Universidad Nacional Autónoma de México

POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

Señales, Imágenes y Ambientes Virtuales

DESARROLLO DE HERRAMIENTA DE GENERACIÓN DE IMÁGENES EN AMBIENTE VIRTUAL PARA EL ENTRENAMIENTO DE REDES NEURONALES

T E S I S

QUE PARA OPTAR POR EL GRADO DE:

MAESTRO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

PRESENTA

Rodrigo Terpán Arenas

Director de Tesis:

Dr. Alfonso Gastélum Strozzi

ICAT, UNAM

Ciudad Universitaria, CDMX, Enero 2021

Agradecimientos

¿Cuál es el propósito de la vida? Durante años me he preguntado esto sin una respuesta clara. En este momento de mi vida, puedo llegar a la conclusión de que el propósito de la vida son las metas que uno se pone en la vida. Lo que uno quiere hacer combinado con lo que hay que hacer. Sin embargo, esta definición está incompleta y mutará con el paso de los años.

Dar el siguiente paso siempre es aterrador. En especial porque uno no sabe lo que va a pasar en el futuro. Uno tiene miedo de fallar, de quedar expuesto. Sin embargo, llegando a donde estoy, me doy cuenta de que, si me esfuerzo, no debo temer, ya que he llegado aquí gracias a mi esfuerzo y a la red de amigos y familiares que me han apoyado en todo este camino. Lo que he aprendido de ellos se refleja en mí. Y espero en el futuro poder seguir apoyándolos a ellos como ellos me apoyaron a mí.

A mi madre Silvia

Por guiarme y apoyarme en el transcurso de mi vida. Por inculcarme valores, conocimiento, cultura y hábitos para ser independiente. Enseñarme el valor del trabajo duro y ponerme en el camino correcto. Siempre has sido una persona muy fuerte, admiro tu dedicación y perseverancia.

A mi padre Leobardo

Por siempre estar allí apoyándome y escuchándome cuando lo necesito. Por inculcarme el valor de los libros y la música. Por perseverar por nosotros aun cuando el tiempo estaba en nuestra contra. Que nunca se te olvide que te quiero y te admiro mucho sin importar lo que pase.

A mi hermana Natalia

Hermanos unidos por siempre. ¿Verdad? Es muy posible que en el futuro cercano ya no podamos vernos de manera constante como lo hemos hecho todas nuestras vidas. Pero sin importar la distancia, siempre serás mi hermana que me ha acompañado toda mi vida, apoyándome y molestándome a la vez. Gracias por compartir tantos momentos conmigo. Espero que en el futuro podamos seguir haciéndolo.

A mi familia

Presente en la ciudad de México, Estados Unidos y Yucatán. Ustedes me han enseñado el valor de apoyarnos en familia. Siempre pienso en ustedes, sin importar la distancia.

A mis amigos

Adrian, Julieta, Jorge Luis, Víctor, Geovas, Jaime, Pablo, Jhovan, Misael, Said, Jennifer, Issac, Mariana. Gracias por ser mis amigos. Siempre se me ha dificultado crear conexiones con otros, por lo que en verdad aprecio la amistad que me han ofrecido. Gracias por reír conmigo, llorar conmigo, escucharme, apoyarme y aguantar mis ataques nerds.

A mi tutor Alfonso Gastelum Strozzi

Por apoyarme, guiarme y hacer posible este trabajo. Los temas que se han desarrollado son de mi interés y espero poder seguir trabajando en realidad virtual y aumentada en el futuro.

A lo profesores que me han guiado

Norma García Vasconcelos, Erik Castañeda de la Isla Puga, Lily María Raquel de Gortari, Jesús Manuel Dorador y Yair Bautista. Gracias a ustedes y a incontables profesores más, he podido avanzar paso a paso como profesional y como persona.

Finalmente, agradezco a las instituciones que me formaron en esta etapa de mi vida: La UNAM y el IIMAS. Por formarme como profesional y cultivar mi pensamiento científico. Aquí es donde aprendí que el camino a tomar no siempre es evidente.

Gracias a todos.

Rodrigo Terpán Arenas

"You can never know everything, and part of what you know is always wrong. Perhaps even the most important part. A portion of wisdom lies in knowing that. A portion of courage lies in going on anyways."

al'Lan Mandragoran (Escrito por Robert Jordan)- Wheel of time, Winters heart

"The truth is that the world is full of dragons, and none of us are as powerful or cool as we'd like to be. And that sucks. But when you're confronted with that fact, you can either crawl into a hole and quit, or you can get out there, take off your shoes, and Bilbo it up."

Patrick Rothfuss

En agradecimiento a la fundación CONACYT, quien permitió la realización de este proyecto.

Resumen

El laboratorio de BIO-Robótica de la Facultad de Ingeniería de la Universidad Nacional Autónoma de México trabaja con la programación y creación de algoritmos para robots de servicio. Uno de los problemas que se enfrenta el laboratorio con respecto al robot es el reconocimiento de diversos objetos en el ambiente. Una de las formas de solucionar este problema es mediante el uso de redes neuronales para el reconocimiento de imágenes. Sin embargo, las imágenes utilizadas para este entrenamiento no son representativas de un entorno realista ya que estas pueden presentar objetos en posiciones ilógicas como techos o paredes. Es necesario generar más datos para que la red pueda detectar objetos sin necesidad de recurrir a las imágenes previamente descritas. Una de las formas de poder resolver este problema es mediante el uso de ambientes virtuales.

En el presente trabajo se desarrolló un ambiente virtual en el motor de desarrollo gráfico Unity con el propósito de poder generar escenas compuestas de modelos 3D de objetos colocados en el ambiente de forma aleatoria. Este ambiente permite capturar imágenes y obtener información espacial sobre los objetos dentro de la imagen, con el propósito de utilizar estos datos para el entrenamiento de redes neuronales para el reconocimiento de objetos.

La metodología para la generación del trabajo es descrita desde la creación inicial del ambiente virtual, creación y búsqueda de modelos virtuales, programación de funciones de captura de información, desarrollo de una interfaz de usuario, generación de información y su implementación en entrenamiento de redes neuronales. Los datos generados fueron sometidos a entrenamientos en la red neuronal YOLOV3, realizándose un análisis comparativo entre los resultados generados, para determinar la utilidad de esta información.

Abstract

The laboratory of BIO-Robotics of the School of Engineering from the National Autonomous University of México (UNAM) currently works on the programming and testing of algorithms for service robots. One of the main problems the laboratory faces is the recognition of objects and agents in the local environment. Among the most prominent solutions lies the use of neural networks for image recognition. However, the images currently used by the lab, while functional, are not representative of a real environment in its entirety, examples of this include imposing objects in unrealistic positions like walls or ceilings. That brings the necessity of generating additional information with the objective of relying less on the modified images. One of the ways of doing this is using virtual environments.

The present work deals with the development of a virtual environment on the graphic engine Unity with the objective of generating scenes consisting of 3D models randomly placed on the environment. This tool allows the capture of both image and text information regarding the objects on the image, with the objective of using this information for training of image recognition neural networks.

The work methodology consists of the description of the creation of the virtual environment, beginning with the search and creation of 3d models, the programming of the necessary algorithms, the creation of a user interface, the capture of information and the use of that information in the training of neural networks. The network of choice for the trainings was the architecture YOLOV3, the results of these experiments were subject of a comparative analysis to determine the usefulness of the generated information.

Índice

Índice VIII
Índice de figuras X
Índice de tablas XI
Índice de Pseudocódigos XII
Índice de ecuaciones XII
Capítulo 1: Introducción 1
Definición del problema 2
Objetivos 3
Objetivo general: 3
Objetivos específicos: 3
Contribución y Relevancia 3
Organización del trabajo 4
Capítulo 2: Antecedentes 5
Robots de servicio 5
Problemas visuales. 7
Complejidad del entorno 8
Métodos de enseñanza para los robots 9
Dato: 9
Conjunto de datos: 10
Base de datos: 11
Preparación de bases de datos 11
Aprendizaje de máquina (Machine learning). 12
Redes neuronales 13
Neurona artificial 14
Redes multicapa. 18
Redes neuronales convolucionales 19
Arquitecturas importantes de RNC 20
YOLOV3 24
Aumento de datos (Data augmentation) 26
Ambientes virtuales 30

Clasificación de presentación de ambientes virtuales 31
Motor de creación de videojuegos para desarrollo 32
Capítulo 3: Desarrollo del sistema virtual 35
Elección del motor de desarrollo 35
Modelado de objetos 36
Organización de módulos 37
Módulo de automatización 39
Módulo de generación de listas 40
Módulo de generación de objetos 43
Preparación del entorno 46
Algoritmos: 47
Módulo de captura de información 48
Proyección a la pantalla 51
Estructura rectangular 54
Guardado de datos 57
Interfaz de usuario 58
Control de dirección 61
Control de entorno 61
Capítulo 4: Metodología de experimentación 64
Características técnicas del entrenamiento: 64
Preparación de la red neuronal 65
Primeras pruebas 65
Experimentación con conjuntos de datos 66
Capítulo 5: Pruebas y resultados 68
Capítulo 6: Discusión y conclusiones 80
Capítulo 7: Trabajo a futuro 85
Anexo 86
Bibliografía 89

Índice de figuras

Figura 1 Robot Toyota HSR..... 2
Figura 2 Jerarquía entre datos, conjunto de datos y una base de datos. 10
Figura 3 Distintos tipos de aprendizaje de máquina. 13
Figura 4 (Izq.) Neurona natural, (Der)Neurona artificial. 15
Figura 5 Ejemplo de una red neuronal multicapa. 18
Figura 6 Le-Net 5, una arquitectura de red neuronal convolucional 20
Figura 7 Arquitectura LeNet5. 21
Figura 8 Arquitectura de AlexNet..... 21
Figura 9 Módulo interno de la red Inception 22
Figura 10 Arquitectura YOLO 23
Figura 11 Ejemplo del proceso de regresión probabilística de YOLO 23
Figura 12 Estructura de YOLOV3 25
Figura 13 Ejemplo de aumento de datos. (Arr) fragmento de la base de datos Fashion MNIST. (Abj) El mismo fragmento con transformaciones aplicadas para aumentar datos 26
Figura 14 Ejemplo del trabajo de Alhaija, et al. En donde se puede observar la unión de datos sintéticos con una fotografía verdadera. 28
Figura 15 Ejemplo de modificación de imágenes del trabajo de Talukdar, et al. Se puede observar las imágenes originales en la fila superior, con modificaciones de color y emborronado en filas inferiores 29
Figura 16 Imagen ejemplo de la base de datos "Falling things" (FAT), la cual consiste en colocar objetos tridimensionales en un ambiente virtual para obtener imágenes e información sobre ubicación y profundidad de los objetos 30
Figura 17 Vista del entorno de trabajo de Unity..... 36
Figura 18 Vista del entorno de trabajo de blender 37
Figura 19 Diagrama de módulos de la aplicación..... 38
Figura 20 Diagrama del funcionamiento del módulo de repetición. 40
Figura 21 Ejemplo de listas de objetos en la interfaz gráfica de Unity. 41
Figura 22 Tres tipos de colisionadores primitivos para un entorno 3D 44
Figura 23 Visualización de la operación de4 Raycasting, imagen de: https://learn.unity.com/tutorial/let-s-try-shooting-with-raycasts#5c7f8528edbc2a002053b469.. 45
Figura 24 Vista superior de la mesa virtual con las 4 paredes invisibles rodeándola. 47
Figura 25 Ejemplo del resultado del algoritmo de generación de objetos. 48
Figura 26 Organización de la sección de captura de datos. 50
Figura 27 Boceto conceptual de la proyección de las coordenadas del objeto al plano de la cámara. 51
Figura 28 Visualización del tronco de visión de cámara y comparación con posición real de los objetos..... 52
Figura 29 Proyecciones de un punto en la base del tronco de visión de una cámara 53
Figura 30 Visualización del espacio de pantalla en Unity 54
Figura 31 Ejemplo de raycasting para detectar oclusión. 56
Figura 32 Visión inicial de la interfaz de usuario..... 59
Figura 33 Menú de selección de objetos. 59

Figura 34 Menú con variables para el proceso automático. 60
Figura 35 Opciones para cambiar la dirección de guardado. 61
Figura 36 Opciones de control de entorno. 62
Figura 37 Ejemplo de una variación de la iluminación y el ángulo de visión. 63
Figura 38 Resultado del entrenamiento de prueba. 66
Figura 39 Gráfica de pérdida del primer entrenamiento 69
Figura 40 Gráfica de pérdida del segundo entrenamiento 70
Figura 41 Gráfica de pérdida del tercer entrenamiento 70
Figura 42 Gráfica de map de los tres entrenamientos realizados 72
Figura 43 Comparación entre los tres entrenamientos sobre una misma imagen. De arriba hacia abajo, primer entrenamiento, segundo entrenamiento, tercer entrenamiento. 74
Figura 44 Ejemplo de reconocimiento fallido de un adorno como manzana 75
Figura 45 Ejemplo de reconocimiento fallido de una taza en imagen. 76
Figura 46 Ejemplo de reconocimiento fallido de una taza. 76
Figura 47 Imagen con texturas simplificadas y mala clasificación. Indicando una necesidad de mejorar la detección en caso de contar con imágenes dañadas o con ruido. 77
Figura 48 Ejemplo de detección en ambiente de cocina. 78
Figura 49 Reconocimiento fallido de una taza a corta distancia 79
Figura 50 Reconocimiento exitoso de dos tazas a distancia alejada. 79

Índice de tablas

Tabla 1 Aplicaciones de cámaras RGB-D en robótica. 6
Tabla 2 Clasificaciones de métodos de aprendizaje de máquina 13
Tabla 3 Equivalencias entre una neurona natural y una artificial. 15
Tabla 4 Estructura de Darknet-53 25
Tabla 5 Tabla de características de motores de desarrollo. 36
Tabla 6 Especificaciones del equipo de cómputo utilizado 64
Tabla 7 Paquetes adicionales instalados. 65
Tabla 8 Objetos por categoría, primer experimento. 67
Tabla 9 Tiempos de entrenamiento. 68
Tabla 10 Imágenes generadas para los experimentos 68
Tabla 11 Resultados del conteo manual del primer entrenamiento. 73
Tabla 12 Resultados del conteo manual del segundo entrenamiento. 73
Tabla 13 Resultados del conteo manual del tercer entrenamiento. 74
Tabla 14 Grupos modificados de fotografías. 77
Tabla 15 Resultado de conteo manual de detección en conjunto pequeño de imágenes reales. 78

Índice de Pseudocódigos

Pseudocódigo 1 Algoritmo Knught Shuffle 42
Pseudocódigo 2 Algoritmo de creación de listas. 43
Pseudocódigo 3 Algoritmo de generación de objetos. 47
Pseudocódigo 4 Algoritmo de destrucción de objetos 48
Pseudocódigo 5 Algoritmo de obtención de datos de objeto en la imagen. 55

Índice de ecuaciones

Ecuación 1 Representación matemática de una neurona artificial. 15
Ecuación 2 Función logística..... 16
Ecuación 3 Función softmax. Una generalización de la función logística que también es usada en redes neuronales..... 16
Ecuación 4 Función ReLU 16
Ecuación 5 Función de entropía cruzada 17
Ecuación 6 Función de entropía cruzada binaria para funciones discretas 17
Ecuación 7 Pérdida cuadrática 17
Ecuación 8 Función de convolución 19
Ecuación 9 Función de pérdida de YOLOV3, ... 26
Ecuación 10 Expresión vectorial de la operación de Raycasting..... 45
Ecuación 11 Características de los cuaterniones 46
Ecuación 12 Transformación a sistema de coordenadas de la cámara. 52
Ecuación 13 Proyección al plano de visión de la cámara para un tronco general (no simétrico)..... 53
Ecuación 14 Homogeneización de las coordenadas. 53
Ecuación 15 Conversión a coordenadas de visión de pantalla. 54
Ecuación 16 Cálculo de la media de precisión promedio..... 71

Capítulo 1: Introducción

La robótica como herramienta para la automatización de procesos industriales ha estado en uso durante varias décadas. El principal propósito de esto ha sido automatizar aquellas tareas en una línea de fabricación. Con el paso del tiempo, el avance de la tecnología ha permitido que los robots realicen tareas más complicadas. Pasando de un ambiente exclusivamente industrial a entornos más variados como el servicio o la medicina.

Un robot de servicio es aquel que funciona de forma parcial o totalmente autónoma con el propósito de asistir a usuarios humanos o reemplazar a estos en la realización de tareas no industriales. Entre estas tareas se puede incluir ayudar a pacientes con tareas de rutina, mantener vigilancia en personas de la tercera edad o facilitar las tareas de personas con cierto tipo de discapacidad.

Para el buen funcionamiento de un robot de servicio, este debe contar con sistemas para reconocer su entorno. Entre estos se incluyen sensores para detectar obstáculos y cámaras para poder obtener información visual del ambiente. Este último es de gran importancia para poder reconocer no sólo a los usuarios humanos, sino también diversos objetos con los cuales es posible interactuar.

El reconocimiento se lleva a cabo mediante la utilización de algoritmos de visión computacional, los cuales utilizan conceptos de inteligencia artificial y procesamiento de imágenes para poder encontrar características específicas de los objetos a reconocer. Unos de los métodos ampliamente utilizados para esto son las redes neuronales convolucionales.

En el laboratorio de robótica del posgrado de la facultad de ingeniería de la UNAM, se trabaja en la continua programación y entrenamiento del robot Toyota HSR, denominado como Takeshi. El robot es capaz de reconocer con su entorno gracias a cámaras y sensores infrarrojos. Un brazo con 6 grados de libertad y un módulo de voz permiten que Takeshi interactúe con su entorno.

Figura 1 Robot Toyota HSR

El robot HSR Takeshi utiliza la red neuronal YOLO (You Only Look Once) para poder reconocer objetos y agentes humanos en su entorno. En este proyecto se desarrolló una herramienta computacional para generar imágenes e información de forma virtual con el propósito de reducir el tiempo de creación de imágenes para el entrenamiento de esta red neuronal.

Definición del problema

Para poder interactuar con su entorno, es necesario que el robot utilice algoritmos de reconocimiento de objetos mediante imágenes.

El algoritmo de reconocimiento utiliza la red neuronal YOLO para poder discernir objetos en una imagen. El entrenamiento de esta red se realiza mediante la creación de imágenes de un entorno con objetos etiquetados superpuestos en la imagen.

Si bien, se puede lograr un entrenamiento exitoso con estos datos, este método de creación de imágenes trae como consecuencia una mala identificación de objetos en posiciones ilógicas como paredes o techos. De esto surge la

necesidad de crear una herramienta para la creación de datos de entrenamiento de forma artificial.

Objetivos

Objetivo general:

El objetivo principal del trabajo es la creación de una herramienta computacional que pueda generar la información requerida para entrenar una red neuronal con el propósito de reconocer objetos en imágenes. Esto se logrará mediante la utilización de un entorno de desarrollo de aplicaciones virtuales que permita la programación de las funciones necesarias para la obtención de la información requerida.

Objetivos específicos:

- Seleccionar el entorno de desarrollo que permita la creación flexible del sistema.
- Programar el sistema para poder generar objetos al azar para la extracción de datos.
- Programar métodos de captura de datos para la obtención automática de los mismos.
- Probar dichos datos generados para el entrenamiento de una red neuronal con el propósito de corroborar la calidad de los mismos.

Contribución y Relevancia

El desarrollo de la herramienta computacional para la creación de datos con el propósito de entrenar una red neuronal permite cambiar la distribución de datos y el tipo de información que se recopila. Esto permite utilizar esta herramienta con toda una gama de arquitecturas diseñadas para el reconocimiento de objetos en imágenes. En resumen, esta aplicación se crea con el propósito de ser usada a futuro y con la posibilidad de adecuarse a otras aplicaciones.

Esta herramienta también se crea con el propósito de reducir el tiempo de preparación de imágenes, el cual puede ser elevado si se realiza de forma manual.

Organización del trabajo

El presente trabajo está organizado de la siguiente manera:

- En el capítulo 2, se explica el concepto de redes neuronales convolucionales, así como el concepto de ambientes virtuales (incluyendo definiciones de realidad mixta, virtual y aumentada), y ejemplos de trabajos previos que intentan resolver este problema.
- En el capítulo 3 se explica a fondo la creación del ambiente virtual, incluyendo la obtención de modelos, la programación y la estructura de la aplicación. Se hace énfasis en los distintos módulos de esta y la obtención de datos.
- En el capítulo 4 se explica la metodología de experimentación, incluyendo la generación de datos, las condiciones de entrenamiento de la red neuronal y los entrenamientos que se realizaron.
- El capítulo 5 contiene los resultados de los entrenamientos realizados en el capítulo 4.
- Finalmente, en los capítulos 6 y 7 se analizan los resultados para llegar a conclusiones y se explica los cambios a futuro que es necesario realizar para mejorar los resultados obtenidos.

Capítulo 2: Antecedentes

Robots de servicio

Los robots han tenido mucho éxito en el ámbito industrial, en donde generalmente forman parte de cadenas de producción automatizadas, con el objetivo de suplir tareas muy monótonas, repetitivas o peligrosas para operativos humanos.

En estos ambientes, las condiciones a las que los robots se enfrentan son limitadas. Por lo general un robot se encarga de una sola parte del proceso y se enfrenta a las mismas condiciones de fabricación durante toda su vida útil. Estos robots no son autónomos ya que están limitados por una serie de instrucciones precisas y no son capaces de interactuar con su entorno más allá de su función principal.

El número de robots autónomos y semi autónomos en la industria ha aumentado en los últimos años. Investigaciones recientes han resultado en la introducción de robots que pueden ser entrenados por agentes humanos [1], [2] y robots que cumplen tareas de forma paralela con ellos [3]. Compañías como Amazon también trabajan en el desarrollo de robots que puedan comprender e interactuar con su entorno de forma dinámica para el manejo de mercancía [4].

En contraste con los robots industriales, los robots denominados “De servicio” requieren interactuar en tiempo real con su entorno y con los agentes que se desenvuelven en este. Los robots de servicio pueden operar en todo tipo de ambientes, desde educación, vigilancia, turismo, apoyo comunitario, recolección de basura o incluso el cuidado de personas con capacidades limitadas o de la tercera edad. Tanto la creciente automatización de la vida diaria como eventos recientes han puesto en evidencia la utilidad de los robots de servicio para limitar el contacto humano. [5], [6], [7] Lo cual hace el desarrollo de robots más avanzados una necesidad.

Además de las capacidades mecánicas del robot, el poder percibir el entorno es muy importante, no sólo para poder moverse en él, sino también para poder interactuar con el entorno. Muchos tipos de sensores son usados, con las cámaras de video (RGB, RGB-D) siendo uno de los más usados por la cantidad de información que proporciona. Algunas de las funciones más importantes en las que ayuda este tipo de sensor se explican en la siguiente tabla.

Aplicaciones de cámaras RGB-D en robótica	
Función	**Descripción**
Reconocimiento del entorno	El detectar el espacio en donde el robot trabaja, así como cualquier cambio que este pueda tener. Esto se usa en conjunto con otros sensores presentes en el robot para generar y actualizar mapas del ambiente.
Interacción con usuarios	El detectar personas a partir de sus características físicas, además de poder interpretar acciones de los usuarios, las cuales son interpretadas como comandos.
Reconocimiento de objetos	Reconocer objetos presentes en el ambiente a partir de una base de datos para futura interacción. Por lo general, el uso de cámaras RGB-D permite obtener datos de profundidad del ambiente, lo cual permite una mayor precisión al interactuar con dichos objetos de manera física.
Retroalimentación de información visual.	Mediante una conexión entre el robot y un usuario remoto, es posible enviar información visual al usuario. Esto puede ser útil en ambientes de alto riesgo o en caso de que se quiera evitar el contacto humano.

Tabla 1 Aplicaciones de cámaras RGB-D en robótica.

Siendo el reconocimiento una tecnología en desarrollo, existen muchas dificultades en la interacción de robots autónomos con su entorno, lo cual se debe principalmente a la infinidad de variables que puede tener una sola interacción en un ambiente no controlado. Esta complejidad se multiplica si consideramos que los robots de servicio avanzados deben de enfrentarse a una gran cantidad de situaciones distintas.

Problemas visuales.

Como ya se mencionó, las cámaras en un robot proporcionan información visual sobre los objetos y personas que se encuentran en los alrededores del robot e incluso pueden proporcionar información de profundidad. Sin embargo, dependiendo de la calidad de la cámara y las condiciones del ambiente, la imagen resultante puede variar mucho. Esto puede llevar a problemas de identificación por parte del robot, lo cual resulta en dificultades para el correcto funcionamiento del mismo.

Algunos de los problemas visuales más comunes son:

- *Cambios de iluminación en el entorno:*
  Los cambios de iluminación son uno de los mayores problemas visuales, no sólo para la robótica, sino para muchos de los campos de reconstrucción y reconocimiento visual. El cambio de iluminación provoca cambios en la intensidad de colores de los objetos observados e incluso puede llevar a ocultar completamente ciertos objetos, lo cual puede llevar a fallas de reconocimiento y fallas de navegación si no se utilizan sensores adicionales.

- *Presencia de ruido visual:*
  Todo tipo de sensores tienen la posibilidad de generar errores de medición. En las cámaras esto puede ser en forma de ruido en la imagen como pérdida de información, efectos de pixelado, o imágenes emborronadas. Esto a su vez puede llevar a fallas de reconocimiento de objetos o agentes.
- *Presencia de materiales transparentes:*

Tal es el caso de materiales como vidrio o plástico transparente. Debido a su falta de coloración y superficies reflejantes, estos objetos son mucho más difíciles de localizar para sensores visuales e infrarrojos. Esto genera dificultades para los sistemas de navegación de los robots, ya que es difícil detectar objetos como puertas de vidrio.

