Benemérita Universidad Autónoma de Puebla

Facultad de Ciencias de la Computación

Robot inteligente recolector de basura asistido por redes neuronales artificiales.

T E S I S

PARA OBTENER EL GRADO DE:

Maestro en Ciencias de la Computación

PRESENTA:

Kevin Arturo Ramirez Zavalza

ASESORES:
Dr. Manuel Isidro Martin Ortiz
Dr. José Luis Carballido Carranza

Puebla, Pue, 2020

Agradecimientos

Al Consejo Nacional de Ciencia y Tecnología (CONACYT) por la beca otorgada durante los dos años de estudios.

Al Dr. Manuel Isidro Martin Ortiz y al Dr. José Luis Carballido Carranza al otorgarme su apoyo para la realización de este trabajo de tesis.

Al Dr. Roberto Olmos Pimentel, ya que gracias a él ha continuado mi interés por conocer más sobre el área de Inteligencia Artificial y también agradecer todo su apoyo otorgado para la realización de este trabajo de tesis.

A mi esposa por siempre estar ahí a mi lado apoyándome y alentándome a siempre dar lo mejor de mí, motivándome a superarme siempre y continuar creciendo.

A mis padres, por su apoyo incondicional para siempre lograr lo que me proponga y por siempre estar ahí para mi cuando los necesite.

Y por último, también agradezco a mis amigos y profesores ya que gracias a su apoyo y enseñanzas aportaron a mis ganas de continuar estudiando.

Índice general

Resumen

En este trabajo se realiza el diseño y construcción de un prototipo de robot recolector de basura basado en algoritmos de inteligencia artificial, principalmente redes neuronales artificiales convolucionales. El propósito del prototipo es generar una alternativa viable para ayudar a solventar el grave problema de contaminación que existe en la tierra debido a la basura sólida. El prototipo presentado es capaz de navegar de forma autónoma en un ambiente de oficina u hogar, identificar la basura, ir hacia ella y recolectarla de forma autónoma.

El trabajo se ha dividido en 2 etapas principales: El diseño y desarrollo del hardware necesario para el funcionamiento electromecánico del prototipo y el diseño, entrenamiento y empleo de distintos algoritmos de inteligencia artificial cuya tarea será lograr que el dispositivo sea capaz de recolectar la basura del entorno.

En la parte del desarrollo del hardware, es importante señalar que mientras en este trabajo se presenta un dispositivo robótico completamente funcional, debido al área de estudio, el tiempo disponible y el foco de la investigación, no se ahonda mucho en la parte electromecánica del dispositivo, por lo que es posible que existan muchas formas de optimizar esa parte para proporcionarle mayores capacidades al dispositivo y soporte para terrenos más diversos. El foco de la presente investigación ha sido el entrenamiento de redes neuronales artificiales para su aplicación en la tarea que se presenta y los componentes electromecánicos elegidos para el prototipo han sido escogidos como un vehículo para la experimentación de los algoritmos. En este trabajo se detallan los componentes electromecánicos escogidos y se explica el por qué fueron elegidos para la función que desempeñan, lo cual siempre estuvo en función de las capacidades requeridas por el dispositivo y mantener costos bajos para que el dispositivo desarrollado pueda ser viable para su implementación más generalizada en hogares y oficinas.

Dentro de la parte del diseño y selección del hardware del robot, en este trabajo se presenta también la selección de hardware capaz de proveer las capacidades computacionales requeridas por el proyecto de forma óptima. Analizando diversas alternativas y considerando sus beneficios hacia los objetivos propuestos, acabando con la elección del empleo de un dispositivo de Edge computing para el funcionamiento del prototipo.

La etapa de diseño, entrenamiento y selección de algoritmos de inteligencia artificial es en la cual se centra principalmente este trabajo, para este fin se hace un repaso de la literatura existente sobre los últimos y más eficientes algoritmos que existen para las

tareas que requiere el prototipo, los retos en esta sección pueden ser clasificados como 3 principales. El primero la tarea de localización de la basura dentro de un entorno, la segunda la determinación del tipo de basura con el que se trata y la tercera el proporcionar un medio de navegación visual para el dispositivo. Basándose en factores como la velocidad del algoritmo, la precisión y la capacidad de cómputo del prototipo se escogió el empleo de un detector de la familia de detectores de objetos de una sola etapa y se complementó con un algoritmo de búsqueda simple basado en tamaños desarrollado para este trabajo.

Como resultado final de este trabajo se obtuvo un prototipo totalmente funcional de robot de bajo costo, capaz de recolectar 2 tipos de objetos considerados como basura sólida con hasta un 75 % de exactitud, navegar de forma eficiente por su espacio de trabajo gracias al empleo de marcadores visuales y clasificar los tipos de basura que recolecta entre sí.

Capítulo 1

Introducción

En la actualidad la generación y el manejo de la basura es un problema a nivel mundial el cual afecta a todos los seres vivos que habitan el planeta no solo a los humanos además de que la generación y la manera en que se maneja la basura produce un gran impacto negativo en el medio ambiente. Al año 2016 se tuvo que el mundo genera aproximadamente 2.01 billones de toneladas de basura sólida anualmente tan solo considerando la basura generada a nivel residencia, comercial e institucional sin incluir la generada a nivel industrial, nivel médico, electrónica, demolición y construcción. Además, al observar el panorama global una persona puede generar en promedio 0.74 kilogramos de basura, pero este valor puede variar en entre 0.11 a 4.54 kilogramos al día por persona en función al país, nivel de urbanización y nivel de ingresos. Mirando a futuro se tiene proyectado que para el año 2050 la producción global de basura crecerá hasta 3.40 billones de toneladas anuales.

La generación de basura es un producto natural derivado del proceso de urbanización, desarrollo económico y del crecimiento de la población. A medida que los países, las ciudades o asentamientos prosperan esto tiene como resultado la creación de una mayor demanda de productos y de servicios para los habitantes, lo cual propicia el crecimiento a nivel económico, pero también trae como consecuencia la necesidad de una mejor administración de la basura con respecto a su transporte, manejo y disposición final.

En la Figura 1 podemos observar la cantidad en kilogramos de la generación de basura per cápita a nivel mundial al año 2018, podemos notar que países como lo son Estados Unidos, Canadá o Alemania generan una cantidad superior a 1.50 kg de basura sólida per cápita mientras que países como China, Etiopia, Somalia o Sudan generan una cantidad de basura per cápita entre 0 y 0.49 kg.

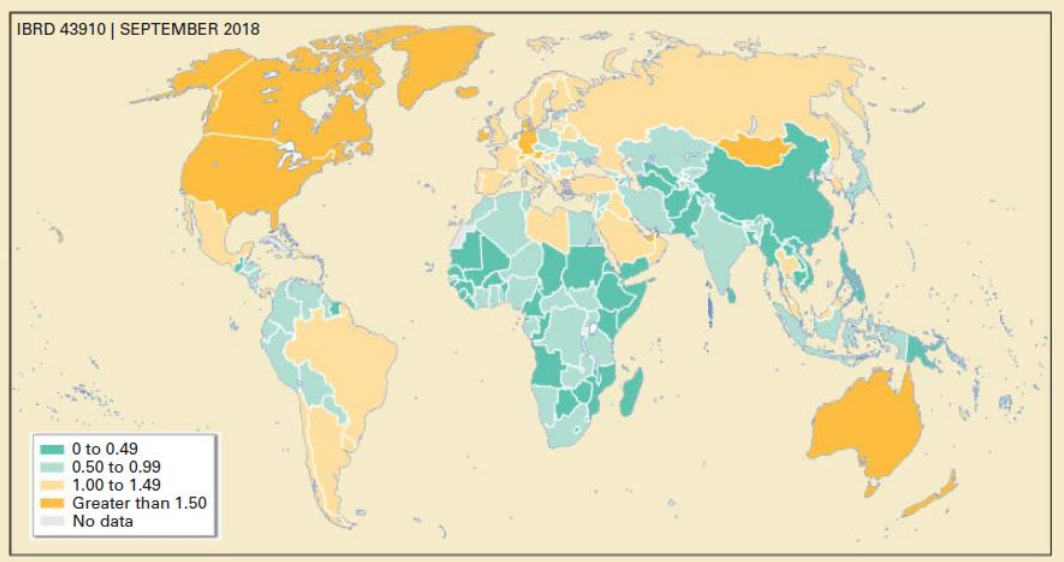


Figura 1: Generación de basura per cápita. [2]

Si se realiza un análisis de la generación de basura a nivel mundial por regiones, como podemos observar en la Figura 2 se puede observar que la región que mas basura genera es el Este de Asia y el Pacifico con el 23 % seguido de Europa y Asia Central con el 20 %, el Sur de Asia con 17 %, Norte América con 14 % y Latino América y el Caribe con 11 %.

[2]Kaza, S., Yao, L., Bhada-Tata, P., & Van Woerden, F. (2018). What a waste 2.0: a global snapshot of solid waste management to 2050. p.19

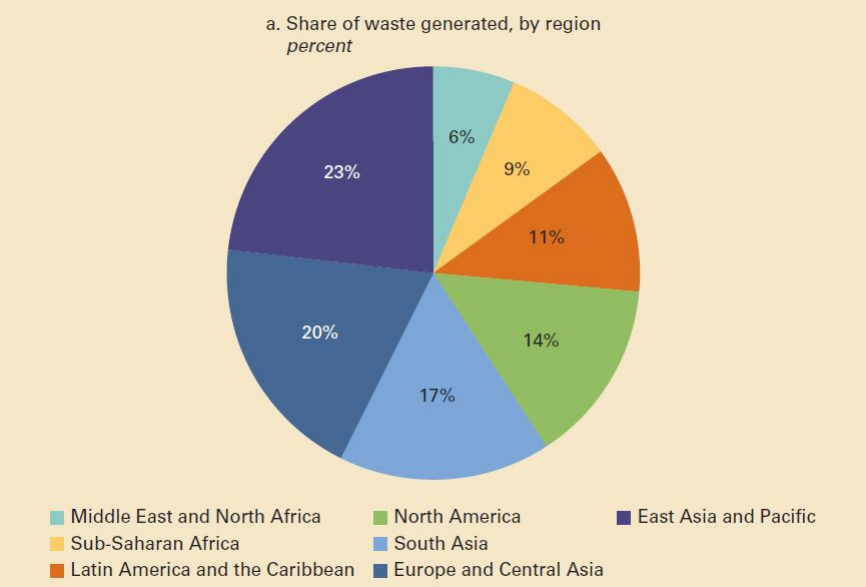


Figura 2: Generación de basura por región. [3]

Además de lo anterior se tiene que a pesar de que los países con altos ingresos solo son un 16 % del total de la población mundial, estos países generan el 34 % de basura a nivel mundial mientras que los países con un bajo nivel de ingresos los cuales solo son un 9 % del total de la población mundial solo generan el 5 % del total de la basura generada a nivel mundial. En la Figura 3 podemos analizar el porcentaje de basura generada con respecto al nivel de ingresos.

[3]Kaza, S., Yao, L., Bhada-Tata, P., & Van Woerden, F. (2018). What a waste 2.0: a global snapshot of solid waste management to 2050. p.19

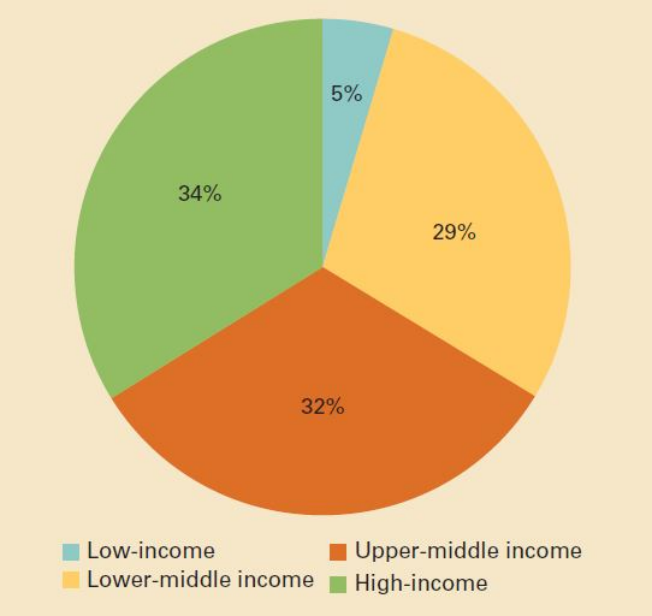


Figura 3: Generación de basura por nivel de ingresos. [4]

Con lo anterior se pudo observar la relación que existe de la generación de basura con el desarrollo económico de un país o región. Mientras mas ingresos se tienen en un país o región mayor es la cantidad de basura generada per cápita mientras que a menor ingreso menor cantidad de basura es generada [1].

Por otro lado, es necesario definir la composición de la basura generada per cápita, esta es categorizada por el tipo de materiales que puede contener y estos tipos son determinados a través de una auditoria estándar a la basura, en donde son tomadas muestras de depósitos y posteriormente son ordenadas en diferentes categorías. Por ello se tienen estas categorías que componen a la basura: Comida y alimentos verdes, vidrio, metal, papel y cartón, plástico, madera, caucho y cuero y otros. En la Figura 4 se puede observar de manera general el porcentaje de composición de la basura de manera general. También se puede observar que de la composición de la basura generada el 38 % está conformado de reciclables secos (plástico, papel, cartón, metal y vidrio.)

[4]Kaza, S., Yao, L., Bhada-Tata, P., & Van Woerden, F. (2018). What a waste 2.0: a global snapshot of solid waste management to 2050. p.21

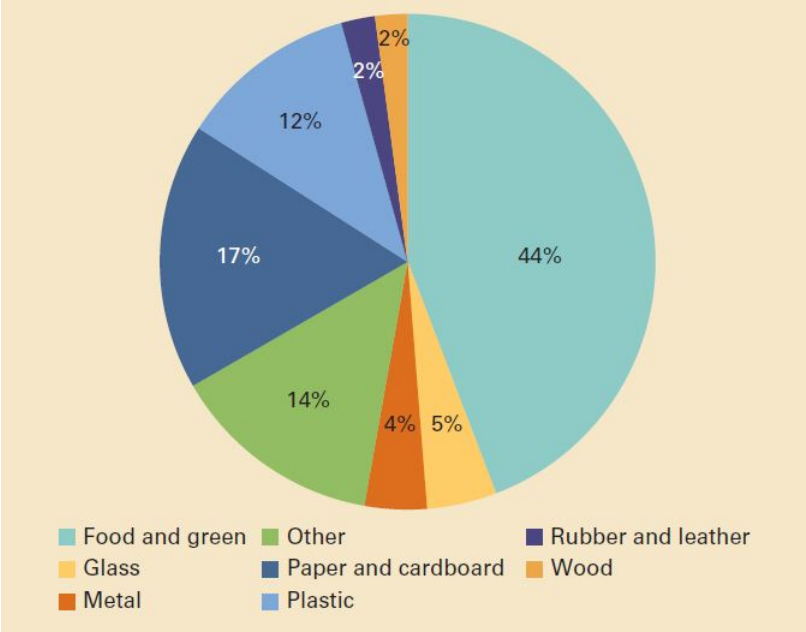


Figura 4: Composición de la basura de manera general. [5]

Aunado a esto, en México en el año 2010 se tuvo un estimado del porcentaje de composición de la basura como se muestra en la Figura 5. El tipo de basura más común en ese año fue la materia orgánica.

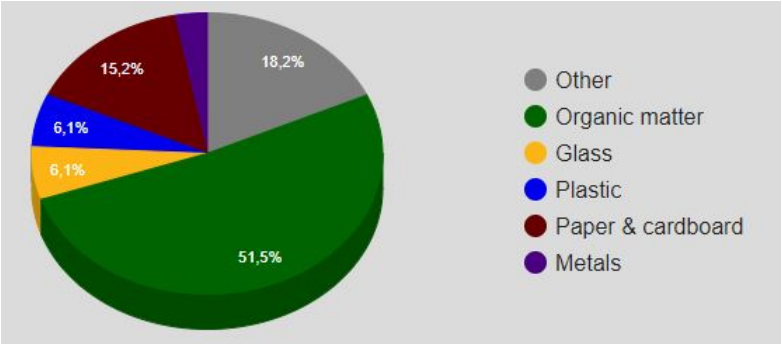


Figura 5: Composición de basura en México en 2010. [6]

En la actualidad en México se han establecido políticas del manejo de la basura con el fin de disminuir la generación de basura, de crear cultura de reducción de basura

[5]Kaza, S., Yao, L., Bhada-Tata, P., & Van Woerden, F. (2018). What a waste 2.0: a global snapshot of solid waste management to 2050. p.29

[6]Recuperada de `http://www.atlas.d-waste.com/index.php?view=country_report&country_id=13`

además del manejo de esta y que sea adecuada con el medio ambiente, promover el reciclaje, así como el manejo y colocación de la basura de manera que se produzca el menor impacto posible al medio ambiente.

El problema actualmente radica en varios aspectos del manejo que se lleva a cabo con la basura, esto surge desde el lugar donde es originada la basura, sean los hogares, tiendas, industrias y hospitales, ya que estos no tienen los espacios adecuados para colocar la basura lo cual los convierte en lugares de riesgo para la salud y el medio ambiente. Asimismo, otro problema del manejo de la basura radica en la recolección, en América latina en las grandes ciudades es recolectada alrededor del 89 % mientras que en las pequeñas solo es recolectado entre el 50 y 70 %. Se estima que en promedio el 75 % de la basura sólida generada es recolectada dejando el otro 25 % de basura por ahí en zonas públicas, lotes baldíos, barrancos, sistemas de drenaje, ríos, entre otros.

Asimismo, es importante mencionar que el manejo de la basura puede tener efectos adversos en la salud de las personas que están en contacto o se encuentran expuestos a ella. Esto puede suceder de manera directa o indirecta, al tener contacto corporal con ella, al tener alguna lesión ocasionada por contacto con la basura, también al inhalar o al ingerirla. La exposición a la basura está en función a que tanta basura es generada, cuanta de esta es recolectada, transportada y en qué proporción es eliminada de forma segura. En un nivel superior, el riesgo a exponerse a la basura tiene mucho que ver sobre como esta es manejada y los recursos que se destinan a esta tarea. Existen diversas razones por las que uno puede estar expuesto a la basura, una de estas es al estar expuesto a la basura generada en sus hogares, trabajos u hospitales. Otra es que están las personas que realizan la labor de recolectarla como lo son los trabajadores del servicio de limpia o aquellas personas que trabajan en la informalidad recolectando basura sin algún tipo de equipo necesario para su protección además de que están las personas que viven cerca de rellenos sanitarios o lugares donde es incinerada la basura.

Así pues, al estar expuesto a la basura puede tener repercusiones serias en las personas, la gravedad de esto depende de diversos factores como lo es el tipo de basura con la que se esta en contacto, el tiempo el que se estuvo expuesto a esta o en contacto con esta, el tipo de población que estuvo expuesta y las posibles contramedidas tomadas por la exposición a esta. El impacto que puede traer esto en la salud puede ser desde efectos psicológicos leves hasta morbilidad, discapacidad o la muerte. Esto se produciría debido a transmisión de infecciones o enfermedades debido a la acumulación de basura, contacto con material médico, por heridas como cortes con objetos afilados, o enfermedades crónicas por exposición a químicos o radiación debido al mal manejo de desechos industriales o daños emocionales o psicológicos ya que se ha reportado que en ocasiones han sido encontradas partes humanas en rellenos sanitarios hasta enfermedades mentales ocasionadas por exposición constante a metales pesados [2].

En ese mismo contexto, en México son llevadas a cabo labores de reciclaje con el fin de poder reusar ciertos tipos de materiales con el fin de reducir la cantidad de basura además de reducir el uso de recursos en el manejo de basura y en los procesos de producción y manufactura de algunas compañías. En México se reporta que se recicla el 10 % de la basura generada de la cual la mayor cantidad de materiales reciclados son

papel, cartón, latas de aluminio y PET [3]. Lamentablemente este porcentaje es muy pequeño comparado con la cantidad de basura que se produce y no se ha visto todo el potencial que se tiene en esta actividad, actualmente se considera que el reciclaje es uno de los negocios más rentables del mundo, al año 2016 solo en México se estima que existe una oportunidad de mercado de 24 mil millones de dólares en la industria de residuos reciclables sin embargo esta oportunidad no está siendo aprovechada [4].

Los métodos usados actualmente para llevar a cabo el reciclaje dependen mayormente del trabajo humano ya que es necesario ir a los lugares donde se encuentra la basura para posteriormente separarla manualmente con el fin de identificar que tipo de basura puede ser reciclada y posteriormente transportarla a centros de acopio. Lamentablemente al ser una actividad que depende de la labor humana es susceptible a errores como el separar erróneamente la basura además del riesgo que conlleva con la salud, debido a esto no se ha logrado sacar el mayor provecho y beneficio económico al reciclaje por lo que existe la necesidad de crear sistemas autónomos o semiautónomos e inteligentes que tengan la capacidad de identificar la basura, que sean capaces de manipularla y separarla de manera más eficiente, rápida y segura.

1.1. Objetivo general

Diseñar, validar e implementar un prototipo de robot y los algoritmos de control asistidos por redes neuronales artificiales e inteligencia artificial. El robot deberá ser capaz de identificar visualmente "basura", ir a ella, recolectarla y llevarla a un área predefinida de forma autónoma.

1.1.1. Objetivos particulares

1. Investigar y recopilar información del estado del arte
   - *a*) de las redes neuronales artificiales y sus aplicaciones.
   - *b*) de las redes neuronales convolucionales y sus aplicaciones.
   - *c*) sobre arquitecturas de detectores de objetos.
2. Plantear un método haciendo uso de arquitecturas de detección de objetos para clasificar e identificar objetos específicos en el mundo real.
3. Generar un dataset con imágenes de diversos objetos a ser detectados por el método planteado para el entrenamiento de los métodos.
4. Investigar y recabar información sobre aprendizaje reforzado y sus aplicaciones.
5. Implementar un sistema de control para el movimiento del robot que resuelva un problema de recolección de "basura" concreto.

