrollar un sistema de detección de objetos con redes neuronales convolucionales para un robot de servicio.*

Debido a estas razones, se comprobó la hipótesis planteada al inicio del proyecto: *El entrenamiento de un modelo neuronal con imágenes sintéticas generadas de manera automática, tendrá altos índices de precisión en la detección de objetos en una escena real. La implementación de este modelo mejorará el sistema de detección de objetos en un robot de servicio, aumentando los índices de confidencia en la etapa de reconocimiento y mejorando la detección de objetos en una escena tridimensional.*

7.2. Trabajo futuro

Como trabajo futuro, se plantean varias mejoras a los sistemas propuestos:

- Módulo de segmentación de objetos. Mejorar el procesamiento de bajo nivel de las escenas capturadas con la cámara compacta, para que de esta manera, se tenga una mejor detección de bordes del objeto de interés, con el objetivo de mejorar los índices de segmentación del sistema.

- Módulo de creación de imágenes sintéticas. Desarrollar un sistema que permita detectar con precisión los lugares específicos donde pueden aparecer los objetos, como mesas, estantes, refrigerados, etc. para que, de esta manera, los objetos sean colocados dinámicamente en estos lugares específicos. Además, incluir más técnicas de aumento de datos para robustecer los conjuntos de entrenamiento y validación.
- Tamaño de la imagen de entrada del modelo neuronal YOLOv3. Aumentar el tamaño de la imagen de entrada y evaluar los resultados.
- Arquitectura de la red neuronal. Explorar otras arquitecturas y evaluar el desempeño de los conjuntos de entrenamiento y validación.

Bibliografía

[1] iRobot. *iRobot Vacuum Cleaning*. 2018. URL: https://www.irobot.mx/ (visitado 08-08-2018).

[2] Ali Sharif Razavian y col. "CNN Features off-the-shelf: an Astounding Baseline for Recognition". En: *CoRR* abs/1403.6382 (2014). URL: http://arxiv.org/abs/1403.6382.

[3] Pierre Sermanet y col. "Overfeat: Integrated recognition, localization and detection using convolutional networks". English (US). En: *International Conference on Learning Representations (ICLR2014), CBLS, April 2014*. 2014.

[4] Olga Russakovsky y col. *Imagenet large scale visual recognition challenge 2013 (ilsvrc2013)*. 2013. URL: http://www.image-net.org/challenges/LSVRC/2013/. (visitado 08-08-2018).

[5] RoboCup Federation. *RoboCup*. 2016. URL: https://www.robocup.org/ (visitado 08-08-2018).

[6] RoboCup Federation. *RoboCup@Home*. 2016. URL: https://athome.robocup.org/ (visitado 08-08-2018).

[7] Federación Mexicana de Robótica. *Torneo Mexicano de Robótica*. 2016. URL: https://www.femexrobotica.org/tmr2019/ (visitado 08-08-2018).

[8] World Robot Summit. *WRS*. 2016. URL: https://worldrobotsummit.org/en/ (visitado 08-08-2018).

[9] Takeshi Team. *HSR-Challenge 3. Facultad de Ingeniería de la UNAM*. 2019. URL: https://www.youtube.com/watch?v=et75sn1KkZE (visitado 13-04-2019).

[10] Jesús Savage Carmona. *Laboratorio de Bio-Robótica de la Facultad de Ingeniería de la UNAM*. 2016. URL: https://biorobotics.fi-p.unam.mx/ (visitado 08-08-2018).

[11] J Canny. "A Computational Approach to Edge Detection". En: *IEEE Trans. Pattern Anal. Mach. Intell.* 8.6 (jun. de 1986), págs. 679-698. ISSN: 0162-8828.

[12] Rafael C. Gonzalez y Richard E. Woods. *Digital image processing*. Prentice Hall, 2008.

[13] Gregorio Molinero Díez. "Segmentación de imágenes en color basada en el crecimiento de regiones." Tesis doct. Universidad de Sevilla, Departamento de ingeniería, 2010.

[14] Lih-Jen Kau y Tien-Lin Lee. "An Efficient and Self-Adapted Approach to the Sharpening of Color Images". En: *TheScientificWorldJournal* 2013 (nov. de 2013), pág. 105945. DOI: 10.1155/2013/105945.

[15] George Stockman y Linda G. Shapiro. *Computer Vision*. 1st. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2001. ISBN: 0130307963.

[16] Ramesh Jain, Rangachar Kasturi y Brian Schunck. *Machine Vision*. Ene. de 1995. ISBN: 978-0-07-032018-5.

[17] Ian Goodfellow, Yoshua Bengio y Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[18] Raúl Rojas. *Neural Networks: A Systematic Introduction*. Berlin, Heidelberg: Springer-Verlag, 1996. ISBN: 3-540-60505-3.

[19] Pawan Jain. *Complete Guide of Activation Functions*. 2019. URL: https://towardsdatascience.com/complete-guide-of-activation-functions (visitado 08-08-2019).