- *Ambigüedad en reconocimiento de características:*
  Dos objetos distintos pueden tener características similares como forma, detalles o color, lo cual hace difícil el poder distinguir entre estos sin que se tenga una base de datos extensa que incluya estos ejemplos específicos. Esto no está limitado a objetos, sino también a agentes humanos. Para superar esta dificultad se requieren de grandes bases de datos cubriendo una gran cantidad de objetos distintos.
- *Objetos camuflados:*
  Puede llegar a presentarse la situación de que uno o más objetos en el campo visual del robot sean muy similares en color y en textura. De ser el caso es posible que no sea posible discernir entre los objetos. Esto también puede presentar un problema de navegación en el caso en que objetos, que pueden llegar a ser obstáculos para el robot, no son discernibles de elementos circundantes a este. Este problema, sin embargo, es menor ya que la detección de obstáculos y planeación de rutas se puede llevar a cabo con otros sensores que no son propensos a este tipo de errores de ambigüedad.

[8], [9]

Complejidad del entorno

Para que un robot de servicio pueda funcionar de manera eficiente, debe de contar con un mapa de su entorno. Dicho mapa puede ser generado de forma manual, sin embargo, esto sólo es útil en condiciones de laboratorio, en donde el entorno es preparado para la operación de un robot, considerando sus capacidades. Por lo general, los robots de servicio se enfrentan a ambientes dinámicos, los cuales

contienen no sólo obstáculos sino también personas y en ocasiones otros agentes (animales, otros robots). Los robots de servicio deben de ser capaces de trabajar en condiciones dinámicas sin llegar a obstruir a otros agentes. Otros objetos no dinámicos como puertas, muebles y escaleras requieren ser reconocidos y diferenciados de forma precisa para una navegación correcta del entorno (un ejemplo de esto es la cantidad de puertas distintas que existen, el reconocimiento del tipo de puerta lleva a la secuencia de movimientos correcta para abrir dicha puerta). [10]

Métodos de enseñanza para los robots

Todos los robots requieren seguir una serie de algoritmos para cumplir con sus funciones. La mayor parte de estos algoritmos son fijos ya que tratan con las mismas instrucciones en condiciones inmutables. Tal es el caso de los robots industriales que tratan con sólo una tarea. Sin embargo, para ambientes dinámicos es necesario contar con algoritmos más complejos que permitan que el robot se pueda adaptar a las circunstancias que lo rodeen.

Al interactuar con su entorno, el robot recaba información mediante los diversos sensores que posee. Esta información es posteriormente presentada en los diversos algoritmos para ser utilizada o guardada para futura referencia. Dependiendo del uso que se le dé, será la representación de esta información.

Dato:

Un dato en el entorno de la informática es definido como *"Las cantidades, caracteres o símbolos los cuales son utilizados por operaciones en una computadora. Estos son transmitidos en forma de señales eléctricas y guardado en medios magnéticos, ópticos o mecánicos."*[11] En el entorno de la estadística, un dato es descrito como *"Una pieza individual de información, la cual se utiliza posteriormente para el análisis e interpretación de la misma"*[12]

Es decir, todo objeto o fenómeno que sea captado por un sensor es convertido en información en el formato correspondiente al tipo sensor que fue utilizado. Por ejemplo, en el caso de una cámara, la información guardada va a ser presentada en forma de una imagen, mientras que un sensor infrarrojo envía información en forma de pulsos eléctricos que son traducidos a datos numéricos.

Estos datos, pueden ser organizados con el propósito de analizar un comportamiento o característica en común. Por ejemplo, datos en un sensor de temperatura pueden ser organizados para poder entender el cambio de la misma en una substancia o un espacio.

Conjunto de datos:

Se denomina conjunto de datos (o dataset) a un conjunto organizado de uno o más tipos de datos que describen un fenómeno. Estos están organizados en distintos formatos como tablas y dependiendo del tipo de datos y de cómo estén relacionados va a ser la forma que tenga esta organización. Ejemplos de esto son una tabla, en donde cada fila describe a un individuo y las columnas describen distintas características que este puede tener, o un dataset de imágenes el cual consiste en un conjunto de imágenes, generalmente del mismo tamaño y formato.

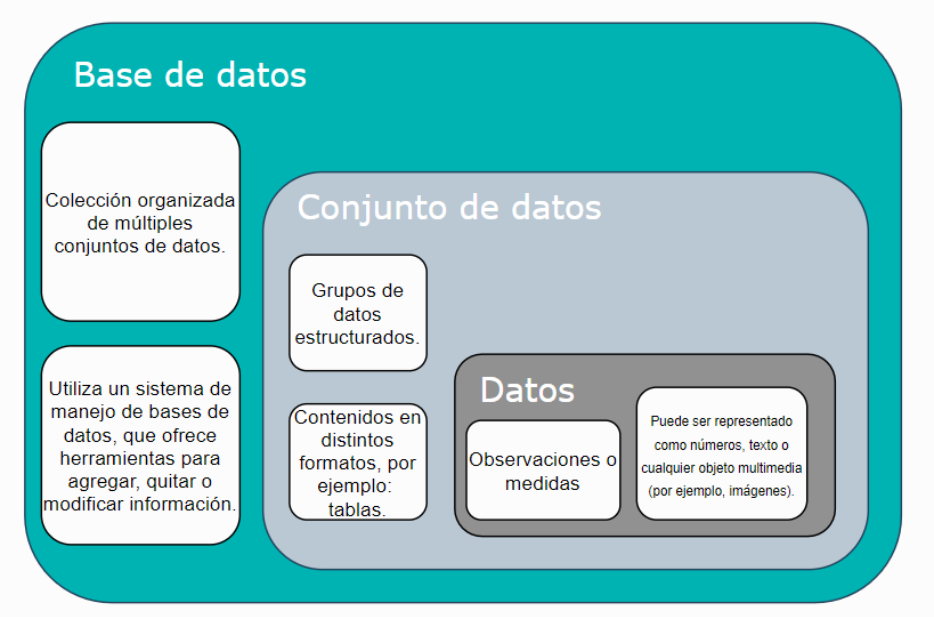


Figura 2 Jerarquía entre datos, conjunto de datos y una base de datos. Imagen original de:[13]

Base de datos:

Una base de datos es una colección de conjuntos de datos, la cual puede ser compuesta por el mismo tipo de conjuntos o por distintos tipos. Estas estructuras de datos cuentan con sistema de manejo de base de datos, el cual permita a un usuario acceder a esta información y manipularla. Las bases de datos han aumentado en número debido a los avances tecnológicos que permiten manejar volúmenes cada vez más grandes de información, como el concepto de Big Data y el cómputo en la nube.

Preparación de bases de datos

Para que los datos se puedan utilizar, es necesario procesarlos posterior a su adquisición. Muchas veces, los datos carecen de organización o cuentan con variables ajenas al fenómeno que se quiere estudiar o a la función para la cual se quiere entrenar. También es importante eliminar información que se encuentra dañada o incompleta. Adicionalmente, la información puede requerir ser normalizada para que las variables que se tengan que estudiar se encuentren en un rango similar o para juntar tipos de información distintos. Una vez tratada, la información puede ser catalogada en bases de datos.

Esta información, es utilizada como base para generar algoritmos de predicción o de toma de decisión. Dependiendo del tipo de dato y de la tarea que se requiera, va a ser necesario utilizar distintas cantidades de datos. Algunas tareas sencillas como el reconocimiento de números sólo requieren de una base de datos como, por ejemplo, la base de datos MNIST[14], mientras que tareas más complicadas como el reconocimiento de usuarios u objetos requieren de bases de datos extensas como PointNet, ShapeNet o ImageNet[15][16], [17].

Aprendizaje de máquina (Machine learning).

El aprendizaje de máquina es una rama de la disciplina de inteligencia artificial (IA) que busca generar algoritmos de toma de decisión o algoritmos discriminatorios, con el objetivo de que las máquinas puedan reaccionar a eventos o cumplir funciones para los cuales no fueron originalmente programadas. Los métodos de aprendizaje de máquina utilizan conjuntos de datos para generar funciones de clasificación o regresión que agrupan los puntos en categorías. Dependiendo de la complejidad de los datos, la función utilizada va a requerir métodos de clasificación más sofisticados.

Los métodos de aprendizaje de máquina se pueden dividir en tres categorías, dependiendo de la naturaleza de la información y la participación de un agente externo al método[13], [18].

Método	Descripción	Ejemplos de métodos
Aprendizaje supervisado	El aprendizaje supervisado cuenta desde un inicio con una serie de datos ya etiquetados por un agente humano. Estos datos se usan para entrenar un algoritmo con el propósito de que este cumpla una función como, por ejemplo, clasificar datos que no hayan sido previamente etiquetados o encontrar relaciones entre conjuntos de variables.	• Máquinas de soporte vectorial. • K vecinos cercanos • Regresión lineal • Regresión logística
Aprendizaje no supervisado	Como su nombre lo indica, el aprendizaje no supervisado utiliza datos no etiquetados y no ocupa intervención alguna de un usuario externo. Estos métodos son ideales cuando la cantidad de información que se utiliza es muy grande para ser etiquetada por agentes humanos.	• Clasificación k-means • Clasificación por clustering • Análisis de componentes principales • Deep learning
Aprendizaje reforzado	El aprendizaje reforzado difiere de los dos anteriores en que en este consiste en dos	• Agentes virtuales • Q-learning

	partes: el algoritmo de aprendizaje y un agente virtual que simula las acciones de una agente real para retroalimentar al algoritmo. Este tipo de aprendizaje aprende en un proceso de prueba y error, en donde se busca llegar a un estado ideal en la clasificación.	• SARSA

Tabla 2 Clasificaciones de métodos de aprendizaje de máquina.

Figura 3 Distintos tipos de aprendizaje de máquina. Imagen original de: [13]

Redes neuronales

Las redes neuronales fueron creadas con el objetivo de emular los procesos neuronales humanos. Esto con el propósito de lograr que las computadoras puedan igualar o incluso superar a la mente humana en ciertas tareas. Si bien, esto se ha logrado en algunos campos (Ciertos juegos de mesa, cálculos complejos, reconocimiento de voz) en otros el desarrollo de las técnicas de inteligencia artificial no es suficiente (reconocimiento de imágenes, reconocimiento de objetos o agentes basados en contexto). Avances recientes en el campo de la inteligencia artificial han permitido que ciertas tareas de reconocimiento de imágenes se puedan resolver

mediante el uso de redes neuronales profundas, las cuales requieren de grandes bases de datos y equipo de cómputo avanzado.

La historia de las redes neuronales se remonta a la época de los años 40 en donde McCulloch y Pitts introdujeron por primera vez el concepto de una red neuronal. Más adelante, a finales de la década de los 50, Rossenblatt introdujo al perceptrón, el cual es el prototipo de las redes neuronales.

Este avance llevó a un auge en la investigación de redes neuronales. Sin embargo, los esfuerzos de investigación amainaron en las décadas siguientes por una falta de resultados y altos requerimientos computacionales para la investigación en redes neuronales. Sin embargo, el desarrollo de estas ha resurgido en las últimas décadas debido al avance acelerado de las capacidades de cómputo y las capacidades de memoria de las computadoras modernas.[19]

Neurona artificial

Se conoce como neurona a la célula básica del tejido nervioso de los seres vivos que poseen un sistema nervioso. La principal función de estas es enviar impulsos eléctricos, los cuales se interpretan como señales que generan una respuesta en el organismo. Desde movimientos involuntarios, sensaciones e incluso pensamientos. Las neuronas cuentan con tres partes principales en su morfología: El Axón, el núcleo y las dendritas. Las señales eléctricas son enviadas por el axón de la neurona y son recibidas por las dendritas de las neuronas que se encuentren conectadas. Estas señales pueden ser inhibidas o aumentadas mediante procesos electroquímicos del sistema nervioso.

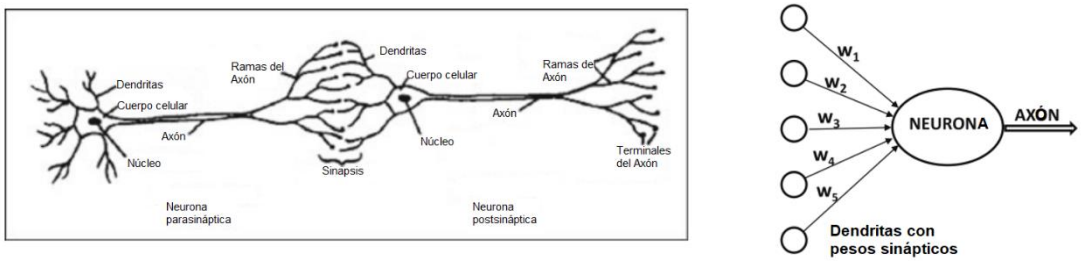


Figura 4 (Izq.) Neurona natural, (Der)Neurona artificial. Imagen de:[19]

Una neurona artificial es una representación aproximada de las neuronas naturales. Cada uno de los componentes de una neurona emula los componentes de su contraparte natural. Los componentes de una neurona artificial están listados en la tabla 3 junto con sus análogos.

Equivalencias entre neurona natural y neurona artificial	
Natural	**Artificial**
Dendritas	Vector de entrada (X).
Axón	Función de activación.
Núcleo	Función de sumatoria (∑).
Inhibición, aumento de señal	Pesos (W).

Tabla 3 Equivalencias entre una neurona natural y una artificial.

$$y = f\left(b + \sum_{i=1}^{n} x_i w_i\right)$$

Ecuación 1 Representación matemática de una neurona artificial.

Existen dos funciones importantes para el funcionamiento de una red neuronal. La función de activación y la función de pérdida. La primera es la función que toma el resultado de la sumatoria de una neurona y dan como resultado una salida dependiente de la función. Usualmente entre 0 y 1. Algunas de las funciones de activación más comunes en redes neuronales incluyen:

Función sigmoide

Una función sigmoide es aquella función real, diferenciable y limitada que cuenta con un solo punto de inflexión, gráficamente se presentan en la forma de una letra s con el punto de inflexión en x=0. Un ejemplo de este tipo de funciones es la función logística.

$$\sigma(x) = \frac{1}{1 + e^{-x}}$$

Ecuación 2 Función logística

Muchas funciones sigmoides presentan valores en el eje y en el rango de 0 a 1 o de -1 a 1, lo cual las hace ideales para respuestas de redes neuronales.

$$S(y)_i = \frac{\exp(y_i)}{\sum_{j=1}^{n} \exp(y_j)}$$

Ecuación 3 Función softmax. Una generalización de la función logística que también es usada en redes neuronales.

ReLU (Rectifier Linear Unit)

La función rectificadora, también conocida como rampa o ReLU, es una de las funciones de activación más usadas en redes neuronales. Se define como la parte positiva de su variable de entrada. Entre las ventajas que presenta, destaca el que es invariante a la escala y es más eficiente en cuestión de operaciones computacionales. Sin embargo, tiene problemas como no estar centrada en cero y no ser diferenciable en cero, lo cual no la hace apta para algunas funciones. Algunas variaciones como Leaky ReLu han sido desarrolladas para combatir estos problemas.

$$f(x) = \max(0, x)$$

Ecuación 4 Función ReLU

La segunda función importante es la función de pérdida, la cual es utilizada para ajustar los pesos *W* y sesgos *b* de cada neurona con el objetivo de optimizar la respuesta de la arquitectura de acuerdo con el modelo que se desee entrenar. La función usada depende de la función de activación utilizada ya que el tipo de datos debe concordar. Algunas de las funciones de pérdida utilizadas son:

Función de entropía cruzada:

Usada comúnmente junto con una función de activación Softmax, la función de entropía cruzada, también conocida como pérdida logarítmica, pérdida logística o log loss. Se usa para comparar las probabilidades de clase de un conjunto de estas con el valor real (ground truth) de los datos de entrenamiento. Penalizando los valores que se alejen del valor esperado.

$$L_{CE} = -\sum_{i=1}^{n} t_i \log(p_i)$$

Ecuación 5 Función de entropía cruzada

En donde ti es el valor verdadero y p es la probabilidad softmax para la clase i.

Una variante de esta ecuación es la entropía binaria cruzada, utilizada para clasificación binaria. Se define como:

$$L = -\frac{1}{N}\left[\sum_{j=1}^{N}[t_j \log(p_j) + (1 - t_j) * \log(1 - p_j)]\right]$$

Ecuación 6 Función de entropía cruzada binaria para funciones discretas

Pérdida cuadrática

También conocida como pérdida L2, esta se define como:

$$L = (y - f(x))^2$$

Ecuación 7 Pérdida cuadrática

Es útil ya que, al tener sólo un mínimo, la función no se atorará en un mínimo local y por lo tanto, el gradiente eventualmente convergirá. Sin embargo, este modelo no es muy efectivo si hay muchos datos dispersos.

Redes multicapa.

Una neurona es útil si la aplicación para la cual se requiere es sencilla. La neurona más sencilla, el perceptrón, es útil si se requiere de un discriminador lineal entre dos clases. Sin embargo, aplicaciones más complejas requieren de la utilización de más neuronas conectadas.

Una red neuronal multicapa es la unión de varias capas compuestas por un conjunto de neuronas. Cada capa realiza los cálculos y pasa los resultados a la capa siguiente (en lo que se conoce como una arquitectura Feed Forward). Las capas entre la entrada y la salida se conocen como capas ocultas porque los cálculos que ocurren en su interior no son visibles a sistemas externos a la red neuronal. El uso de estas capas ha permitido el desarrollo de redes neuronales cada vez más complejas.

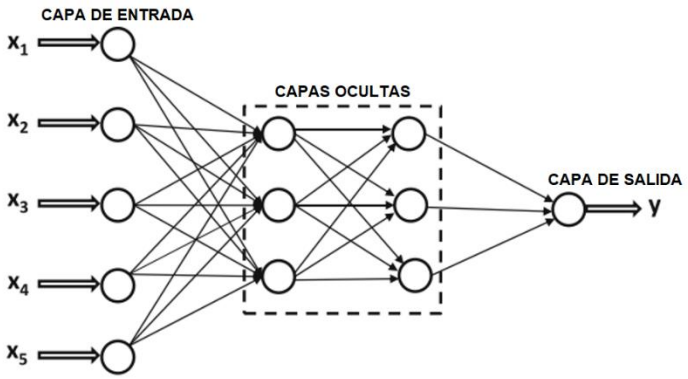


Figura 5 Ejemplo de una red neuronal multicapa. Imagen original de: [19]

Esta organización en capas y bloques con ciertas funciones se volvió la base para el desarrollo de muchas otras arquitecturas más complejas.

Redes neuronales convolucionales

Las redes neuronales convolucionales se han popularizado en las últimas dos décadas debido al avance en poder computacional y a la gran cantidad de datos que se pueden manejar hoy en día. Estas redes son ideales para tratar problemas de procesamiento reconocimiento de objetos en imágenes e incluso son usadas en aplicaciones de procesamiento de texto.

La operación de convolución consta de la integral del producto de dos funciones en donde una de estas se desliza una distancia *t*. La convolución indica la magnitud de la superposición de las funciones. Para funciones no continuas (como en el caso de la multiplicación de arreglos) se utiliza la convolución discreta, la cual se define como:

$$f[m] * g[m] = \sum_{n} f[n]g[m-n]$$

Ecuación 8 Función de convolución

Este tipo de convolución también es posible realizarse con matrices. El proceso se da mediante la multiplicación de una matriz por la matriz objetivo, repitiendo esto y moviendo la primera matriz en cada iteración. Esto es la base de la operación realizada en las capas convolucionales de una RNC. La matriz que se desliza en la capa se conoce como filtro y está compuesto por los pesos W. El resultado de la convolución del filtro y una región específica de la capa se pasa por una función ReLU de activación para definir un espacio en la siguiente capa.

Las capas en este tipo de arquitecturas son tridimensionales. En el caso de la capa de entrada, las dimensiones representan las dimensiones de la imagen de entrada y el número de canales de esta imagen. En las capas ocultas, la profundidad de cada capa representa los mapas de características, los cuales codifican ciertas formas de las imágenes para su futuro aprendizaje.

El otro tipo de capa importante en esta arquitectura son las capas de Max Pooling. Estas toman una sección de una capa y regresan un elemento de esta, ya sea un

máximo o promedio para la siguiente capa. De esta forma, se reduce la dimensión de las capas manteniendo las características más importantes de la imagen.

El resultado de estas capas convolucionales es un vector conocido como vector de características, el cual es introducido posteriormente a capas de clasificación (llamadas ”*Fully Connected*”).

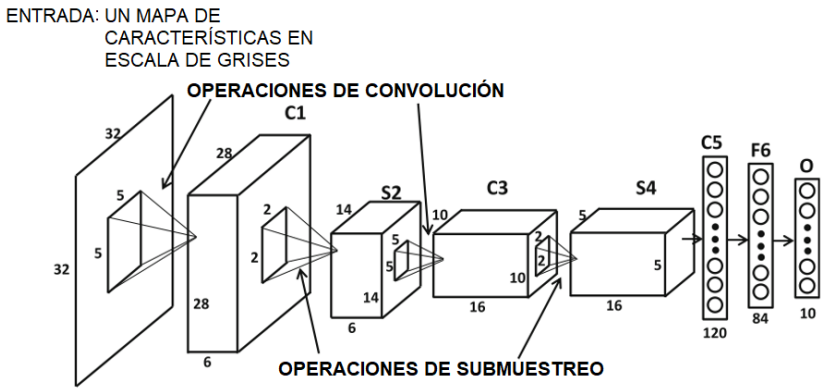


Figura 6 Le-Net 5, una arquitectura de red neuronal convolucional. Imagen original de: [19]

Otra característica importante de las RNC es la operación llamada transferencia de conocimiento, en la cual se entrena una red neuronal con pesos correspondientes a las primeras capas de una red similar (o incluso de una red correspondiente a una terea completamente ajena. Debido a que la gran mayoría de las capas codifican formas y características comunes entre objetos es posible aplicar estas características a múltiples bases de datos. Para muchas aplicaciones es más eficiente tomar estos pesos previamente entrenados y entrenar los pesos de las últimas capas para optimizarlos a la tarea deseada.

Arquitecturas importantes de RNC

Algunas de las arquitecturas más importantes de RNC son las siguientes:

Le-Net5

Le-Net 5 es una de las primeras arquitecturas de RNC desarrolladas. Desarrollada en el año de 1998, esta arquitectura con 5 capas, dos de ellas convolucionales y tres de ellas *fully connected*. Esta red se utilizó para procesar imágenes de 32x32 pixeles. Una aplicación de esto es el reconocimiento de letras y dígitos escritos a mano.

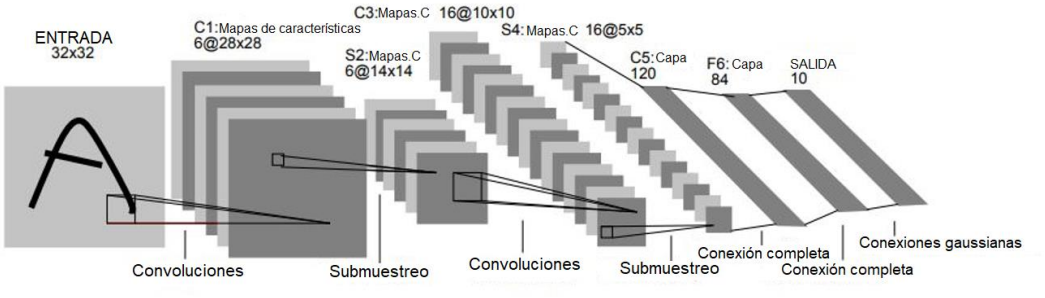


Figura 7 Arquitectura LeNet5. Imagen originalmente publicada en[20]

AlexNet

Ganadora de la competencia de ImageNet en el año 2012, AlexNet cuenta con 8 capas computacionales (5 de ellas convolucionales y 3 de ellas *fully connected*). Parte de lo que hizo exitosa a esta arquitectura fue la implementación de funciones ReLU.

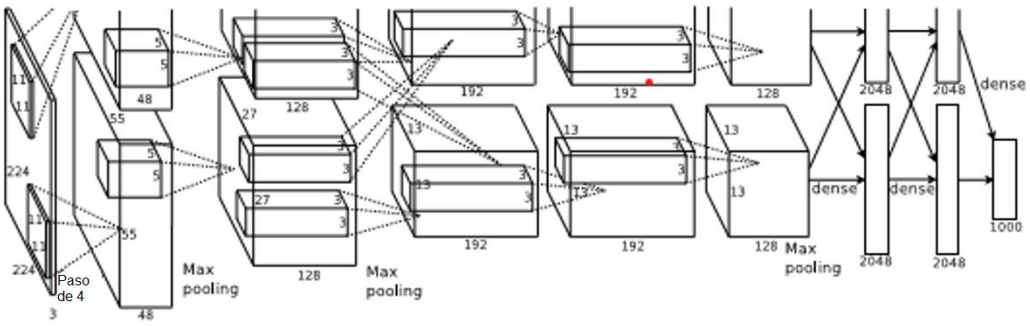


Figura 8 Arquitectura de AlexNet, imagen originalmente publicada en:[21]

GoogLeNet (Inception)

Ganadora de la competencia de ImageNet 2014, esta arquitectura cuenta con 22 capas. Uno de los principales avances que esta red tuvo fue la implementación de los módulos *Inception*, los cuales son una red neuronal en miniatura con convoluciones en paralelo, cuyos resultados son posteriormente concatenados.

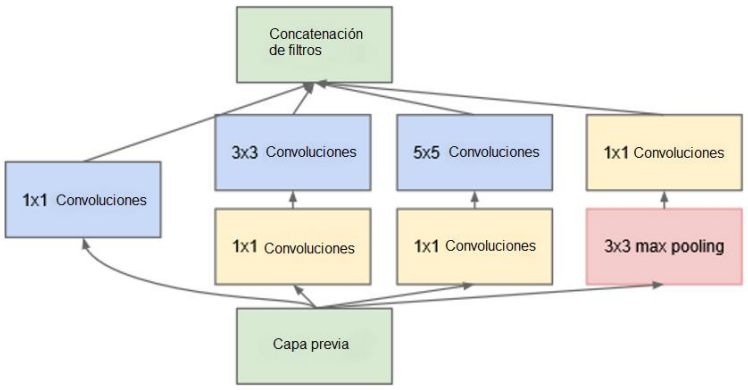


Figura 9 Módulo interno de la red Inception, imagen originalmente publicada en: [22]

La red también cuenta con dos clasificadores adicionales que toman datos de distintas partes de la arquitectura. Estos ayudan a mejorar la discriminación de características y ayudan a propagar el gradiente en la etapa de retro propagación.