Para una mejor lectura del presente documento se ha dividido el trabajo de tesis de la siguiente manera: El primer capítulo contempla el fundamento teórico en el cual se basa este trabajo. El segundo capítulo contempla el análisis y diseño de requisitos para la elaboración del trabajo de tesis. El tercer capítulo detalla los procedimientos llevados a cabo para la implementación del trabajo de tesis. En el cuarto capítulo se presentan las pruebas realizadas y los resultados obtenidos experimentalmente. Y finalmente se presenta un capítulo con las conclusiones y comentarios del presente trabajo de tesis.

Capítulo 2

Marco teórico

En este capítulo se identifica de forma concisa el problema que busca resolver el desarrollo tecnológico asociado a este trabajo además de que se estudian algunos de los trabajos que han generado soluciones relacionadas. Posteriormente a lo largo de este capitulo se presenta una breve revisión de cada uno de los elementos teóricos que son necesarios y en los cuales se fundamenta el desarrollo e implementación de este trabajo de tesis.

2.1. Planteamiento del problema

Como se presentó anteriormente en este trabajo, la basura actualmente es un problema que se presenta a nivel mundial debido a la cantidad que es generada anualmente y los efectos que esta tiene por su pobre manejo. Con respecto a esto, al tener un pobre manejo de la basura ocasiona la aparición de focos de infecciones y otras enfermedades en humanos, no propicia el reciclaje de los distintos tipos de basura lo cual ocasiona un incremento de espacio requerido por esta lo que a su vez favorece la creación de más rellenos sanitarios lo que a su vez puede causar la contaminación del subsuelo además de la creación de mas incineradores de basura lo cual puede producir un daño irreversible al medio ambiente al incrementar la contaminación del aire. Estas cantidades de basura pueden ser reducidas si la basura es separada debidamente desde los orígenes como lo son los hogares y la industria para su posterior procesamiento y reciclaje con el fin de reutilizar estos recursos. Por lo anterior es necesario empezar a proponer nuevas soluciones en cuando a recolección y manejo de basura con el fin de disminuir el impacto que esta produce en nuestras vidas y en nuestro medio ambiente a la par de generar un beneficio económico mediante el reciclaje. Actualmente contamos con diversos avances tecnológicos los cuales nos permitirán abordar el problema y sus posibles soluciones.

2.2. Antecedentes y trabajos relacionados

Desde el surgimiento del área de visión por computadora se ha tenido un claro objetivo y el cual actualmente se sigue persiguiendo, este es la detección y localización de objetos, ya que esta es una tarea crucial para nuevas tecnologías que están siendo desarrolladas como lo son la conducción autónoma, robots de asistencia y uso personal, así como para la industria robótica entre otras. Al momento en que surge este objetivo el problema se planteó de la siguiente manera: dada una imagen de entrada, predecir la clase a la que pertenecen los objetos presentes y la ubicación de cada uno de estos en la imagen mediante el uso de cajas delimitadoras o bounding boxes. De entre los primeros trabajos formales existentes enfocados a resolver el problema se encuentra el trabajo "Rapid Object Detection Using a Boosted Cascade of Simple Features" por Viola y Jones [5] en el 2001 enfocado originalmente para la detección de rostros, el método que propusieron es conocido como "Haar Cascade". Este es un algoritmo del área de aprendizaje automático el cual es entrenado con ejemplos de imágenes positiva y negativas, para esto son extraídas diversas características de las imágenes mediante la aplicación de un filtro en una porción de la imagen y comparando las intensidades de los pixeles del filtro con la imagen. Posteriormente varios años después en el 2005 en el paper "Histograms of Oriented Gradients for Human Detection" por Dalal y Triggs [6] surgió un nuevo método comúnmente conocido como HOG el cual es un descriptor de características, este tuvo como objetivo la detección de peatones en imágenes mediante el uso de los gradientes de una imagen a la cual le son aplicados diversos filtros, la magnitud de estos gradientes sirvieron para la detección de bordes y esquinas, posteriormente se genera un histograma basado en los gradientes de la imagen para posteriormente calcular el vector de características de la imagen. Posteriormente gracias al incremento de poder de cómputo tanto en CPU's como en GPU's y debido a la disminución de su costo dio pie a la implementación de nuevos modelos que pudieran aprovechar esa capacidad de cómputo además de surgir la necesidad de procesar grandes cantidades de información. De este modo se empezó a hacer más énfasis en el desarrollo de las redes neuronales artificiales y posteriormente con el surgimiento y auge del Aprendizaje profundo gracias a los sistemas de reconocimiento de voz y la creación de conjuntos de datos etiquetados como ImageNet [7] el cual contenía un total de 3.2 millones de imágenes las cuales estaban etiquetadas en 5247 diferentes categorías al momento en que se publicó el paper. Debido a la creación de conjuntos de imágenes como este propiciaron la creación de nuevas arquitecturas de redes neuronales enfocadas a la detección de objetos mediante el uso de Redes Neuronales Convolucionales como lo son AlexNet y VGG en 2012, GoogleNet y ResNet en 2015 las cuales fueron entrenadas con ImageNet para competir en el ILSVRC(ImageNet Large-Scale Visual Recognition Challenge) el cual tiene como objetivo que se logre clasificar de manera correcta la mayor cantidad de imágenes posibles, esto ha propiciado el constante desarrollo y mejora de arquitecturas de redes neuronales profundas enfocadas en la detección de objetos. Asimismo, gracias a las anteriores arquitecturas en combinación con otros métodos propuestos surgieron nuevas arquitecturas en las cuales se logran ambos objetivos: la detección y localización de objetos [8].

Para esto surgieron arquitecturas como lo son R-CNN, Fast R-CNN, Faster R-CNN entre otros los cuales usan métodos basados en regiones para indicar en que parte de la imagen se encuentra el objeto, además de estas surgieron otras como YOLO o SSD las cuales están basadas en métodos de regresión y clasificación. Actualmente estas y otras arquitecturas más están siendo aplicadas para resolver problemas como detección de cáncer de mama [9], detección de cáncer en los pulmones [10], lectura de placas de automóviles [11] o reconocimiento de señales de tráfico para el manejo autónomo de vehículos [12]. Como estos ejemplos hay muchos otros problemas por resolver y que pueden ser resueltos mediante la aplicación del Aprendizaje Profundo en sus distintas vertientes.

En la actualidad se están buscando diversas soluciones para resolver los problemas generados por la basura, por su transporte, por su reciclaje y reutilización además de el manejo de esta al llegar a su destino final. Debido a esto se lanza la interrogante al aire de qué opciones existen actualmente para resolver este problema, que utilizan para lograr el objetivo y de si es posible hacer uso del aprendizaje profundo como una posible solución mas a estos problemas. Gracias al crecimiento presentado en el área de visión artificial haciendo uso de redes neuronales profundas se han propuesto diversas soluciones para abordar la problemática de la basura, de su detección, de su recolección y de su clasificación para posterior reciclaje. Haciendo un estudio de los trabajos más recientes, partiendo del año 2016, se encuentran trabajos en donde haciendo uso de una red neuronal profunda se realiza la detección de basura y su concentración tomando como base una fotografía y el procesamiento realizado mediante una aplicación móvil [13]. Tomando un camino diferente al aprendizaje profundo, se han propuesto soluciones en donde hacen uso de algoritmos tradicionales los cuales permiten la generación de un mapa mediante un sensor ultrasónico el cual les permite determinar en donde se encuentra un obstáculo denominado basura para posteriormente manipularlo con una herramienta de succión por aire [14]. Asimismo, han sido presentados trabajos enfocados en el sector industrial con el fin de implementar líneas automatizadas con el fin de realizar clasificación y manipulación de distintos tipos de basura haciendo uso de visión artificial junto con redes neuronales profundas y usando un brazo robótico para manipularla [15]. De manera similar han sido propuestos diversos trabajos de robots semiautónomos y autónomos los cuales hacen uso de visión artificial con el fin de recolectar basura individualmente o recolectar contenedores de basura. Entre algunos de estos trabajos podemos encontrar el robot E-Swachh [16] el cual tiene como objetivo recolectar la basura indiscriminadamente en un área determinada haciendo uso de una minicomputadora para la labor de visión artificial. Igualmente se puede encontrar trabajos en donde se ha realizado la implementación de un robot autónomo con el fin de recolectar los contenedores de basura en una oficina mediante el uso de sensores LIDAR para la creación de un mapa en tiempo real además de ser utilizado para la detección de objetos [17]. También se encuentran trabajos donde el robot esta diseñado para funcionar en un área de trabajo específica, un ejemplo de esto es un robot autónomo basado en aprendizaje profundo diseñado específicamente para la detección y recolección de basura en entornos donde predomina el pasto, sea en lugares públicos como parques o

en escuelas [18]. Además, existen trabajos que buscan el reducir la basura por medio de sistemas que se enfocan en identificar el tipo de material de la basura como por ejemplo las máquinas expendedoras inversas la cuales pueden ser encontradas en distintas regiones de Europa como Estados Unidos, estas máquinas reciben envases vacíos para posteriormente regresar dinero al usuario [19]. Este tipo de máquinas hacen uso de visión artificial mediante redes neuronales profundas para poder determinar el tipo de material del envase que el usuario esta ingresando, gracias a este tipo de máquinas se motiva a la gente a reciclar la basura que genera a cambio de obtener algún beneficio monetario o un descuento en algún servicio. En el estudio de lo mas reciente al año 2020 se puede encontrar el desarrollo de un robot inteligente conocido como IWSCR [20] el cual funciona en superficies acuáticas el cual tiene como objetivo recolectar basura plástica que flota en el agua haciendo uso de un manipulador y realizando la detección de la basura mediante el uso de visión artificial basada en redes neuronales profundas.

Entre las principales ventajas que se pueden encontrar en los trabajos realizados a la actualidad es que varios de estos han buscado implementar soluciones basadas en aprendizaje profundo para sus sistemas de visión artificial los cuales les permite tener una mejor detección en comparación con los métodos utilizados en la época anterior a este lo que nos permite ver que es posible hacer uso de esta para resolver al problema propuesto. Asimismo, entre las desventajas que se pudieron encontrar con respecto a los trabajos realizados actualmente se encuentra que algunos de estos haces uso de plataformas de hardware con poco poder de cómputo como lo es Arduino además de que algunos de los sistemas de visión funcionan de manera genérica, solo detectan concentraciones de basura o se enfocan en un tipo muy específico de esta para su funcionamiento por lo que la cantidad de trabajo que estos pueden realizar es limitada además que del tiempo en que estos trabajos fueron realizados a la actualidad ha permitido la creación, optimización y actualización de diversas arquitecturas para detección y localización de objetos lo cual puede permitir un incremento en el porcentaje de detección de objetos a la vez de reducir el costo computacional requerido. Debido a esto y con los recursos disponibles actualmente en cuanto a hardware y software es realizable una nueva implementación de este tipo de sistemas con el fin de mejorar su desempeño además de ampliar sus funciones.

2.3. Aprendizaje Automático

En la actualidad con el desarrollo de la tecnología diariamente se genera una gran cantidad de datos en medios como texto, audio, fotografía o video, cada uno de estos medios almacenan un tipo de información específica como correos electrónicos, secuencias se sonido, fotografías o una secuencia de imágenes, de estos datos es posible extraer información de interés por lo que se tiene la necesidad de analizar grandes cantidades de datos para lograr este fin. En algunos casos esta es una tarea sencilla de realizar ya que es posible crear y hacer uso de algoritmos los cuales pueden ser utilizados para buscar información específica entre una gran cantidad de información recolectada, esto signi-

fica, en otros términos, establecer una serie de instrucciones las cuales nos permitirán convertir información de entrada (lo que deseamos analizar) en una o varias salidas (saber si se obtuvo un resultado satisfactorio o no). Pero para realizar algunas tareas en específico, no se posee algún algoritmo en específico que nos permita realizarlas correctamente ya que se puede dar el caso en el que no existan marcadores o patrones claros en la información los cuales puedan ser usados como una referencia. También puede ser que la información varíe con respecto a cada individuo por lo que no siempre se pueden seguir las mismas reglas para completar la tarea. Un ejemplo que está presente en la actualidad es la gestión del correo electrónico, uno de los problemas que se presentan es saber detectar y separar los correos basura de los que no lo son, esta no es una tarea sencilla ya que la información contenida en cada uno de los correos electrónicos muchas veces no posee patrones específicos que permitan detectar rápidamente este tipo de correos por lo que en algunos casos nuestra falta de conocimiento de lo que se tiene que buscar haciendo uso de un algoritmo tradicional no es suficiente por lo que es necesario aproximarse al problema de diferente manera. Al no poseer un algoritmo tradicional que nos permita realizar exitosamente ese trabajo entonces es necesario buscar una manera de realizar la tarea con los mejores resultados posibles. En la actualidad para poder resolver esta clase de problemas se desarrolló un área de la inteligencia artificial conocida como Aprendizaje Automático o Machine Learning que tiene como fin permitir a un sistema tener la capacidad de aprender del entorno buscando replicar la manera en la que aprenden los humanos, este se dedica al estudio y a la creación de algoritmos que puedan aprender de casos anteriores y que además sea capaz de realizar predicciones a futuro. Para esto es necesario recabar una gran cantidad de elementos que permitan ser usados de guía para que el sistema sea capaz de aprender por sí misma y sea capaz de generalizar y realizar predicciones. En otras palabras, esto significa que mediante el uso de Machine Learning se es capaz de generar un algoritmo para una tarea en específico haciendo uso como base diversos ejemplos buscando abstraer toda la información posible de estos, mediante la búsqueda de similitudes, características únicas, patrones y regularidades presentes en la información.

2.4. Métodos de Aprendizaje

La forma en que aprenden los humanos siempre ha sido objeto de estudio, esto con el fin de aprender cómo aprendemos, pero además tener la capacidad de generar métodos que puedan ser aplicados a otros sistemas, como son los computacionales y así dotarlos de capacidad de aprendizaje y de saber responder ante nueva información que ingrese al sistema y no solo dependiendo de casos pre establecidos en la programación.

El aprendizaje para este tipo de sistemas está clasificado en tres:

1. Aprendizaje Supervisado
2. Aprendizaje no Supervisado
3. Aprendizaje por Refuerzo

2.4.1. Aprendizaje Supervisado

El método de aprendizaje supervisado es un método el cual hace uso de conjuntos de ejemplo o de entrenamiento para generar una función con el fin de obtener una salida ya establecida. Para poder realizar este procedimiento es necesario establecer cómo será la entrada y posteriormente aplicar un algoritmo con el fin de generar un vector de características, este vector de características estará relacionado con una clase específica a detectar y clasificar, pero además se ha de asignar una salida para cada uno de los diferentes vectores de características para las clases que se deseen identificar y clasificar. Esto se realiza mediante el análisis de datos de entrada y usando como referencia la salida establecida, esto con el fin de inferir una función que nos proporcione la salida deseada, pero que además esta función puede ser usada para generalizar nuevas salidas ante la presencia de nuevos casos, esto quiere decir que el sistema después de haber sido entrenado, y ante la presencia de entradas similares a el set de entrenamiento, este puede generar una respuesta similar En el aprendizaje supervisado, existen diferentes tipos de clasificadores, de entre los cuales la elección para la resolución de este problema son las Redes Neuronales.

2.4.2. Aprendizaje no Supervisado

El aprendizaje no supervisado es un método en el cual las entradas son conocidas y las salidas son desconocidas. El objetivo de este método es el encontrar regularidades o patrones en la entrada, esto con el fin de identificar una periodicidad o uniformidad y esto suceda de manera constante. El aprendizaje no supervisado está estrechamente relacionado con el problema de estimación de densidad en el área de estadística.

Para realizar el aprendizaje no supervisado es necesario aplicar una estimación de densidad, esta es una técnica en el Análisis Exploratorio de Datos (DEA por sus siglas en inglés). Esta técnica busca generar un valor de salida mediante el análisis de las entradas, esto buscando patrones o regularidades presentes en la entrada y se pueda obtener una salida consistente [42]. Para este método de aprendizaje se puede realizar desde diferentes aproximaciones, entre las principales se encuentran:

- Clustering o Algoritmo de Agrupamiento
- Algoritmo E-M

2.4.3. Aprendizaje por Refuerzo

Este es un método aprendizaje perteneciente al área de aprendizaje autónomo, este método está inspirado en el área de psicología, específicamente en el conductismo, este método busca cómo se realiza la toma de decisiones en un entorno con el fin de resolver un problema de manera satisfactoria.

Algunas de las áreas en las que es aplicado este método son la teoría de juegos, teoría de control, optimización basada en simulación y algoritmos genéticos.

Para algunas aplicaciones, se requiere de una secuencia de pasos con el fin de llegar a un resultado satisfactorio, por lo que para este método no es importante que cada uno de estos pasos sea correcto, lo que busca este método de aprendizaje es una norma la cual permita generar una serie de pasos correctos para alcanzar un resultado positivo, pero además con este método, el sistema es capaz de generar diversas normas para llegar a un mismo resultado y también es capaz de distinguir cuál de estas normas generadas es más efectiva. Este método es iterativo, ya que requiere de varios intentos con el fin de llegar a resolver un problema o cumplir su objetivo [42].

2.5. Redes Neuronales

Las redes neuronales artificiales están inspiradas en el funcionamiento y estructura de una red neuronal biológica. Una red neuronal biológica es un conjunto de neuronas que se encuentran conectadas y se comunican entre sí. Para poder comprender lo que es una red neuronal primero tenemos que enfocarnos en las unidades fundamentales que conforman a las redes neuronales biológicas, la neurona.

La neurona es una célula viva que tiene algunos elementos que la diferencian de las otras células vivas, esta tiene una forma más o menos esférica y esta posee un tamaño entre 5 y 10 micras, en contraste con una compuerta lógica de silicio las neuronas son más lentas ya que estas funcionan en orden de mili-segundos mientras que una compuerta está en el orden de nanosegundos. Una neurona está compuesta por una parte llamada axón que está conectada al cuerpo de la célula y que en su extremo contiene ramificaciones más pequeñas con las cuales emiten la salida de la neurona y realizan la conexión con otra u otras neuronas. También del cuerpo de la célula salen otras ramificaciones llamadas dendritas las cuales son utilizadas para recibir la conexión de otra neurona y las cuales llevan la señal al interior de la célula. Por lo general una neurona recibe miles de conexiones e información de otras neuronas y a su vez puede enviar información a miles de neuronas. En la Figura 6 se puede observar la forma general de la neurona en donde se puede ver el cuerpo de la célula, el axón, las dendritas y los puntos de conexión de entrada y de salida de la neurona.

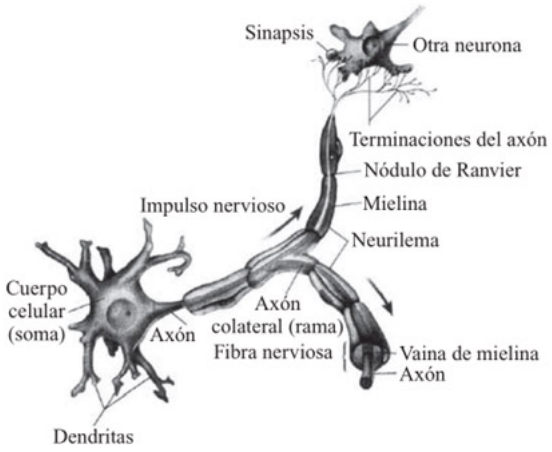


Figura 6: Forma general de la neurona.[2]

Las señales que utiliza una neurona son de dos naturalezas diferentes, de tipo eléctrica y de tipo química. La señal que se transmite desde el axón es de tipo eléctrica, mientras que la señal que se transmite desde las ramificaciones del axón hasta las dendritas de la siguiente neurona son de tipo químicas, a esta conexión se le llama sinapsis. En la Figura 7 se puede ver cómo se lleva a cabo el proceso de la sinapsis, la distancia que existe entre la terminal del axón y de la dendrita de la siguiente neurona es de apenas 50 nm.

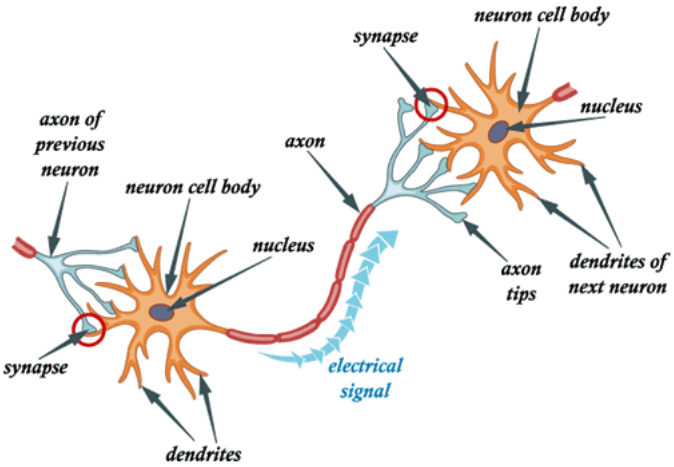


Figura 7: Sinapsis entre dos neuronas.[3]

[2]Cruz, P. P. (2011). Inteligencia artificial con aplicaciones a la ingeniería. Alfaomega. p.194

[3]Recuperada de `https://medium.com/autonomous-agents/mathematical-foundation-for-activation-functions-in-artificial-neural-networks-a51c9dd7c089`

La sinapsis química es el tipo más común, como se mencionó anteriormente una señal eléctrica viaja a través del axón hasta llegar al botón sináptico en la Figura 7, ahí se rompen las vesículas sinápticas representadas de color azul y así liberándose una sustancia llamada neurotransmisor, esta sustancia es captada por la dendrita de la siguiente neurona y esta estimula la emisión de una nueva señal eléctrica en la nueva neurona y que va en dirección derecha para poder pasar a la siguiente neurona. En la neurona existen dos comportamientos que son importantes:

El primero nos dice que el pulso que llega a la sinapsis y el que sale no son necesariamente iguales, el pulso de salida dependerá de la cantidad de neurotransmisor liberada, la cantidad del neurotransmisor varía durante el proceso de aprendizaje de la neurona haciendo que el pulso sea reforzado o debilitado.