[20] Sumit Saha. *A Comprehensive Guide to Convolutional Neural Networks*. 2018. URL: https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way (visitado 08-08-2019).

[21] Sabyasachi Sahoo. *Residual blocks — Building blocks of ResNet*. 2018. URL: https://towardsdatascience.com/residual-blocks-building-blocks-of-resnet (visitado 08-08-2019).

[22] Simeon Kostadinov. *Understanding Backpropagation Algorithm*. 2019. URL: https://towardsdatascience.com/understanding-backpropagation-algorithm (visitado 08-08-2019).

[23] Standford Vision Lab. *ImageNet*. 2016. URL: http://image-net.org (visitado 08-08-2018).

[24] Tsung-Yi Lin y col. "Microsoft COCO: Common Objects in Context". En: *European Conference on Computer Vision (ECCV)*. Oral. Zürich, 1 de ene. de 2014. URL: /se3/wp-content/uploads/2014/09/coco_eccv.pdf,http://mscoco.org.

[25] Na8. *Qué es overfitting y underfitting y cómo solucionarlo*. 2019. URL: https://www.aprendemachinelearning.com/que-es-overfitting-y-underfitting-y-como-solucionarlo/ (visitado 08-08-2019).

[26] Joseph Redmon y Ali Farhadi. "YOLOv3: An Incremental Improvement". En: *arXiv* (2018).

[27] Python Lessons. *YOLOv3 theory explained*. 2018. URL: https://medium.com/@pythonlessons0/yolo-v3-theory-explained (visitado 08-08-2019).

[28] OpenCV. *Open Source Computer Vision Library*. 2019. URL: https://opencv.org/ (visitado 08-08-2019).

[29] ROS. *ROS (Robot Operating System*. 2018. URL: https://www.ros.org/ (visitado 08-08-2018).

[30] Ui Yamaguchi y col. "HSR, Human Support Robot as Research and Development Platform". En: *The Abstracts of the international conference on advanced mechatronics : toward evolutionary fusion of IT and mechatronics : ICAM* 2015.6 (dic. de 2015), págs. 39-40. DOI: 10.1299/jsmeicam.2015.6.39.

[31] Sony. *Sony DSC-RX100M5*. 2019. URL: https://www.sony.com.mx (visitado 08-08-2019).

[32] Asus. *Xtion-PRO LIVE*. 2019. URL: https://www.asus.com/3D-Sensor/XtionPRO (visitado 08-08-2019).

[33] Joseph Redmon. *Darknet: Open Source Neural Networks in C*. http://pjreddie.com/darknet/. 2013–2016.

[34] Marko Bjelonic. *YOLO ROS: Real-Time Object Detection for ROS*. https://github.com/leggedrobotics/darknet_ros. 2016–2018.

[35] Alexey A.B. *Yolo Mark*. https://github.com/AlexeyAB/Yolo_mark. 2019.

[36] Matthew D. Zeiler y Rob Fergus. "Visualizing and Understanding Convolutional Networks". En: *CoRR* abs/1311.2901 (2013). arXiv: 1311.2901. URL: http://arxiv.org/abs/1311.2901.

[37] Jesús Cruz Navarro. "Detección y reconocimiento de objetos usando imágenes RGB y nubes de puntos organizadas". Tesis doct. Instituto de Investigaciones en Matemáticas Aplicadas y en Sistemas (IIMAS). Universidad Nacional Autónomas de México, 2016.

[38] Jungong Han y col. “Enhanced Computer Vision with Microsoft Kinect Sensor: A Review”. En: *IEEE Trans. Cybernetics* 43 (2013). URL: https://www.microsoft.com/en-us/research/publication/enhanced-computer-vision-with-microsoft-kinect-sensor-a-review/.

[39] David A. Forsyth y Jean Ponce. *Computer Vision - A Modern Approach, Second Edition.* Pitman, 2012, págs. 1-791. ISBN: 978-0-273-76414-4.

[40] Kurt Hornik. “Approximation Capabilities of Multilayer Feedforward Networks”. En: *Neural Netw.* 4.2 (mar. de 1991), págs. 251-257. ISSN: 0893-6080. DOI: 10.1016/0893-6080(91)90009-T. URL: http://dx.doi.org/10.1016/0893-6080(91)90009-T.

[41] Zachary Chase Lipton. “A Critical Review of Recurrent Neural Networks for Sequence Learning”. En: *CoRR* abs/1506.00019 (2015). arXiv: 1506.00019. URL: http://arxiv.org/abs/1506.00019.

[42] F. Rosenblatt. “The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain”. En: *Psychological Review* (1958), págs. 65-386.

[43] Nicholas T. Carnevale y Michael L. Hines. *The NEURON Book*. New York, NY, USA: Cambridge University Press, 2006. ISBN: 0521843219.

[44] Kaiming He y col. “Mask R-CNN”. En: *2017 IEEE International Conference on Computer Vision (ICCV)* (2017), págs. 2980-2988.