ResNet

Esta arquitectura cuenta con 152 capas. Fue una de las primeras redes en tener más de 100 capas de profundidad sin caer en el problema de desvanecimiento del gradiente (En retro propagación, el gradiente para corregir los pesos va disminuyendo a medida que retrocede, en redes muy grandes, esto puede llevar a que los pesos no cambien). Esto se logró mediante el uso de conexiones entre capas no contiguas y el uso de normalización de bache para mantener sus capas de la misma dimensión en bloques de convolución, cambiando sólo el número de capas que cada bloque tendrá. [23]

Yolo

Desarrollada en 2016, YOLO (You Only Look Once) es una arquitectura diseñada para la clasificación y detección de objetos en una imagen de forma veloz. Esto lo hace con una sola red neuronal convolucional. La arquitectura recibe la imagen y esta es dividida en sectores, los cuales generan bounding boxes (cajas delimitantes) cada una con una etiqueta y un porcentaje de confianza (qué tanta confianza se tiene de que la caja contenga al objeto).[24], [25]

Estas probabilidades son posteriormente comparadas y la más probable es la elegida para clasificar al objeto en la imagen.

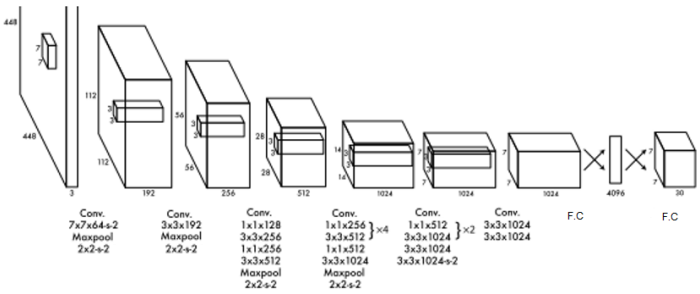


Figura 10 Arquitectura YOLO, imagen originalmente publicada en:[24]

El resultado de detección es extremadamente rápido, permitiendo detectar imágenes a un ritmo de 45 cuadros por segundo.

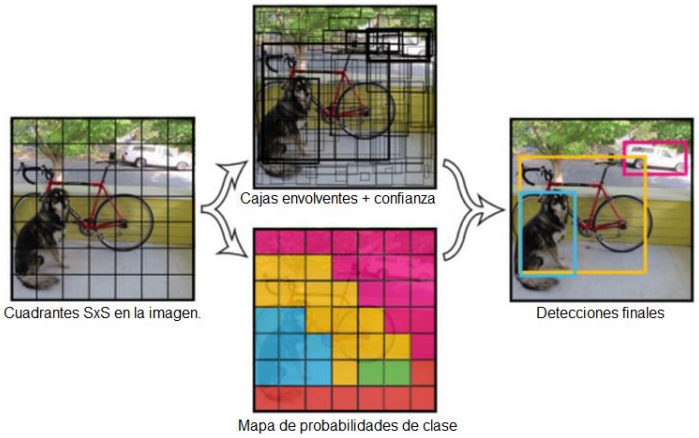


Figura 11 Ejemplo del proceso de regresión probabilística de YOLO, imagen original de: [24]

YOLOV3

Desarrollada en 2018, la tercera iteración de la red neuronal YOLO consiste en una mejora a varios problemas de detección que presentaba la primera versión. Siendo los principales problemas fallas de detección y un índice de exhaustividad bajo. Esto fue progresivamente solucionado en la siguiente iteración (YOLO9000) las cual agregó normalización de baches, aumentando la resolución durante el entrenamiento y removiendo las capas fully connected para cambiar la detección de bounding boxes por puntos de anclaje.[26]

Yolo V3 itera en la arquitectura de V2 agregando detección en múltiples escalas, las cuales son obtenidas durante distintos puntos del procesos y escaladas para coincidir. Esto permite obtener una mejor predicción de clases y cajas envolventes en las imágenes. [25]

Estructura de la red:

YOLOV3 está compuesta por varios bloques de operaciones. La entrada toma la imagen y esta entra en un bloque conocido como Darknet 53. La cual está compuesta de 53 capas convolucionales con filtros consecutivos de 3x3 y 1x1. Este bloque es el responsable de extraer las características de las imágenes.

	Tipo	Filtros	Tamaño	Salida
	Convolucional	32	3x3	256x256
	Convolucional	64	3x3/2	128x128
1x	**Convolucional**	32	1x1	
	Convolucional	64	3x3	
	Residual			128x128
	Convolucional	128	3x3/2	64x64
2x	**Convolucional**	64	1x1	
	Convolucional	128	3x3	
	Residual			64x64
	Convolucional	256	3x3/2	32x32
	Convolucional	128	1x1	

8x	**Convolucional**	256	3x3	
	Residual			32x32
	Convolucional	512	3x3/2	16x16
8x	**Convolucional**	256	1x1	
	Convolucional	512	3x3	
	Residual			16x16
	Convolucional	1024	3x3/2	8x8
4x	**Convolucional**	512	1x1	
	Convolucional	1024	3x3	
	Residual			8x8

Tabla 4 Estructura de Darknet-53

En dos puntos del bloque convolucional y al final de este, tres ramas procesan la imagen a distintas resoluciones, realizando operaciones de escalamiento para acomodar los resultados de cada bloque y poder concatenarlos con el bloque anterior. Esto permite que la arquitectura aprenda a reconocer objetos a diferentes tamaños, lo cual mejora la detección.

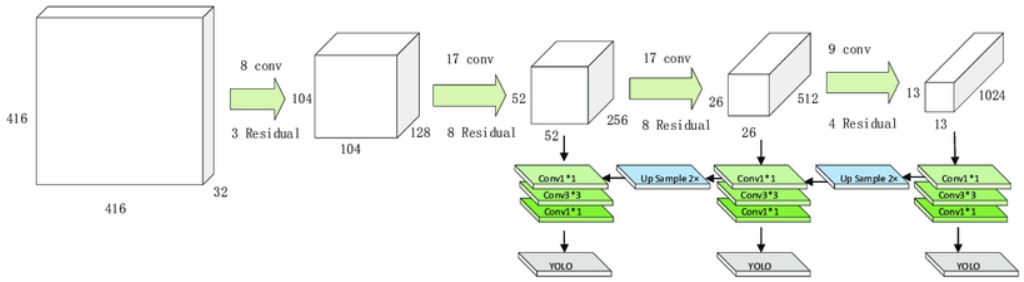


Figura 12 Estructura de YOLOV3. Fuente: Researchgate.

Al final del proceso, los tres mapas de características generan 3 cajas en cada celda de la imagen. Esto incluye información del centro y tamaño de estas, la confianza de que hay un objeto en esas cajas (denominado *"objectness"*) y las probabilidades de clase (de las clases entrenadas) en la caja [23, 24].

La función de pérdida de YOLOV3 toma estos 4 datos, y calcula la diferencia con los valores

$$\lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2]$$
$$+ \lambda_{coord} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} [(\sqrt{w_i} - \sqrt{\hat{w}_i})^2 + (\sqrt{h_i} - \sqrt{\hat{h}_i})^2]$$
$$+ \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{obj} (C_i - \hat{C}_i)^2 + \lambda_{noobj} \sum_{i=0}^{S^2} \sum_{j=0}^{B} 1_{ij}^{noobj} (C_i - \hat{C}_i)^2$$
$$+ \sum_{i=0}^{S^2} 1_i^{obj} \sum_{c \in classes} (p_i(c) - \hat{p}_i(c))^2$$

Ecuación 9 Función de pérdida de YOLOV3,

Fuente: https://towardsdatascience.com/yolo-v3-explained-ff5b850390f, consultado el 24/11/2020

Aumento de datos (Data augmentation)

Se denomina aumento de datos (Data Augmentation) a la modificación de datos de entrenamiento existente o a la inserción de datos ajenos a una base de datos con el propósito de aumentar el número de datos para el entrenamiento de una red neuronal. Esto se puede lograr de varias formas, entre las que destacan la transformación de imágenes (traslación, rotación, escala), cambios de color o iluminación, inserción de ruido, entre otras técnicas [27].

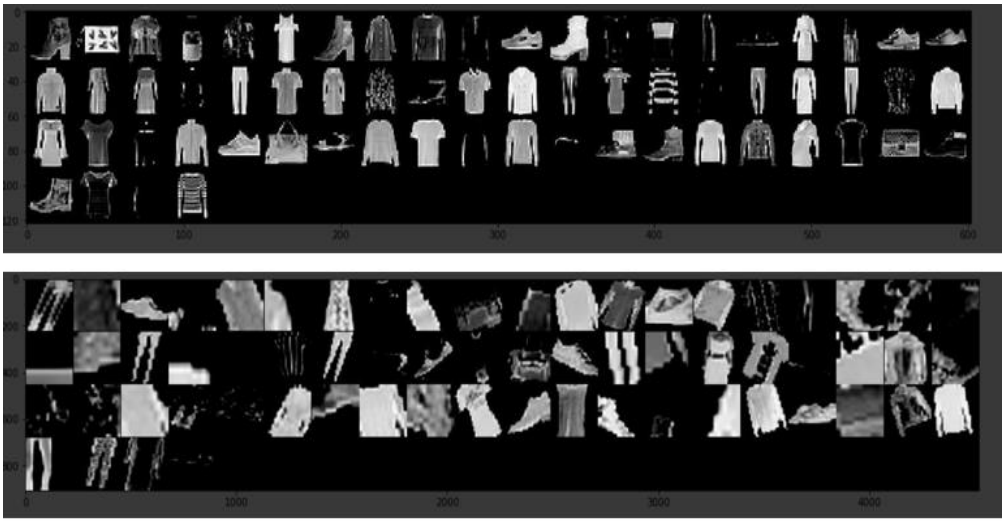


Figura 13 Ejemplo de aumento de datos. (Arr) fragmento de la base de datos Fashion MNIST. (Abj) El mismo fragmento con transformaciones aplicadas para aumentar datos. Imágenes de :[28]

Muchas veces, el desarrollo de bases de datos para entrenamiento profundo implica la creación y etiquetado de cientos, o incluso miles de imágenes. Esto por lo general se hace de forma manual, lo cual implica una gran inversión de tiempo. Por esta razón, muchos investigadores han optado por generar imágenes u objetos de forma digital con el propósito de disminuir el tiempo ocupado en generar una base de datos.

Uno de estos ejemplos es el trabajo de Mueller y Metzger [29] quienes utilizaron un ambiente virtual para generar imágenes adicionales para entrenar redes neuronales convolucionales con el propósito de mejorar la estimación de pose para navegación. Los resultados del estudio indican que el augmentar los datos de entrenamiento para varias arquitecturas mejora el resultado de estimación de la pose. Sin embargo, las poses estimadas no se acercan a las posiciones reales, por lo que queda abierta la opción de mejorar los resultados por otros métodos.

Otro ejemplo es el trabajo de Abu, Karthick, Mescheder, Geiger y Rother[30], quienes realizaron la inserción de modelos digitales de autos en la base de datos KITTI 2015 para entrenar el reconocimiento de peatones, autos y demás vehículos en entornos de calle. El conjunto de entrenamiento con imágenes augmentadas resultó ser más efectivo que el conjunto sin datos aumentados.

La creación de datos adicionales no sólo se limita a generar imágenes. También es posible generar herramientas completas o ambientes virtuales con el propósito de generar información. Tal es el caso del trabajo de Alhaija, Mustikovela, et al [30], quienes crearon una herramienta que combina datos generados por computadora (Objetos 3d) con fotografías 3D de escenarios, con el propósito de ahorrar en tiempo de creación de escenarios. Estas fotografías en 360 grados se utilizaron como mapas para poder generar combinaciones realistas de iluminación en los objetos generados. Todo esto con el propósito de generar una base de datos para entrenar algoritmos de conducción de automóviles en ambientes urbanos.

Figura 14 Ejemplo del trabajo de Alhaija, et al. En donde se puede observar la unión de datos sintéticos con una fotografía verdadera. Imagen de: [30]

Un trabajo que utiliza la generación de datos sintéticos para ayudar al reconocimiento de objetos manipulables por robots es el trabajo de Talukdar, et al[31]. Este trabajo genera imágenes con modelos tridimensionales relacionados con productos comestibles empaquetados. Estos son acomodados en el software blender, en donde se generaron 2000 imágenes. Las cuales luego fueron modificadas en diversas formas (blanco y negro, saturación de canales, transformaciones de traslación y rotación, inversión de colores, etc.). Posteriormente, estos datos son entrenados en una red neuronal llamada DetetcNet. Los resultados de los experimentos realizados indican que la combinación de varias técnicas de modificación de datos aumenta considerablemente la precisión de la red neuronal para localizar objetos.

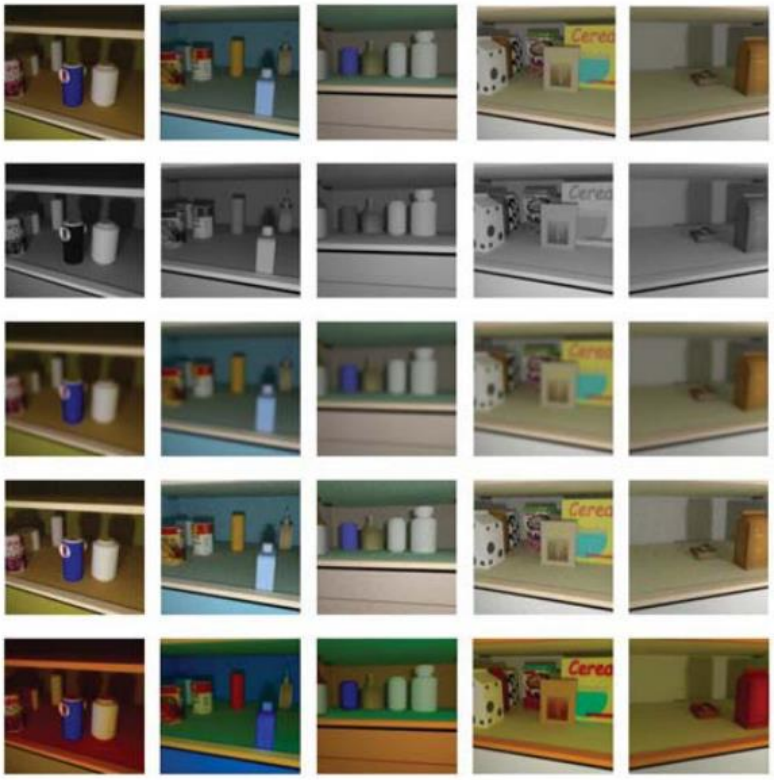


Figura 15 Ejemplo de modificación de imágenes del trabajo de Talukdar, et al. Se puede observar las imágenes originales en la fila superior, con modificaciones de color y emborronado en filas inferiores. Imagen de: [31]

Otros trabajos también tratan con aumentar datos para mejorar los resultados de tareas de reconocimiento de imágenes[8], [31], [32]. El primero trata con la creación y prueba de varios conjuntos de datos sintéticos para diversas redes de clasificación de imágenes. El segundo genera un cuarto completamente virtual para generar nuevas imágenes de entrenamiento y el tercero genera escenas que constan de varios objetos de sobremesa generados por computadora. Los tres trabajos reportan una mejora en el entrenamiento en comparación con métodos de entrenamientos sin aumentar datos.

Finalmente, un trabajo con características similares al presente trabajo es la base de datos llamada “Falling Things”, creada por Trembley, To y Borchfield [33], investigadores de la corporación NVIDIA. El trabajo para generar esta base de datos consistió en colocar de manera aleatoria una serie de 21 objetos de la base de datos YCB [34], los cuales fueron modificados con anterioridad, para mantener consistencia de coordenadas, en una serie de ubicaciones en tres mapas de alta

fidelidad gráfica. Los objetos fueron capturados con tres tipos distintos de cámaras simuladas (Estéreo, RGB y RGB-D) para generar un conjunto de 61,500 imágenes únicas. Las cuales han puesto a disposición de la comunidad para su uso en entrenamiento para el reconocimiento de objetos.

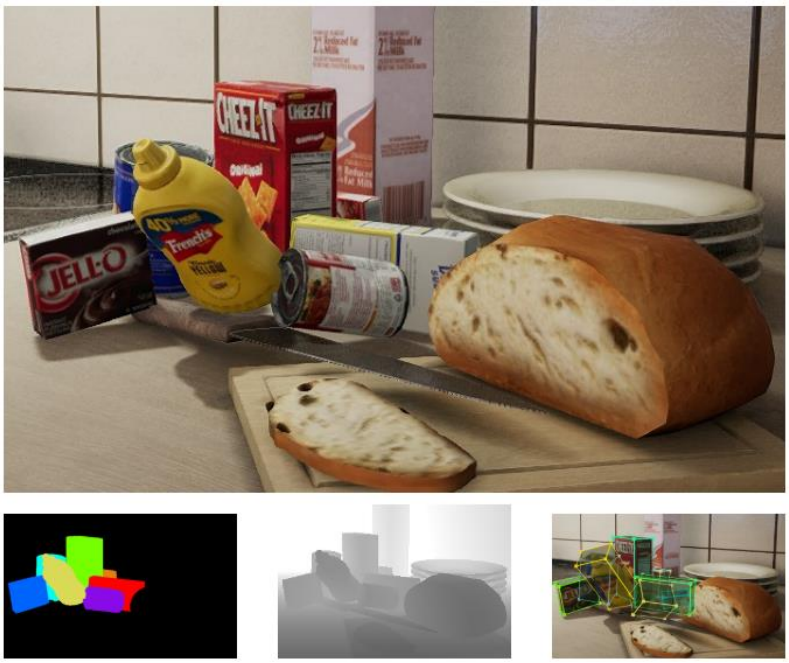


Figura 16 Imagen ejemplo de la base de datos "Falling things" (FAT), la cual consiste en colocar objetos tridimensionales en un ambiente virtual para obtener imágenes e información sobre ubicación y profundidad de los objetos. Imagen de: [33]

Ambientes virtuales

Un ambiente virtual se define como una representación del mundo real creada utilizando gráficos generados por computadora. Estas representaciones están diseñadas pensando en que el usuario interactuará con el ambiente, por lo tanto, cuentan con formas para controlar el ambiente. La inmersión del usuario se logra con una combinación de dispositivos que otorgan retroalimentación visual, auditiva y en ocasiones háptica al usuario.

El origen de los ambientes virtuales como concepto se remonta a principios del siglo pasado, en donde el avance en iluminación y tecnología permitieron crear

simulaciones mecánicas para el entrenamiento de pilotos. Las primeras de estas fueron simuladores de vuelo mecánicos. Pero no fue sino hasta la década de los 50 que el avance tecnológico permitió utilizar métodos de visualización electrónicos. En 1958 la compañía Philco creó lo que se considera el primer casco de realidad virtual, mientras que en 1968 Ivan Shuterland desarrolló el primer display de realidad aumentada [33, 34].

Avances en décadas siguientes incluyeron la utilización de pantallas electrónicas para presentar información al usuario, mejoras mecánicas en los sistemas de simulación y la adición de sistemas de retroalimentación junto con mejoras gráficas en los sistemas de desarrollo para la industria de la computación y los videojuegos.

Todos estos avances han permitido la diversificación de cómo se presentan los ambientes virtuales ante el usuario. Y gracias a esto, el número de aplicaciones de estos ha aumentado considerablemente, desde aplicaciones militares y en la industria hasta aplicaciones de ocio, educación y turismo.

Clasificación de presentación de ambientes virtuales

Representación en computadora

Los ambientes virtuales, por naturaleza, son generados en ordenadores utilizando técnicas de modelado y programación. Por lo general, muchos de estos son diseñados para ser accedidos por el usuario mediante el uso de sus propias computadoras sin necesidad de equipos especializados. Ejemplos de esto incluyen videojuegos, simuladores, programas de entrenamiento, tours virtuales, etc. En estos casos, la principal vía de interacción del usuario con el entorno es mediante el monitor, el ratón y el teclado.

Realidad Virtual (VR)

La realidad virtual se define como una simulación del mundo real en donde el usuario se ve inmerso en este mundo mediante la utilización de dispositivos electrónicos como pantallas, headsets, lentes, celulares u otros dispositivos que pueda transmitir una imagen. Estos dispositivos usualmente bloquean

completamente la visión del mundo real mediante una combinación de imágenes y sonidos y en el caso de headsets, cuentan con sensores de movimiento para registrar los movimientos del usuario y simularlos en el ambiente virtual, lo cual aporta a la ilusión. Adicionalmente, muchos dispositivos de realidad virtual cuentan con controles que permiten interactuar con el entorno virtual y proveen retroalimentación háptica al usuario [37].

Realidad aumentada (AR)

La realidad aumentada es la técnica mediante la cual se sobreponen elementos digitales en superficies del mundo real. En un principio, esto se limitaba a los HUDs (Heads-up-displays) presentes en equipos militares, pero avances tecnológicos han permitido que esta tecnología llegue al sector comercial mediante dispositivos más ligeros como teléfonos celulares [38].

Realidad mixta (XR)

La realidad mixta es una combinación de VR y AR. Mediante el uso de dispositivos con cámaras y sensores de profundidad se crean modelos del mundo real y sobre este se sobrepone la información digital. Los dispositivos para esta tecnología se presentan en forma de gafas o cascos que sobreponen una superficie transparente en donde se sobrepone las imágenes. Un ejemplo de esto son los Microsoft Hololens [39].

Motor de creación de videojuegos para desarrollo

Para el correcto desarrollo de la aplicación, es necesario contar con un entorno de desarrollo que permita crear entornos tridimensionales virtuales de forma sencilla. Es posible desarrollar estos entornos mediante código, sin embargo, hacer un ambiente de desarrollo desde cero, sale de los objetivos de este trabajo, ya que se requiere no sólo el ambiente, sino también la lógica para poder simular interacciones entre objetos y sistemas de física e iluminación.

Por lo tanto, la opción idónea es la utilización de entornos de desarrollo ya existentes.

Motor de desarrollo de videojuegos:

Se entiende como Motor de desarrollo de videojuegos a un sistema que reúne varias bibliotecas de software, las cuales permiten manejar diversos aspectos gráficos, auditivos y lógicos con el propósito de generar un videojuego o software interactivo. Muchas de las funciones de estos programas permiten integrar modelos tridimensionales, efectos, luces, física y programación en un entorno visual, lo cual facilita la unión multidisciplinaria en la creación de videojuegos y software.

Existen varios entornos para el desarrollo de aplicaciones 3D, sin embargo, los más adecuados para esta tarea son los entornos dedicados al desarrollo de videojuegos. Entre estos, destacan los siguientes:

Unity:

Unity es una de las herramientas para desarrollar videojuegos más utilizadas hoy en día. En comparación con otros entornos de desarrollo, es fácil de usar y cuenta con una comunidad activa, documentación detallada para varias versiones y un entorno en el cual se pueden obtener recursos adicionales ya sean gratuitos o de paga. Unity maneja principalmente los lenguajes de programación C# y Java.[40]

Unreal Engine:

Otro motor de desarrollo popular, Unreal Engine es un motor de desarrollo especializado para el desarrollo de aplicaciones 3D. Utilizado por muchas compañías profesionales de videojuegos, este motor permite tener un gran control sobre el aspecto visual del entorno.

El lenguaje principal de programación utilizado por Unreal es C++, pero también cuenta con un sistema de programación gráfica llamada Blueprints, la cual permite controlar variables y funciones en forma de diagrama de flujo.

Unreal Engine también cuenta con un portal para obtener recursos adicionales llamado Marketplace [41].

Godot:

Godot es un entorno de desarrollo gratuito de código libre bajo la licencia del MIT, es compatible con una gran cantidad de sistemas operativos como Windows y Linux y tiene compatibilidad para C# y C++ además de contar con una interfaz de programación gráfica. Tiene la capacidad de construir juegos en 2D y 3D, lo cual lo hace una herramienta muy flexible [42].

Algo importante a recalcar es que, a pesar de ser plataforma abierta, Godot no cuenta con un portal para obtener recursos adicionales análogo a la Tienda de Complementos (Asset Store) de Unity y el Marketplace de Unreal Engine. Debido al tiempo requerido para crear estos modelos, se decidió no usar Godot para el desarrollo del entorno.

Capítulo 3: Desarrollo del sistema virtual

En este capítulo se explicará a detalle el desarrollo de la aplicación de generación de información para el entrenamiento de redes neuronales.

Primero, se hablará de la elección del entorno de desarrollo, luego se explicará el proceso de obtención y adecuación de modelos digitales en el entorno y posteriormente se explicarán de forma individual las diversas funciones de la aplicación, así como la estructura conjunta de estas funciones y la forma en la que se guarda la información.

Elección del motor de desarrollo

Considerando las características de los tres motores mencionados en el capítulo anterior, se terminó escogiendo Unity como el motor de desarrollo. Debido a su facilidad de uso, amplia documentación, fácil acceso a modelos gratuitos de calidad similar y el hecho de que el software no requiere el pago de algún derecho o comisión, siempre y cuando el software generado no se publique para su venta. Unreal, en contraste cuenta con una curva de aprendizaje más lenta, lo cual alentaría el desarrollo del trabajo.

Característica	**Unity**	**Unreal Engine**
Precio	Gratis, con una cuota anual si se publica software	Gratis, con una cuota si se publica software
Plataformas objetivo	Xbox 360; Xbox One; Wii U; PlayStation 3; PlayStation 4; PlayStation Vita; Nintendo Switch; Nintendo 3DS, Windows, OSX, Linux.	PlayStation 4, Xbox One, Nintendo Switch, Windows, Mac OS X, Linux, SteamOs, HTML5.
Disponibilidad en plataformas	Windows; OSX; Linux	Windows; OSX; Linux
Acceso a recursos adicionales	Sí	Sí

Acceso a documentación	Sí	Sí
Lenguajes de programación	C#	C++, Blueprints
Soporte gráfico API	DirectX, Metal, OpenGL, Vulkan	DirectX 10, DirectX 11, Vulkan (SM5), OpenGL

Tabla 5 Tabla de características de motores de desarrollo.