El segundo comportamiento nos dice que se suman todas las entradas de las dendritas de una neurona y si la suma de esas entradas sobrepasa un umbral entonces se mandará un pulso a través del axón, en caso de no superar el umbral no se mandara un pulso. Las neuronas poseen un tiempo refractario, esto significa que durante la transmisión entre mensaje y mensaje requieren un tiempo de entre 0.5 y 2 mili-segundos para poder transmitir un nuevo mensaje [21].

2.5.1. Redes Neuronales Artificiales

La motivación por la cual se desarrollaron las redes neuronales artificiales es la búsqueda de modelar el funcionamiento de una red neuronal biológica, esto específicamente por la forma de funcionar del cerebro humano ya que este es un sistema complejo, no lineal y funciona de manera paralela, esto significa que puede realizar varias operaciones de manera simultánea a diferencia de las operaciones realizadas por una computadora que son realizadas de manera secuencial, lo que significa que para poder realizar operaciones tienes que realizarlas una a la vez, con esto podemos decir que una red neuronal artificial puede funcionar como un procesador, esta red está compuesta por neuronas que serían las unidades encargadas del procesamiento. Algunas de las principales características de una red neuronal artificial son:

1. Estas poseen la capacidad de adquirir conocimiento a través de la experiencia mediante el uso de datos de ejemplo, con los cuales la red neuronal ajusta las conexiones entre las neuronas para responder a determinados estímulos.

2. Poseen capacidad de plasticidad y adaptabilidad, esto significa que una red neuronal puede ser moldeada para aprender a ejecutar nuevos procesos como el reconocimiento de patrones, clasificaciones y asociaciones de la información de entrada además de poder responder de forma correcta a nueva información por su capacidad de generalización.

3. Tienen un alto nivel de tolerancia a fallas, esto significa que la red puede llegar a presentar fallos por causas externas y la red podrá seguir teniendo un buen funcionamiento, esto imitando la red neuronal biológica.

A continuación, se presenta un modelo sencillo para la construcción de redes neuronales artificiales, este modelo busca modelar las características más importantes del comportamiento biológico.

Para el modelo de la red neuronal artificial, como se puede observar en la Figura 8 contamos con n neuronas de entrada denotadas como $x_1 ... x_n$ y estas transmiten los valores de entrada a la neurona de salida y_j. Los elementos denotados w_{j0}, w_{j1}, w_{ji} son los pesos sinápticos de las conexiones con la neurona y_j, en la notación de los pesos el primer índice indica a qué neurona se dirige y el segundo índice indica de qué neurona proviene la información.

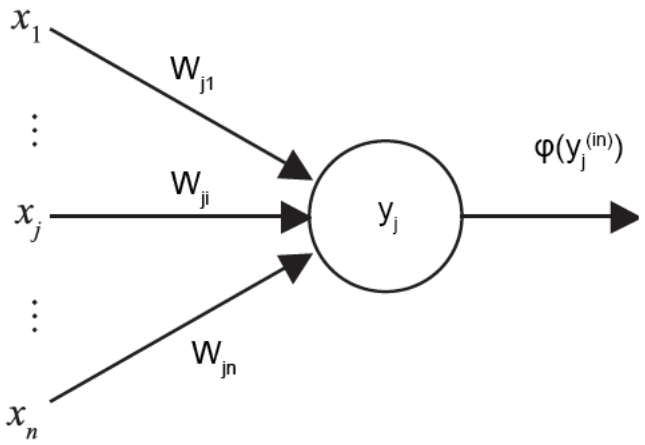


Figura 8: Esquema de la neurona artificial.[4]

La función de cada uno de los pesos sinápticos w_{ji} es realizar una multiplicación con su correspondiente entrada y este peso define la importancia de cada entrada. Si se recuerda de lo mencionado anteriormente, en una neurona biológica se suman todas las entradas, para el caso del modelado de la neurona artificial se realiza la misma operación, por lo que para obtener la entrada total y_j se tiene la ecuación 1.

$$y_j^{(in)} = \sum_{i=1}^{n} w_{ji} x_i \tag{1}$$

Se incluye el índice (in) para denotar que es la entrada. Como se mencionó en el funcionamiento de la neurona biológica, para poder generar una salida es necesario que la suma de las entradas supere un umbral. Para poder realizar esto es necesario aplicar una función de activación φ la cual se aplica sobre $y_j^{(in)}$, para realizar esto existen diversas funciones de activación. La ecuación 2 representa la señal de salida de la neurona y_j.

[4]Elaboración propia.

$$y_j = \varphi(y_j^{(in)}) \tag{2}$$

Entre los componentes más importantes que podemos encontrar en una red neuronal son:

- Función de activación
- Tipo de red neuronal artificial
- Regla de aprendizaje
- Algoritmo de entrenamiento

La red neuronal puede estar organizada en función de:

- Número de capas
- Número de neuronas por capa
- Patrones de conexión
- Flujo de información

Para poder realizar la distribución de las neuronas en una red neuronal se realiza mediante capas o niveles y cada una con un determinado número de neuronas. En una red neuronal existen tres tipos de capas: de entrada, ocultas y de salida.

La capa de entrada es donde se recibe la información de la entrada original proveniente de otro medio y que puede haber sido pre-procesada para su uso en la red. Las capas ocultas son internas y no poseen alguna conexión directa con el exterior de la red, con la posibilidad de que pueden existir varias capas ocultas y estas pueden estar conectadas entre sí. Y por último la capa de salida la cual transfiere la información interna de la red al exterior.

Para el flujo de información, dependiendo del tipo de red existen las redes con conexiones hacia adelante o feedforward en las cuales se propaga la información hacia adelante, desde la capa de entrada hasta la capa de salida. También existen las redes con conexiones hacia adelante y hacia atrás, o que significa que la información puede propagarse en ambos sentidos de las conexiones de la red.

Con esto se da la topología de las redes neuronales las cuales son:

- Redes simples o unicapa
- Redes multicapa

Redes simples o unicapa: Estas son las redes neuronales mas básicas, esta está conformada por solo dos capas, una de entrada para recibir señales y posteriormente distribuir estas señales a la capa de salida.

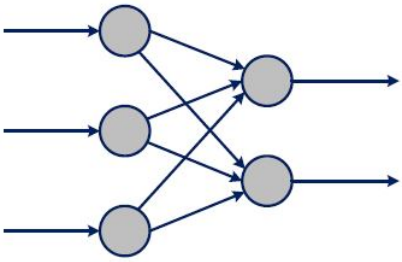


Figura 9: Red neuronal simple.[5]

Redes multicapa: Esta es una red neuronal simple, esta conformada por 3 capas, 2 de estas son visibles, la de entrada y de salida además de contar con una capa intermedia que se encuentra oculta, esta capa oculta dota de la red la capacidad de aprender, al no tener acceso a la capa oculta, es necesario el uso de algoritmos de aprendizaje para ajustar los parámetros internos de la red.

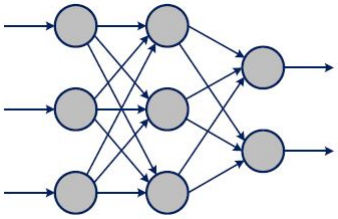


Figura 10: Red neuronal multicapa.[6]

2.5.1.1. Funciones de activación

Como se mencionó anteriormente, para que una neurona artificial pueda generar una salida se requiere que la entrada total supere un umbral. Por lo que para hacer esto es necesario aplicar una función de activación sobre $y_j^{(in)}$ o en algunos casos donde recibe el nombre de x . Entre las funciones de activación más comunes se encuentran:

- Función sigmoidal
- Función ReLU
- Función Leaky ReLU
- Función softmax

[5]Redes neuronales & deep learning. Independently published. p.205

[6]Redes neuronales & deep learning. Independently published. p.206

Función sigmoidal

En la función de activación sigmoidal (Figura 11) el valor de salida de la función es cercano a uno de los valores asintóticos por lo que la salida está comprendida en la zona alta o baja de la sigmoidal, esta función toma valores comprendidos entre 0 y 1. Esta función es muy utilizada para cuando se realizara el entrenamiento de una red para reconocimiento de patrones y clasificación y es la cual será utilizada para el desarrollo de la parte práctica.

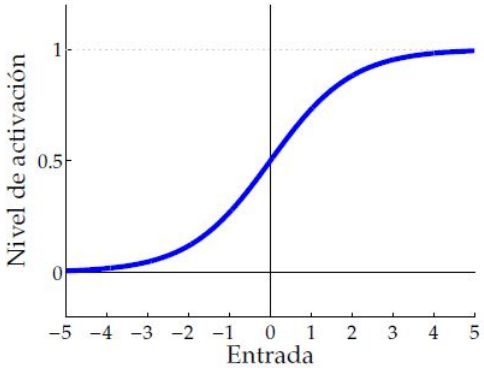


Figura 11: Gráfica de la función de activación sigmoidal.[7]

La función sigmoidal se define de la siguiente manera:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \tag{3}$$

Esta función es utilizada principalmente en implementaciones básicas de redes neuronales con pocas capas como lo son redes neuronales para regresión logística.

El principal problema que se tiene con esta función de activación al ser aplicada en redes profundas es que estas requieren de mayores recursos computacionales al hacer uso de una función transcendental: la función exponencial además de que las neuronas que hacen uso de esta función tienden a saturarse demasiado pronto durante el entrenamiento de la red, lo que implica quedarse atascado en un valor fijo lo cual ralentiza el aprendizaje[21].

Función ReLU

La función de unidades lineales rectificadas mejor conocida como ReLU (Figura 12) por su nombre en inglés es usada frecuentemente en el área de redes neuronales, más comúnmente en el aprendizaje profundo ya que han demostrado un buen desempeño en redes neuronales convolucionales.

[7]Redes neuronales & deep learning. Independently published. p.197

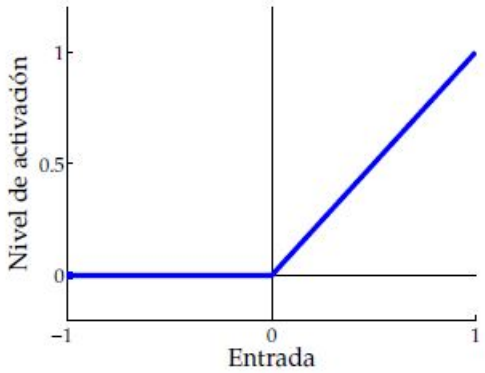


Figura 12: Gráfica de la función de activación ReLU.[8]

Esta se encuentra definida por la función:

$$\sigma(z) = max(0, z) \tag{4}$$

La principal ventaja de esta función de activación es que esta es computacionalmente barata ya que esta no requiere de hacer uso de funciones transcendentales, esta simplemente asigna a la salida un 0 si la entrada es negativa y si es positiva, se mantiene el mismo valor a la salida. Gracias a esto se puede tener un entrenamiento y evaluación más rápida que con funciones como lo son las funciones de activación sigmoidal.

Una de las principales limitaciones de esta función de activación es que estas son incapaces de aprender cuando ingresa las entradas son 0 o valores negativos lo que propicia que al ejecutar el algoritmo de aprendizaje las neuronas con esta función de activación no aporten nada al aprendizaje de la red, cuando esto sucede se dice que las neuronas están muertas. Para resolver esta situación se han propuesto varias variantes de la función ReLU como lo es la función de unidades lineales rectificadas con fugas mejor conocida como leaky ReLU [22].

Función Leaky ReLU

La función de unidades lineales rectificadas con fugas o leaky ReLU (Figura 13) es una variante de la función ReLU que busca solucionar el problema de las neuronas muertas al recibir valores de entrada 0 o negativos.

[8]Redes neuronales & deep learning. Independently published. p.349

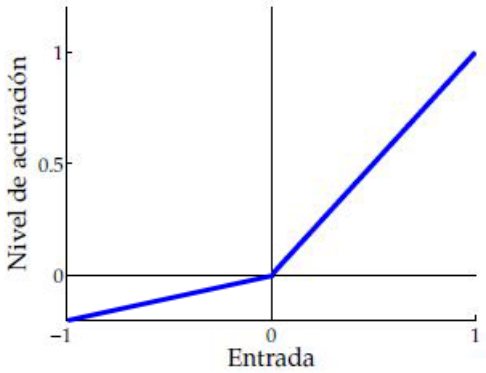


Figura 13: Gráfica de la función de activación Leaky ReLU.[9]

Esta funciona se encuentra definida de la siguiente manera:

$$\sigma(z) = \begin{cases} z & x > 0 \\ cz & x \leq 0 \end{cases} \tag{5}$$

Como podemos ver en la ecuación X esta es una función definida por partes, cuando a nuestra entrada recibimos un mayor valor a cero, a nuestra salida tenemos nuestro mismo valor de entrada pero al recibir un 0 o un número negativo a nuestra entrada, a nuestra salida tenemos nuestro valor de entrada multiplicada por una constante c, esta constante c conocida como factor de fuga, usualmente es usado un valor muy pequeño (e.g 0.01) el cual evitará que la salida de activación sea 0 para prevenir que las neuronas estén "muertas" y puedan seguir aportando al aprendizaje de la red [22].

El principal problema que surge con esta función de activación es que el factor de fuga no produce una salida consistente y proporcional al valor negativo de entrada por lo que el aprendizaje no puede ser fiable al existir la posibilidad de entregar predicciones erróneas. Así como esta existe otras variantes de ReLU que buscan mejorar sus resultados y evitar el problema de las neuronas muertas, estas no se mencionarán ya que, a pesar de existir tantas variaciones, la mejora de desempeño obtenida por estas variaciones con la versión original es ínfima.

Función softmax

La función de activación exponencial normalizada (Figura 23) mejor conocida como softmax, esta es utilizada comúnmente para resolver problemas de clasificación multiclase especialmente en la construcción de redes neuronales convolucionales ya que esta nos da a las salidas una estimación de la probabilidad de que una entrada pertenezca a cada clase.

[9]Redes neuronales & deep learning. Independently published. p.349

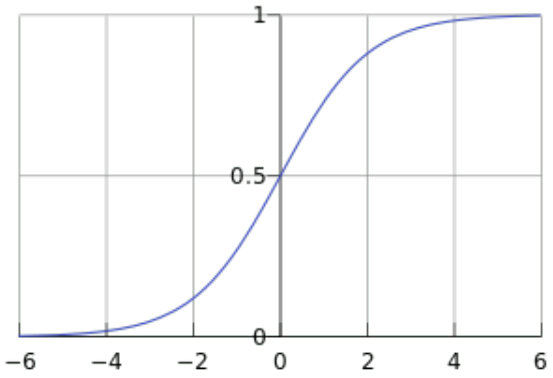


Figura 14: Gráfica de la función de activación Softmax.[10]

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_1^K e^{z_k}} \tag{6}$$

En donde Z es un vector de entrada con los valores de salida de la última capa de la red de tamaño K, K es el número de clases deseadas y la salida σ es un vector de probabilidades de pertenecer a cada una de las clases. El uso de exponenciales permite que todos los valores sean positivos, esta función permite normalizar los pesos de la última capa en valores en la escala [0, 1] los cuales representan la probabilidad de pertenecer a cada una de las clases definidas en una red, al ser probabilidades la suma de éstas es 1 [22].

2.6. Deep Learning

Gracias al conocimiento obtenido del desarrollo del área de aprendizaje automático se concibieron las redes neuronales artificiales las cuales mostraron buen desempeño para la realización de tareas básicas como clasificación o regresión, pero eventualmente las arquitecturas propuestas llegaron a su límite lo que propició el surgimiento de una nueva área de trabajo derivada del aprendizaje automático e inteligencia artificial (Figura 15, esta es el aprendizaje profundo.

[10] Recuperada de `https://danerineuronal.wordpress.com/2017/09/16/ml-04-red-neuronal-basica/`

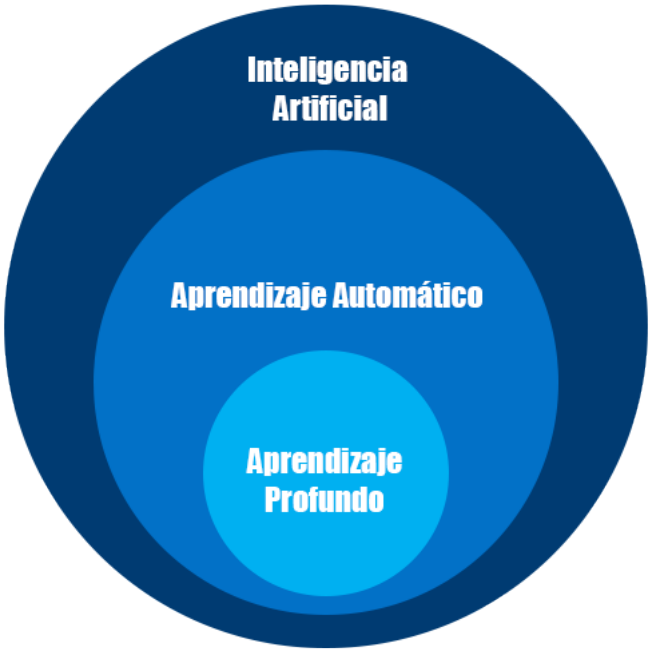


Figura 15: Esquema de áreas de IA.[11]

El aprendizaje profundo es un área de trabajo del aprendizaje automático e inteligencia artificial el cual está basado en el uso de redes neuronales artificiales profundas con el fin de imitar el aprendizaje humano. Esta área surgió de la necesidad de mejorar el desempeño de las redes neuronales tradicionales para realizar ciertas tareas como lo son el reconocimiento de voz, reconocimiento de imágenes, procesamiento natural del lenguaje entre otras más. El aprendizaje profundo nos permite enseñar a las máquinas cómo hacer tareas complejas sin la necesidad de programarles explícitamente la solución, debido a esto se desarrolló una nueva topología de red, las redes neuronales profundas [23].

Redes profundas

Este tipo de redes son las más utilizadas actualmente ya que usan las técnicas proporcionadas por el aprendizaje profundo, la diferencia de una red profunda con una multicapa es que la red profunda puede tener múltiples capas ocultas lo que les permite aprender mucha más información e información más completa. Estas también requieren de algoritmos de aprendizaje para poder actualizar los parámetros internos de cada una de las capas ocultas que la conforma.

[11] Elaboración propia.

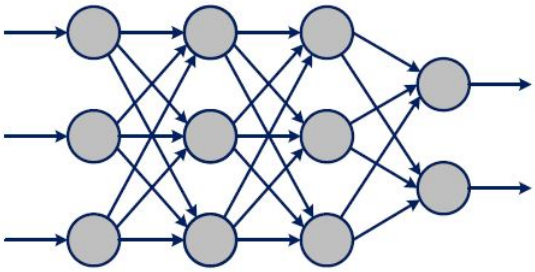


Figura 16: Red neuronal profunda.[12]

Las principales ventajas que se tienen con el aprendizaje profundo son:

1. Simplicidad: Las redes profundas ofrecen elementos de construcción como lo son bloques básicos, capas predefinidas, las cuales son repetidas a lo largo de la arquitectura de la red por lo que no es necesario colocar elemento por elemento.

2. Escalabilidad: Los modelos de aprendizaje profundo pueden funcionar correctamente con grandes conjuntos de datos.

3. Transferencia de dominio: Un modelo entrenado para una tarea puede ser utilizado para tareas relacionadas ya que las características aprendidas son lo suficientemente generales para funcionar con otras tareas en donde no se tienen suficientes datos.

Gracias a todas las bondades que ofrece el aprendizaje profundo se han desarrollado diversas técnicas las cuales han permitido obtener excelentes resultados para tareas como: procesamiento de imágenes, procesamiento natural del lenguaje, conducción autónoma, entre otras. Con respecto al procesamiento de imágenes es posible realizar diversas tareas entre los cuales se tienen la clasificación y localización de objetos en una imagen o secuencia de video. Para lograr este objetivo surgieron las redes neuronales convolucionales las cuales fueron inspiradas en el funcionamiento de la corteza visual del cerebro de los mamíferos [22].

2.7. Redes Neuronales Convolucionales

Las redes neuronales convolucionales conocidas más comúnmente como CNN por sus siglas en inglés, estas son un tipo de arquitectura de redes neuronales profundas las cuales están diseñadas especialmente para trabajar con datos en varias dimensiones como lo son las imágenes.

Este tipo de redes funcionan de manera similar a una red neuronal básica, su diferencia principal radica en que estas han demostrado ser adecuadas para aprovechar

[12]Redes neuronales & deep learning. Independently published. p.205

la estructura bidimensional de los pixeles de una imagen especialmente ya que en las imágenes es importante la relación de adyacencia entre pixeles por lo que una red neuronal convolucional es capaz de capturar con éxito las dependencias espaciales y temporales de una imagen a través del uso de diferentes filtros [23].

Comúnmente el aprendizaje realizado por una red neuronal convolucional para clasificación de imágenes es de tipo supervisado, para esto es necesario etiquetar la información de entrada para poder tener un aprendizaje haciendo un mapeo entre la entrada y la etiqueta. Para lograr esto la red aprende características de las imágenes con las que se entrena, desde características de bajo nivel como lo son bordes o color hasta características de alto nivel como lo son formas. En la Figura 17 podemos observar un ejemplo del proceso que lleva a cabo una CNN para clasificar una imagen.