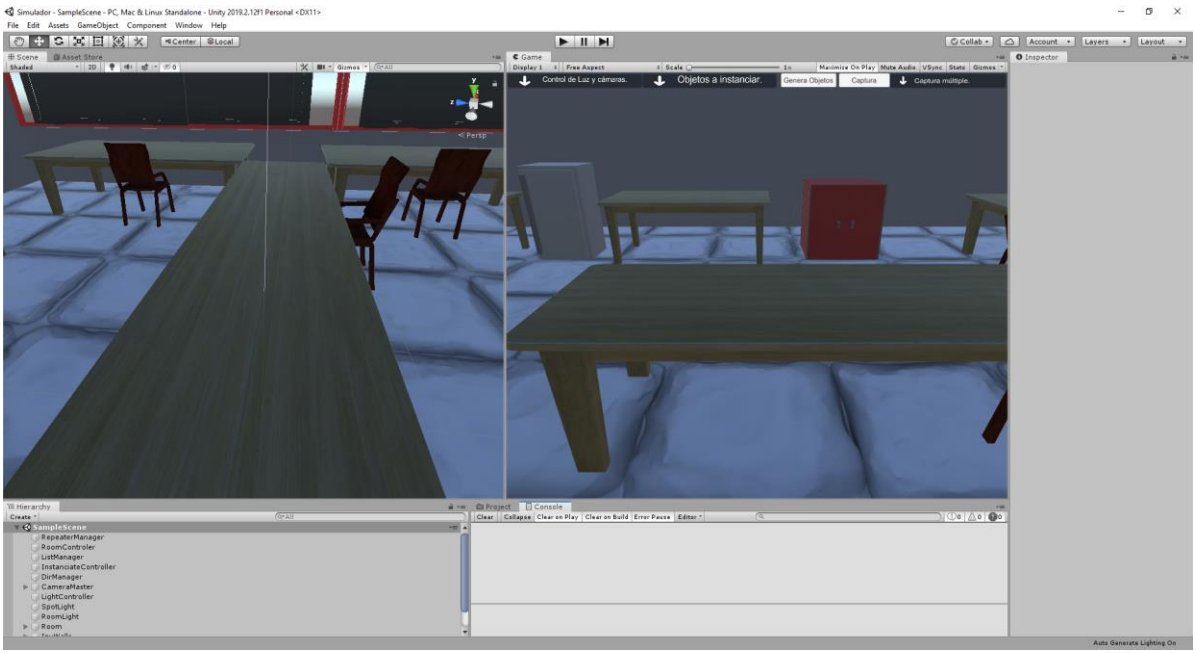


Figura 17 Vista del entorno de trabajo de Unity

Modelado de objetos

Si bien muchos de los modelos utilizados fueron obtenidos en la Tienda de Complementos de Unity, se crearon varios modelos utilizando la herramienta de modelado Blender. La cual es una herramienta gratuita para crear modelos 3D, animaciones y películas.

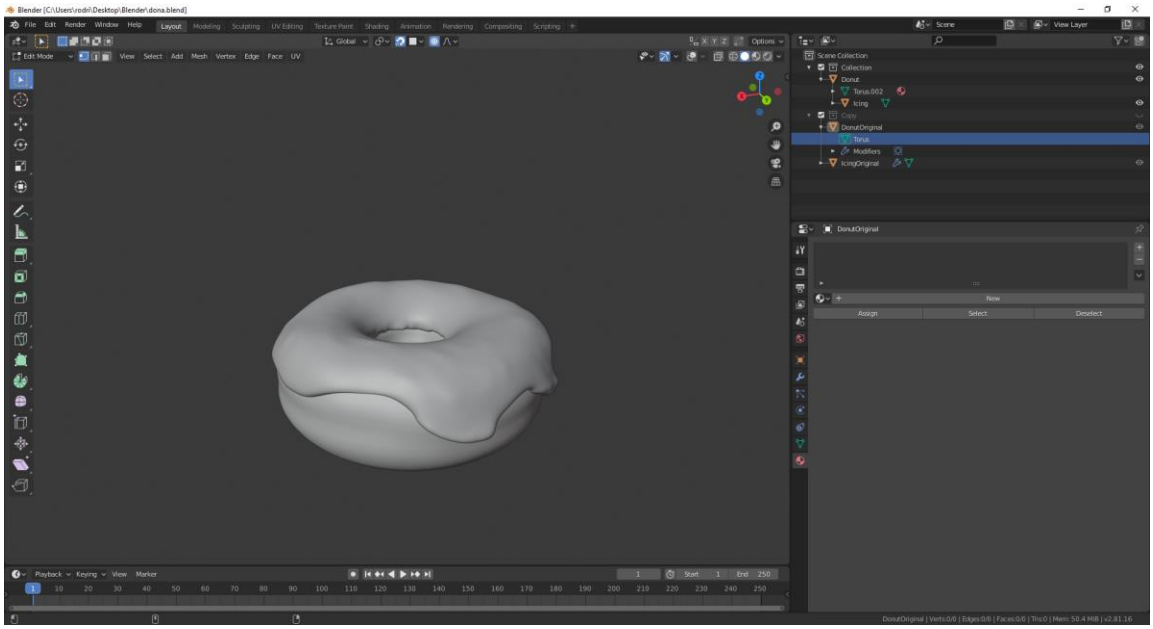


Figura 18 Vista del entorno de trabajo de blender

Los modelos generados se guardaron en formato OBJ para poder usarlos en Unity.

Organización de módulos

Durante el desarrollo de la aplicación computacional se programaron de forma individual los módulos a cargo de las funciones principales. Esto con el propósito de poder tener un mejor control sobre cada función y poder corregir de forma fácil algún error en caso de presentarse.

El desarrollo de los módulos principales (Listado, generación y captura) fue realizado de manera simultánea, mientras que los demás módulos (Repetición, actualización de directorio y reorganización del entorno) fueron agregados posteriormente.

La organización de estos módulos se puede observar en la figura 19.

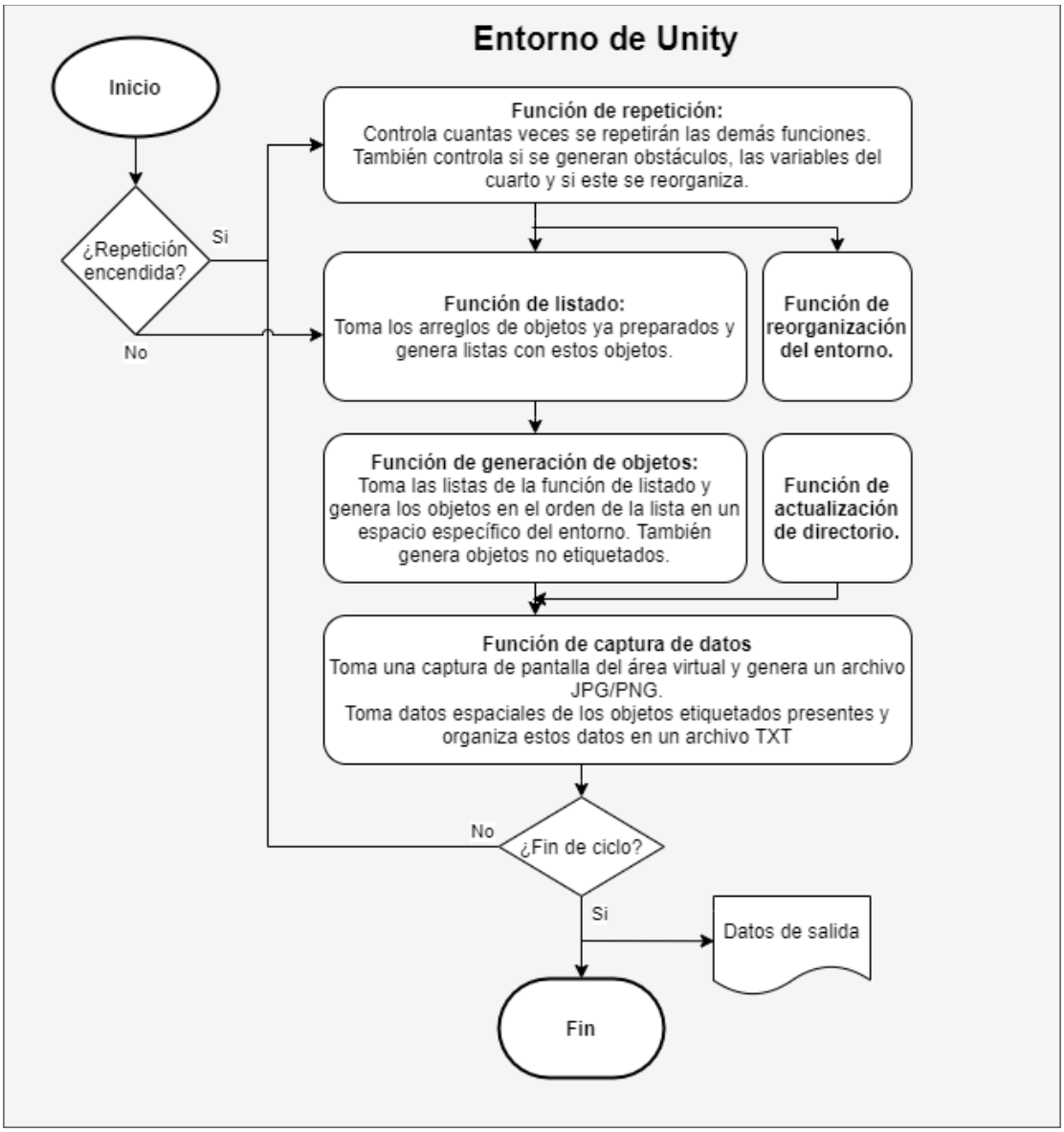


Figura 19 Diagrama de módulos de la aplicación.

Los detalles de cada módulo, incluyendo sus funciones y relación con el entorno de Unity serán explicados a continuación.

Módulo de automatización

El primer módulo en el orden de operación es el módulo de repetición o automatización. Este módulo tiene como tarea controlar las variables del entorno de forma automática además de llamar a las funciones principales de cada uno de los módulos posteriores.

El uso de este módulo es opcional. Si no se usa, las funciones posteriores se deben activar de forma manual mediante la interfaz gráfica.

Como datos de entrada, se toman el número de iteraciones que se requiere hacer, si se requiere crear obstáculos, si se requiere reorganizar el cuarto y cada cuántas iteraciones se debe reorganizar. Todos estos datos son recabados de la interfaz de usuario.

Con estos datos, el módulo realiza dos funciones principales:

- Cambiar las variables de entorno (iluminación, posición de cámara).
- Ejecutar las funciones de módulos posteriores.

Para poder cambiar las variables del entorno de manera aleatoria, el módulo toma información de los objetos *slider* correspondientes en la interfaz de usuario. Con esa información, genera un valor al azar dentro de los límites de cada *slider* y lo asigna en tiempo real.

Para poder mandar a llamar a cada módulo posterior, el código utiliza la función *onClick.Invoke()*, la cual se utiliza para activar botones desde el código. Ya que la función principal de cada módulo está ligada a un botón en la interfaz de usuario, el módulo de automatización simplemente manda a llamar cada botón en el orden de los módulos.

Para evitar que el proceso se vea interrumpido, se desactiva la interfaz de usuario mientras se realizan las iteraciones.

El orden de operaciones se puede observar en la figura 20.

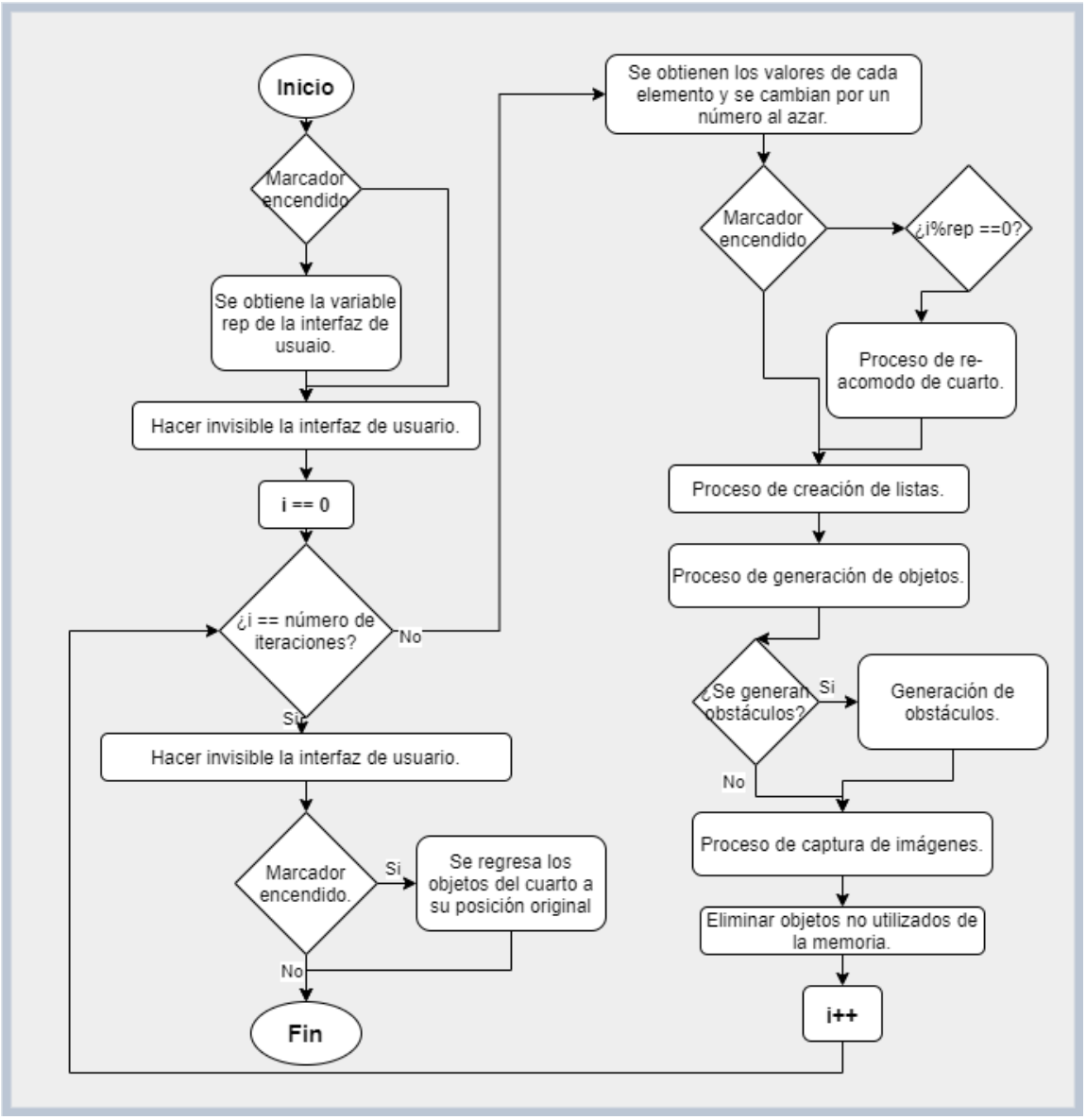


Figura 20 Diagrama del funcionamiento del módulo de repetición.

Módulo de generación de listas

El segundo módulo en el orden de operación es el módulo de creación de listas. Este módulo tiene la tarea de tomar los elementos listados en la interfaz de usuario y generar un número de listas que se van a usar para poder generar los objetos en el ambiente virtual.

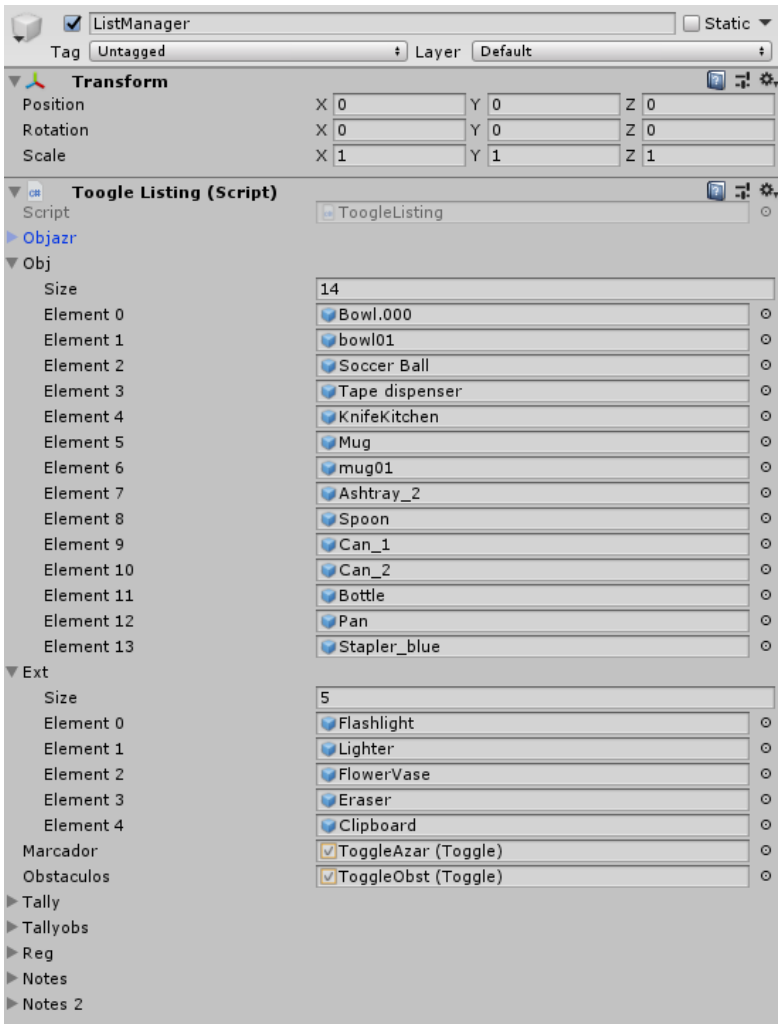


Figura 21 Ejemplo de listas de objetos en la interfaz gráfica de Unity.

El primer paso para poder manipular estas listas es tener los objetos que se necesitan generar en arreglos. Esto se puede lograr en la interfaz de usuario una vez que el código necesario indique que estos arreglos son públicos, lo cual nos permite ver el agregar los objetos ya preparados a este.

Se requieren tres listas para cumplir con las distintas funciones de la aplicación. La primera lista cuenta con un número grande de objetos y es utilizada para generar un número al azar de objetos. La segunda lista consta de 15 elementos y esta lista sirve para generar los elementos seleccionados por el usuario en la interfaz de la aplicación.

La última lista es para aquellos objetos que serán utilizados como obstáculos. Los objetos colocados en esta lista serán generados, pero no serán registrados por el algoritmo de captura de datos.

Para poder seleccionar objetos al azar de la primera lista y para poder generar los objetos de cada lista en un orden aleatorio, es necesario contar con un método para acomodar estas de forma aleatoria. Para este trabajo se escogió el algoritmo Knuth Shuffle.

Algoritmo Knuth shuffle:

También conocido como el algoritmo Fisher Yates, este corto algoritmo permite mezclar los elementos de un arreglo[43]. En este módulo, se utiliza este algoritmo para poder mezclar las listas de objetos a generar con el propósito de obtener un cierto orden aleatorio en el número y orden de aparición de estos.

Algoritmo Knuth Shuffle
Datos de entrada: Un arreglo de tamaño n
Datos de salida: Un arreglo tamaño n con los elementos del arreglo de entrada en un orden distinto.
Complejidad del algoritmo: O(n)

```
for (int i = 0; i < largo del arreglo; i++)
        {
            int temp = inicial[i];
            int r = Random.Range(i, inicial.Length);
            inicial[t] = inicial[r];
            inicial[r] = temp;

        }
```

Pseudocódigo 1 Algoritmo Knught Shuffle

El algoritmo para poder crear estas listas se compone de una serie de pasos idéntica para cada lista, con el único cambio siendo la longitud del arreglo necesario y la longitud de las listas creadas. El algoritmo general utiliza sentencias if-else para poder generar los casos de forma independiente.

Pseudocódigo del algoritmo de creación de listas
Datos de entrada: Un arreglo de entrada tamaño m, una lista de Objetos llamada Tally
Datos de salida: Una lista de tamaño n
Complejidad del algoritmo: O(n)

- Se crea una lista A con longitud igual al tamaño del primer arreglo
- Se llena esta lista con los dígitos 0 a la longitud menos 1
- Se revuelve esta lista utilizando el algoritmo Knuth shuffle
- Se crea un número entero al azar x
- Se crea una nueva lista B con una longitud igual a x
- Con un ciclo for se copian los primeros x elementos de la lista A a la lista B
- Se limpia la lista Tally
- Con un ciclo for se agrega a Tally los elementos del primer arreglo correspondientes a los índices en B
- Se regresa la lista Tally con los Objetos ordenados.

Pseudocódigo 2 Algoritmo de creación de listas.

Las listas generadas son utilizadas por el siguiente módulo.

Módulo de generación de objetos

El tercer módulo tiene la tarea de generar los objetos en el ambiente virtual para que estos puedan ser visualizados por el usuario. También es necesario manejar el borrado de dichos objetos para mantener una baja utilización de la memoria del sistema y poder tener una captura de datos correcta.

Motor de física de Unity:

En el entorno de Unity es necesario simular de manera realista el comportamiento de objetos según las leyes de la física, por esta razón, el entorno cuenta con un motor de física, el cual simula acciones como aceleración de objetos, movimiento y colisiones.

Algunos de los elementos que conforman el motor de física incluyen;

Colisionadores:

Un colisionador o collider es un componente de un objeto que permite que el objeto pueda entrar en contacto con otros objetos del entorno que también cuenten con un colisionador y un cuerpo rígido. Por lo general, estos colisionadores cubren en su

totalidad al objeto y no poseen la misma forma que el objeto, para hacer los cálculos más eficientes.

Los colisionadores más básicos que se pueden utilizar son llamados colisionadores primitivos. Estos presentan formas básicas como cubos, esferas o cápsulas. Estos a su vez, pueden combinarse en colisionadores compuestos para simular de forma más exacta los límites físicos de un objeto.[44]

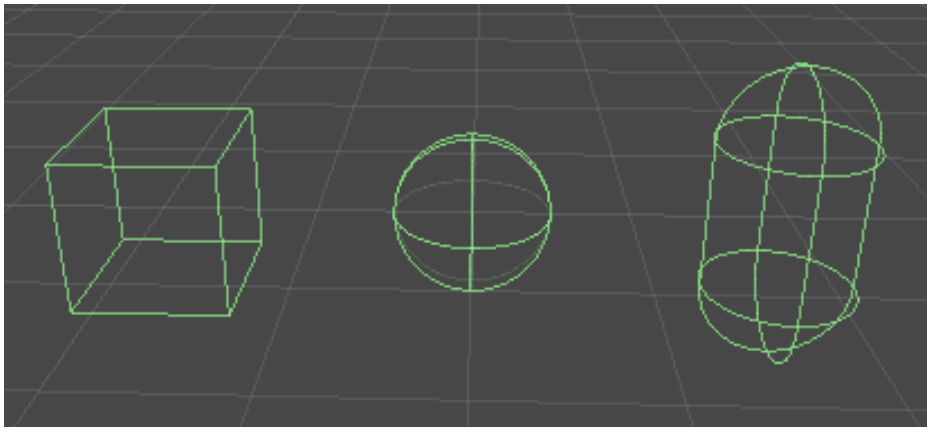


Figura 22 Tres tipos de colisionadores primitivos para un entorno 3D

Cuerpos rígidos:

Un cuerpo rígido o RigidBody, es un componente que permite que el motor de física controle el objeto al cual este componente esté agregado. Si a un objeto se le agrega un componente RigidBody, este de inmediato reaccionará a fuerzas externas como la gravedad. La presencia de este componente permite al objeto interactuar con colisionadores para simular colisiones o activar comportamientos, dependiendo de lo que se requiera.[45]

Raycasting:

En el entorno virtual de Unity, un rayo o Ray se define como una línea que conecta dos puntos A y B en el espacio. Raycasting es una operación en la cual se envía un

rayo a partir de un punto en específico con una dirección determinada por un vector. El rayo generado es invisible y su duración es muy corta.

El propósito de esta operación es recopilar información de todo objeto con un componente colisionador que el rayo intercepte. A esta colisión se le conoce como RayCastHit. La información recopilada describe las características del objeto con el cual colisionó incluyendo su posición en el espacio, la longitud del rayo o el punto en el cual ocurrió la colisión. [46]

Esta operación es útil para poder determinar la posición de objetos en el espacio y para determinar si ciertos objetos se encuentran en determinada dirección. Para la generación de objetos en este proyecto, se utilizó la función de raycasting para determinar si el espacio en donde el algoritmo iba a generar el objeto se encontraba cerca del centro del entorno virtual para poder visualizar el objeto con facilidad.

Figura 23 Visualización de la operación de4 Raycasting, imagen de: https://learn.unity.com/tutorial/let-s-try-shooting-with-raycasts#5c7f8528edbc2a002053b469

$$X = X_0 + t * D_x$$

$$Y = Y_0 + t * D_Y$$

$$Z = Z_0 + t * D_Z$$

Ecuación 10 Expresión vectorial de la operación de Raycasting

Cuaternión

Los cuaterniones son números hipercomplejos de la forma $a + bi + cj + dk$ en donde los coeficientes a, b, c y d son números reales y las tres unidades imaginarias cumplen con las siguientes características:

$$i^2 = j^2 = k^2 = -1$$

$$ij = k = -ji, \qquad jk = i = -kj, \qquad ki = j = -ik$$

Ecuación 11 Características de los cuaterniones

Los cuaterniones son un campo no conmutativo con propiedades asociativas y distributivas, lo cual hace el concepto una de las bases del álgebra vectorial.

De forma más relevante a este trabajo, los cuaterniones son utilizados en el modelado y manipulación de objetos en ambientes virtuales para representar rotaciones en el espacio.