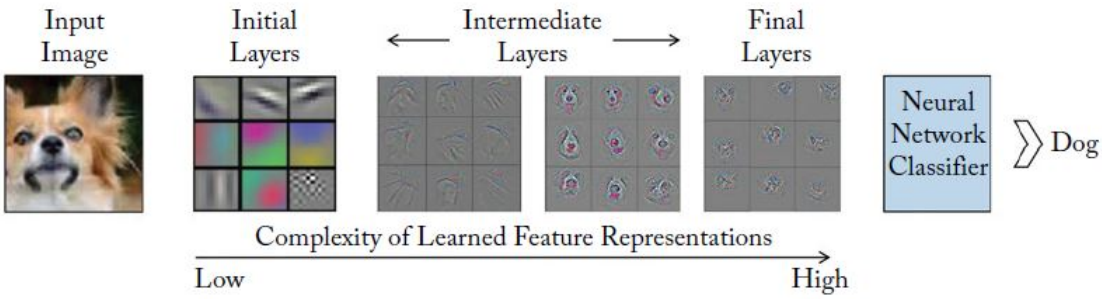


Figura 17: Ejemplo de clasificación de imagen con CNN.[13]

Una red neuronal convolucional está compuesta por varios bloques de construcción básicos llamadas capas o layers. Una CNN está conformada por Capas convolucionales o CNN Layers, Capas pooling, una capa flatten y capas completamente conectadas o FC Layers, a continuación, se explicarán las funciones de cada una de estas:

Un filtro es una matriz de dimensiones $n_k \times n_k$ usualmente pequeña como se muestra en la Figura 18. Los pesos de cada filtro son aprendidos durante el entrenamiento de la CNN, estos son inicializados de manera aleatoria durante el entrenamiento. Estos filtros son aplicados sobre la imagen mediante una operación llamada convolución.

1	0
0	1

Figura 18: Filtro de 2 x 2.[14]

[13] A guide to convolutional neural networks for computer vision. Synthesis Lectures on Computer Vision. p.44

La convolución es una operación matemática comúnmente utilizada en el área de procesamiento de imágenes, la convolución es realizada entre dos matrices (la entrada y el filtro) las cuales son multiplicadas elemento a elemento y posteriormente los números multiplicados son sumados, el filtro se desliza a lo largo del ancho y del alto de la imagen de entrada y este proceso continua hasta que ya no puede deslizarse más sobre la entrada lo que nos da como resultado una nueva matriz conocida como mapa de características, esto se puede observar en la Figura 19.

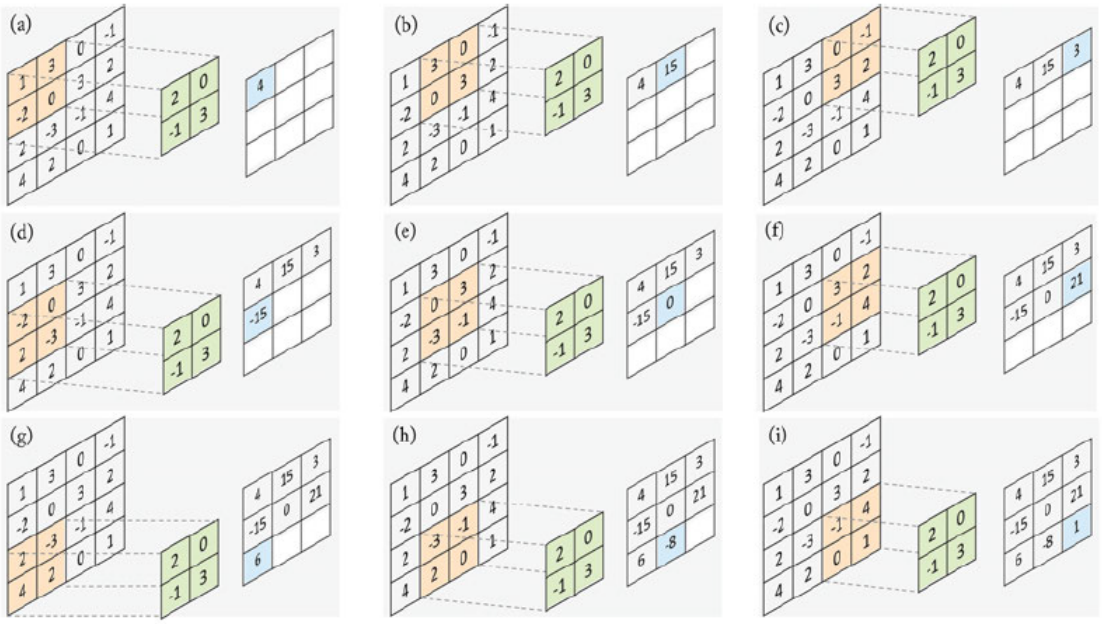


Figura 19: Ejemplo de aplicación de convolución.[15]

En el ejemplo de la Figura 19 se puede observar que para calcular cada valor del mapa de características el filtro da un paso a lo largo de la posición horizontal o vertical, este término es conocido como **stride o paso**, además de eso se puede observar que la dimensionalidad del mapa de características se redujo con respecto a la entrada, esto es debido al tamaño del filtro y el paso utilizado, el tamaño de salida esta dado por el tamaño de la entrada $h \times w$, el paso es s y el tamaño del filtro es f:

$$h' = \lfloor \frac{h - f + s}{s} \rfloor, w' = \lfloor \frac{w - f + s}{s} \rfloor \tag{7}$$

Si es necesario que el tamaño de salida del mapa de características se mantenga del mismo tamaño a la entrada es necesario aplicar **zero-padding o rellenado por ceros**

[14]Elaboración propia.

[15]A guide to convolutional neural networks for computer vision. Synthesis Lectures on Computer Vision. p.47

alrededor del mapa de características, esto nos permite incrementar las dimensiones de la salida, el tamaño de salida esta dado por el tamaño de la entrada $h \times w$, el paso es s y el tamaño del filtro es f y p es el rellenado:

$$h^{'} = \lfloor \frac{h - f + s + p}{s} \rfloor, w^{'} = \lfloor \frac{w - f + s + p}{s} \rfloor \tag{8}$$

Capas de pooling (Pooling Layers) Esta capa tiene como objetivo disminuir el costo computacional necesario para procesar los datos, esto se realiza mediante la reducción de dimensionalidad del mapa de características de entrada. Esto además permite extraer las características dominantes del mapa de características las cuales son rotacional y posicionalmente invariantes. Para esta capa es necesario especificar la región de pooling de tamaño $f \times f$ y el paso, la región de pooling se desliza a lo alto y ancho de la entrada con un paso s definido, el tamaño de salida está dado por la ecuación 7.

La región de pooling al deslizarse sobre la entrada puede ejecutar dos operaciones diferentes, el valor máximo o el promedio de la región, estos tipos de pooling son llamados: Max pooling y Average pooling. Max pooling regresa el valor máximo de la porción de la entrada cubierta por región mientras que Average pooling calcula el promedio de los valores que se encuentran dentro de la región. En la Figura 20 se puede observar un ejemplo de cómo se ejecuta el pooling de región 2x2 y paso 1 para reducir el mapa de características a un tamaño 3x3.

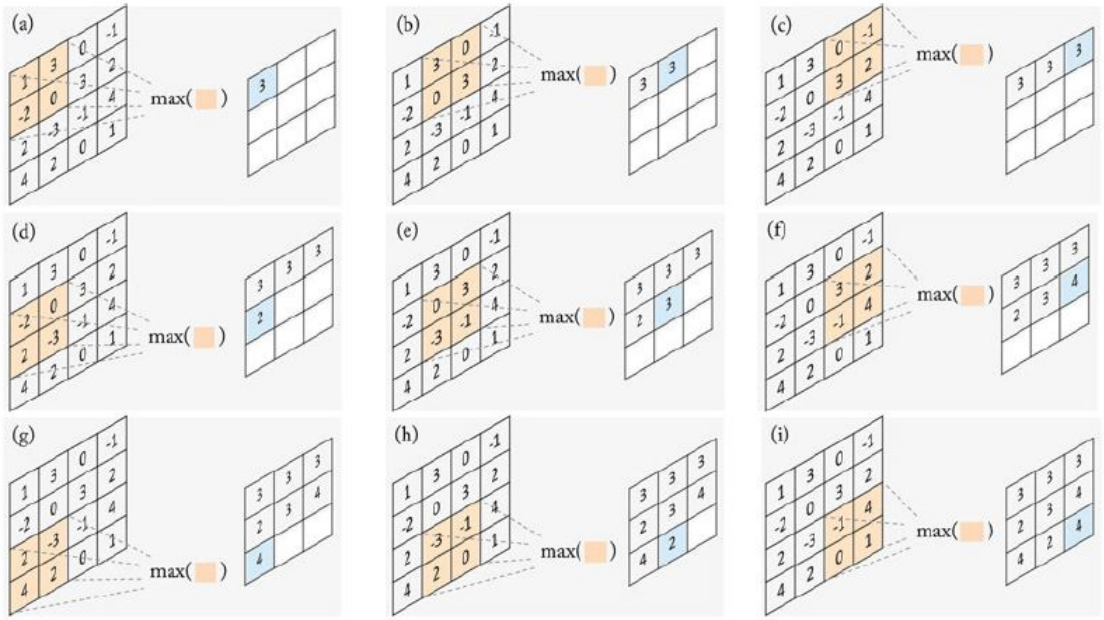


Figura 20: Operaciones realizadas en la capa Max pooling para reducción de dimensionalidad.[16]

Capa Flatten (Flatten Layer)

Esta capa se implementa usualmente a la salida de la última capa convolucional o capa pooling, esta capa tiene como fin convertir el último mapa de características en un vector con el fin de que este pueda ser posteriormente ingresado a una red completamente conectada para realizar el paso de clasificación de la imagen. Se puede observar un ejemplo en la Figura 21.

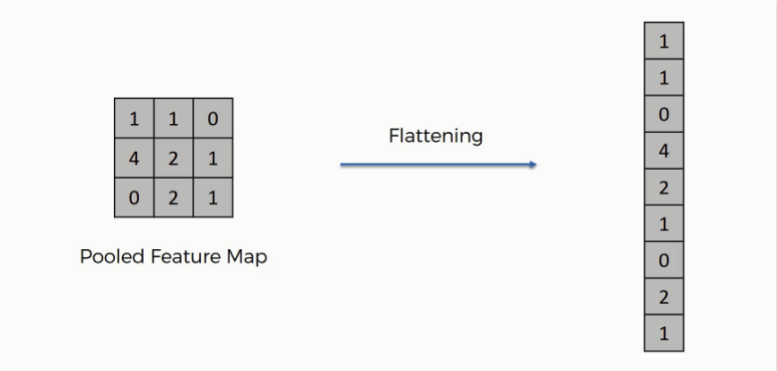


Figura 21: Conversión de mapa de características a vector.[17]

Capas completamente conectadas (Fully Connected Layers)

Esta capa tiene como objetivo el funcionar como un clasificador, para esto son colocadas varias capas completamente conectadas, en esta cada neurona de cada capa está conectada a cada una de las neuronas de la siguiente capa, esto significa que cada neurona influye en la activación de las neuronas a las que se encuentra conectada. Es necesario entrenar también esta red para que pueda desempeñarse como clasificador. En la Figura 22 se puede observar la arquitectura conformada por la capa flatten y las capas completamente conectadas y el resultado de la clasificación al aplicar una activación softmax en las neuronas de salida.

[16] A guide to convolutional neural networks for computer vision. Synthesis Lectures on Computer Vision. p.53

[17] Recuperada de `https://www.superdatascience.com/convolutional-neural-networks-cnn-step-3-flattening/`

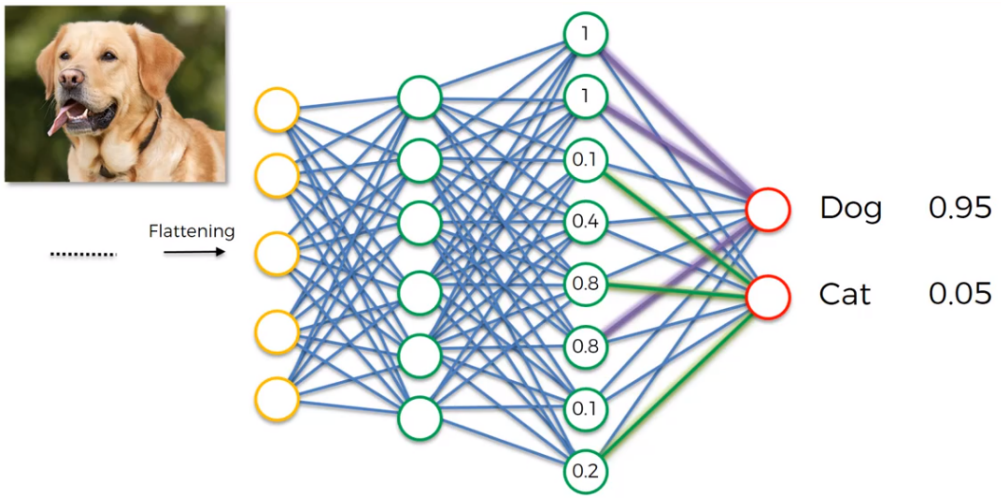


Figura 22: Arquitectura formada con las capas completamente conectadas.[18]

[18]Recuperada de `https://www.andreaperlato.com/aipost/cnn-and-softmax/`

Capítulo 3

Análisis y Diseño

En este capítulo se verá todo lo relacionado con el análisis y diseño del prototipo móvil, para conocer qué elementos lo conforman, tanto en hardware como en software. En la primera parte se identificarán los requerimientos que se tienen para el prototipo en móvil en función a los objetivos que tiene que cumplir este. Posteriormente se procederá a realizar el diseño del hardware y del software mediante su representación en diagramas. A continuación, se mostrará cómo se lleva a cabo la integración del hardware con el software. Y por último se presentarán las propuestas posibles para el sistema de visión artificial que serán implementadas en el prototipo móvil.

3.1. Análisis de requisitos

En función a los objetivos establecidos para este trabajo, a continuación, se muestran los requisitos que debe de cumplir el prototipo móvil para poder lograrlos. Con respecto al hardware se tiene que el prototipo debe de contar con las siguientes características:

1. El prototipo debe de tener un sistema mecánico que le permita ser capaz de desplazarse en todas las direcciones posibles en superficies no rugosas con una inclinación máxima de 15°.

2. Debe de ser capaz de censar su entorno, con esto se refiere a poder medir las variables existentes en el entorno en el que se desempeñará como por ejemplo la distancia hacia los objetos u obstáculos que se le presenten.

3. El prototipo debe de poseer un manipulador mecánico el cual le permitirá tomar los objetos detectados para su posterior transporte.

4. Debe de llevar una tarjeta de desarrollo la cual funcionará como maestro, la cual debe tener las capacidades suficientes para poder ejecutar un sistema de IA de manera fluida, esta se encargará de ejecutar el sistema de visión artificial además de mandar las órdenes para que el prototipo pueda manejarse en un entorno controlado.

5. Está obligado a montar una cámara de video la cual será utilizada para alimentar al sistema de visión artificial.

6. Debe de llevar una tarjeta secundaria que funcionará como esclavo, la cual se encargará de enviar las órdenes recibidas por el maestro, esta acoplada a los sensores, motores y actuadores con los que contará el prototipo móvil.

7. Debe de poseer un sistema de alimentación móvil el cual otorgue la energía suficiente para que el prototipo móvil pueda funcionar de manera ininterrumpida durante una cantidad de tiempo finita.

Con respecto al software este debe de cumplir los siguientes requisitos:

1. Todo el software debe de ser ejecutado bajo el sistema operativo Ubuntu 18.04.

2. El código implementado debe de estar bajo el lenguaje de programación Python ya que es el lenguaje con más herramientas disponibles de ciencia de datos e inteligencia artificial actualmente.

3.2. Diseño

Una vez identificados los requerimientos para el prototipo móvil se procederá a indicar cada uno de los elementos utilizados tanto para el hardware como para el software como se puede observar a continuación.

3.2.1. Diagrama de hardware y sofware

En la Figura A se puede observar un diagrama en el que se ve de manera breve como se encuentra interconectado el hardware del prototipo móvil.

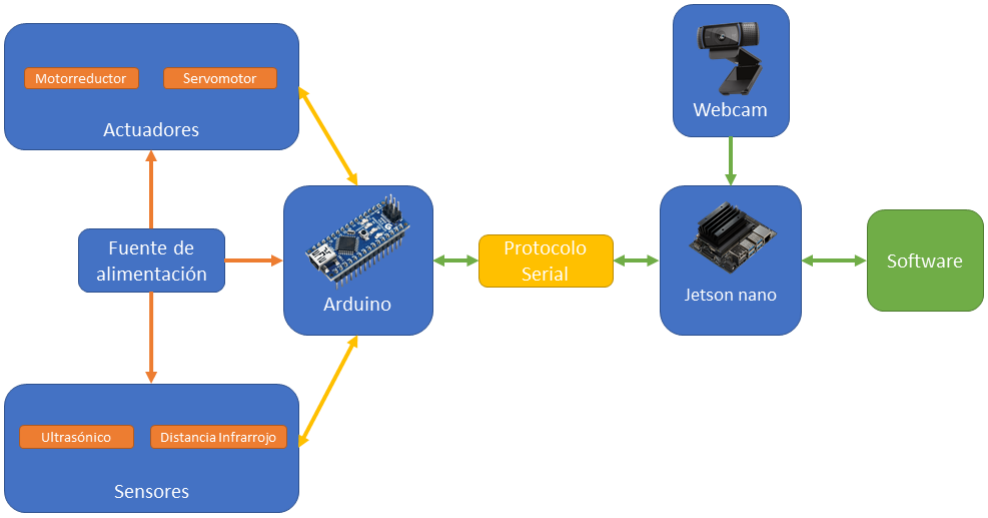


Figura 23: Diagrama de hardware.[2]

Con respecto a los sensores que serán utilizados, se tienen dos tipos de sensores de distancia, unos de tipo ultrasónico los cuales ayudarán a medir el entorno para determinar si existe algún objeto con el que se pueda colisionar, estos son los sensores SR-04 como se puede observar en la Figura 24. Este tipo de sensor tiene un rango de medición de 2 a 450 cm aproximadamente con una precisión de +- 3mm.

Figura 24: Sensor de distancia ultrasónico.[3]

Además de este, se tiene otro tipo de sensor de distancia como el que se muestra en la Figura 25, este el sensor de distancia infrarrojo el cual será utilizado para medir la distancia desde el manipulador de objetos hasta el objeto a manipular. Esto proporcionará al sistema de control un valor de referencia para poder acercarse al objeto y tomarlo, este sensor puede medir distancias de 10 a 180 cm, se hace uso de este sen-

[2]Elaboración propia.

[3]Recuperadas de `https://naylampmechatronics.com/741-large_default/sensor-ultrasonido-hc-sr04.jpg`

sor por su reducido tamaño además de su rápida respuesta comparada con el sensor ultrasónico.

Figura 25: Sensor de distancia infrarrojo.[4]

Posteriormente en el apartado de actuadores se hará uso de un chasis con un dispositivo tractor oruga el cual permitirá el desplazamiento de manera estable del prototipo móvil aun en terrenos irregulares. Este se puede observar en la Figura 26.

Figura 26: Chasis de prototipo.[5]

Dentro de este chasis, en las orugas serán montados dos motorreductores como los mostrados en la Figura 27 los cuales darán al prototipo la capacidad de carga de hasta 5kg.

Figura 27: Motorreductor.[6]

[4]Recuperadas de `https://images-na.ssl-images-amazon.com/images/I/5131pBVhuUL.__AC_SY300_QL70_ML2_.jpg`

[5]Recuperadas de `https://imgaz1.staticbg.com/thumb/large/oaupload/banggood/images/CB/FB/140b9459-af50-48ee-a0c7-aee7d372dcfa.jpg.webp`

Con respecto al manipulador, se tendrá un brazo robot de 4 grados de libertad como se puede observar en la Figura 28. Esto permitirá que tenga un mayor rango de movimiento para poder manipular al objeto.

Figura 28: Brazo robot.[7]

Dentro de este brazo se colocan 5 servomotores como los mostrados en la Figura 29 los cuales dotarán al brazo del torque necesario para poder agarrar y levantar al objeto.

Figura 29: Servomotor.[8]

Una vez descritos los elementos anteriores es importante mencionar que para poder mandar las señales necesarias para operar los sensores y los actuadores es requerido de una tarjeta de desarrollo, para este trabajo se hará uso de una tarjeta Arduino Nano como se muestra en la Figura 30 la cual lleva un microcontrolador programable y la

[6]Recuperadas de `https://rees52-fbcb.kxcdn.com/8978-thickbox_default/25mm-9v-dc-reduced-gear-motor-for-robot-with-hall-sensor-diy-toy-car-tank-chassis-rs1562.jpg`

[7]Recuperadas de `https://imgaz2.staticbg.com/thumb/large/oaupload/banggood/images/F2/1A/35482e98-85cc-4729-afea-3ee24460b1ac.jpg`

[8]Recuperadas de `https://imgaz3.staticbg.com/thumb/large/oaupload/banggood/images/CE/15/234b8cc5-f3b4-47dc-aa0f-070ff14e3f56.jpg.webp`

cual es diseñada específicamente para proyectos electrónicos y de robótica. Esta tarjeta cuenta con distintos protocolos de comunicación y puede funcionar como maestro o como esclavo, para este caso funcionará como esclavo ya que esta ejecutará las órdenes recibidas por parte del sistema de visión artificial, de navegación y de control del prototipo mediante comunicación serial.

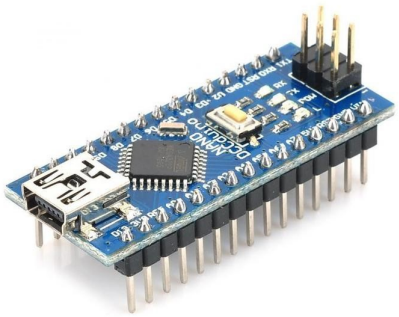


Figura 30: Tarjeta de desarrollo Arduino Nano.[9]

Como maestro se hace uso de la tarjeta de desarrollo Jetson Nano (Figura 31), esta será la encargada de ejecutar los sistemas de visión artificial, de navegación y de control del prototipo. Se hace uso de esta tarjeta debido a las capacidades que esta posee, a su reducido tamaño y a su reducido consumo además de que esta tarjeta es desarrollada específicamente para realizar tareas relacionadas con Deep Learning ya que esta puede otorgar hasta 472 GFLOPS de desempeño computacional con operaciones de medio punto flotante[36].

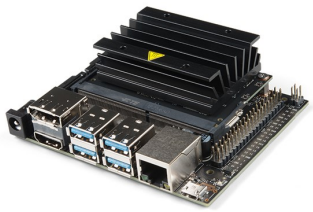


Figura 31: Nvidia Jetson Nano.[10]

[9]Recuperadas de `https://cdn.shopify.com/s/files/1/0557/2945/products/oem-9380-5624042-2-zoom_x700.jpg?v=1546431938`

Entre las principales características[35] con las que cuenta esta tarjeta se encuentran:

- CPU ARM Cortex-A57 de cuatro núcleos
- GPU Nvidia Maxwell con 128 núcleos Nvidia CUDA
- 4 Gb de memoria RAM LPDDR4 a 1600Mhz
- Almacenamiento mediante micro SD hasta 128 GB
- Codificación y decodificación de video hasta 4K hasta 60 cuadros por segundo
- 4 puertos usb 3.0
- 1 puerto HDMI
- Puertos GPIO con soporte para protocolos I2C, SPI, UART
- Consumo de 5W o 10W dependiendo del modo de funcionamiento

A esta tarjeta se le conectará una cámara web (Figura 32) mediante uno de sus puertos USB, esta será utilizada para alimentar al sistema de visión artificial, la cámara cuenta con una resolución máxima de 1080p hasta 30 cuadros por segundo, tiene un campo de visión de 78°. Se escogió esta cámara por su interfaz USB ya que al ser un dispositivo plug and play no requiere de configuración extra, se eligió también por su precio ya que esta cámaras es asequible además de su capacidad de ser configurada en distintas resoluciones estándar como lo son 1080p, 720p y 480p.

Figura 32: Cámara web.[11]

[10]Recuperadas de `https://cdn.sparkfun.com//assets/parts/1/4/9/4/6/16271-NVIDIA_Jetson_Nano_Developer_Kit__V3_-01.jpg`

[11]Recuperadas de `https://aludrastorage.blob.core.windows.net/aludra-files/2/Product/6A94CE6B-C67C-4507-89A1-27613C86103B/original/26052020212245.jpg`

Para el desarrollo del software necesario para que el prototipo funcione de manera correcta se propone realizarlo de la manera como se muestra en la Figura 33.

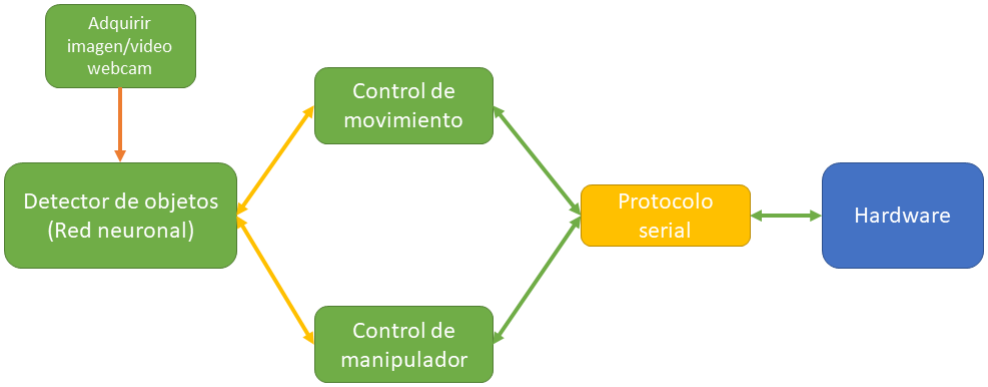


Figura 33: Diagrama de funcionamiento del software.[12]

Se tiene primero que un módulo del software estará encargado de realizar la conexión con la cámara web además de la adquisición de imágenes. Posteriormente esta imagen será pasada al siguiente módulo el cual estará encargado de ejecutar el detector de objetos en función a la imagen pasada por la cámara, dependiendo del estado en que se encuentre el robot podrá tomar dos acciones diferentes. La primera acción llevará al módulo encargado de controlar el desplazamiento del prototipo, este movimiento se realizará para la búsqueda de objetos, marcadores o para acercarse lo más posible al objeto a manipular. La segunda acción se controlará cuando el prototipo se encuentre cerca del objeto identificado, tendrá como objetivo ejecutar las acciones de control necesarias para poder tomar el objeto. Todo esto será transmitido mediante comunicación serial al Arduino Nano el cual está encargado de ejecutar las acciones propuestas por el software.

Para la adquisición de imágenes con la cámara web se hará uso de la librería OpenCV 4.4[C] la cual es una librería libre de visión artificial la cual funciona con lenguajes de programación como: C++, Python, Java y JavaScript. Para este trabajo se hace uso de la librería para python. Esta nos permitirá inicializar la cámara web y posteriormente acceder al stream de video.

Para el detector de objetos se hace uso de la librería TensorFlow 1.14[38] la cual es una librería de código abierto para aprendizaje automático que puede ser ejecutada tanto en CPU como en GPU, esta se usa con el fin de construir y entrenar redes neuronales para distintas tareas como reconocimiento de voz, análisis de sentimientos, detección de fraudes, detección de movimiento o reconocimiento de imágenes. Para este trabajo nos enfocaremos en hacer uso de Tensorflow para GPU para reconocimiento de imágenes. El entrenamiento de la red neuronal para la detección de objetos hará uso

[12]Elaboración propia.

de TensorFlow Model Zoo [39] el cual provee de una colección de modelos de detección pre-entrenados en distintos datasets de imágenes.

Para la comunicación serial de la Jetson Nano con la tarjeta Arduino nano se hace uso de la librería pySerial[40] la cual puede ser ejecutada tanto el Windows, Linux o OSX, esta nos permitirá acceder al puerto serial de la Jetson en la que se encuentra conectada la tarjeta secundaria.

3.2.2. Integración de hardware y sofware

Con lo anterior ya definido entonces se propone la integración del hardware y software como se muestra en la Figura 34.

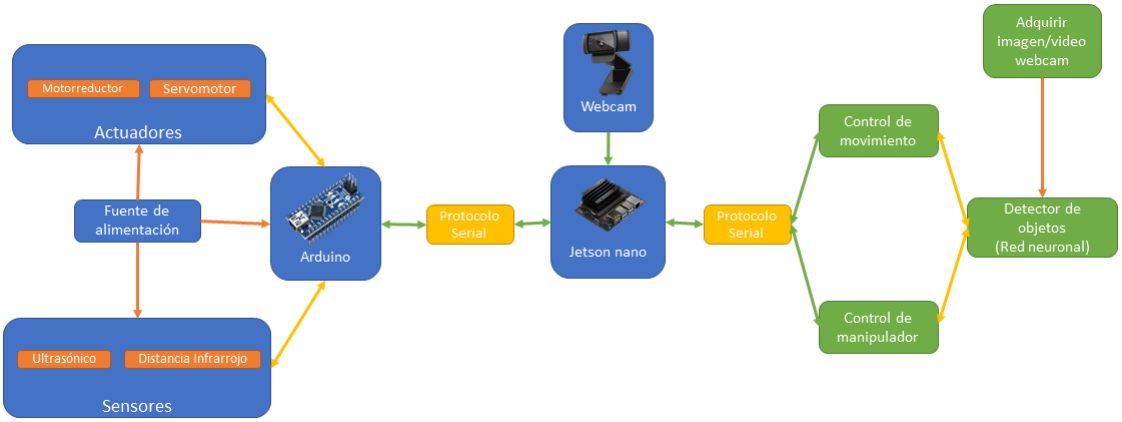


Figura 34: Diagrama de integración de hardware con software.[13]

La conexión del hardware y del software se realizará mediante el modelo maestro-esclavo como se mencionó anteriormente, y mediante comunicación serial se tendrá una comunicación del software de la tarjeta encargada de ejecutar el sistema de IA la cual funcionará como maestro, esta enviará las órdenes mediante puerto serial al esclavo, la tarjeta Arduino Nano y está a su vez mandara las órdenes necesarias para solicitar la lectura de algún sensor o activar alguno de los actuadores con los que cuenta el prototipo y posteriormente responderá al maestro que realizó su solicitud.

3.3. Propuestas posibles

Con lo anterior se definieron los requerimientos que tiene que cumplir el prototipo para que este pueda funcionar de manera óptima, se indicaron con que elementos contará

[13] Elaboración propia.

el hardware y software. Para el sistema de visión artificial se tienen dos propuestas de arquitecturas:

- Faster R-CNN, detector basado en propuesta de regiones.
- SSD, detector de una sola toma.

Ambas arquitecturas serán implementadas en el prototipo, estas serán evaluadas y comparadas a fin de determinar cuál de estas es la óptima para el funcionamiento del prototipo.

Capítulo 4

Implementación

En este capítulo haremos una breve descripción de qué es un sistema de visión artificial y cuáles son los elementos y conceptos básicos que están presentes en este. Posteriormente se hará un análisis extenso y detallado del funcionamiento de dos arquitecturas de visión artificial para detección de objetos: Faster R-CNN y SSD las cuales toman aproximaciones diferentes para resolver el problema. Y por último se presenta el sistema de marcadores el cual le permitirá al prototipo ubicarse en un espacio determinado, se explicará el algoritmo propuesto para lograr este objetivo.

4.1. Sistema de visión artificial para basura

Debido al surgimiento de grandes conjuntos de datos de imágenes como lo es ImageNet y con el auge del aprendizaje profundo surgió la necesidad de crear sistemas de visión artificial con la finalidad de lograr estas tareas: tener la capacidad de clasificar los objetos que se encuentran dentro de una imagen además de determinar en qué parte se encuentran localizados estos, estas tareas se denominaron como detección y la localización de objetos.

De manera formal se define que la detección de objetos es la tarea de detectar objetos presentes en una fotografía mientras que la localización de objetos es la tarea la cual consiste en identificar en que parte de la imagen existen los objetos y dibujar una caja delimitadora mejor conocida como "bounding box" alrededor de cada objeto. En la Figura 35 se puede ver un ejemplo de cómo se ve la detección y localización en una imagen estándar. En estas se puede observar el objeto encerrado dentro del bounding box, se ve la etiqueta la cual indica la clase a la cual pertenece el objeto y por último se observa la probabilidad de pertenecer a esa clase.

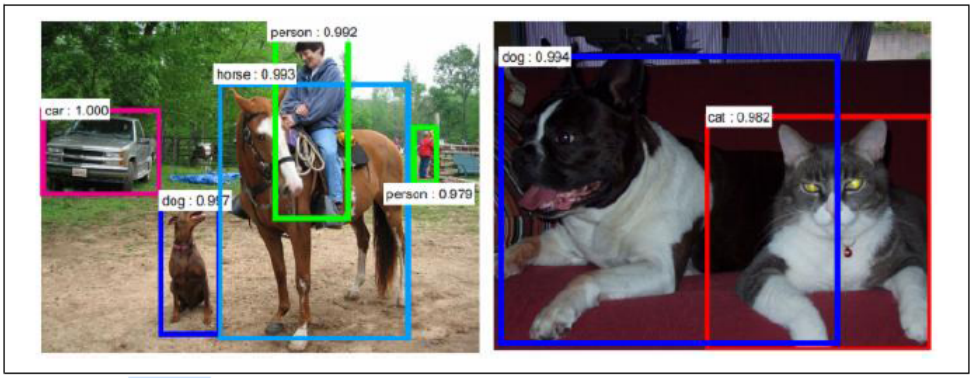


Figura 35: Ejemplo de objetos detectados y localizados en la imagen. [2]

4.1.1. Arquitecturas de detectores y localizadores de objetos

Actualmente para poder resolver estas tareas se han realizado varios intentos para abordar este problema usando redes neuronales convolucionales (CNNs). Con respecto a las arquitecturas que existen actualmente se encuentran aquellas que usan la aproximación mediante el uso de propuestas de regiones como lo son R-CNN con sus respectivas mejoras que son Fast R-CNN y posteriormente Faster R-CNN. Otra aproximación diferente a esta son los detectores de objetos basados en regresión también conocidos como detectores de una sola toma, entre los principales se encuentran SSD y YOLO.

4.1.1.1. Faster R-CNN

Faster R-CNN es una arquitectura compleja de redes neuronales profundas que tiene como fin realizar la detección y localización de objetos dentro de una imagen. A continuación, se describirá de manera general como funciona esta arquitectura para posteriormente entrar a detalle en cada uno de sus componentes que la conforma. El proceso realizado por esta arquitectura de manera simple consiste en: ingresar una imagen y de esta se busca obtener: una lista de bounding boxes de los objetos presentes, una etiqueta para cada uno de estos objetos la cual indica a que clase pertenece y por último la probabilidad de pertenecer a la clase.

[2] Ramsundar, B. and Zadeh, R., n.d. Tensorflow For Deep Learning. p.127.

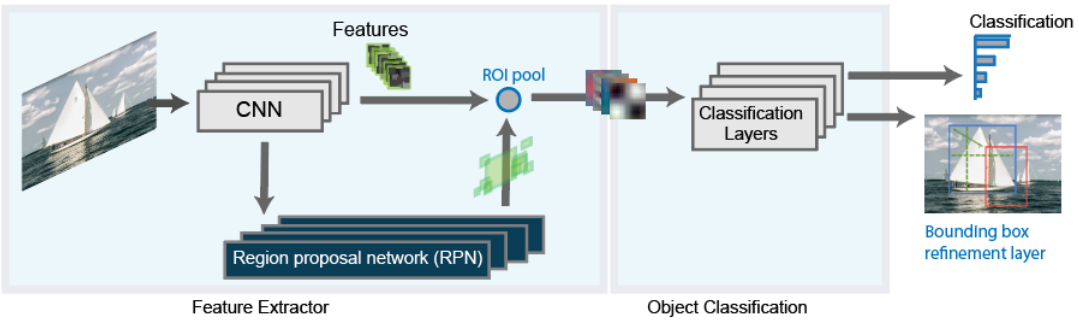


Figura 36: Elementos que conforman la arquitectura Faster R-CNN.[3]

En la Figura 36 se puede observar primeramente que los elementos presentes en la arquitectura se encuentran separados en dos: el extractor de características y el clasificador de objetos, dentro del extractor de características se parte del ingreso de una imagen y sus elementos consecuentes como lo son la CNN la cual nos dará como resultado uno o varios mapas de características, la red de propuesta de regiones conocida como RPN haciendo uso de las características obtenidas por la CNN obtendrá un número predefinido de regiones las cuales podrían contener un objeto, la capa de regiones de interés o ROI la cual nos permitirá determinar si existe algún objeto relevante en función a lo obtenido por la CNN y la RPN para posteriormente pasar al clasificador de objetos el cual puede ser una red neuronal completamente conectada la cual se encargada de clasificar o descartar el contenido del bounding box y ajustar las coordenadas de esta. Posteriormente se llega a la salida con los resultados los cuales arrojan la clase y la bounding box de una o varias instancias presentes en la imagen.

El primer paso para poder hacer uso de esta arquitectura es entrenar una CNN en caso de tener una gran cantidad de imágenes para su entrenamiento, en caso contrario se hace uso de una CNN pre-entrenada en la tarea de clasificación de imágenes haciendo uso de algún dataset ya existente (como ImageNet[25] o COCO[26]).

La arquitectura originalmente propuesta por Faster R-CNN hace uso de la arquitectura VGG-16[29] con el fin de obtener el mapa de características. Para este trabajo de tesis se hace uso de la CNN Inception pre-entrenada con el dataset COCO.

En la Figura 37 se muestra un diagrama simplificado con la arquitectura de Inception V1[28] la cual será desglosada posteriormente para una mejor comprensión de esta.

[3]Recuperada de `https://la.mathworks.com/help//vision/ug/faster.png`

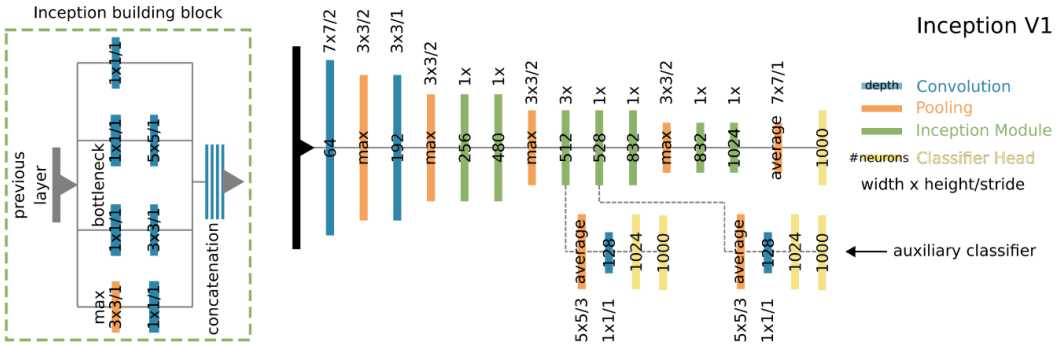


Figura 37: Diagrama simplificado de arquitectura Inception V1.[4]

En la Figura 38 se puede observar un poco más a detalle la topología de Inception V1 para poder conocer detalles de esta como el tamaño de salida obtenido para cada capa, la cantidad de parámetros internos de la red y número de operaciones realizadas.

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

Figura 38: Topología interna de Inception V1.[5]

[4]Hoeser, T., & Kuenzer, C. (2020). Object Detection and Image Segmentation with Deep Learning on Earth Observation Data: A Review-Part I: Evolution and Recent Trends. Remote Sensing, 12(10), 1667.

[5]Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015). Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1-9).

Entre las principales ventajas que tiene Inception para ser elegida para este trabajo de tesis con respecto a otras arquitecturas como ResNet, AlexNet o VGGNet es que Inception requiere una menor cantidad de parámetros entrenables lo que se traduce en una menor cantidad de memoria requerida con respecto a otras CNNs, esto a la vez también se traduce en una menor cantidad de tiempo de entrenamiento requerido además de la reducida cantidad de operaciones de punto flotante requeridos para cada pasada como se puede observar en la Figura 39.

Comparison					
Network	Year	Salient Feature	top5 accuracy	Parameters	FLOP
AlexNet	2012	Deeper	84.70%	62M	1.5B
VGGNet	2014	Fixed-size kernels	92.30%	138M	19.6B
Inception	2014	Wider - Parallel kernels	93.30%	6.4M	2B
ResNet-152	2015	Shortcut connections	95.51%	60.3M	11B

Figura 39: Comparativa entre diversas arquitecturas de CNNs.[6]

Para que Inception pudiera lograr estas ventajas se introdujeron una serie de mejoras con respecto a las otras arquitecturas, entre estas mejoras se tuvo la implementación del uso de convoluciones de 1×1. Esta es usada como un módulo de reducción de dimensionalidad con el fin de reducir la cantidad de operaciones requeridas. En la Figura 40 se puede ver un ejemplo de una convolución de tamaño 5×5.

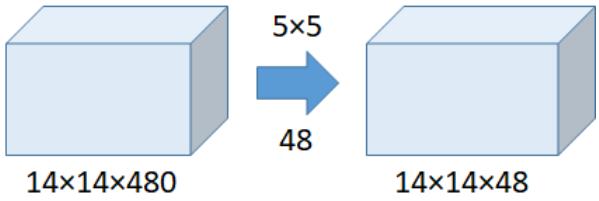


Figura 40: Convolución de 5x5.[7]

Con los datos presentados en la Figura 40 y realizando el cálculo se tienen: $(14 \times 14 \times 48) \times (5 \times 480) = 120,422,400$ total de operaciones a realizar haciendo uso de una convolución de 5×5. Por otro lado, en la Figura 41 se puede ver un ejemplo de la aplicación de una convolución de 1×1 antes de la convolución de 5×5.

[6]Recuperada de `https://miro.medium.com/max/700/1*p-2QjvJ4nDCfn3F5oIxvYA.png`

[7]Recuperada de `https://miro.medium.com/max/422/1*s6_8m_0EpwrzZPunRGgFCQ.png`

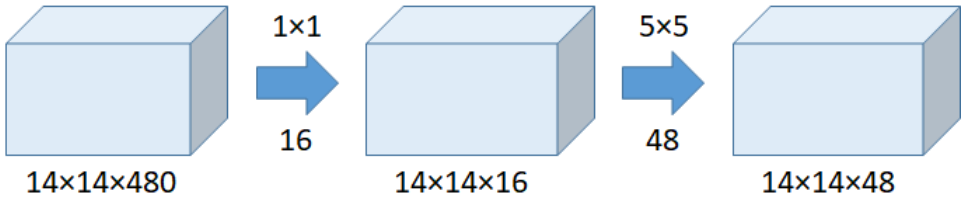


Figura 41: Convolución de 1x1 y de 5x5.[8]

Con los datos presentados en la Figura 41 y realizando el cálculo para la primera convolución se tiene $(14 \times 14 \times 16) \times (1 \times 1 \times 480) = 1,505,280$ operaciones mientras que para la segunda convolución se tienen $(14 \times 14 \times 48) \times (5 \times 5 \times 16) = 3,763,200$ operaciones, por lo que el número total de operaciones es de $1,505,280 + 3,763,200 = 5,268,480$ lo cual es mucho menor que los 120,422,400 operaciones si solo se realizara la convolución de 5×5.

Para la arquitectura Inception otra mejora que se propuso fue la creación de algo conocido como "Módulo Inception" y por la cual esta recibe su nombre, este módulo tiene como objetivo permitir un cómputo más eficiente y la creación de redes más profundas a través de la reducción de dimensionalidad. Originalmente este módulo fue propuesto como se presenta en la Figura 42.