Unity hace uso de los cuaterniones para representar y controlar internamente las rotaciones de los objetos en el entorno virtual. Por lo general, esto no es modificado por el usuario, ya que es preferible que se haga una interpolación entre rotaciones para generar una nueva posición. Sin embargo, existen funciones que interactúan directamente con los cuaterniones, ya sea para obtener información de estos o para generar un cuaternión nuevo. En el caso de este trabajo, cada vez que se genera un objeto, es necesario declarar la rotación que va a tener con respecto a la rotación original del objeto en el entorno, característica que es guardada en todos los objetos que Unity utiliza. Para propósitos de facilidad de pruebas, se restringió esta rotación a solamente ser sobre el eje y, sin embargo, esto puede ser modificado fácilmente para futuras pruebas.

Preparación del entorno

Para poder asegurar que los objetos generados fuesen visibles para la cámara y el usuario, se delimitó un área específica en el entorno.

El área escogida fue delimitada por 5 colisionadores de caja. Uno de estos correspondía a una mesa virtual colocada para que los objetos generados cayeran sobre esta. Los otros cuatro colisionadores rodean la mesa formando paredes con el propósito de que los objetos no se salieran de la misma.

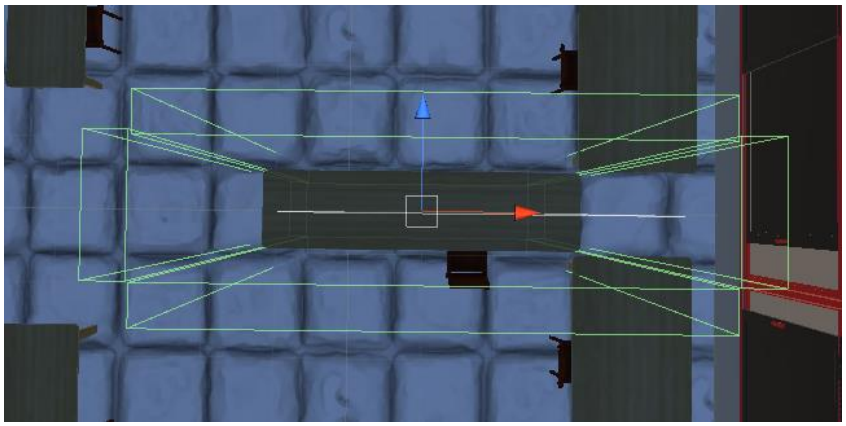


Figura 24 Vista superior de la mesa virtual con las 4 paredes invisibles rodeándola.

Algoritmos:

El funcionamiento del módulo se puede resumir en dos algoritmos: Un algoritmo para generar los objetos y otro para quitarlos del entorno cuando ya no es necesario tenerlos en escena.

Pseudocódigo del algoritmo de generación de objetos
Datos de entrada: Una lista de Objetos llamada Registro, un entero n
Datos de salida: Una lista de tamaño n
Complejidad del algoritmo: O(n)

- Se crea un entero contador con valor igual a 0
- Mientras contador sea menor a n (bucle while).
    - Se crean seis números flotantes al azar, tres para posición y tres para ángulo
    - Se inicializa una variable tipo Ray, para una operación de raycasting con inicio en el vector de posición y dirección (0,-1,0).
    - Si se detecta una colisión, y la variable "tag" del objeto es "Table":
        - Se genera un número tipo flotante n al azar con un valor entre 0.8 y 1.2.
        - Se genera el objeto de la lista correspondiente a la variable contador por debajo del rayo.
        - Se multiplica n por la escala local del objeto.
        - Se agrega este objeto a una lista local pública llamada "conteo"
        - Se aumenta contador por 1

Pseudocódigo 3 Algoritmo de generación de objetos.

Al generarlos, estos objetos son afectados por la gravedad del entorno y caen hasta colisionar con la mesa. Como cada objeto generado cuenta con componentes Rigidbody y BoxCollider, estos pueden llegar a colisionar entre ellos, lo cual lleva a aumentar la variación de escenas que se pueden crear.

Figura 25 Ejemplo del resultado del algoritmo de generación de objetos.

Pseudocódigo del algoritmo de destrucción de objetos
Datos de entrada: Un entero n
Datos de salida: Una lista de tamaño n
Complejidad del algoritmo: O(n)

- Si n>0
    - Se crea un número entero llamado contador.
    - Do-while (contador>0)
        - Se asigna a una variable GameObject el objeto en la lista pública Tally correspondiente al índice contador.
        - Se elimina ese objeto mediante el comando Destroy().
        - Se reduce la variable contador por 1.
    - Se limpia la lista Tally.

Pseudocódigo 4 Algoritmo de destrucción de objetos

Estos dos algoritmos también son utilizados para generar al azar los objetos que componen el entorno virtual. La única diferencia es que para eso se utiliza otra lista de obstáculos que también proviene del módulo anterior.

Módulo de captura de información

El cuarto módulo es el encargado de tomar imágenes de la cámara, recabar los datos de posición de los objetos generados por el módulo anterior y guardar estos

datos en la ubicación correspondiente. Este módulo se puede dividir en varios métodos que cumplen una función en particular:

- El primer método se encarga de recabar la información espacial de cada objeto con respecto a la cámara.
- El segundo método tiene la tarea de tomar la captura de pantalla del punto de vista de la cámara.
- El tercer método se encarga de guardar los datos provenientes de los primeros dos módulos en los archivos correspondientes.

La organización del módulo se puede ver en la figura 25.

Algoritmo para la obtención de datos de objetos

Los datos necesarios para entrenar una red neuronal para la detección de objetos son:

- Una imagen del objeto en cuestión en donde este se pueda ver de forma clara.
- Un documento de texto en donde se describa cuáles objetos se encuentran en la imagen, la posición de los mismos en coordenadas x,y además de las dimensiones de una caja que rodea al objeto en la imagen.

Cada objeto que se quiera identificar debe de tener una etiqueta distinta para poder diferenciarlos. Estas etiquetas se asignan a cada objeto en a interfaz de usuario de Unity.

Por lo que la línea de datos que se requiere para describir a un objeto en una imagen es:

Etiqueta, Coordenada centro x, coordenada centro y, ancho de caja, alto de caja.

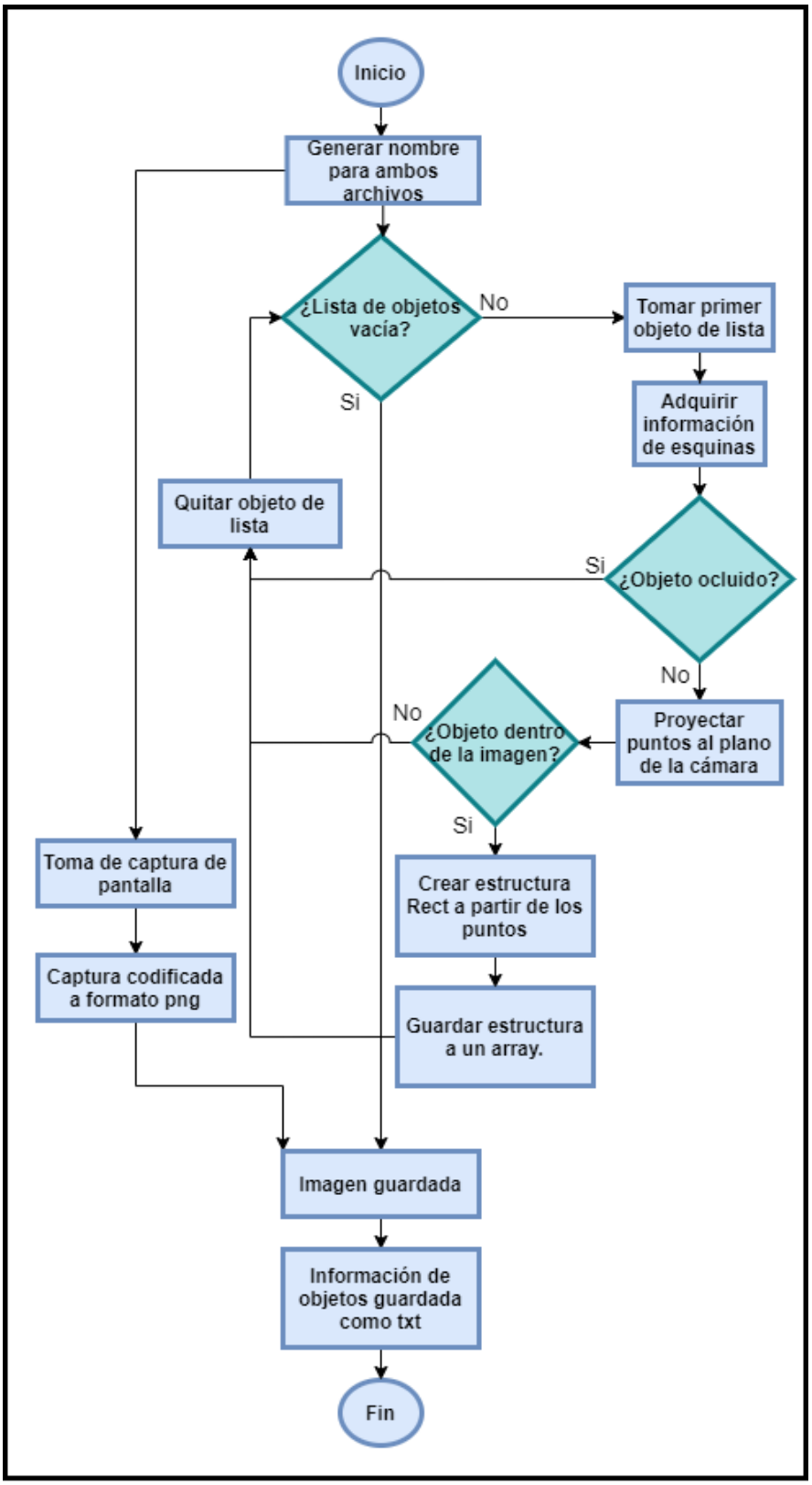


Figura 26 Organización de la sección de captura de datos.

Para poder obtener estos datos, es necesario obtener puntos que describan la posición del objeto y posteriormente proyectar estos puntos al plano de la cámara para que estos coincidan con las coordenadas del plano de la cámara.

Los puntos que se utilizan para cada objeto es el centro del objeto y cada una de las 8 esquinas del colisionador cúbico que rodea al objeto. Las coordenadas de estos puntos están dadas con respecto a una coordenada cero que se encuentra en el centro del ambiente virtual.

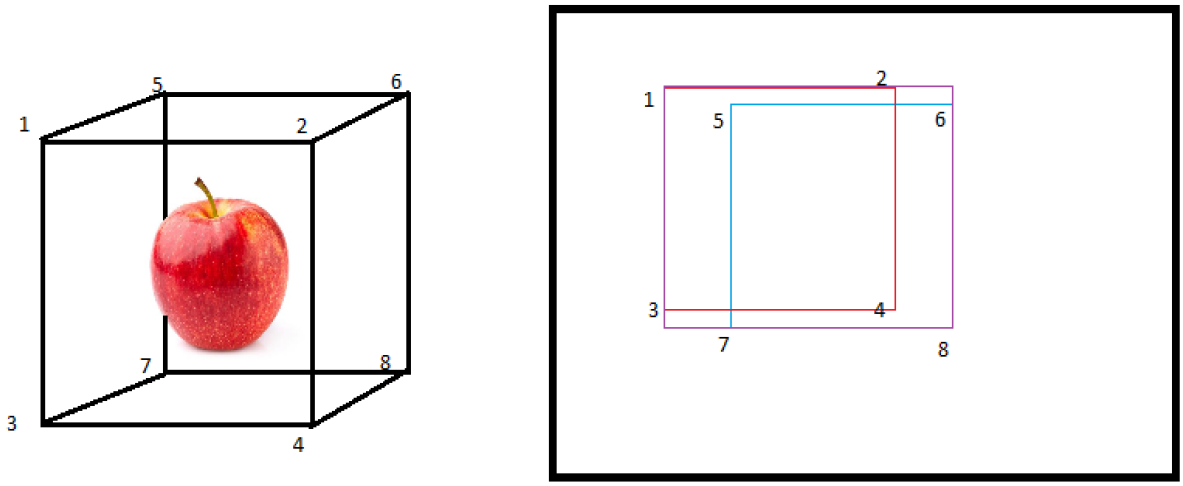


Figura 27 Boceto conceptual de la proyección de las coordenadas del objeto al plano de la cámara.

Para que estas coordenadas sean correctas, en el algoritmo, se multiplica cada coordenada por la matriz de rotación correspondiente al objeto y por la escala que ese objeto presenta.

Proyección a la pantalla

El sistema de visión de Unity maneja proyecciones matriciales basadas en el modelo de OpenGL. Para que un punto en el ambiente se traduzca a un punto en la visión del usuario, se debe pasar por una serie de transformaciones de espacio.

El primer paso es transformar las coordenadas del punto en el ambiente virtual del sistema de coordenadas absoluto al sistema de coordenadas de la cámara. Esto se logra multiplicando por una matriz de rotación correspondiente al ángulo y distancia de la cámara con respecto al centro del espacio virtual.

$$\begin{bmatrix} X_C \\ Y_C \\ Z_C \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_1 \\ r_{21} & r_{22} & r_{23} & t_2 \\ r_{31} & r_{32} & r_{33} & t_3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

Ecuación 12 Transformación a sistema de coordenadas de la cámara.

Estas coordenadas son proyectadas al plano de visión de la cámara. El tronco de visión de una cámara en Unity (y en OpenGL) se representa como una pirámide truncada con la posición de la cámara en el punto en donde se encontraría la punta de la pirámide. Los planos del corte y la base se denominan n y f respectivamente. El plano N es el plano de visión de la cámara mientras que el plano f corresponde al máximo de visión. Los ejes en el plano N se rigen bajo las coordenadas l y r para el eje x y en las coordenadas b y t para el eje y. Ambos conjuntos son equivalentes a -1 y 1 respectivamente, como se puede ver en la figura 45.

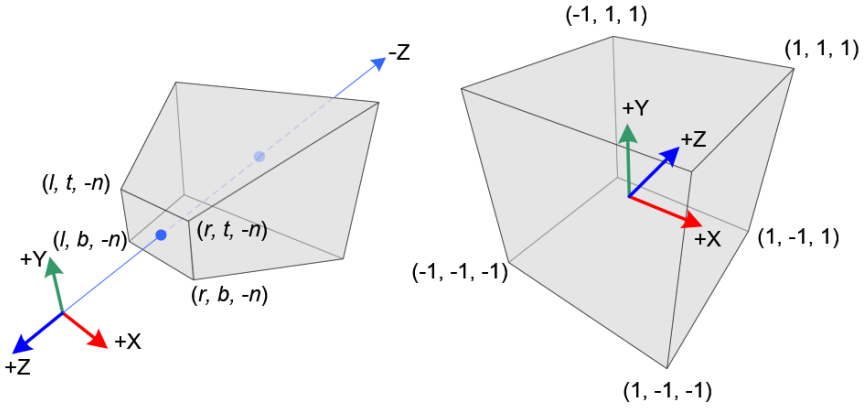


Figura 28 Visualización del tronco de visión de cámara y comparación con posición real de los objetos. Imagen de: [47]

Utilizando trigonometría, triángulos similares, se consiguen las transformaciones para poder proyectar el punto al plano cercano del tronco de visión.

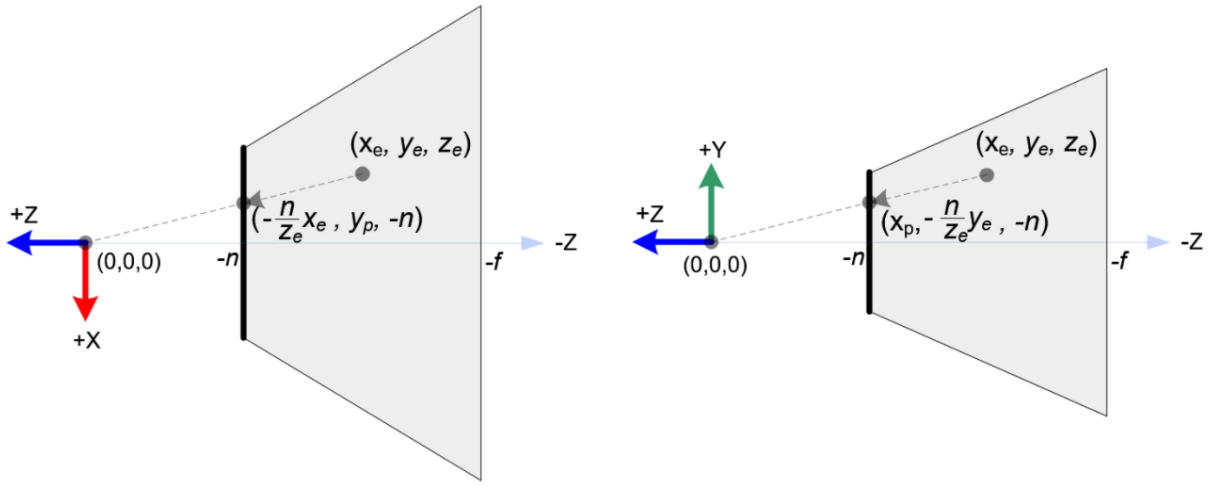


Figura 29 Proyecciones de un punto en la base del tronco de visión de una cámara. Imagen de:[47]

$$\begin{bmatrix} X_p \\ Y_p \\ Z_p \\ W_p \end{bmatrix} = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{-(f+n)}{f-n} & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} X_c \\ Y_C \\ Z_c \\ 1 \end{bmatrix}$$

Ecuación 13 Proyección al plano de visión de la cámara para un tronco general (no simétrico).

Posteriormente, las coordenadas proyectadas son divididas por el componente W del vector para generar las coordenadas normalizadas del dispositivo (NDC), estas son las coordenadas homogéneas de los puntos en el plano, las cuales son invariantes a la escala.

$$\begin{bmatrix} X_h \\ Y_h \\ Z_h \end{bmatrix} = \begin{bmatrix} X_p/W_p \\ Y_p/W_p \\ Z_p/W_p \end{bmatrix}$$

Ecuación 14 Homogeneización de las coordenadas.

Finalmente, para convertir estas coordenadas a las coordenadas de la visión de pantalla, simplemente se hace una conversión basándose en la estructura (x,y,w,h).

$$\begin{bmatrix} X_{vp} \\ Y_{vp} \\ Z_{vp} \end{bmatrix} = 0.5 \begin{bmatrix} w * X_h \\ h * Y_h \\ (f-n)Z_h \end{bmatrix} + \begin{bmatrix} X_0 + w * 0.5 \\ Y_0 + h * 0.5 \\ (f-n)0.5 \end{bmatrix}$$

Ecuación 15 Conversión a coordenadas de visión de pantalla.

En donde X_0 y Y_0son los puntos iniciales de las coordenadas de pantalla (en la mayor parte de los casos este se encuentra en la esquina inferior izquierda), w y h son el ancho y alto de la pantalla medido en escala de 0 a 1 y n y f son la distancia mínima y máxima del tronco rectangular de visión, respectivamente.[47], [48]

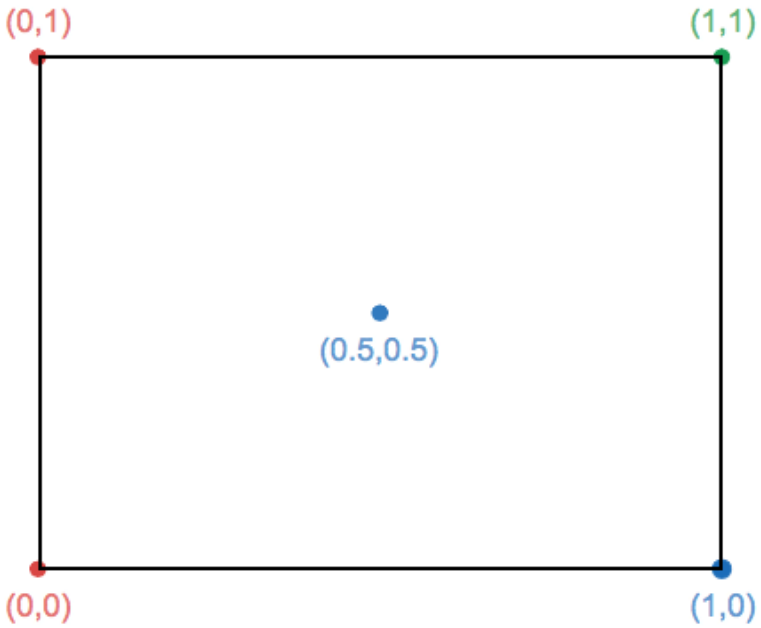


Figura 30 Visualización del espacio de pantalla en Unity, imagen de: [49]

Estructura rectangular

Estas coordenadas son posteriormente proyectadas en el plano de la cámara. En este nuevo marco de referencia se hace una serie de comparaciones para obtener la mayor distancia en ambos ejes entre los ocho puntos. Con estas nuevas coordenadas, finalmente se puede generar la estructura rect (x,y ancho, alto) que se requiere. Los datos adicionales para la estructura se agregan en el algoritmo de guardado de datos.

Pseudocódigo del algoritmo para la obtención del rectángulo correspondiente a un objeto
Datos de entrada: El colisionador C, la matriz de rotación m, la escala e, la cámara cam
Datos de salida: Una estructura rect con el rectángulo que rodea al objeto
Complejidad del algoritmo: O(n)

- Se obtiene la posición P=(x,y,z) del centro de C.
- Se crea una estructura M tipo Vector2 (coordenadas x,y).
- Por cada esquina del colisionador cúbico:
    - Dentro de la estructura, al vector P se le suma la mitad de las aristas del colisionador C para apuntar a una esquina. Este vector es multiplicado por la escala del colisionador y por la matriz m para que apunte a la dirección correcta.
    - Este nuevo vector es proyectado al plano de la cámara con la función worldToViewPoint, la cual toma como argumento el vector de entrada y la cámara cam.
- Se inicializan dos vectores de dos componentes a y b.
- Por cada vector i en M se utiliza la función a= Vector2.min(i,a) y b = Vector2.max(i,b) las cuales toman dos vectores y regresan un tercer vector compuesto por los elementos mínimos y máximos (respectivamente) de los vectores de entrada.
- Se regresa la estructura: a.x, b.y, b.x-a.x, b.y-a.y

Pseudocódigo 5 Algoritmo de obtención de datos de objeto en la imagen.

Esta estructura resultante es convertida a cadena de caracteres junto con la etiqueta del objeto y guardada en un arreglo de tipo *string* para su futuro uso.

Algoritmo para detección de obstáculos

No todos los objetos van a poder observarse en todas las capturas. Dada la naturaleza aleatoria de la generación de los objetos y la posición de la cámara, es probable que uno o más de ellos quede parcial o totalmente obstruido por otro objeto desde la perspectiva de la cámara.

Para poder hacer un buen entrenamiento de la red neuronal es recomendable no tener objetos parcialmente ocluidos para el entrenamiento. Y al no aparecer en la imagen, los objetos totalmente ocluidos no deben ser tomados en consideración para el entrenamiento.

Este algoritmo tiene como objetivo filtrar aquellos objetos que no sean visibles para la cámara. Tomando datos del ejemplo anterior, se utilizan los 8 puntos

correspondientes a las esquinas del colisionador cúbico del objeto junto con su centro.

Posteriormente, se hacen operaciones de raycasting de cada uno de estos puntos a la cámara. Si estos rayos encuentran un obstáculo en el camino, se aumenta un contador. En caso de que este contador supere una cierta cantidad de puntos (por ejemplo, la mitad), el algoritmo regresa un falso booleano.

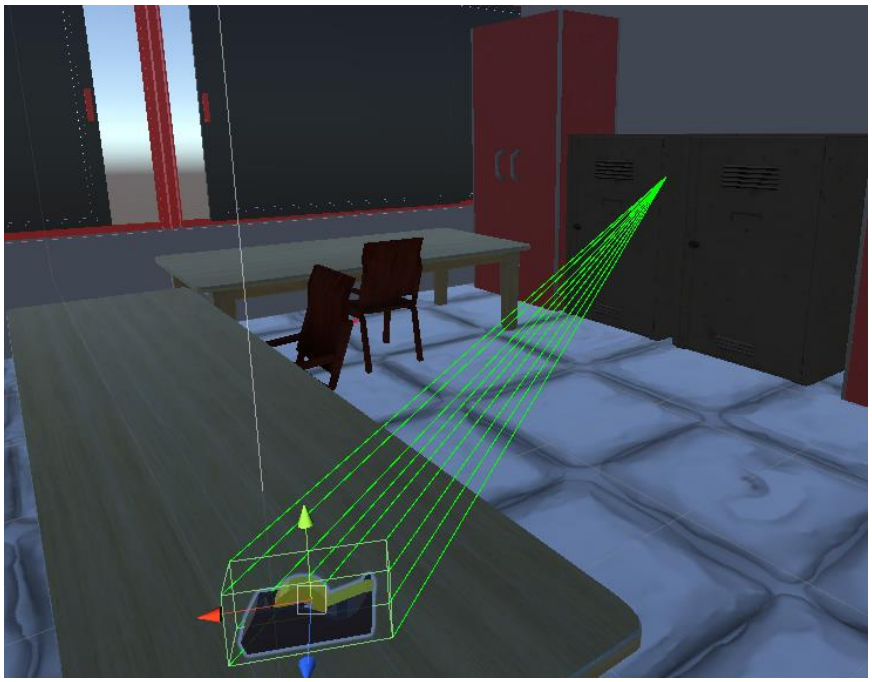


Figura 31 Ejemplo de raycasting para detectar oclusión.

Este algoritmo puede ser modificado para aceptar más o menos cantidad de esquinas ocluidas antes de ignorar el objeto. Por lo que es posible realizar pruebas sin objetos ocluidos o con objetos parcialmente ocluidos.