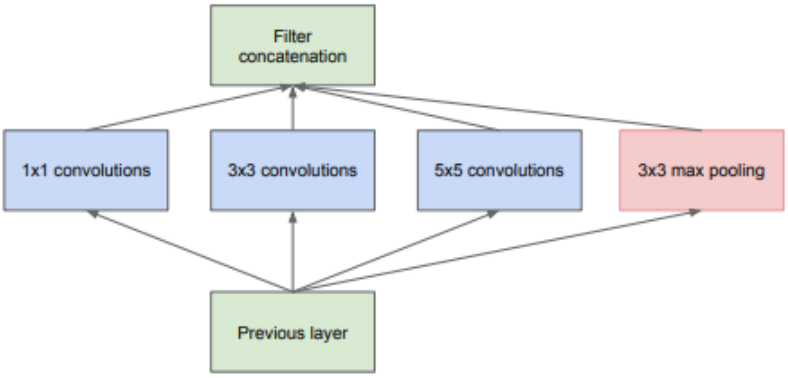


Figura 42: Versión original del módulo Inception.[9]

Al observar detalladamente la versión del módulo original se pueden notar como lo mencionado con el ejemplo de la Figura 40 es que este requeriría de ejecutar una gran cantidad de operaciones. Debido a ello se propuso una nueva versión del módulo

[8] Recuperada de `https://miro.medium.com/max/683/1*SJDJyBGM__wRX9wJYIKv8w.png`

[9] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015). Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1-9).

Inception aplicando la reducción de dimensionalidad como se puede observar en la Figura 43.

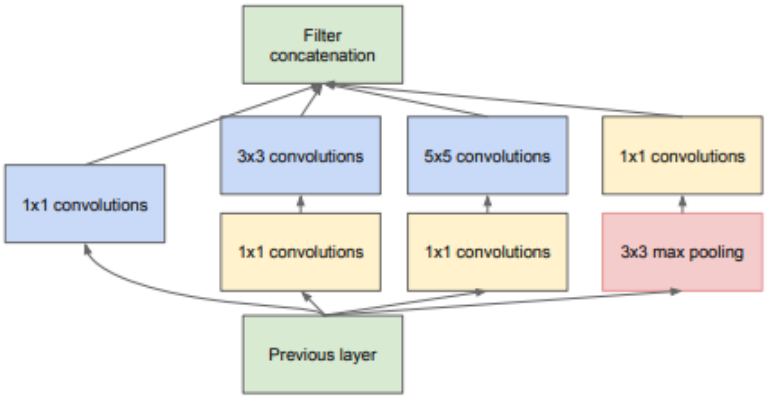


Figura 43: Versión mejorada con reducción de dimensionalidad.[10]

La nueva versión del módulo Inception implementa convoluciones 1×1 antes de las convoluciones de 3×3 y 5×5 con el fin de reducir la cantidad de operaciones requeridas. Se hace uso de diferentes tamaños de convolución con el fin de extraer diferentes tipos de características. Al finalizar el resultado de cada una de las convoluciones realizadas son concatenadas para poder pasar como una sola entrada en los siguientes módulos. En la arquitectura de Inception son usados 9 módulos Inception.

Además de esto la arquitectura implementa en la mitad de su estructura dos clasificadores auxiliares usados exclusivamente para el entrenamiento, el loss obtenido por estos clasificadores auxiliares es añadido al loss total durante el entrenamiento y como se puede observar en la ecuación 9.

$$total\_loss = real\_loss + 0.3 \times aux\_loss\_1 + 0.3 \times aux\_loss\_2 \tag{9}$$

En donde real_loss es el loss obtenido por el clasificador principal de la arquitectura y donde aux_loss_1 y aux_loss_2 es el loss obtenido por cada uno de los clasificadores auxiliares respectivamente, estos son añadidos con un peso de 0.3. En la Figura 44 se puede observar la arquitectura que lleva cada uno de los clasificadores auxiliares.

[10]Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015). Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1-9).

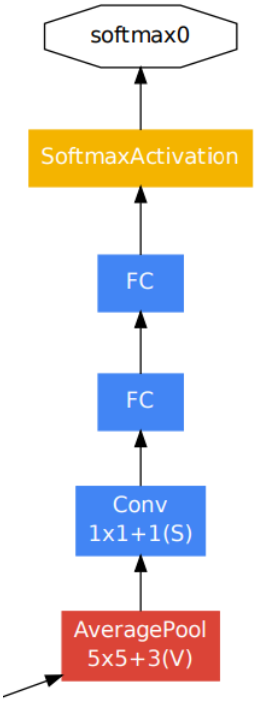


Figura 44: Clasificador auxiliar en Inception.[11]

El objetivo de incluir estos clasificadores auxiliares es solucionar el problema del desvanecimiento de gradiente, este se presenta en el entrenamiento de redes neuronales que hacen uso de métodos basados en gradiente en su entrenamiento y con propagación hacia atrás. El problema surge debido a que en ocasiones el gradiente puede tomar valores muy pequeños lo que impide que los pesos de la red se actualicen de manera eficaz y en el peor de los casos puede impedir que la red continúe con su entrenamiento.

En este mismo contexto surge una nueva mejora propuesta para la CNN Inception, se da el surgimiento de Inception V2 e Inception V3[30], para la arquitectura Faster R-CNN es usada la versión 2. El autor de estas arquitecturas propone una serie de actualizaciones a la V1 con el fin de incrementar la precisión además de reducir el poder de cómputo requerido para esta.

Para lograr esto se realizaron unas mejoras a los módulos Inception, en la Figura 45 se puede observar el módulo Inception propuesto en Inception V1.

[11]Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015). Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1-9).

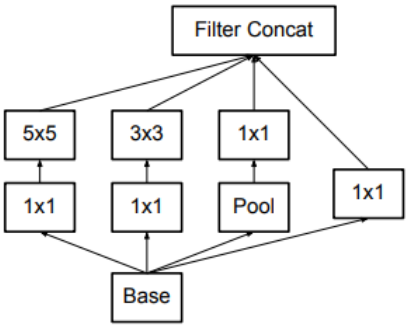


Figura 45: Módulo Inception en Inception V1.[12]

En la primera mejora de ese módulo se propuso factorizar las convoluciones de 5×5 en dos convoluciones de 3×3 apiladas una sobre otra para incrementar la velocidad de ejecución ya que una convolución de 5×5 es 2.78 veces más costosa computacionalmente que una convolución de 3×3 aplicando la misma cantidad de filtros, esto se puede observar en la Figura 46.

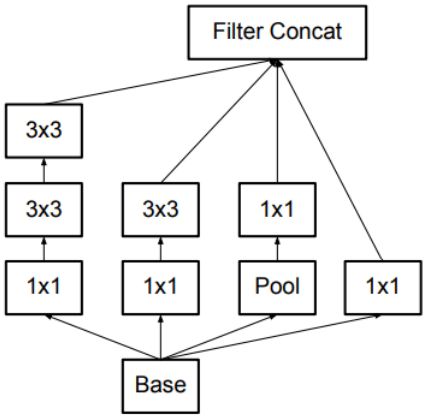


Figura 46: Módulo Inception en donde se reemplaza la convolución de 5x5 en dos de 3x3.[13]

[12]Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 2818-2826)

[13]Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 2818-2826)

Además, al factorizar convoluciones de tamaño $n \times n$ en una combinación de convoluciones de tamaño $1 \times n$ y $n \times 1$ se encontró que este método es un 33 % más barato para la misma cantidad de filtros de salida. Para ejemplificar esto, si se tiene una matriz de 3×3 esta puede ser factorizada en una matriz de 1×3 y una matriz de 3×1. Esto se puede observar en la Figura 47.

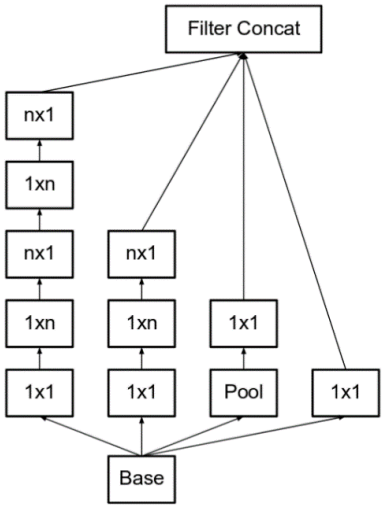


Figura 47: Módulo Inception con factorización 1xn y nx1.[14]

El problema que surge de apilar más convoluciones es que estas representan un cuello de botella ya que si el módulo se hace más profundo esto provocará una excesiva reducción de dimensionalidad lo que a su vez ocasionaría una pérdida de información por cada pasada. Por ello otra mejora que se propuso fue expandir a lo ancho las convoluciones en lugar de apilar estas y hacer que el módulo crezca en profundidad, este cambio puede ser visualizado en la Figura 48.

[14]Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 2818-2826)

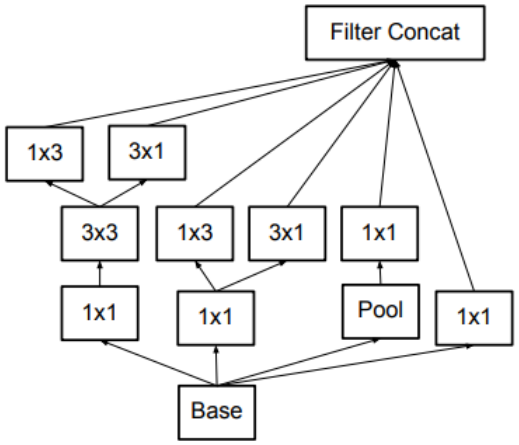


Figura 48: Módulo Inception expandido en ancho en lugar de profundidad.[15]

Con las mejoras anteriormente mencionadas surgieron tres diferentes tipos de módulos Inception, el presentado en la Figura 46, la Figura 47 y la Figura 48. Estos tres tipos de módulos fueron implementados en la arquitectura Inception V2 quedando de la forma presentada en la Figura 49.

type	**patch size/stride** or remarks	**input size**
conv	3×3/2	299×299×3
conv	3×3/1	149×149×32
conv padded	3×3/1	147×147×32
pool	3×3/2	147×147×64
conv	3×3/1	73×73×64
conv	3×3/2	71×71×80
conv	3×3/1	35×35×192
3×Inception	As in figure 5	35×35×288
5×Inception	As in figure 6	17×17×768
2×Inception	As in figure 7	8×8×1280
pool	8 × 8	8 × 8 × 2048
linear	logits	1 × 1 × 2048
softmax	classifier	1 × 1 × 1000

Figura 49: Arquitectura de Inception V2.[16]

[15]Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 2818-2826)

Observando la Figura 49, en el apartado donde dice “As in figure 5” corresponde al módulo Inception de la Figura 46, el apartado “As in figure 6” corresponde al módulo Inception de la Figura 47 y por último el apartado “As in figure 7” representa el módulo Inception de la Figura 48.

Posteriormente otra parte fundamental de la arquitectura Faster R-CNN es la red de propuesta de regiones mejor conocida como RPN por sus siglas en inglés, esta posee un clasificador y un regresor. Esta red tiene como objetivo generar diversas propuestas de regiones en donde se puede o no encontrarse un objeto, estas propuestas también son conocidas como anclas o cajas, pero son más comúnmente encontradas en la literatura como anchors[31]. La RPN predice la posibilidad de que un anchor sea el objeto o el fondo además de que puede refinar el tamaño de este para ajustarse más al tamaño del objeto. En la Figura 50 se puede observar en qué nivel se encuentra el módulo RPN dentro de la arquitectura Faster R-CNN.

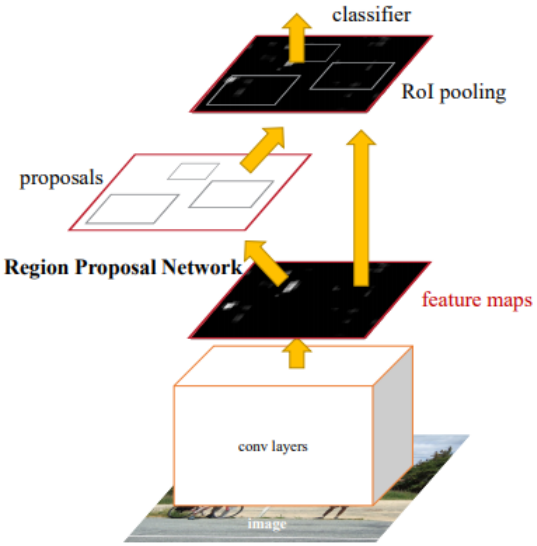


Figura 50: Módulo RPN dentro de arquitectura Faster R-CNN.[17]

Los anchors son generados a partir del mapa de características el cual es entregado por el extractor de características, para este caso Inception V2. En la configuración por defecto de Faster R-CNN se generan 9 anchors diferentes usando dos parámetros: escalas y relación de aspecto como se puede observar en la Figura 51.

[16]Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 2818-2826)

[17]Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. In Advances in neural information processing systems (pp. 91-99)

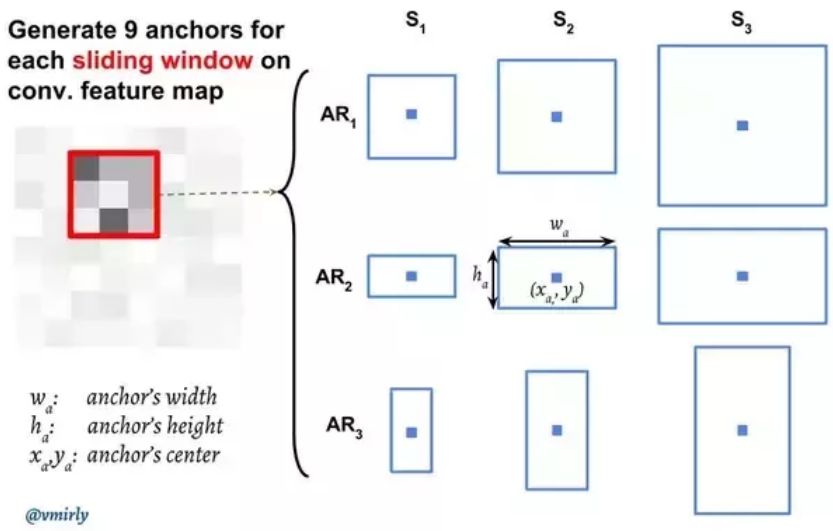


Figura 51: Anchors generados tomando en cuenta la escala y la relación de aspecto.[18]

Para generar todos los posibles anchors es necesario definir otro parámetro llamado stride o paso, en función de este se tendrá una n cantidad de posibles posiciones de los anchors, e.g. si tomamos la salida del mapa de características de tamaño $18 \times 25 \times 512$ se tendrá un total de 450 puntos, estos puntos son el centro donde se toman las cajas donde existe posibilidad de que exista un objeto, entonces para este ejemplo se tienen un total de $18 \times 25 \times 9 = 4050\, cajas$, la cantidad de propuestas de cajas entonces está dada por $W \times H \times k$ donde k es el número de anchors con sus respectivas escalas (3 escalas y 3 relaciones de aspecto, $3 \times 3 = 9$)

Todo este proceso se lleva a cabo dentro de la RPN y una vez que se tienen las cajas generadas en base a los anchors y las relaciones de aspecto la RPN procede a comprobar si existe un objeto o no en función a las propuestas de cajas generadas y la imagen de entrada, para esto el clasificador dentro de la RPN hace uso de dos etiquetas: objeto y no objeto, para poder determinar a cual pertenece se hace uso de la operación IoU (Intersection over Union) conocida como Intersección sobre Unión, esta se encuentra definida como lo muestra la ecuación 10.

$$IoU = \frac{A \cap Gt}{A \cup Gt} \tag{10}$$

En donde A es la propuesta de caja dada por el RPN y Gt es la caja que uno marcó manualmente en la imagen que se está utilizando en la entrada, esta caja es mejor conocida como ground truth o ground truth box, en la Figura 52 se puede observar un ejemplo de esto.

[18]Recuperada de `https://qph.fs.quoracdn.net/main-qimg-5f23c56676fcf1369a836f625500c19e.webp`

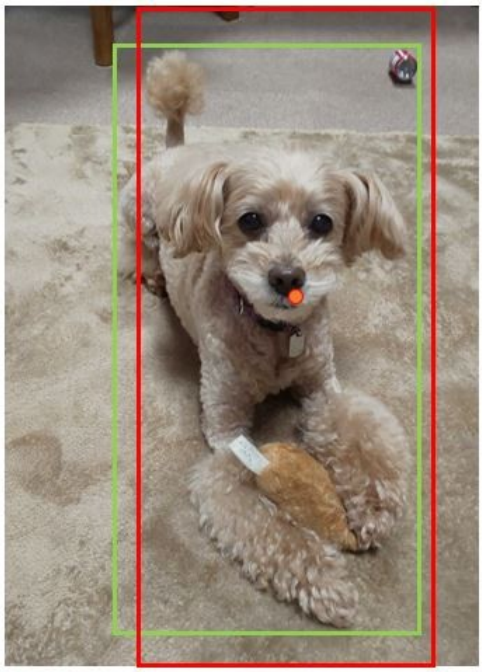


Figura 52: Operación de Intersección sobre Unión.[19]

En la Figura 52 se puede observar una caja de color rojo, esta caja es la propuesta obtenida por la RPN mientras que la caja en verde es la ground truth box. Para la operación IoU mientras más traslape exista entre la propuesta y la ground truth box entonces más posibilidades exista de que el objeto se encuentre dentro de la propuesta. Para que la RPN pueda clasificar como un objeto el IoU tiene que ser mayor a 0.7, cualquier valor menor a este será clasificado como no objeto.

Además, como se mencionó anteriormente la RPN está compuesta por un clasificador y por un regresor por lo que se requiere de hacer uso de una función de loss, esta se encuentra definida como se observa en la ecuación 11:

$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}}\sum_i L_{cls}(p_i, p_i^*) + \lambda\frac{1}{N_{reg}}\sum_i p_i^* L_{reg}(t_i, t_i^*) \tag{11}$$

En la ecuación 11 toma en cuenta el loss para el clasificador además del loss para el regresor, se tiene que i es el índice del anchor utilizado, Se tiene que L_{cls} es la función Log Loss (Logarithmic Loss) sobre dos clases (objeto vs no objeto), p_i es la probabilidad de que exista o no un objeto, p_i^* es la etiqueta de la ground truth box (1 o 0). Posteriormente viene el loss de regresión el cual solo es activado si el valor de p_i^* es 1. Dentro de esta se tiene el termino t_i el cual es la predicción de salida de la capa de regresión, esta consta de un vector representando las coordenadas parametrizadas de la bounding box predicha, los valores del vector son $t_x\,, t_y\,, t_w\,, t_h$, de estas $t_x\, y\, t_y$

[19]Recuperada de `https://miro.medium.com/max/353/1*9WnJWGGF5nLUvGi1bET6ag.jpeg`

describen la coordenada de inicio del bounding box mientras que t_w y t_h indican el ancho y el alto que debe tener. El termino t_i^* representa los mismos valores que el vector que t_i solo que para la ground truth box, los valores que conforman estos vectores dependen de la geometría del anchor la cual está representada por las siguientes ecuaciones:

$$t_x = \frac{(x - x_a)}{w_a} \qquad t_y = \frac{(y - y_a)}{h_a}$$
$$t_w = log\left(\frac{w}{w_a}\right) \qquad t_h = log\left(\frac{h}{h_a}\right)$$
$$t_x^* = \frac{(x^* - x_a)}{w_a} \qquad t_y^* = \frac{(y^* - y_a)}{h_a}$$
$$t_w^* = log\left(\frac{w^*}{w_a}\right) \qquad t_h^* = log\left(\frac{h^*}{h_a}\right)$$

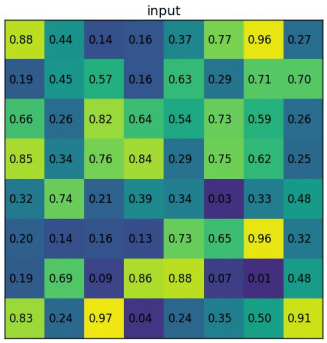
Donde x e y son las coordenadas del centro de la caja mientras que w y h son el ancho y el alto de la caja y se tiene que x^*, y^*, w^*, h^* corresponden al ground truth box y x_a, y_a, w_a, h_a corresponden a la geometría del anchor.

Otro paso llevado a cabo dentro de la RPN es la Supresión no máxima (NMS por sus siglas en inglés) la cual tiene como objetivo remover cajas que se traslapan con otras cajas mediante la medición de su traslape aplicando IoU sobre cada posible par de cajas, si este traslape es mayor a 0.5 entonces las cajas comparadas están apuntando a un mismo objeto por lo que se remueve la caja con menor probabilidad de clase. Este proceso se repite hasta que se hayan eliminado todas las cajas repetidas.

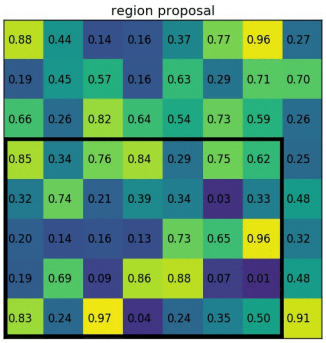
Asimismo, otro componente importante que forma parte de la arquitectura Faster R-CNN es la capa de regiones de interés mejor conocida como ROI (Region of interest), esta capa tiene como objetivo producir mapas de características de tamaño fijo mediante la aplicación de max pooling en las entradas. La capa ROI toma dos entradas: un mapa de características obtenido de la CNN después de varias convoluciones y capas de pooling y un número N de propuestas de la RPN a las cuales se les nombra región de interés, cada una de estas propuestas posee 5 valores, el primero indica el índice y los otros cuatro contienen las coordenadas de la propuesta, estas coordenadas representan generalmente la parte superior izquierda y la esquina en la zona inferior derecha de la propuesta.

La capa ROI entonces toma todas las regiones de interés de la entrada y toma una sección del mapa de características de entrada correspondiente a esa región y convierte esa sección del mapa de características en un mapa de dimensión fija, esto nos permite resolver el problema de dimensionalidad ya que las entradas no necesariamente son siempre del mismo tamaño, al realizar esto siempre se tendrá un tamaño fijo sin importar el tamaño de la entrada. En la Figura 53a se puede observar un ejemplo de entrada, en la Figura 53b se puede observar la propuesta de región, en la Figura 53c se puede observar el número de secciones en función al tamaño de salida deseado, en la

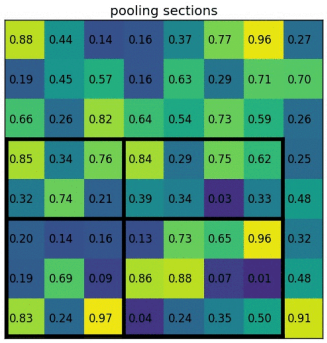
Figura 53d se puede observar la aplicación del max pooling y en la Figura 53e se puede observar la salida obtenida.



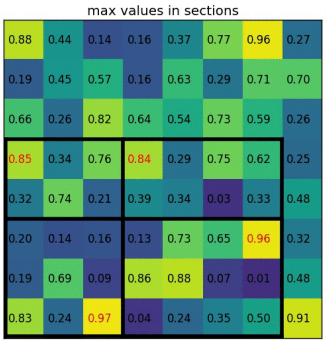
(a) Mapa de entrada



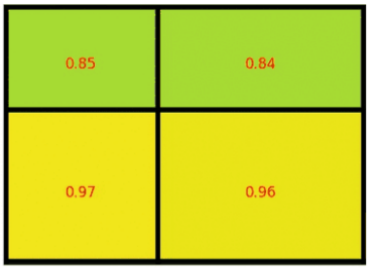
(b) Propuesta de región



(c) Secciones de pooling



(d) Aplicación de max pooling



(e) Salida final

Figura 53: Proceso llevado a cabo por ROI[20]

[20]Recuperadas de `https://deepsense.ai/region-of-interest-pooling-explained/`

Por último, la salida obtenida por la capa ROI es pasada a dos ramas en donde se encuentran una capa completamente conectada en cada una de ellas, una de estas ramas es utilizada para realizar la clasificación mediante un clasificador softmax el cual nos entrega las probabilidades de pertenecer a cada una de las clases definidas mientras que la otra rama es utilizada como un regresor con el fin de obtener el bounding box del o de los objetos detectados. En la Figura 54 se puede observar lo que se comenta anteriormente para la clase, perro, botella y fondo la cual representa que no hay objeto.

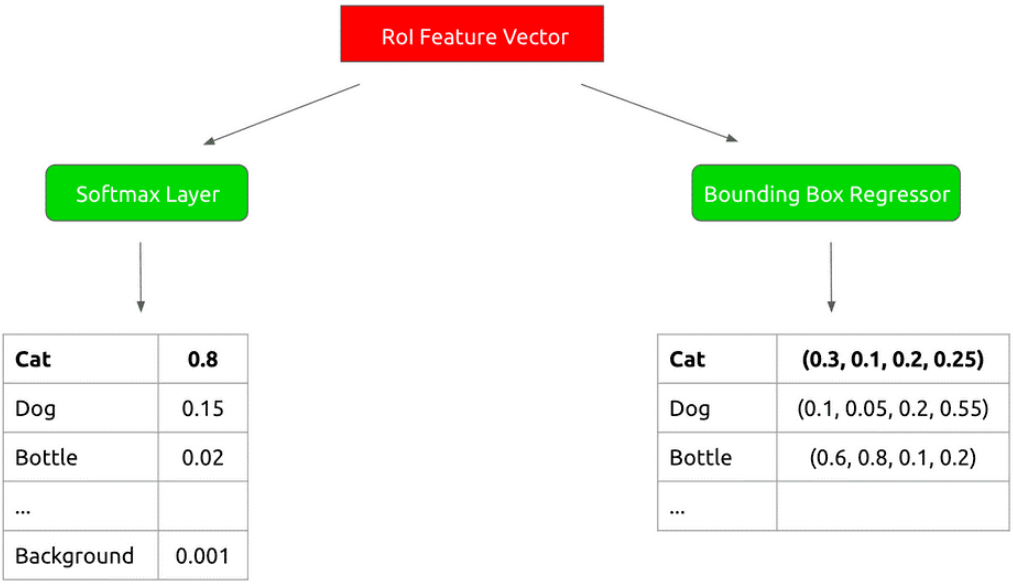


Figura 54: Salida obtenida en las capas completamente conectadas al final de la arquitectura.[21]

4.1.1.2. SSD

En contra parte con las arquitecturas basadas en R-CNN y propuestas basadas en regiones se tienen los detectores de una sola toma, entre ellos se encuentra la arquitectura SSD[32], esta requiere de una sola captura para poder detectar múltiples objetos dentro de la imagen, esta arquitectura puede llegar a ser mucho más rápida que las basadas en propuestas de regiones.

La arquitectura SSD está conformada por tres componentes: la columna vertebral (o backbone) el cual consiste en un extractor de características el cual ya se encuentra preentrenada con algún dataset como ImageNet o COCO, para este caso en el paper donde

[21] Recuperada de `https://www.mihaileric.com/static/softmax_and_bbox_regressor-972e4f6607eaac361c42e8aeb92ee0a0-3d61e.png`

se propuso esta arquitectura se hace uso de VGG-16 en donde fue removida la parte final, la cual consta de una red completamente conectada. El segundo componente que conforma esta arquitectura son diversas capas convolucionales auxiliares, permitiendo así extraer características a múltiples escalas y disminuir progresivamente el tamaño de la entrada a cada capa subsiguiente, además de unas capas convolucionales extras, las cuales están encargadas de la localización y clasificación. En la Figura 55 se puede observar un diagrama de cómo está conformada la arquitectura.

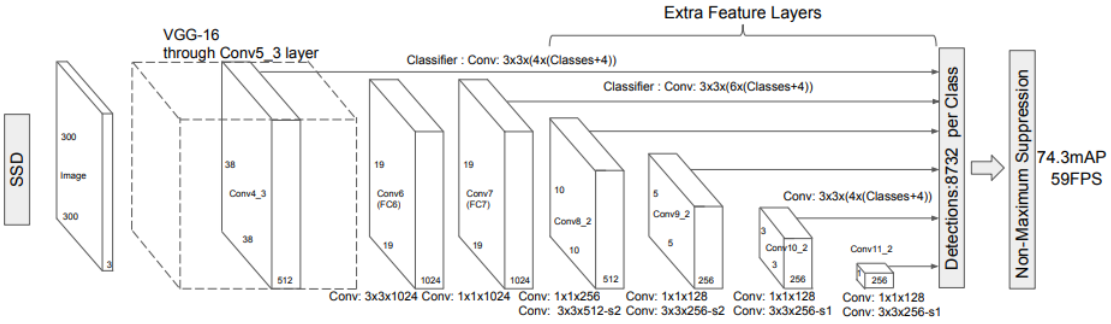


Figura 55: Arquitectura SSD.[22]

Como se mencionó anteriormente la primera parte de esta arquitectura es una red base pre-entrenada con algún dataset, para este caso se usó VOC2007[33], en el paper original se propuso el uso de VGG-16, pero a la cual se le realizaron las siguientes modificaciones: El tamaño de entrada de la imagen fue propuesto en dos versiones, una de tamaño 300×300 y otra de 512×512 pixeles. Posteriormente se modificó la tercera capa de pooling de la arquitectura para cambiar la función encargada de determinar el tamaño de salida, paso de una función piso(floor) a una función techo(ceiling), esto es particularmente útil si las dimensiones del mapa de características anterior son impares ya que el tamaño de mapa de características al ser pasado de una capa a otra es reducido a la mitad y el uso de esta función nos permite tener dimensiones pares las cuales son más convenientes para su manejo y posterior reducción, e.g. el mapa de características conv3_3 de VGG-16 tendrá una sección transversal de 75 x 75 que es reducida a la mitad a 38 x 38 gracias a la función techo en lugar de una dimensión 37 x 37 que puede ser inconveniente para la reducción en las siguientes capas. El siguiente cambio a realizar fue en la quinta capa de pooling en donde se cambió el kernel de tamaño 2 x 2 y paso 2 por un kernel de tamaño 3 x 3 y de paso 1. Este cambio tiene como propósito evitar dividir a la mitad las dimensiones del mapa de características de la capa convolucional anterior.

[22]Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. Y., & Berg, A. C. (2016, October). Ssd: Single shot multibox detector. In European conference on computer vision (pp. 21-37). Springer, Cham.

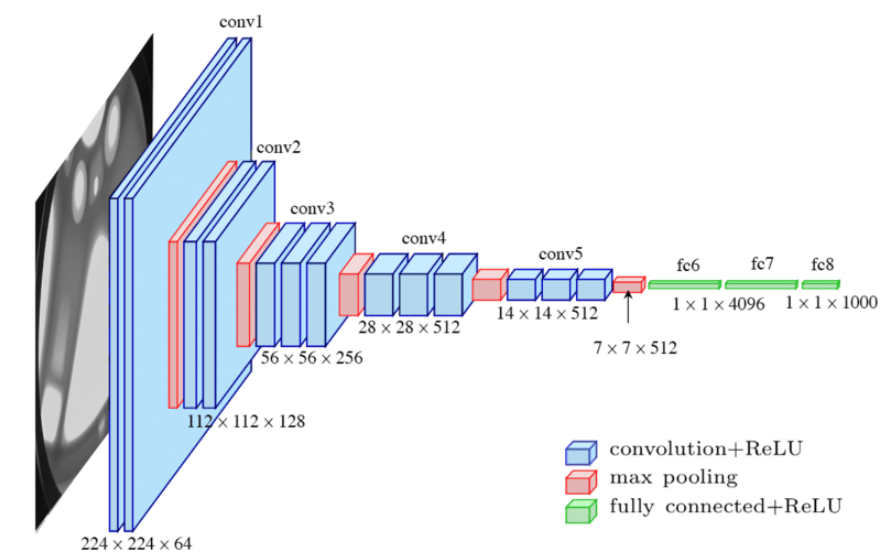


Figura 56: Arquitectura en capas de VGG16.[23]

La última modificación realizada consiste en quitar la capa completamente conectada fc8 de VGG-16, que se puede observar en la Figura 56 y posteriormente convertir las capas completamente conectadas fc6 y fc7 en capas convolucionales conv6 y conv7, para poder lograr esto se remodelaron los parámetros de las capas completamente conectadas.

Originalmente VGG-16 operaba con imágenes de 224 x 224 pixeles, en su capa fc6 recibe una entrada aplanada (convertida en un vector de 1xn) de 7 x 7 x 512 proveniente de la capa max pooling anterior y tiene un tamaño de salida de 1 x 1 x 4096 por lo que su capa equivalente conv6 para poder obtener el mismo tamaño de salida requiere de un kernel de 7 x 7 y 4096 canales de salida para poder obtener la misma salida con respecto a fc6. En su capa fc7 recibe un tamaño de entrada 1 x 1 x 4096 y su salida es de 1 x 1 x 4096 por lo que su equivalente conv7 tiene un kernel de 1 x 1 y 4096 canales de salida con el fin de obtener la misma salida con respecto a fc7. Con lo anterior se puede observar que la cantidad de filtros es grande y por lo tanto computacionalmente cara por lo que en la arquitectura SSD se optó por reducir la cantidad y el tamaño de los filtros aplicados en las capas convolucionales al submuestrear los parámetros de las capas fc6 y fc7. Para esto en la capa conv6 se usaron 1024 filtros en lugar de 4096, cada uno con dimensión 3 x 3 x 512 así pasando de 4096 filtros de 7 x 7 x 512 a 1024 filtros de 3 x 3 x 512 mientras que para conv7 también se usaron 1024 filtros cada uno con dimensión 1 x 1 x 1024 por lo que se pasó de 4096 filtros de 1 x 1 x 4096 a 1024 filtros de 1 x 1 x 1024. Para el submuestreo se toma cada n-ésimo parámetro a lo largo de una dimensión particular, este proceso también es conocido como diezmado. Para el

[23]Recuperada de `https://cdn-images-1.medium.com/max/850/1*_Lg1i7wv1pLpzp2F4MLrvw.png`

caso de conv6 esta fue diezmada de 7 x 7 a 3 x 3 al solo mantener cada tercer valor.

Posteriormente se apilaron más capas convolucionales auxiliares sobre la red base, estas capas extras tienen como objetivo proporcionar mapas de características adicionales, cada uno progresivamente más pequeño que el anterior y como se puede observar en la Figura 57.

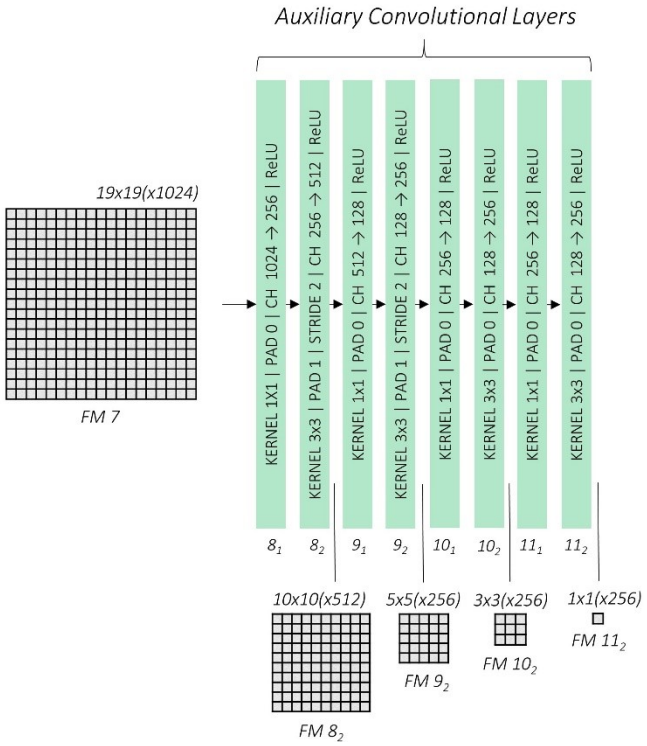


Figura 57: Capas convolucionales auxiliares.[24]

Para lograr lo anterior se introdujeron cuatro bloques convolucionales, cada uno conformado por dos capas convolucionales, la primera capa obtiene el mapa de características mientras que la segunda capa convolucional de cada bloque esta encargada de reducción de dimensionalidad, mientras que en VGG-16 de hace uso de capas de pooling para la reducción de dimensión, en este caso se logra el mismo objetivo mediante una capa convolucional con stride 2. Estas capas se pueden observar en la Figura 57 con los subíndices 2. Dentro de estos bloques se realiza el proceso de detección de objetos, para poder realizar este proceso SSD hace uso de bounding boxes predeterminadas haciendo el uso de una técnica llamada Multibox.

Multibox es una técnica de regresión para bounding box el cual define varias bounding box las cuales son conocidas como priors (anchors o anclas en Faster R-CNN).

[24]Recuperada de `https://raw.githubusercontent.com/sgrvinod/a-PyTorch-Tutorial-to-Object-Detection/master/img/auxconv.jpg`

La técnica de Multibox propone que los priors son escogidos de manera que coincidan estrechamente con la distribución de las cajas ground truth que se encuentran en el dataset de entrenamiento. Para que puedan ser seleccionadas de esta manera la caja generada es comparada con las cajas ground truth mediante IoU, si este es mayor a 0.5 es una buena propuesta, de caso contrario es descartada, pero sirve de punto de partida para el algoritmo de regresión de las bounding boxes. Por lo tanto, Multibox empieza con los priors como predicciones y mediante regresión intenta obtener las bounding boxes más cercanas a las cajas ground truth. Para poder generar los priors a cada mapa de características obtenido en cada uno de los bloques se le asigna una escala especifica la cual se encuentra entre 0.2 y 0.9 a excepción de la capa conv4_3 la cual puede tener escala de 0.1 al ser el mapa de características más grande. Para poder determinar que escala se asigna a cada capa se tiene que para el k-ésimo mapa de entre m mapas la escala es elegida como lo indica la ecuación iv en donde S_{min} es la escala mínima, S_{max} es la escala máxima. Mientras más grande sea el mapa de características, este tendrá priors con menores escalas lo que permitirá detectar objetos más pequeños.

$$s_k = s_{min} + \frac{s_{max} - s_{min}}{m - 1}(k - 1),\ k\,\epsilon[1, m] \tag{12}$$

Para cada mapa le corresponde una escala en específico además de que puede tener 4 o 6 cajas predeterminadas asociadas para cada escala. Igualmente, los priors tienen diferentes relaciones de aspecto en función del nivel de la capa, estas relaciones de aspecto están predefinidas de la siguiente manera:

$$a_r = \{1, 2, 3, \frac{1}{2}, \frac{1}{3}\} \tag{13}$$

Todos los mapas de características tienen las relaciones de aspecto: 1,2, $\frac{1}{2}$ mientras que los mapas intermedios de conv7, conv8_2 y conv9_2 también tienen los aspectos: 3 y $\frac{1}{3}$ además de que todos los mapas de características tienen un prior extra cuando se tiene aspecto 1 y con una escala la cual es la media geométrica de las escalas del mapa de características actual y posterior. Para poder obtener cuantos priors se tienen por mapa de características se calcula de la siguiente manera:

$$N_p = f \times b \times (4 + (c + 1)) \tag{14}$$

Donde f es el tamaño del mapa de características m x n, b es el número de priors por posición (4 o 6) y c es el número de clases a clasificar más 1, donde ese +1 representa a la clase: no hay objeto. Independiente del número de clases que tengamos, podemos obtener el número total de priors predeterminados por c clases para cada mapa de características como se muestra en la Tabla 1.

Sumando la cantidad de priors por capa de la tabla anteriormente mostrada se tiene un total de 8732 priors por clase. Para poder visualizar los priors es necesario conocer el ancho y el alto de cada una de las priors predeterminadas, para poder calcular el ancho y el alto se hace uso de las siguientes ecuaciones:

$$w_k^a = s_k\sqrt{a_r} \tag{15}$$

Mapa	**Tamaño de mapa**	**Priors predeterminados**	**Priors predeterminados totales por mapa**
conv4_3	38 x 38	4	5776
conv7	19 x 19	6	2166
conv8_2	10 x 10	6	600
conv9_2	5 x 5	6	150
conv10_2	3 x 3	4	36
conv11_2	1 x 1	4	4

Tabla 1: Cantidad de priors por mapa de características

$$h_k^a = \frac{s_k}{\sqrt{a_r}} \tag{16}$$

Donde w_k^a es el ancho de la caja a del mapa k, h_k^a es el alto de la caja, s_k es la escala del mapa k y a_r es la relación de aspecto para ese k mapa. Además de esto se agrega que cuando la relación de aspecto es 1 se agrega un nuevo prior con una nueva escala definida de la siguiente manera:

$$s_k^{'} = \sqrt{s_k s_{k+1}} \tag{17}$$

Donde $s_k^{'}$ es la nueva escala, s_k es la escala del mapa k y s_{k+1} es la escala del mapa k siguiente.

En la Figura 58 podemos observar cómo se ve un prior en el mapa de características conv9_2, dentro de este vemos 5 priors con las relaciones de aspecto 1, 2, 3, $\frac{1}{2}$ y $\frac{1}{3}$ además del sexto prior con la escala extra.

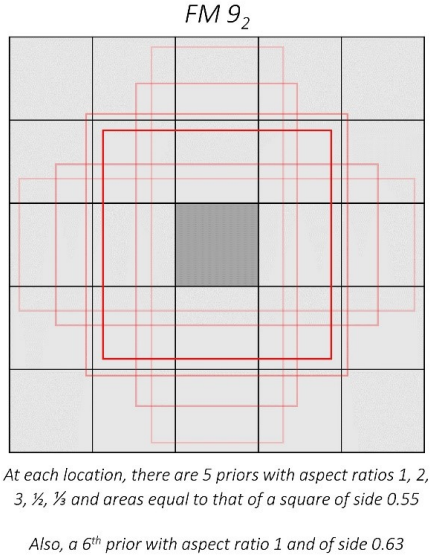


Figura 58: Priors sobre el mapa de características de conv9_2.[25]

Con todo lo mencionado anteriormente se obtienen las aproximaciones como punto de partida en donde se puede encontrar un objeto mediante los priors, pero ahora es necesario encontrar que tanto es necesario ajustar el prior para coincidir con la caja ground truth, estos ajustes necesarios son conocidos como offsets.

La siguiente parte de la arquitectura consiste en poder predecir para cada prior en cada parte en la que se encuentra ubicado en cada mapa de características los offsets necesarios para la bounding box, además de obtener un conjunto de n puntajes para los bounding boxes donde n representa el número de clases de objetos. Para poder realizar esto se hace uso de dos capas convolucionales por cada mapa de características obtenido. La primer capa convolucional a la cual nos referiremos como LOC(localization) tiene como objetivo realizar la predicción de ubicación del objeto, para poder hacer eso la capa convolucional hace uso de un kernel de 3 x 3 con 4 filtros para poder evaluar cada ubicación y mediante 4 filtros por cada prior presente en esa ubicación y los cuales permiten calcular cada uno de los 4 offsets requeridos g_{c_x}, g_{c_y}, g_w, g_h para el bounding box predicho para el prior.