Algoritmo para la captura de imagen

Para poder obtener la captura de pantalla, es necesario trabajar con texturas. Una textura es simplemente un conjunto de pixeles que forman una imagen 2D. Por lo tanto, la captura de pantalla se obtiene mediante la creación de una textura con tres dimensiones (altura, anchura, canales).

El proceso de generación de esta textura es el siguiente:

- Se crea una textura de renderizado con el ancho y alto de la pantalla y 24 canales.
- Renderizar la imagen de la cámara en la textura creada.
- Crear una textura 2D con los mismos parámetros de dimensión que la primera textura, y con formato RGB24.
- Activar la textura de renderizado para la grabación de datos.
- Grabar los datos que ve la cámara en la textura 2D y aplicar estos cambios.
- Finalmente, codificar los datos de esta segunda textura en el formato requerido.
- Desactivar y eliminar la textura de renderizado.

El dato de salida de este proceso es un arreglo, el cual representa la imagen en el formato de imagen requerido.

Guardado de datos

Finalmente, los datos generados en los algoritmos anteriores se guardan en la carpeta correspondiente. En el caso de las imágenes, estas se guardan en formato PNG, mientras que los archivos de texto se guardan en texto plano.

Los archivos correspondientes tienen el mismo nombre que sigue la siguiente convención:

Ancho de imagen x alto de imagen_Var.extensión

En donde Var es una variable tipo entero (int) que incrementa cuando se termina de guardar los datos de una captura.

Dependiendo de la selección en la interfaz de usuario, los datos se guardan en la carpeta de entrenamiento o en la carpeta de validación.

Interfaz de usuario

Una aplicación tiene como propósito su utilización por usuarios distintos al desarrollador. En este caso, el usuario destino son los investigadores que requieran crear imágenes para entrenar redes neuronales.

Para que una aplicación pueda ser usada de forma efectiva, la funcionalidad de esta debe ser clara para el usuario. Por lo cual es necesario contar con una interfaz gráfica para controlar ciertos aspectos del entorno virtual, la generación de objetos y la captura de imágenes. En esta sección se describirá de forma breve el diseño de la interfaz de usuario de la aplicación.

Canvas

En Unity, los elementos de interfaz de usuario se crean en una capa particular denominada *Canvas*. Esta capa es independiente del resto del entorno virtual y se acomoda de forma automática a la pantalla. Loe elementos presentes en esta capa se acomodan con un sistema de coordenadas en dos dimensiones. Adicional a esto, estos elementos se pueden anclar a ciertas partes de la pantalla, permitiendo que mantengan sus posiciones relativas en distintas resoluciones.

La organización de la interfaz se realizó por categorías, un menú proporciona opciones para cambiar de forma manual las variables del entorno, el segundo controla los objetos que se van a generar y el tercero controla diversos aspectos de la captura de imágenes y la repetición del algoritmo.

Figura 32 Visión inicial de la interfaz de usuario

El menú colapsable con la etiqueta “objetos a instanciar” cuenta con 14 marcadores, cada uno de estos indica un objeto en particular que se puede generar. En el módulo de creación de listas, estos 14 marcadores forman parte de un arreglo que se utiliza para crear una lista.

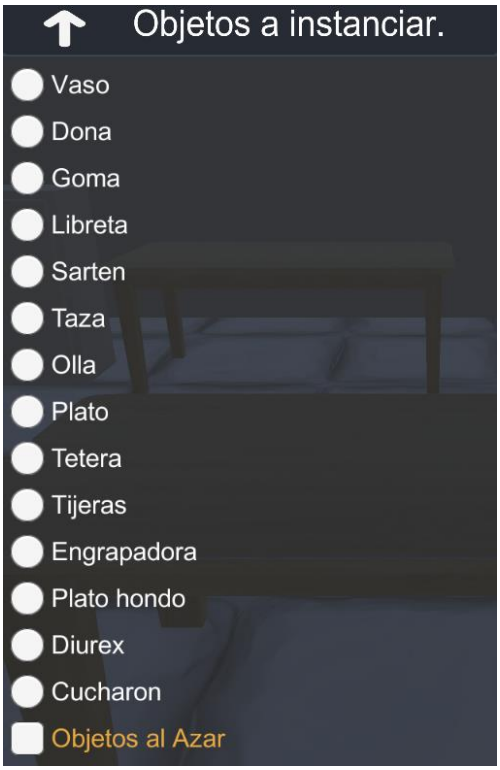


Figura 33 Menú de selección de objetos.

El segundo menú colapsable con la etiqueta “captura múltiple” cuenta con campos para modificar opciones utilizadas en el módulo de automatización del proceso.

El campo de número de capturas indica cuántas veces se repiten los módulos, el primer marcador es usado para guardar los datos capturados en la carpeta de datos de entrenamiento o en la carpeta de datos de validación (si no está marcada). El tercer marcador indica que la numeración de archivos continuará desde el número que se guardó en memoria, si este se deja en blanco, la numeración de las imágenes se reiniciará. Los otros elementos de este menú controlan la adición de obstáculos a los objetos instanciados y el reacomodo de muebles en la escena.

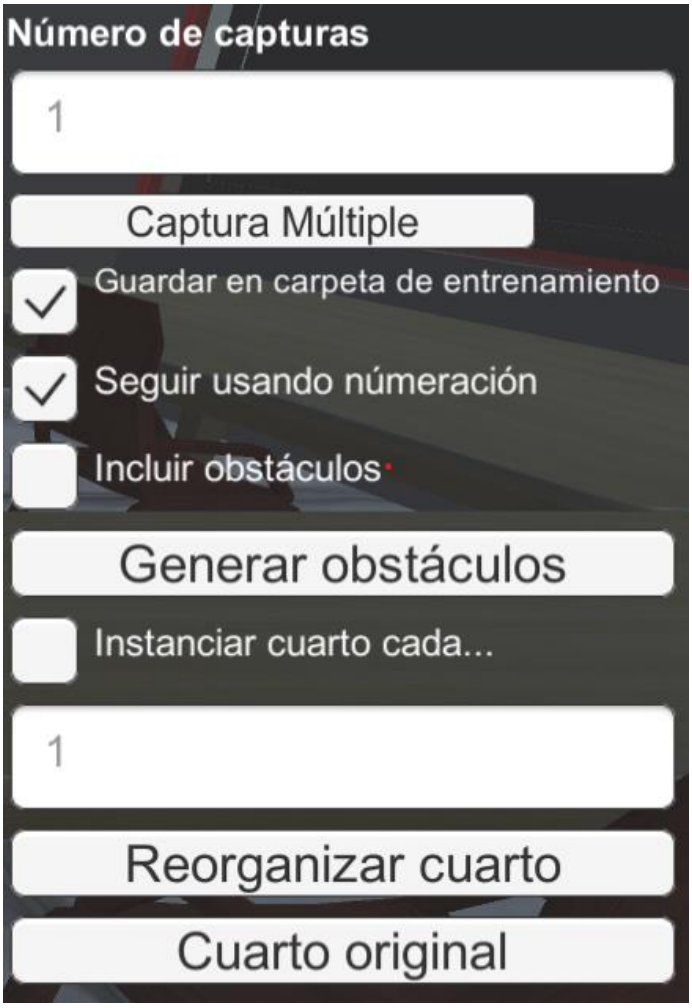


Figura 34 Menú con variables para el proceso automático.

Control de dirección

Otra parte importante del menú es la sección dedicada a cambiar la dirección de guardado de los datos recabados por el módulo de captura.

Figura 35 Opciones para cambiar la dirección de guardado.

El campo en este menú sirve para poder modificar la dirección de guardado de los datos. Si se escribe una dirección válida, la cadena de caracteres será modificada y todas las imágenes y documentos de texto provenientes del módulo de captura se guardarán en esta nueva dirección.

En caso de que no existan las carpetas requeridas en la dirección seleccionada, el código las genera de forma automática.

Control de entorno

El control de ciertos aspectos del entorno se puede hacer de forma manual o de forma automática mediante el módulo de automatización. En caso del segundo, los cambios de iluminación y de rotaciones de cámara se hacen al azar al inicio de cada iteración de código.

En caso del control manual, este se logra a partir de la manipulación de elementos de la interfaz de usuario tipo slider.

Cada slider se liga a una variable en particular. En la interfaz gráfica de Unity se puede modificar la variable ligada, así como los límites máximo y mínimo del componente slider.

Para la modificación del entorno se usaron las siguientes variables:

- Dos variables de intensidad de luz, una de luz general y la otra para una luz enfocada sobre la mesa en el centro del entorno.
- Dos elementos de rotación sobre un eje, los cuales controlan la posición de la cámara con respecto a la mesa de centro. La rotación se realiza sobre un objeto que se encuentra en el centro de la mesa y del cual la cámara es un objeto hijo.
- Un control de aumento, el cual afecta la variable “Profundidad de campo” de la cámara, simulando un efecto de aumento.

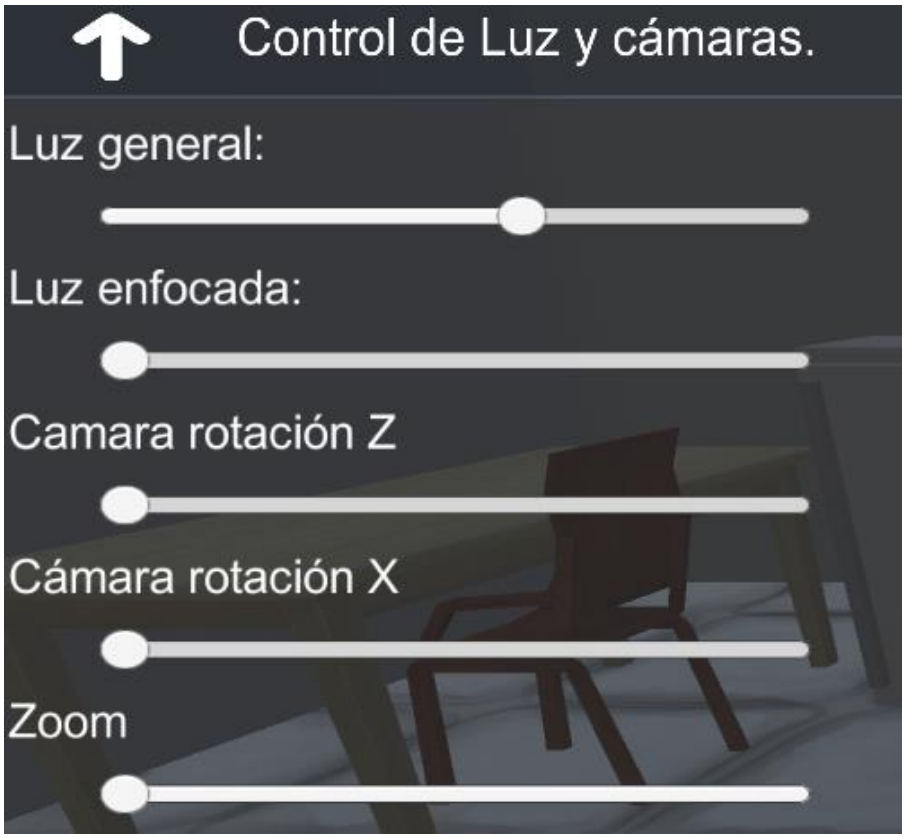


Figura 36 Opciones de control de entorno.

Figura 37 Ejemplo de una variación de la iluminación y el ángulo de visión.

Capítulo 4: Metodología de experimentación

Este capítulo explica los diversos experimentos realizados para poder verificar la efectividad del simulador creado. Esto incluye la generación de imágenes y el entrenamiento de la red neuronal con estas.

Características técnicas del entrenamiento:

Debido a problemas de compatibilidad del software, durante el tiempo de escritura de este trabajo, se trabajó en dos distintos sistemas operativos. La generación de imágenes se realizó en una cuenta de Windows 10, con la versión de Unity 2019-2, mientras que la instalación y entrenamiento de la red neuronal se realizó en Linux Ubuntu 18.

Ambas partes fueron desarrolladas en el mismo equipo computacional, cuyas características son desglosadas en la siguiente tabla.

Datos técnicos de la computadora usada en los experimentos	
Especificación	**Valor**
Modelo de computadora	ACER Predator Helios 300
Tarjeta gráfica	Nvidia GTX 1050-ti
Procesador	Intel Core i7-8750H
Número de núcleos del procesador	6
Velocidad del reloj	2.2GHz
Memoria RAM (de la tarjeta de video)	4GB

Tabla 6 Especificaciones del equipo de cómputo utilizado.

Preparación de la red neuronal

La red neuronal elegida para los experimentos fue el sistema YOLOV3, debido a que esta se encuentra en uso por el equipo de desarrollo del robot Takeshi. Previo a la instalación, en un ambiente virtual en la consola de Ubuntu, se instalaron los siguientes componentes:

Componente	**Versión**
CUDA	10.0
CMake	3.12 o superior
CUDNN	7.0 o superior
OpenCV	2.4 o superior

Tabla 7 Paquetes adicionales instalados.

Primeras pruebas

Para verificar que las imágenes generadas se pueden utilizar para el entrenamiento de la red YOLOV3, se realizó una primera prueba en la cual se entrenó con una serie de imágenes generadas en el simulador.

Estas imágenes se generaron de forma automática con todas las variables aleatorias activadas. También se marcó la opción de reconfigurar el cuarto cada 100 iteraciones del algoritmo.

En total se generaron 8000 imágenes conteniendo un total de 10 objetos en 4 categorías (taza, lata, plato hondo y sartén). La red entrenó con estas imágenes durante un período de 30 horas.

El resultado de esta prueba fue exitoso en comprobar la compatibilidad de las imágenes generadas, pero puso en evidencia la falta de escala adecuada entre distintos objetos, lo cual se corrigió previo a los siguientes experimentos.

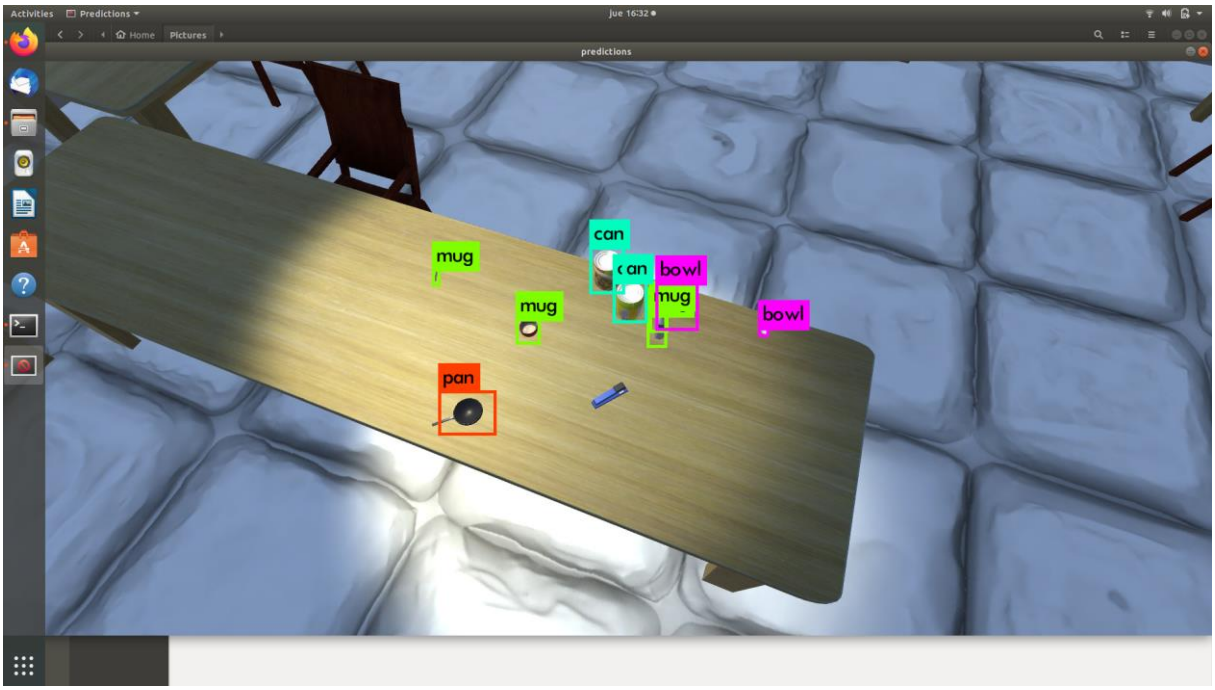


Figura 38 Resultado del entrenamiento de prueba.

Experimentación con conjuntos de datos

La meta del primer experimento es comparar la efectividad de distintas bases de datos digitales en la red neuronal YOLOV3. Para probar esto se entrenaron tres conjuntos de imágenes distintos, cada uno con el mismo número de imágenes. La diferencia principal entre los conjuntos es el tipo de información que se va a presentar en cada uno:

- El primer conjunto consta de imágenes de los objetos capturadas utilizando el algoritmo aleatorio. Ningún elemento adicional es agregado a estas imágenes más allá de los cambios de luz y de la posición de la cámara. Para este conjunto se ignoran los objetos parcialmente ocultos.
- El segundo conjunto es similar al primero con la diferencia de que también se va a recopilar información sobre los objetos parcialmente ocultos. Este cambio se logra modificando la cantidad de puntos que se tienen que contar para ignorar un objeto.
- El tercer conjunto agrega objetos adicionales como obstáculos que se generan al mismo tiempo que los primeros objetos. En este caso no se hará caso de los objetos parcialmente ocultos en la imagen.

Los objetos elegidos para este conjunto de experimentos son 30 objetos pertenecientes a 5 categorías, con el propósito de que los entrenamientos se puedan llevar a cabo de forma rápida.

Los resultados de estos entrenamientos se evaluaron con un conjunto de validación de 1000 imágenes. Este conjunto de validación constaba de imágenes incluyendo estos mismos objetos mezclados con un número mayor de objetos al azar.

Categoría	**Etiqueta**	**Cantidad de objetos**
Platos	0	10
Tazas	1	7
Platos hondos	2	7
Latas	3	3
Manzanas	4	3

Tabla 8 Objetos por categoría, primer experimento.

Generación de imágenes

Para los experimentos se generaron las siguientes imágenes:

- 5000 imágenes para cada conjunto de entrenamiento, todas ellas generadas de forma aleatoria utilizando la función de automatización de la herramienta.
- Dos conjuntos de validación con 500 y 200 imágenes respectivamente.

Capítulo 5: Pruebas y resultados

Tiempo de entrenamiento.

Los entrenamientos se realizaron de forma continua en el equipo mencionado en el capítulo anterior. Los tiempos de entrenamiento se muestran en la siguiente tabla:

Entrenamiento	Tiempo de entrenamiento
1	51 horas
2	49 horas
3	48 horas

Tabla 9 Tiempos de entrenamiento.

Generación de imágenes

Para los experimentos se generaron las siguientes imágenes:

Conjunto	**Generación manual/automática**	**Número de imágenes**
1	Automática	5000
2	Automática	5000
3	Automática	5000
Validación	Automática	500
Validación manual	Automática	200

Tabla 10 Imágenes generadas para los experimentos

Curvas de pérdida

El sistema YOLOV3 cuenta con funciones de graficación para la pérdida del entrenamiento. Estas gráficas se presentan en las figuras 39-41.

Se puede observar que independientemente del tipo de datos generados que se utilicen para entrenar, la curva de pérdida entre los tres entrenamientos no difiere por mucho. En los tres casos, la curva sufre de un rápido descenso durante las primeras mil iteraciones y posteriormente, el valor de pérdida se estabiliza alrededor de un valor de 2.

Otro aspecto importante que notar en estas gráficas es el tiempo de estabilización de cada curva de pérdida. Mientras que, en los primeros dos entrenamientos, la curva se estabiliza alrededor de 4000 iteraciones, la tercera gráfica se estabiliza hasta los 6000, indicando una mejora en el proceso de aprendizaje con el tercer grupo de datos.

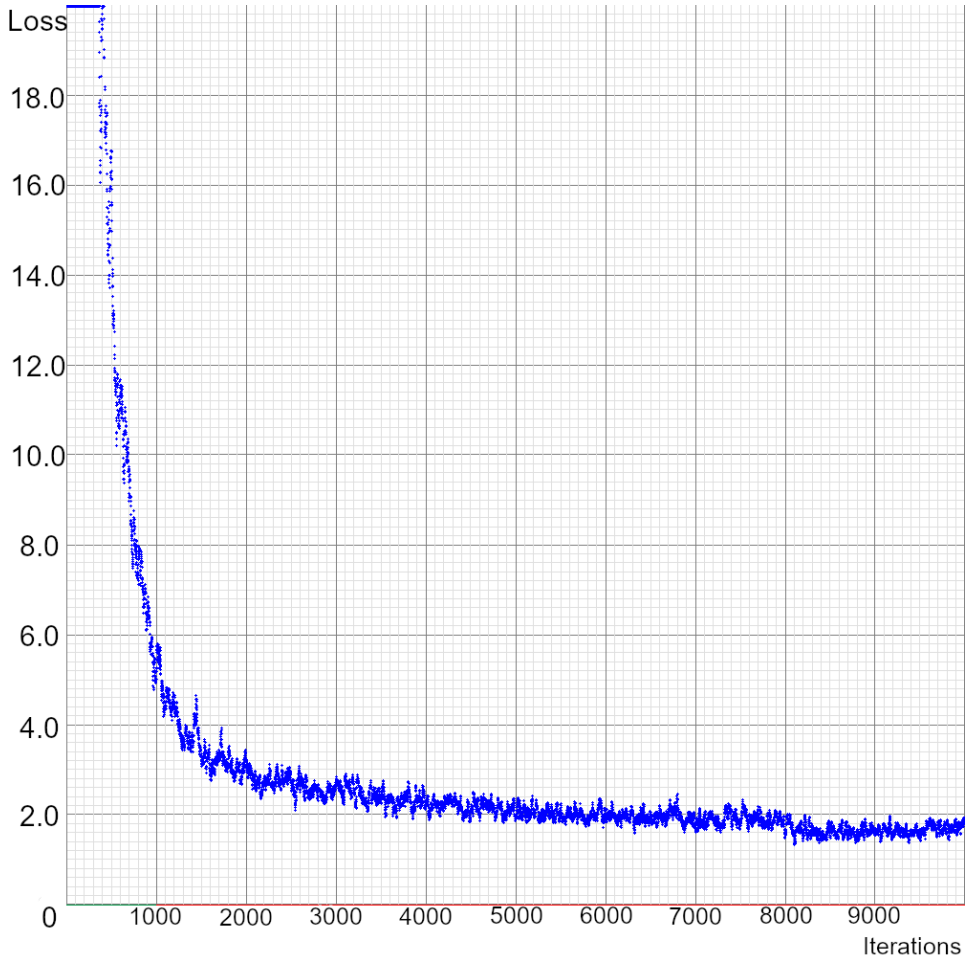


Figura 39 Gráfica de pérdida del primer entrenamiento

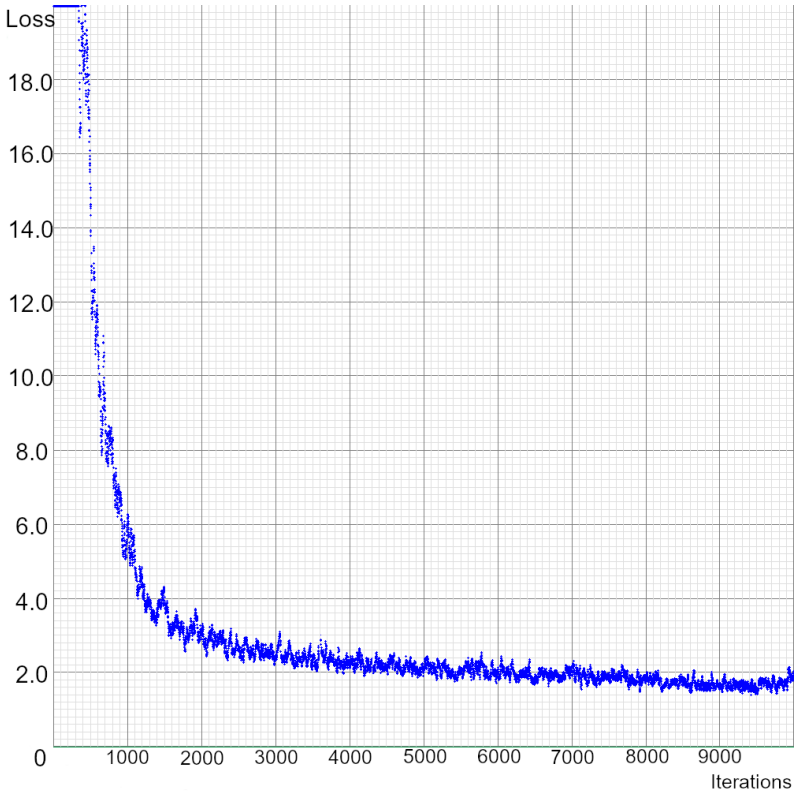


Figura 40 Gráfica de pérdida del segundo entrenamiento

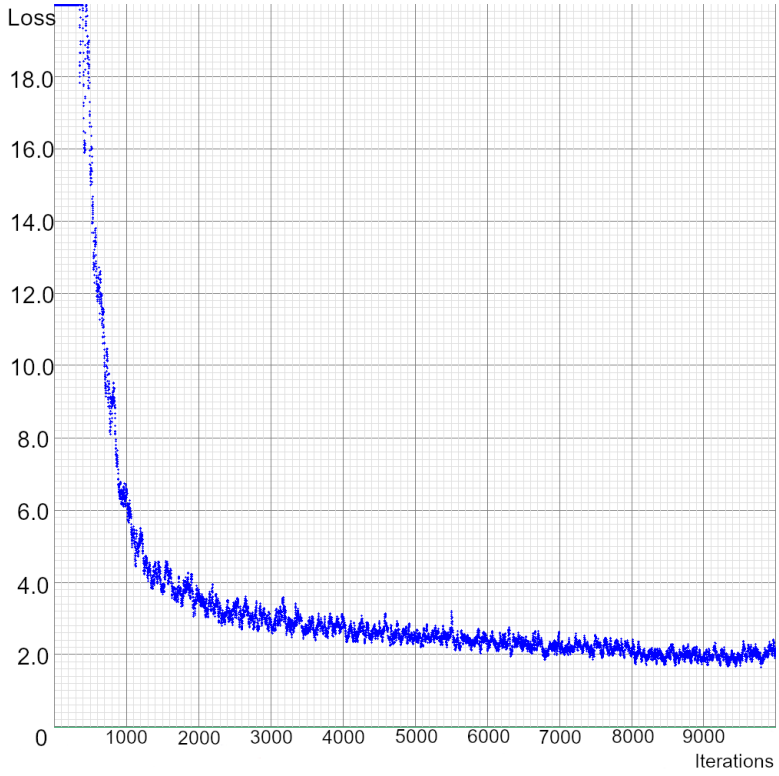


Figura 41 Gráfica de pérdida del tercer entrenamiento

Media de precisión promedio

Adicional al cálculo de la pérdida, la arquitectura YOLOV3 también cuenta con la capacidad de calcular de forma automática la media de precisión promedio (MAP, Mean Average Precision), la cual es una medida popular para medir el rendimiento de modelos de recuperación de información y detección de objetos. Esta se define como el promedio de las puntuaciones medias de precisión para cada consulta.

$$MAP = \frac{\sum_{q=1}^{Q} AveP(q)}{Q}$$

Ecuación 16 Cálculo de la media de precisión promedio

Esta medida se obtiene promediando el resultado de los cálculos de precisión de las categorías de detección de un entrenamiento en particular. En el caso del experimento realizado, las categorías corresponden a los 5 objetos que fueron entrenados. La precisión de cada uno se dio como el cálculo del promedio de IoU (Intersection over Union/Intersección sobre unión) de cada categoría en el conjunto de validación de 500 imágenes.

En cada entrenamiento se realizó un cálculo de MAP cada 1000 iteraciones. Los resultados se pueden observar en la figura 37.

Se puede observar en la imagen que, en el tercer entrenamiento, se obtuvo un mayor MAP que en el resto de los entrenamientos. La similitud entre las curvas del primer y el segundo entrenamiento indican que incluir objetos parcialmente ocultos en los datos de entrenamiento no afecta en gran medida los resultados de detección en la arquitectura, mientras que agregar objetos adicionales como obstáculos mejora la detección de los objetos entrenados. La precisión aumentó de un 87% a un 94% con la adición de obstáculos.

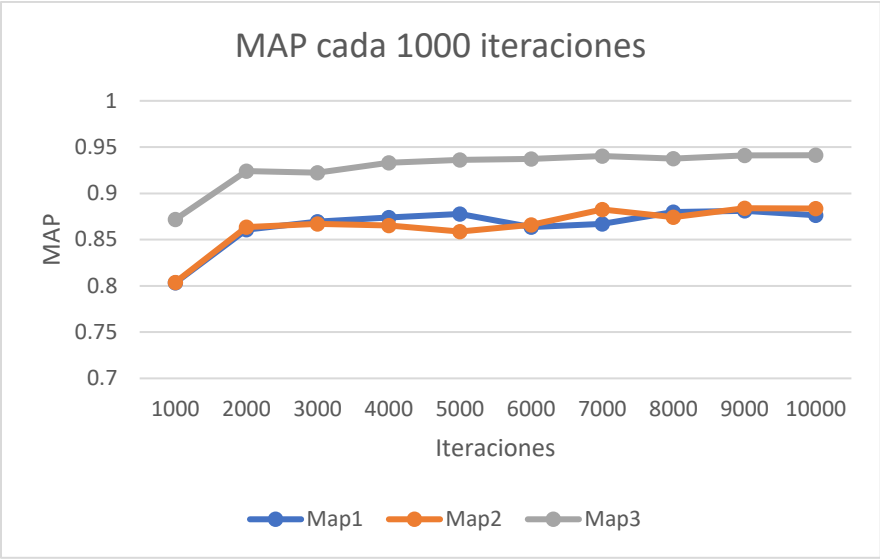


Figura 42 Gráfica de MAP de los tres entrenamientos realizados

Las gráficas correspondientes a las precisiones promedio de cada categoría se encuentran en el anexo al final del documento.

Conteo manual de resultados

En adición al cálculo de MAP automático que realiza el sistema, se realizó un conteo manual de objetos detectados en un conjunto de 200 imágenes independientes a los conjuntos previos. Estas imágenes también fueron generadas con la herramienta desarrollada, bajo las mismas condiciones que las imágenes utilizadas para el tercer entrenamiento.

Las imágenes fueron evaluadas de manera individual, utilizando los pesos finales de cada entrenamiento. Los resultados en las imágenes fueron clasificados de acuerdo con la siguiente regla:

- Verdadero positivo: Aquellos objetos que fueron correctamente identificados.
- Verdadero negativo: Objetos no clasificados que no se encontraban entre los objetos por clasificar.
- Falso positivo: Objetos que no fueron etiquetados durante el entrenamiento, los cuales fueron clasificados.
- Falso negativo: Objetos que tenían que ser clasificados que no fueron clasificados o se les asignó la etiqueta de otra categoría.

Los totales de estos datos fueron utilizados para calcular la precisión, la especificidad, la exactitud y la sensibilidad de la detección con los datos respectivos.

Los resultados resumidos de este conteo se pueden ver en las tablas 11, 12 y 13.

	Platos	**Tazas**	**Plato hondo**	**Lata**	**Manzana**	**Total**
Vp	256	172	183	81	79	771
Fn	11	3	4	0	2	20
Vn	-	-	-	-	-	140
Fp	502	943	163	168	130	1906
Precisión	0.3377	0.1542	0.5289	0.325	0.3779	0.2880
Sensibilidad	0.6464	0.5512	0.5665	0.3665	0.3607	0.8463
Especificidad	0.2180	0.1292	0.4620	0.4545	0.5185	0.068
Exactitud	0.4356	0.2480	0.6591	0.5681	0.6239	0.3211

Tabla 11 Resultados del conteo manual del primer entrenamiento.

	Platos	**Tazas**	**Plato hondo**	**Lata**	**Manzana**	**Total**
Vp	257	170	182	84	74	767
Fn	7	1	5	0	3	16
Vn	-	-	-	-	-	149
Fp	521	981	153	169	122	1946
Precisión	0.3303	0.1476	0.5432	0.3320	0.3775	0.2827
Sensibilidad	0.6330	0.5329	0.5498	0.3605	0.3318	0.8373
Especificidad	0.2223	0.1318	0.4933	0.4685	0.5498	0.0711
Exactitud	0.4346	0.2452	0.6768	0.5796	0.6408	0.3182

Tabla 12 Resultados del conteo manual del segundo entrenamiento.

	Platos	**Tazas**	**Plato hondo**	**Lata**	**Manzana**	**Total**
Vp	256	178	182	84	76	776
Fn	5	2	10	0	4	21
Vn	-	-	-	-	-	1280
Fp	48	225	75	29	36	443
Precisión	0.8421	0.4416	0.7081	0.7433	0.6785	0.6526

Sensibilidad	0.1666	0.1220	0.1244	0.0615	0.0560	0.3774
Especificidad	0.9638	0.8504	0.9446	0.9778	0.9726	0.7560
Exactitud	0.9666	0.8652	0.9450	0.9791	0.9713	0.8257

Tabla 13 Resultados del conteo manual del tercer entrenamiento.

Estos valores, no coinciden con la precisión calculada por el sistema de YOLOV3. Sin embargo, nos dan una idea aproximada de cómo se comporta el clasificador con los distintos grupos de datos. Al igual que las gráficas de MAP, se observa que los resultados del tercer entrenamiento son mucho más favorables, produciendo muchas menos identificaciones falsas. Esto se puede observar claramente al comparar una misma imagen bajo las tres clasificaciones (Fig43).

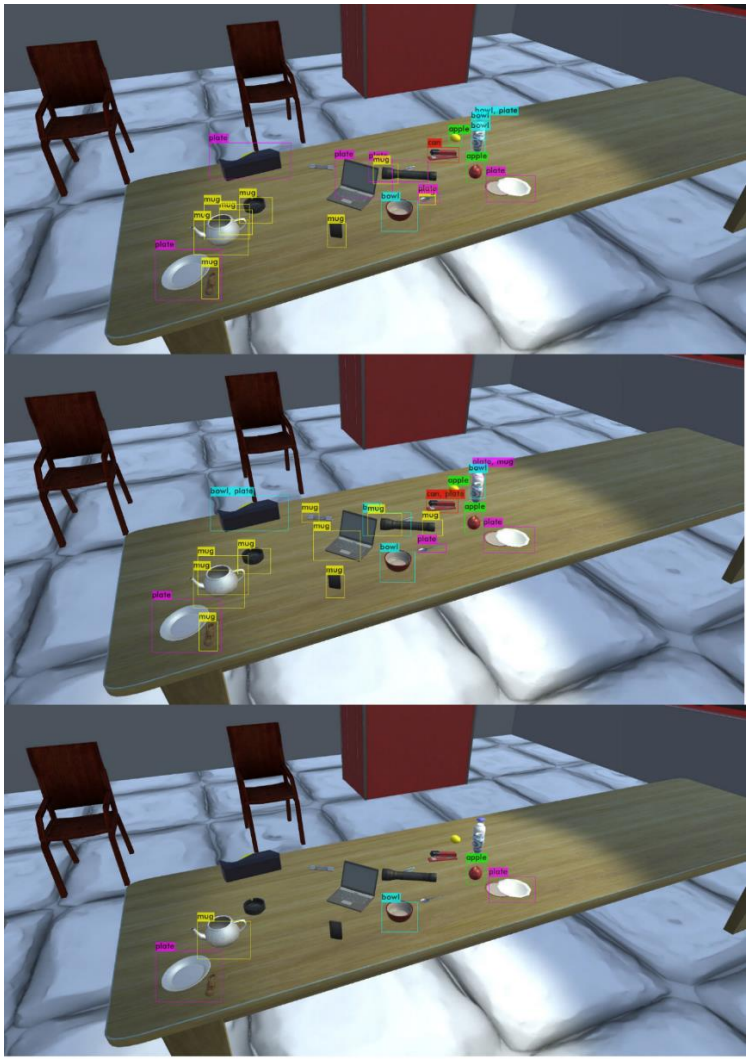


Figura 43 Comparación entre los tres entrenamientos sobre una misma imagen. De arriba hacia abajo, primer entrenamiento, segundo entrenamiento, tercer entrenamiento.

Experimentación con imágenes reales

Si bien, el objetivo principal del trabajo fue crear una herramienta capaz de entrenar redes neuronales, un objetivo ajeno al desarrollo inmediato del trabajo, pero relacionado con este es el de probar la red entrenada con imágenes reales. Para probar esto, se una serie de pequeñas pruebas para verificar el desempeño hipotético de la arquitectura en entornos reales.

Con los pesos del tercer entrenamiento se hicieron diversas pruebas para determinar la efectividad del reconocimiento en entornos reales. Se utilizaron una serie al azar de imágenes de platos, tazas y frutos para verificar la efectividad de la clasificación al igual que una serie de capturas de pantalla de la cámara del equipo de cómputo utilizado.

Los resultados de estas imágenes indican que los datos generados sólo son útiles para el entorno virtual en el cual fueron generados, ya que no se observaron resultados favorables en las imágenes utilizadas.

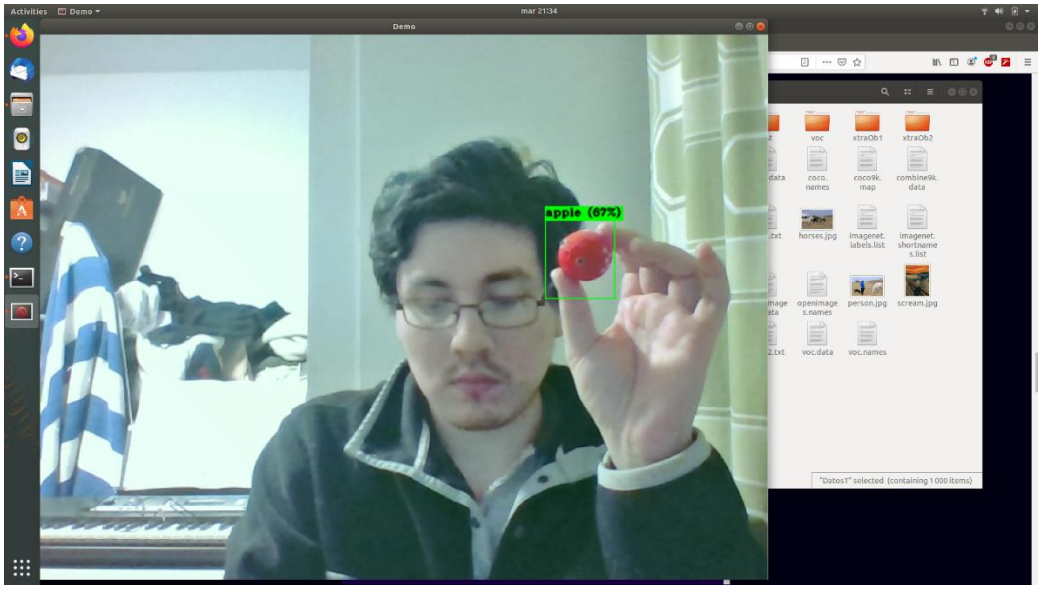


Figura 44 Ejemplo de reconocimiento fallido de un adorno como manzana

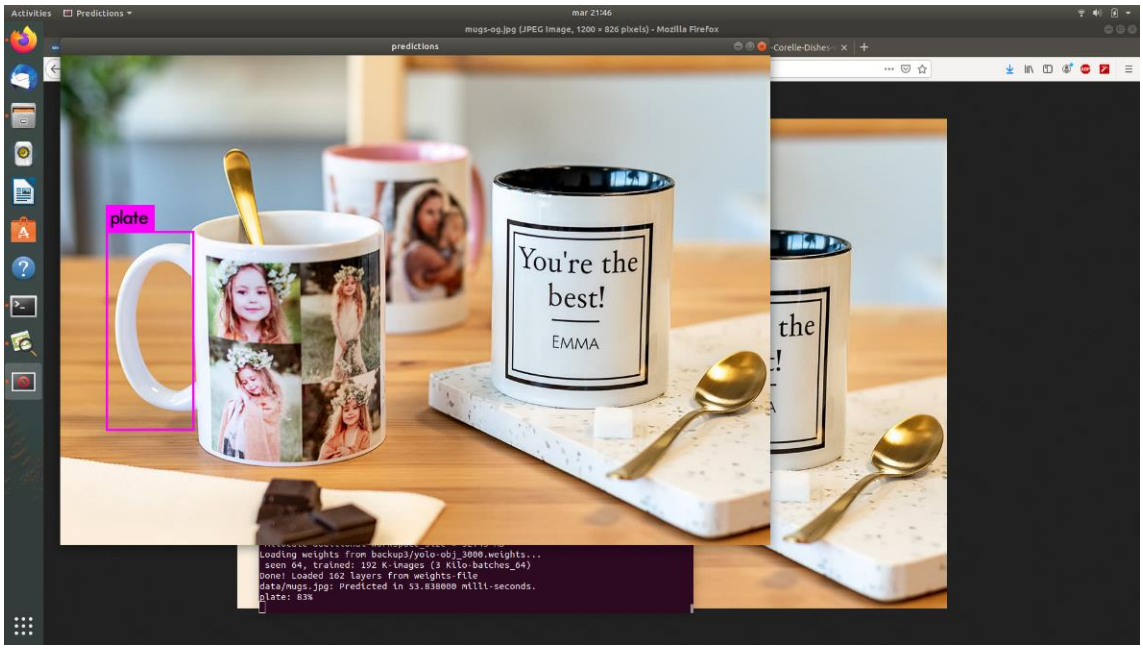


Figura 45 Ejemplo de reconocimiento fallido de una taza en imagen.

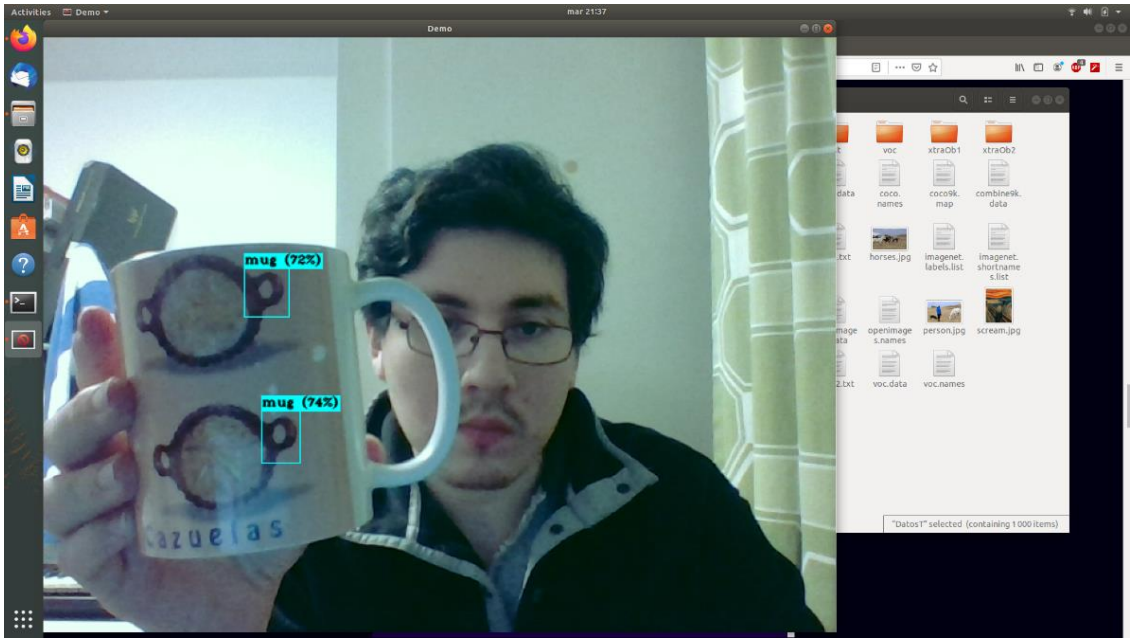


Figura 46 Ejemplo de reconocimiento fallido de una taza.

Para verificar si la escala y la calidad de las imágenes utilizadas para verificar era importante, se realizó un último experimento. Se tomaron capturas a distintas distancias de un grupo de tazas colocadas sobre una mesa. Este conjunto de imágenes fue modificado por una serie de filtros, generando imágenes con las siguientes modificaciones:

Conjunto	Modificación
1	Sin modificaciones
2	Operación de afilado de imagen.
3	Simplificación de texturas mediante K-Means

Tabla 14 Grupos modificados de fotografías.

Figura 47 Imagen con texturas simplificadas y mala clasificación. Indicando una necesidad de mejorar la detección en caso de contar con imágenes dañadas o con ruido.

Estas imágenes fueron procesadas con los mejores pesos del tercer entrenamiento. Los resultados de las imágenes en el ambiente de cocina fueron sometidas a un conteo manual para verificar la hipotética capacidad de la arquitectura para detectar objetos entrenados. En estas imágenes se mezclaron res tazas y una lata entre otros objetos típicamente encontrados en una cocina.

	Conjunto 1	**Conjunto 2**	**Conjunto 3**	**Total**
Vp	12	13	12	37
Fn	23	23	26	72
Vn	102	102	102	306
Fp	29	26	23	78
Precisión	0.2926	0.3333	0.3428	0.3217
Sensibilidad	0.3428	0.3611	0.3157	0.3394
Especificidad	0.7786	0.7968	0.816	0.7968
Exactitud	0.6867	0.7012	0.6993	0.6957

Tabla 15 Resultado de conteo manual de detección en conjunto pequeño de imágenes reales.

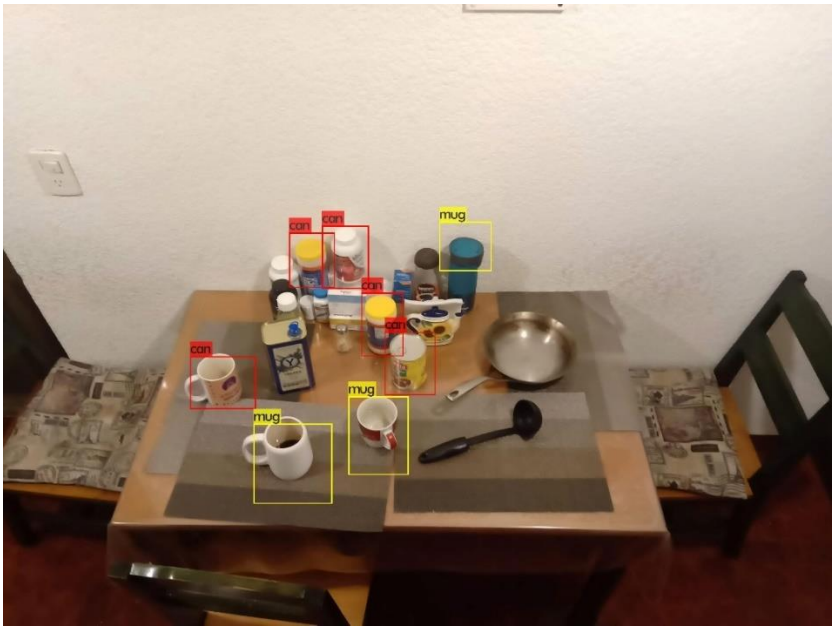


Figura 48 Ejemplo de detección en ambiente de cocina.

Se pudo observar que independientemente de la calidad de las imágenes y el efecto que se le otorgaba, el resultado de la clasificación era la misma. Los filtros utilizados

no cambiaban la imagen lo suficiente como para modificar en gran parte la clasificación final. Sin embargo, uno de los factores que podría afectar la clasificación es la distancia del objeto. En fotografías más alejadas, la arquitectura YOLOV3 fue capaz de reconocer dos de las tres tazas presentes en la imagen de forma correcta. Por lo que se puede suponer que los datos proporcionados por la herramienta virtual entrenaron al algoritmo a reconocer objetos a distancias alejadas y se requerirían nuevos datos a distancias más cercanas para mejorar el reconocimiento de objetos a distancias cortas.

Figura 49 Reconocimiento fallido de una taza a corta distancia.

Figura 50 Reconocimiento exitoso de dos tazas a distancia alejada.

Capítulo 6: Discusión y conclusiones

Para que un robot de servicio pueda interactuar con su entorno, es necesario que este cuente con información visual de la forma de este y de los agentes que se mueven dentro de este. En la mayoría de los casos, esta información visual es suplementada por algoritmos de reconocimiento de objetos. Los objetivos de este trabajo trataron sobre el desarrollo de una herramienta capaz de suplementar los datos introducidos para entrenar arquitecturas de detección de objetos.

Durante el presente el trabajo encontramos lo siguiente:

Expectativas:

La expectativa general del trabajo fue generar una herramienta desarrollada en el motor gráfico Unity, capaz de generar información que se pueda utilizar en el entrenamiento de redes neuronales para el reconocimiento de objetos.

Resultantes:

La herramienta en sí cumple con este primer objetivo. El usuario es capaz de interactuar de forma limitada con el ambiente virtual y puede generar información visual y descriptiva de los objetos que se encuentran diseminados en la zona de trabajo del ambiente virtual.

El ambiente en sí tiene la suficiente calidad grafica para poder diferenciar los distintos objetos que se presentan en esta. Gracias al motor de física de Unity se puede generar una implementación realista de los objetos simulando un ambiente real, lo cual contribuye con la variación de la disposición de objetos y por consiguiente de los datos de entrenamiento.

Los datos generados por la herramienta son compatibles en su totalidad con el sistema YOLOV3, el cual fue utilizado debido a su uso en el desarrollo de software para el robot Takeshi. Con los datos generados fue posible entrenar el sistema en 3 ocasiones distintas, generando resultados con distintos desempeños.

Para los entrenamientos realizados se generaron un total de 15,700 imágenes, todas con objetos generados al azar en el entorno y con cambios al azar de iluminación y posición de cámara. Junto con cada imagen se generó un archivo de texto correspondiente, el cual listaba las coordenadas de cada objeto que poseía una etiqueta para entrenamiento. El entrenamiento se dio sin problemas de compatibilidad, mostrando que los datos generados contaban con el formato y los datos correctos.

Los resultados de los entrenamientos indicaron que la adición de objetos parcialmente ocultos a los datos de entrenamiento no afectaba en gran medida a la precisión de la detección, indicando una calidad similar en los datos. Sin embargo, el agregar obstáculos en el entorno virtual mejoraba de forma drástica la capacidad de detección del algoritmo. A futuro, los datos generados contarán con este tipo de obstáculos de forma más frecuente.

Es importante recalcar que estos datos fueron generados para su uso en un solo sistema de redes neuronales. Por lo que su uso es limitado a sistemas que utilicen la misma disposición de datos (etiqueta, coordenadas de la caja, ancho y alto). Es necesario pensar en generar información apta para otros sistemas en futuras iteraciones del trabajo.

Fallos y limitantes:

A pesar del éxito que se tuvo en adecuar información generada al sistema YOLO3, los resultados que otorgaron los entrenamientos fueron poco satisfactorios. Las gráficas de pérdida del algoritmo mostraron cambios poco significativos entre los tras entrenamientos, lo cual puede indicar que la calidad de los datos no es lo suficiente para generar una pérdida menor. También es posible que la repetitividad de los mismos datos genere un sobreajuste del sistema, limitando el reconocimiento a objetos similares a los que se utilizaron para el entrenamiento.

Las gráficas midiendo la media de precisión promedio durante los entrenamientos muestran un desempeño relativamente pobre. Mientras que el tercer entrenamiento

muestra una pequeña mejoría al incluir obstáculos, esta mejora no es lo suficiente como para considerarse significativa.

Algunas de las posibles causas para este pobre desempeño son las siguientes:

- Conjunto limitado de objetos de entrenamiento: El conjunto de entrenamiento que se utilizó para crear la información consistió en un conjunto de 30 objetos etiquetados y alrededor de 100 objetos sin etiquetar que sirvieron como obstáculos. Al crear más de 15,000 imágenes con un conjunto limitado de datos, es muy probable que exista un sobreajuste de datos para los objetos que más se asemejen al conjunto reducido de objetos que se utilizaron para el entrenamiento.
- Preferencia a ciertos objetos durante el entrenamiento: El conjunto más numeroso de objetos utilizado durante el entrenamiento fue el de platos, contando con 10 objetos distintos en comparación con otras categorías como platos hondos o manzanas (7 y 3 objetos, respectivamente). Esta mayor variabilidad de objetos de entrenamiento pudo contribuir a una mejor detección de las categorías que poseían más objetos. También es importante notar que la poca cantidad de objetos en las categorías muy probablemente llevo a un sub-ajuste de la arquitectura. Es importante ofrecer una distribución más equivalente de objetos por categoría para evitar una preferencia por parte del sistema.
- Condiciones gráficas insuficientes: Siendo este un primer acercamiento a la creación de la herramienta, la calidad gráfica del ambiente y los objetos que se encuentran en este no fue un factor prioritario en su desarrollo. Sin embargo, es posible que la baja calidad gráfica y la discrepancia entre la calidad de modelos y texturas de distintos objetos hayan contribuido a fallas en la detección de objetos. Es importante trabajar para estandarizar la calidad gráfica de la herramienta con el fin de proporcionar información del mismo nivel de calidad.
- Problemas de iluminación: La iluminación es un factor importante para la detección de imágenes y una de las variables que se pueden modificar en la

herramienta. Sin embargo, las funciones de luz pueden resultar insuficientes ya que sólo se puede modificar la intensidad de dos fuentes de luz en el ambiente. Esto limita la cantidad de variables en cuestión de luz y sombras que podría presentar la imagen. La baja calidad gráfica en el ambiente también contribuye a un mal manejo de la luz ya que se pierden muchos detalles como reflexiones de luz.

- Distancias de captura de imágenes limitadas: La distancia de la cámara a la zona de trabajo central se encuentra restringida por la distancia establecida del objeto al cual se encuentra anclado en el ambiente virtual. Si bien es posible modificar esta distancia y las coordenadas del objeto, hacerlo requiere varios pasos en el editor de Unity, por lo que no es una opción fácil de usar. La distancia y los ángulos limitados restringen la variabilidad de información que es posible capturar y limitan a el sistema a trabajar con objetos a ciertas escalas. Lo cual se ve reflejado en los resultados de detección.

Todo esto también se ve reflejado en los resultados de aplicar el sistema en imágenes reales. Cabe recalcar que estos, durante este trabajo, son tratados como ejemplos de metas a futuro debido al pobre desempeño que se tuvo en datos sintéticos. Es necesario mejorar el sistema y ampliar la cantidad y calidad de datos usados para el entrenamiento antes de pasar a detección en entornos reales.

Posibles mejoras:

Es necesario definir mejoras al sistema para superar estas limitaciones de la herramienta y poder generar mejor información. En primera instancia, es necesario ampliar la cantidad y variabilidad de objetos que son utilizados para el entrenamiento de la red neuronal ya que la cantidad limitada des estos lleva a resultados pobres en la detección. También es importante conseguir un equilibrio de objetos en distintas categorías para evitar limitar las características que pueda aprender el sistema. Junto con la adición de objetos es necesario mejorar la calidad gráfica de los mismos y del entorno. También es necesario diversificar la distancia

con la cual se toman capturas de imágenes en el algoritmo para poder mejorar el reconocimiento de objetos.

Otras mejoras que deben ser consideradas es la adición de ruido a las imágenes. Esto se puede lograr mediante la manipulación de las imágenes después de su captura y no requiere modificar los datos recopilados. Estos efectos pueden variar desde la adición de filtros gaussianos para emborronar la imagen y agregar ruido granular a las imágenes hasta efectos de cámara como fallas de lente o efectos de sobre exposición.

También es necesario mejorar la interactividad del entorno. De momento, la herramienta es muy limitada en términos del espacio en donde se pueden localizar objetos, por lo que es imperativo ampliar el espacio de interactividad y agregar más ambientes a la herramienta para elevar la variedad visual del entorno. De igual manera se requiere de un mayor control de la cámara en el modo no aleatorio para mejorar la toma de imágenes.

Conclusiones:

En conclusión. Si bien, el objetivo principal del trabajo se cumplió (la primera implementación de la herramienta es funcional y cumple con la función principal para la cual fue creada), no se puede ignorar el desempeño que se tuvo en los experimentos con la información generada por esta herramienta. Es necesario mejorar varios aspectos de esta e introducir una mayor cantidad de objetos de mejor calidad para mejorar la efectividad de la herramienta y mejorar a futuro la detección de objetos en imágenes reales. Este desarrollo es posible y se espera que, en el futuro cercano, esta herramienta permita que grupos de investigación que utilicen sistemas de detección de imágenes puedan generar información propia para futuros proyectos.

Capítulo 7: Trabajo a futuro

Como trabajo futuro para el mediano y largo plazo, se tienen las siguientes propuestas:

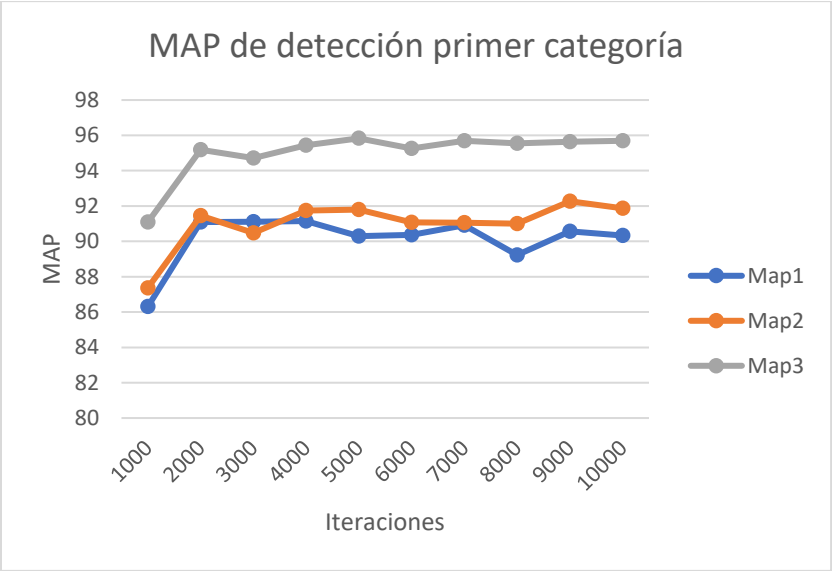
- Mejorar la calidad gráfica del entorno de la herramienta.
- Mejorar la calidad gráfica de los modelos utilizados en la herramienta.
- Estandarizar los modelos utilizados para poder modificar de forma sencilla propiedades como texturas.
- Agregar efectos de cámara y/o ruido a las imágenes generadas.
- Incrementar el nivel de interactividad permitido en la herramienta.
- Mejorar la interfaz de usuario para permitir la adición de más funciones y mejorar la experiencia de usuario.
- Agregar opciones para poder generar datos compatibles con distintas bases de datos.
- Realizar entrenamientos con grupos mixtos de datos (imágenes generadas e imágenes reales) para verificar la efectividad de estos.
- Realizar experimentos con la capacidad de modificar más elementos de los objetos como las texturas, para verificar que tanto afecta a la detección.
- Migrar la herramienta a Unity 2020, lo cual permitiría tener más opciones de iluminación, control de efectos de cámara y modularidad de código.

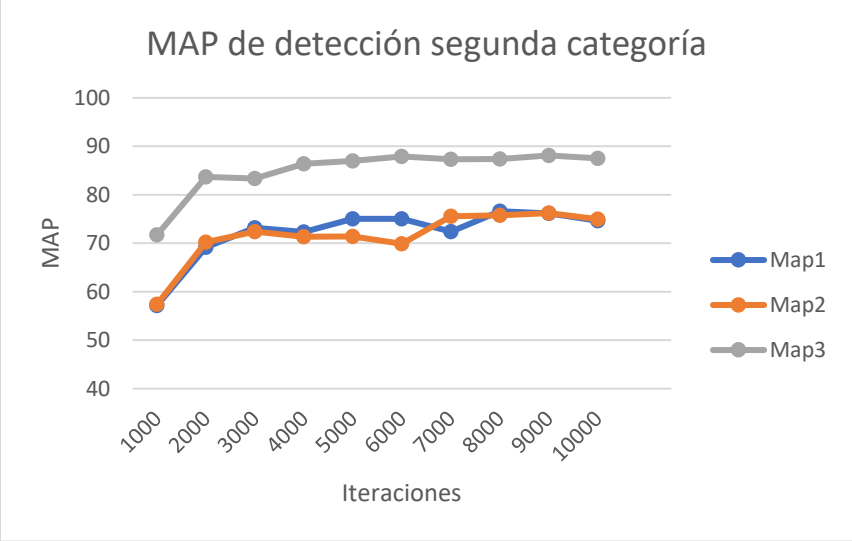
Anexo

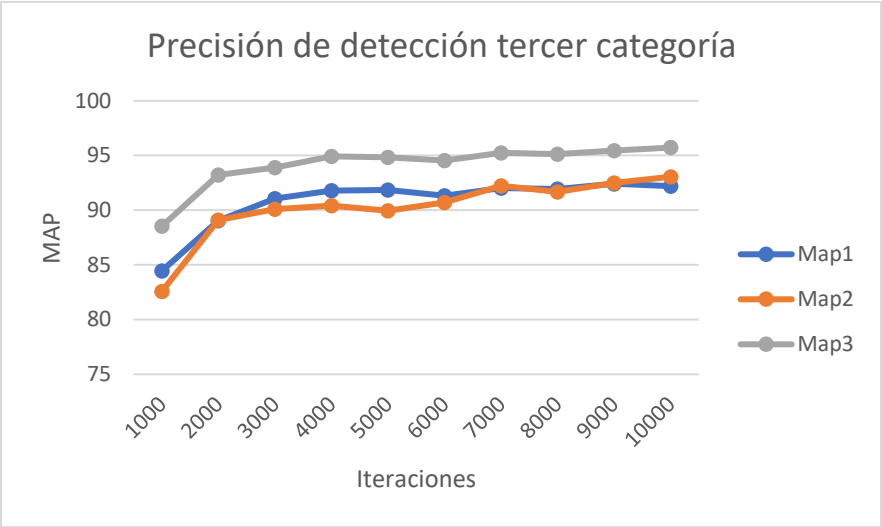
Gráficas de media de precisión promedio en detección por categoría:

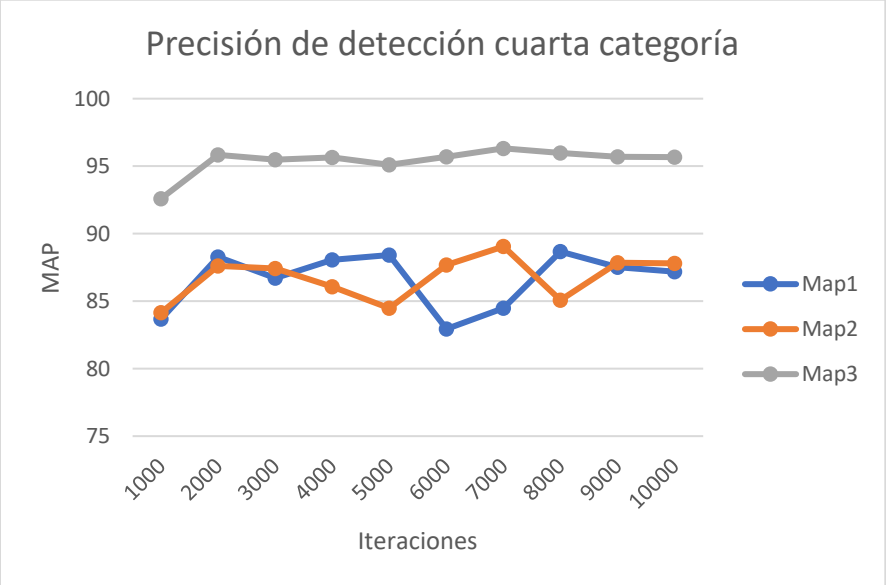
Estas gráficas representan la medición de la media de precisión promedio calculada para cada categoría de entrenamiento. Esto corresponde al conjunto de validación de 500 imágenes sintéticas con el cual se obtuvieron los valores de forma automática durante el proceso de verificación. El eje x corresponde a las iteraciones transcurridas y el eje y corresponde al porcentaje de precisión. Las categorías son las siguientes:

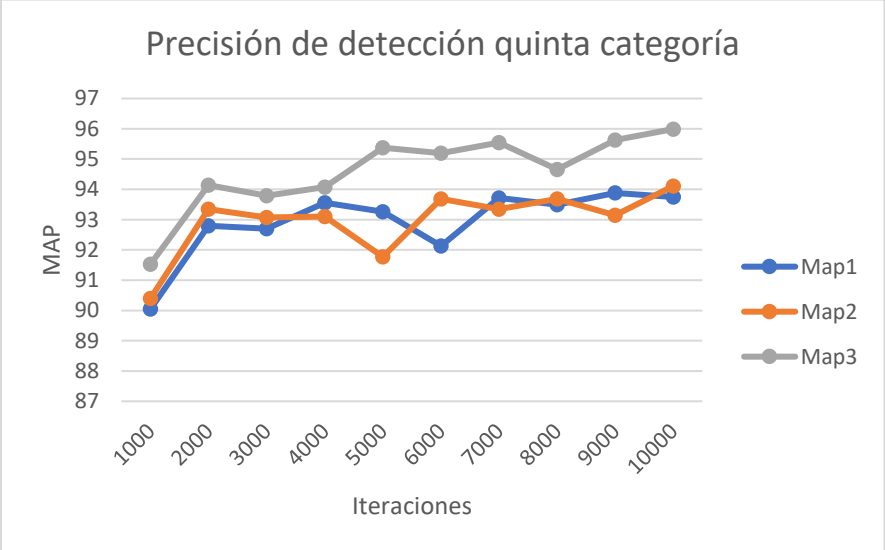
1. Reconocimiento de platos.
2. Reconocimiento de tazas.
3. Reconocimiento de platos hondos.
4. Reconocimiento de latas.
5. Reconocimiento de manzanas.






Precisión de detección tercer categoría
100
95
90
85
80
75
MAP
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
Iteraciones
Map1
Map2
Map3


Precisión de detección cuarta categoría
100
95
90
85
80
75
MAP
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
Iteraciones
Map1
Map2
Map3


Precisión de detección quinta categoría
97
96
95
94
93
92
91
90
89
88
87
MAP
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
Iteraciones
Map1
Map2
Map3

Bibliografía

[1] rethink robotics, “Baxter Product Datasheet,” 2013. [Online]. Available: http://www.rethinkrobotics.com/wp-content/uploads/2015/11/Baxter_Datasheet_Oct2015.pdf.

[2] rethink robotics, “Rethink Robotics ® meets German Engineering The new Sawyer BLACK Edition,” 2018. [Online]. Available: https://www.rethinkrobotics.com/.

[3] P. Tsarouchi, A.-S. Matthaiakis, S. Makris, and G. Chryssolouris, “On a human-robot collaboration in an assembly cell,” *Int. J. Comput. Integr. Manuf.*, vol. 30, no. 6, pp. 580–589, 2017.

[4] L. Cahrlotte, “How Much Has Amazon Invested in Automation? - Amazon Picking Challenge,” 2019. [Online]. Available: http://amazonpickingchallenge.org/how-much-has-amazon-invested-in-automation/. [Accessed: 07-Jul-2020].

[5] “Coronavirus: un robot verifica temperatura y uso de tapabocas en pacientes.” [Online]. Available: https://www.semana.com/mundo/articulo/coronavirus-un-robot-verifica-temperatura-y-uso-de-tapabocas-en-pacientes/675171. [Accessed: 08-Jul-2020].

[6] “Un perro robot distribuye gel para las manos en un centro comercial en Bangkok.” [Online]. Available: https://www.semana.com/tecnologia/articulo/un-perro-robot-distribuye-gel-para-las-manos-en-un-centro-comercial-en-bangkok/676128. [Accessed: 08-Jul-2020].

[7] B. Casey, “Remote-control VR robots to start working in Japanese convenience stores this summer - Japan Today,” 2020. [Online]. Available: https://japantoday.com/category/tech/remote-control-vr-robots-to-start-working-in-japanese-convenience-stores-this-summer?comment-order=oldest. [Accessed: 08-Jul-2020].

[8] N. Mayer *et al.*, “What Makes Good Synthetic Training Data for Learning Disparity and Optical Flow Estimation?,” *Int. J. Comput. Vis.*, vol. 126, no. 9, pp. 942–960, 2018.

[9] A. Owen-Hill, “Top 10 Challenges,” *IBM Developer*. [Online]. Available:

https://blog.robotiq.com/top-10-challenges-for-robot-vision. [Accessed: 04-Dec-2020].

[10] A. J. Soroka, R. Qiu, A. Noyvirt, and Z. Ji, “Challenges for service robots operating in non-industrial environments,” *IEEE Int. Conf. Ind. Informatics*, pp. 1152–1157, 2012.

[11] “Definition of data - data, computing and information.” [Online]. Available: https://web.archive.org/web/20121006073603/http://oxforddictionaries.com/definition/american_english/data. [Accessed: 02-Aug-2020].

[12] “Data and Statistics - Data Module #1: What is Research Data? - All Guides at Macalester College.” [Online]. Available: https://libguides.macalester.edu/c.php?g=527786&p=3608657. [Accessed: 02-Aug-2020].

[13] D. Soriano-Valdez, I. Pelaez-Ballestas, A. Manrique de Lara, and A. Gastelum-Strozzi, “The basics of data, big data, and machine learning in clinical practice,” *Clin. Rheumatol.*, 2020.

[14] Y. LeCun, C. Cortes, and C. J. C. Burges, “MNIST handwritten digit database, Yann LeCun, Corinna Cortes and Chris Burges,” 1998. [Online]. Available: http://yann.lecun.com/exdb/mnist/. [Accessed: 02-Aug-2020].

[15] C. R. Qi, H. Su, K. Mo, and L. J. Guibas, “PointNet: Deep learning on point sets for 3D classification and segmentation,” *Proc. - 30th IEEE Conf. Comput. Vis. Pattern Recognition, CVPR 2017*, vol. 2017-Janua, pp. 77–85, 2017.

[16] “ShapeNet.” [Online]. Available: https://www.shapenet.org/. [Accessed: 02-Aug-2020].

[17] “ImageNet.” [Online]. Available: http://www.image-net.org/. [Accessed: 02-Aug-2020].

[18] “¿Qué es Machine Learning? - México | IBM.” [Online]. Available: https://www.ibm.com/mx-es/analytics/machine-learning?p1=Search&p4=43700052827912798&p5=e&cm_mmc=Search_Google-_-1S_1S-_-LA_MX-_-reinforcement learning_e&cm_mmca7=71700000065289008&cm_mmca8=kwd-342837665&cm_mmca9=Cj0KCQjwyJn5BRDrARIsADZ9ykHMkS8Qi9f-gXekSYEPEtUJrirMRmfomngMj5YGRtw1qU25W6_vlIIaAhtlEALw_wcB&cm_mmca10=451233588239&cm_mmca11=e&gclid=Cj0KCQjwyJn5BRDrARIsADZ9ykHMkS8Qi9f-gXekSYEPEtUJrirMRmfomngMj5YGRtw1qU25W6_vlIIaAhtlEALw_wcB&gclsrc=aw.ds. [Accessed: 02-Aug-2020].

[19] R. E. Neapolitan and R. E. Neapolitan, *Neural Networks and Deep Learning*. New York, New York, USA: Springer US, 2018.

[20] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-Based Learning Applied to Document Recognition,” *Proc. IEEE*, 1998.

[21] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet Classification with Deep Convolutional Neural Networks,” *Adv. Neural Inf. Process. Syst.*, 2012.

[22] C. Szegedy *et al.*, “Going deeper with convolutions,” in *IEEE Conference on Computer vision and Pattern recognition*, 2015.

[23] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016, vol. 2016-Decem, pp. 770–778.

[24] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You Only Look Once: Unified, Real-Time Object Detection,” 2015. [Online]. Available: http://pjreddie.com/yolo/.

[25] J. Redmon and A. Farhadi, “YOLOv3: An Incremental Improvement,” 2018. [Online]. Available: http://arxiv.org/abs/1804.02767.

[26] J. Redmon and A. Farhadi, “YOLO9000: Better, faster, stronger,” 2018. [Online]. Available: http://pjreddie.com/yolo9000/%0AAbstract.

[27] S. C. Wong, A. Gatt, V. Stamatescu, and M. D. McDonnell, “Understanding Data Augmentation for Classification: When to Warp?,” *2016 Int. Conf. Digit. Image Comput. Tech. Appl. DICTA 2016*, 2016.

[28] “Data Augmentation Increases Accuracy of your model — But how ? | by sourav kumar | Secure and Private AI Writing Challenge | Medium.” [Online]. Available: https://medium.com/secure-and-private-ai-writing-challenge/data-augmentation-increases-accuracy-of-your-model-but-how-aa1913468722. [Accessed: 01-Aug-2020].

[29] M. S. Mueller, A. Metzger, and B. Jutzi, “CNN-BASED INITIAL LOCALIZATION IMPROVED by DATA AUGMENTATION,” *ISPRS Ann. Photogramm. Remote Sens. Spat. Inf. Sci.*, vol. 4, no. 1, pp. 117–124, 2018.

[30] H. Abu Alhaija, S. K. Mustikovela, L. Mescheder, A. Geiger, and C. Rother, “Augmented

Reality Meets Computer Vision: Efficient Data Generation for Urban Driving Scenes," *Int. J. Comput. Vis.*, vol. 126, no. 9, pp. 961–972, 2018.

[31] J. Talukdar, A. Biswas, and S. Gupta, "Data Augmentation on Synthetic Images for Transfer Learning using Deep CNNs," *2018 5th Int. Conf. Signal Process. Integr. Networks, SPIN 2018*, pp. 215–219, 2018.

[32] C. Jiang *et al.*, "Configurable 3D Scene Synthesis and 2D Image Rendering with Per-pixel Ground Truth Using Stochastic Grammars," *Int. J. Comput. Vis.*, vol. 126, no. 9, pp. 920–941, 2018.

[33] J. Tremblay, T. To, and S. Birchfield, "Falling things: A synthetic dataset for 3D object detection and pose estimation," *IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit. Work.*, vol. 2018-June, pp. 2119–2122, 2018.

[34] B. Calli *et al.*, "Yale-CMU-Berkeley dataset for robotic manipulation research," *Int. J. Rob. Res.*, vol. 36, no. 3, pp. 261–268, Mar. 2017.

[35] "Virtual World Design - Ann Latham Cudworth - Google Libros." [Online]. Available: https://books.google.com.mx/books?id=BozNBQAAQBAJ&pg=PA2&lpg=PA2&dq=first+virtual+environments+1950&source=bl&ots=QgAboGlk8f&sig=ACfU3U0q805-pl6GNxO5XGQMtZMkU20-0Q&hl=es-419&sa=X&ved=2ahUKEwjUqsaUr4_qAhUG7qwKHeX4DrUQ6AEwCnoECAoQAQ#v=onepage&q&f=false. [Accessed: 19-Jun-2020].

[36] "Historia – Realidad Virtual." [Online]. Available: https://realidavmultimedia.wordpress.com/historia-4/. [Accessed: 20-Jun-2020].

[37] Henry E. Lowood, "Virtual reality | computer science | Britannica.com," *Encyclopædia Britannica*, 2017. [Online]. Available: https://www.britannica.com/technology/virtual-reality. [Accessed: 21-Jun-2020].

[38] W. L. Hosch, "Augmented reality | computer science | Britannica," *Encyclopædia Britannica*, 2017. [Online]. Available: https://www.britannica.com/technology/augmented-reality. [Accessed: 21-Jun-2020].

[39] "Microsoft HoloLens | Mixed Reality Technology for Business." [Online]. Available: https://www.microsoft.com/en-us/hololens?SilentAuth=1&f=255&MSPPError=-

2147217396. [Accessed: 14-Jul-2020].

[40] Unity, “Unity - Game Engine,” 2020. [Online]. Available: https://unity3d.com/es. [Accessed: 07-Jul-2020].

[41] Epic Games, “Unreal Engine: The most powerful real-time 3D creation platform,” 2020. [Online]. Available: https://www.unrealengine.com/en-US/. [Accessed: 12-Jun-2020].

[42] J. Linietsky and A. Manzur, “Godot Engine - Free and open source 2D and 3D game engine,” 2020. [Online]. Available: https://godotengine.org/. [Accessed: 12-Jun-2020].

[43] “Knuth shuffle - Rosetta Code.” [Online]. Available: https://www.rosettacode.org/wiki/Knuth_shuffle. [Accessed: 29-May-2020].

[44] Unity 3D, “Unity Manual: Colliders,” 2018. [Online]. Available: https://docs.unity3d.com/Manual/CollidersOverview.html. [Accessed: 02-Jun-2020].

[45] “Unity - Manual: Rigidbody overview.” [Online]. Available: https://docs.unity3d.com/Manual/RigidbodiesOverview.html. [Accessed: 02-Jun-2020].

[46] “Let’s Try: Shooting with Raycasts - Unity Learn.” [Online]. Available: https://learn.unity.com/tutorial/let-s-try-shooting-with-raycasts#5c7f8528edbc2a002053b469. [Accessed: 02-Jun-2020].

[47] Song Ho Ahn, “OpenGL Projection Matrix,” 2019. [Online]. Available: http://www.songho.ca/opengl/gl_projectionmatrix.html#ortho. [Accessed: 28-Nov-2020].

[48] Song Ho Ahn, “OpenGL Transformation,” 2019. [Online]. Available: http://www.songho.ca/opengl/gl_transform.html. [Accessed: 28-Nov-2020].

[49] Code, “Understanding Screen Point, World Point and Viewport Point in Unity3D – Code Saying,” 2020. [Online]. Available: http://codesaying.com/understanding-screen-point-world-point-and-viewport-point-in-unity3d/. [Accessed: 28-Nov-2020].