La segunda capa convolucional a la cual nos referiremos como CLS (class) tiene como objetivo predecir la clase a la que pertenece el objeto dentro del bounding box, para esto hace uso de un kernel de 3 x 3 aplicado en cada ubicación, con n_clases filtros por cada prior utilizado en esa ubicación. Esos n_clases filtros de un prior calculan un conjunto de n_clases puntajes para ese prior determinado. En la Figura 59 se puede observar un ejemplo de cómo se ven las salidas de estas convoluciones, en azul se observa la salida para la predicción de la ubicación mientras que en amarillo se observa la salida

[25]Recuperada de `https://raw.githubusercontent.com/sgrvinod/a-PyTorch-Tutorial-to-Object-Detection/master/img/priors1.jpg`

para la predicción de clase, para este caso se hace uso de la capa conv9_2.

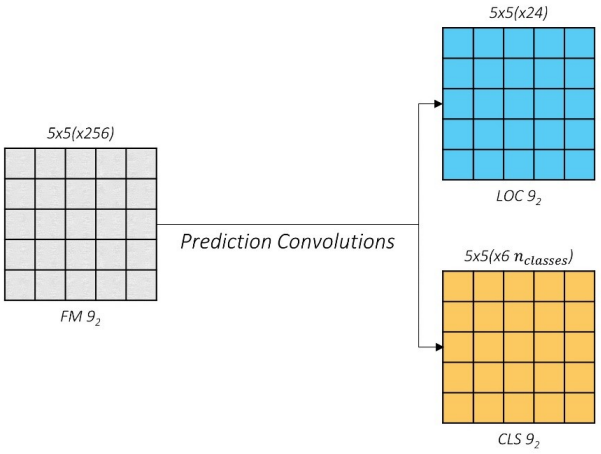


Figura 59: Salida de la convolución de predicción y localización usando la salida de conv9_2.[26]

Cada uno de los canales de la salida LOC de color azul de la Figura 59 de tamaño 5 x 5 x 24 contienen las predicciones, esos 24 canales que se tienen en cada ubicación representan las 6 bounding boxes predichas y cada una de estas contiene los 4 offsets necesarios para ajustar el bounding box con respecto al prior utilizado de los 6 priors predeterminados. Con respecto a la capa de convolución para la predicción de clases CLS se tiene que se tiene un total de n_clases x 6 filtros, e.g. si n_clases = 3 para conv9_2 se tiene una salida de 5 x 5 x (3 x 6) donde cada uno de estos 3 x 6 canales en cada ubicación representan 6 conjuntos de 3 puntajes para los 6 priors predefinidos. Posteriormente en ambas capas convolucionales se procede a remodelar los valores obtenidos. En la Figura 60 se puede observar un ejemplo de la remodelación de los valores obtenidos en CLS y LOC, se pasó de una representación de 5 x 5 x 24 a 150 x 4 en LOC mientras que en CLS de paso de una representación de 5 x 5 x 18 a una representación de 150 x 3.

[26]Recuperada de `https://raw.githubusercontent.com/sgrvinod/a-PyTorch-Tutorial-to-Object-Detection/master/img/predconv2.jpg`

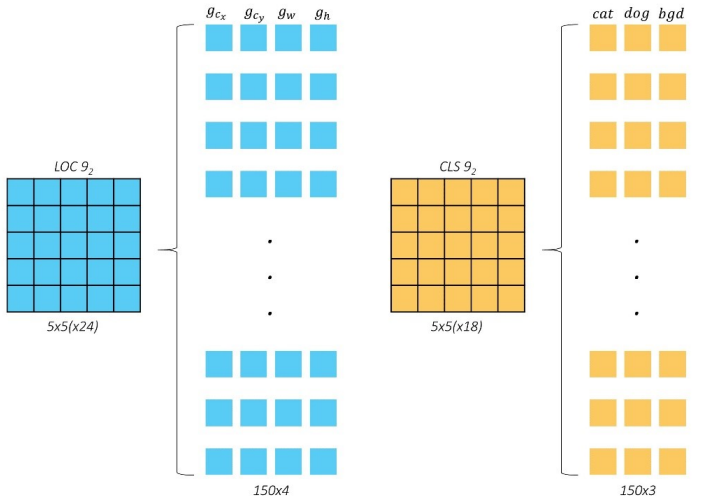


Figura 60: Remodelación de predicciones de CLS y LOC.[27]

Posteriormente se apilan ambas predicciones remodeladas de CLS y LOC, de acuerdo a los cálculos realizados anteriores se mostró que se tienen 8732 priors para la versión de SSD de 300x300 entonces en nuestra remodelación se tendrán 8732 bounding boxes predichos y 8732 conjuntos de puntajes de clases, en la Figura 61 se puede observar un ejemplo de esto.

[27]Recuperada de `https://raw.githubusercontent.com/sgrvinod/a-PyTorch-Tutorial-to-Object-Detection/master/img/reshaping1.jpg`

Figura 61: CLS y LOC apilados para remodelado.[28]

Como se mencionó anteriormente, para poder generar los priors predefinidos se hace uso de la técnica modificada de Multibox, para el entrenamiento de SSD para las predicciones tanto del bounding box como de las clases es necesario definir una función de loss que nos permita identificar que tan bien o que tan mal se está desempeñando la red al momento de ser entrenada y con esto poder determinar que ajustes son necesarios para aproximarse más a las etiquetas ground truth y sus cajas ground truth. A continuación, en la ecuación 18 se muestra la función loss definida para el Multibox.

$$L(x, c, l, g) = \frac{1}{N}(L_{conf}(x, c) + \alpha L_{loc}(x, l, g)) \tag{18}$$

En esta ecuación se tiene que N es el número de cajas predefinidas coincidentes con la caja ground truth, L_{conf} es la función loss específica para la clasificación de objetos, L_{loc} es la función de loss para el regresor encargado de predecir las coordenadas de los bounding boxes, se tiene que x es un indicador 0,1 el cual indica si hay un box coincidente o no, c es la confidencia de múltiples clases, l es la bounding box predicha, g es la caja ground truth y α es un parámetro que nos ayuda a balancear la contribución del loss de ubicación L_{loc} a el loss general. De manera más específica, L_{conf} está definido por la siguiente ecuación:

$$L_{conf}(x, c) = -\sum_{i \epsilon Pos}^{N} x_{ij}^{p} log(\hat{c}_i^p) - \sum_{i \epsilon Neg}^{N} log(\hat{c}_i^0) \; donde \; \hat{c}_i^p = \frac{exp(c_i^p)}{\sum_p exp(c_i^p)} \tag{19}$$

[28]Recuperada de `https://raw.githubusercontent.com/sgrvinod/a-PyTorch-Tutorial-to-Object-Detection/master/img/reshaping2.jpg`

En esta ecuación se tiene que $x_{ij}^{p} = 1{,}0$ el cual es un indicador de coincidencia de la i-ésima caja predefinida con la j-ésima caja ground truth y perteneciente a la clase p, esta sumatoria aplica para los casos positivos, la siguiente sumatoria aplica para los casos negativos donde c_i^0 es la confidencia para la clase 0, que indica que no hay objeto. El loss de ubicación está definido por la función smooth L1 loss y está definido por la ecuación:

$$L_{loc}(x, l, g) = \sum_{i \epsilon Pos} \sum_{m \epsilon \{cx,cy,w,h\}} x_{ij}^{k} smooth_{L1}(l_i^m - \hat{g}_j^m) \tag{20}$$

$$\hat{g}_j^{cx} = \frac{(g_j^{cx} - d_i^{cx})}{d_i^w} \qquad \hat{g}_j^{cy} = \frac{(g_j^{cy} - d_i^{cy})}{d_i^h}$$

$$\hat{g}_j^{w} = log\left(\frac{g_j^w}{d_i^w}\right) \qquad \hat{g}_j^{h} = log\left(\frac{g_j^h}{d_i^h}\right)$$

calculado, l_i^m son los offsets predichos y $\hat{x}_j^m$ son los offsets objetivo, g_j^m los las coordenadas del ground truth, d_i^m son las coordenadas de la caja predefinida, $x_{ij}^k = 1{,}0$ el cual es un indicador de que si la caja predefinida i coincide con el ground truth j de clase k.

Una vez que el modelo fue entrenado, este se encuentra listo para poder realizar el proceso de inferencia en nuevas imágenes sin embargo estas predicciones se encuentran en forma de datos en bruto en dos tensores los cuales contienen los offsets de las bounding boxes y los puntajes de los 8732 priors propuestos, estos datos aun necesitan ser procesados para poder ser interpretados. Para poder realizar este proceso primero se decodifican los offsets para poder ser convertidos en las coordenadas que representan al bounding box. Posteriormente se toman todos los datos en los que se detectó un objeto para así entonces extraer los puntajes de cada una de sus 8732 cajas, al ser muchas cajas es necesario eliminar aquellas que no superen un umbral de puntaje y entonces así obtener las bounding boxes candidatas para una clase determinada del objeto detectado. A pesar de que se reduce el número de bounding boxes, si trazas estas sobre la imagen de entrada es probable que se encuentren cajas que se traslapen sobre el mismo objeto, esto sucede ya que es altamente probable que varios de los priors propuestos correspondan al mismo objeto con pequeñas variaciones en su tamaño y forma. En la Figura 62 se puede observar que se tienen varias predicciones para el mismo objeto, se ve la etiqueta a la que corresponde el objeto además de su puntaje el cual nos indica que tanto pertenece a la clase.

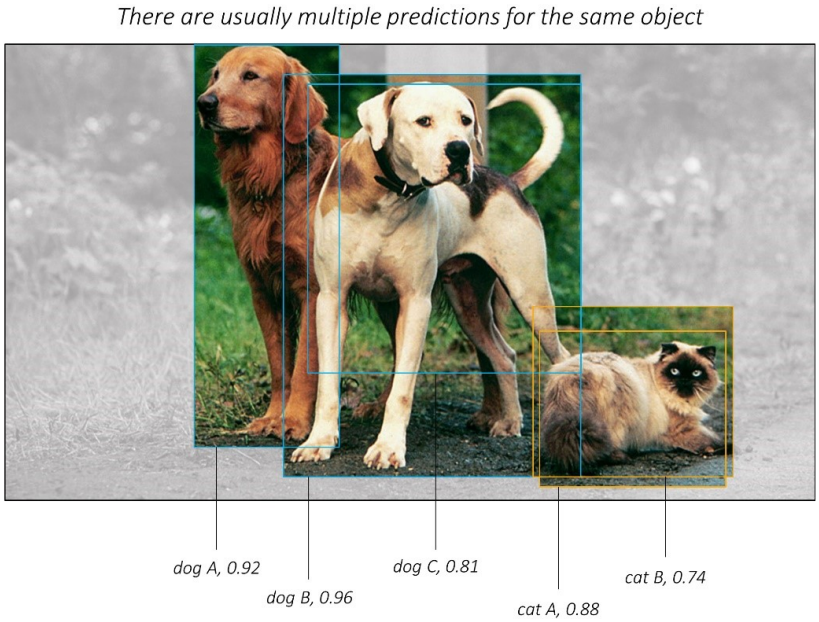


Figura 62: Traslape de bounding boxes para la misma clase de objeto.[29]

Para evitar esto es necesario encontrar las bounding boxes candidatas que se traslapan sobre el mismo objeto y elegir la mejor de ellas. Para hacer esto es necesario comparar cada posible par de cajas (a excepción de consigo mismas) para encontrar cuales se traslapan más entre sí, esto mediante la IoU de cada par de imágenes, si el IoU es mayor a 0.5 entonces se encontró dos bounding boxes candidatas para el mismo objeto entonces es necesario eliminar la bounding box que tenga un menor puntaje, como se mencionó en Faster R-CNN, este proceso se llama supresión no máxima (NMS). Este proceso se repite hasta que se eliminen todas las bounding boxes que se traslapen con un IoU mayor a 0.5 y entonces con eso se tiene el mejor bounding box para cada clase de objeto predicha en la imagen. Una ves realizado esto ya se obtiene la salida final de la arquitectura la cual nos entregará las coordenadas de la bounding box, su clase y su puntaje de la clase.

4.2. Sistema de marcadores y miniaturización del sistema

Para que el robot pueda ubicarse dentro de un espacio determinado y pueda moverse requiere de un punto de referencia para poder determinar donde se encuentra, sea un sistema de coordenadas absolutas o relativas, coordenadas GPS conformadas por latitud y longitud o puede ser un sistema de marcadores el cual nos permita identificar en qué

[29]Recuperada de `https://raw.githubusercontent.com/sgrvinod/a-PyTorch-Tutorial-to-Object-Detection/master/img/nms1.PNG`

lugar o habitación se encuentra. Para el desarrollo de este trabajo se propone hacer uso de un sistema de marcadores físicos los cuales puedan ser detectados y localizados mediante el sistema de visión artificial con algunas de las arquitecturas propuestas anteriormente.

Para que estos marcadores puedan ser detectados por el sistema de visión artificial primero es necesario definir una nueva clase única para poder identificar a estos, posteriormente es necesario definir como serán estos marcadores, para hacer esto es necesario tomar en cuenta que el marcador debe tener una forma geométrica que no se encuentre fácilmente en el entorno en el que funcionara ya que si este se define de manera incorrecta este podría ser confundido con otro objeto, por ejemplo, si se definiera el marcador con una manera cuadrada, rectangular u octagonal este podría ser confundido con alguna señalética, con un cuadro, con una foto o con muchos otros objetos más por lo que se propone que el marcador debe de tener una forma pentagonal ya que esta no se encuentra comúnmente en el entorno. Posteriormente se definieron las dimensiones físicas del marcador, este se definió como un pentágono regular con sus lados de 12cm. Posteriormente es necesario definir otro tipo de características específicas para el marcador y así este pueda ser abstraído y aprendido más eficientemente por la red neuronal, por esto mismo se definió que este marcador tendrá un borde de 1.5 cm de color rojo. Por último, es necesario definir qué información nos indicará el marcador, para este trabajo de tesis los marcadores indicarán lugares específicos, además de que indicarán en que parte del lugar se encuentra, y también indicarán la dirección que tiene que seguir para poder ir a la siguiente ubicación. Para poder lograr esto la parte interna del marcador es dividida en dos regiones, cada una de las regiones se encontrará rellena con un color, por lo que se busca tener n cantidad de pares de colores diferentes y sin repetir ya que se utilizará esa combinación interna de dos colores para asignarle una ubicación, mientras coincidan los pares de colores se sabrá la ubicación asignada a ese par. En la Figura 63 se puede observar el ejemplo de un marcador, en la parte interna de este cuenta con dos colores: blanco y negro por lo que a ese par se le asigna la ubicación A y esto mismo se hará para todos los marcadores deseados.

Figura 63: Ejemplo de marcador, ubicación “A” asignada al par de colores negro y blanco.[30]

Para poder identificar que colores se encuentran dentro del marcador es necesario establecer un método que nos permita extraer el color de cada una de las dos regiones establecidas y posteriormente este sea comparado con una lista la cual contendrá una tupla la cual contiene el nombre de la ubicación y los pares de colores que lo identifican, para poder generar esta lista se recortó un cuadrado de 5 x 5 pixeles de cada región, posteriormente se promedió el valor de los pixeles de cada cuadrado para poder tener un color promedio de cada región. Este proceso se realiza de manera manual para cada marcador nuevo que se desee agregar y se hace de manera automática durante la inferencia. Una vez que se tienen estos colores promedio se les asigna un nombre a la ubicación que representan y son agregados a la lista.

Después de tener la lista con los datos de los marcadores, es necesario establecer un método por el cual se puedan obtener los colores que se encuentran en las dos regiones, para hacer esto se propuso el siguiente método:

1. Se obtiene la imagen dentro del bounding box detectado mientras pertenezca a la clase del marcador como el objeto de la Figura 63.

2. Se recorta automáticamente la imagen para que quede de forma cuadrada.

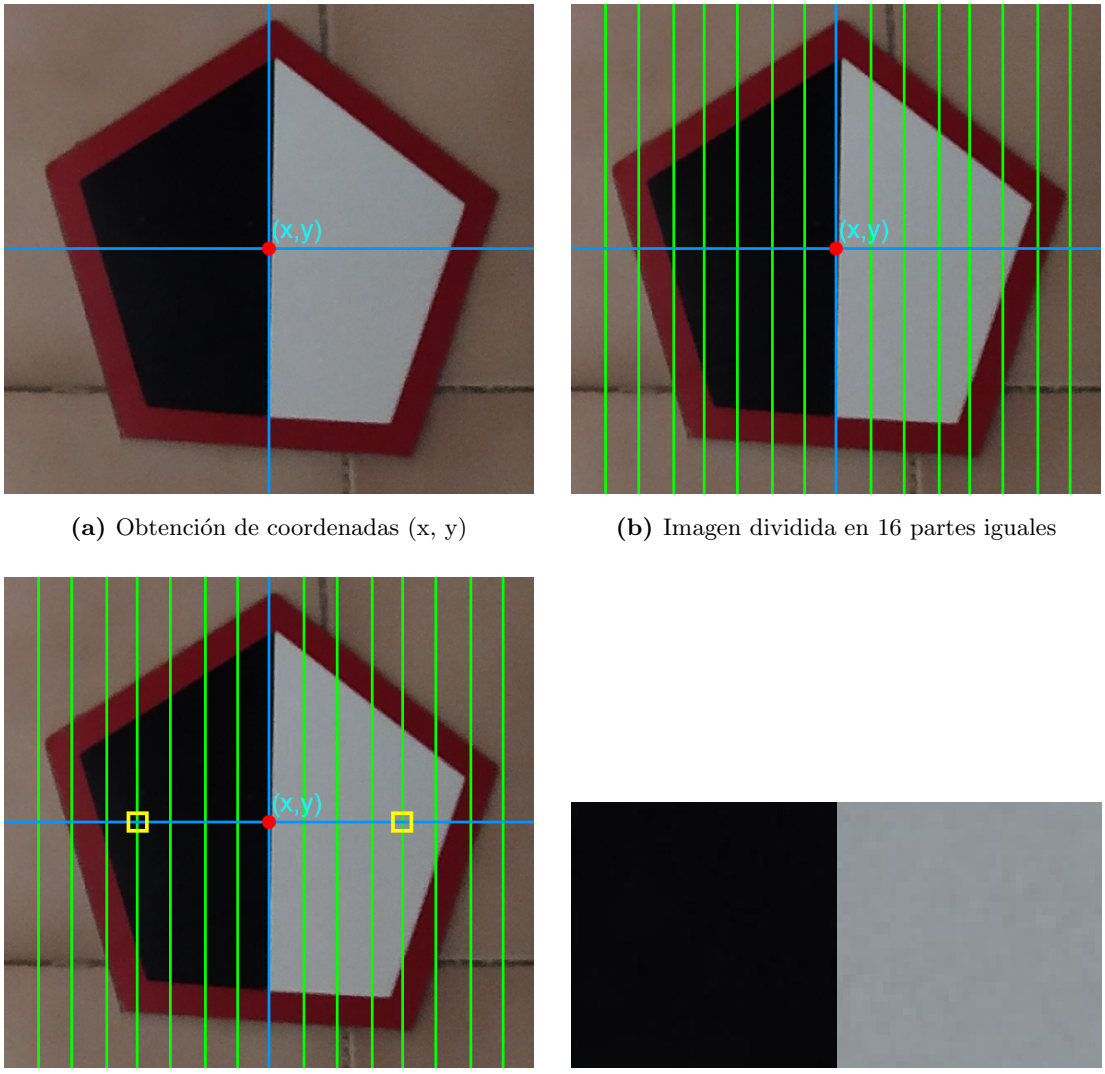
3. Se calcula la mitad de la anchura y altura de la imagen y se almacenan sus coordenadas (x, y) como se observa en la Figura 64a con las líneas azules.

4. Posteriormente la anchura de la imagen se divide en 16 partes iguales como se ve en la Figura 64b marcado con las líneas verdes.

5. Se toma la división 4/16 y la 12/16 además de tomar la coordenada y que indica donde está la mitad de la imagen en altura para así entonces recortar una región de 3 x 3 de cada una de las divisiones seleccionadas como se ve en la Figura 64c con el cuadro amarillo.

6. Por ultimo se promedia el valor de los pixeles de cada canal de cada una de las regiones recortadas las cuales se pueden observar en la Figura 64d y así obtener el color promedio de cada región del marcador.

[30]Elaboración propia



(a) Obtención de coordenadas (x, y)

(b) Imagen dividida en 16 partes iguales

(c) Elección de regiones para extraer colores

(d) Colores obtenidos de la imagen

Figura 64: Proceso llevado a cabo para obtención de colores[31]

Con el procedimiento realizado anteriormente se obtienen los dos colores de las regiones del marcador, el siguiente paso a seguir es realizar la comparación con los elementos de la lista de marcadores para poder estimar a cuál se aproxima más. Para poder hacer esto primero se itera cada elemento de la lista de marcadores el cual será comparado con los colores extraídos, en la Figura 65 se puede ver un ejemplo de los colores obtenidos y los colores de un marcador separados por canales.

[31] Elaboración propia.

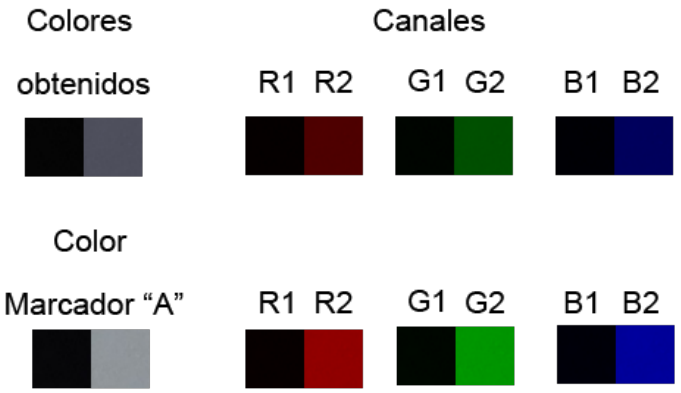


Figura 65: Comparación de colores obtenidos con colores almacenados en la lista de marcadores. [32]

Para obtener que tan cerca se encuentra un canal del otro se aplica la siguiente ecuación para cada canal:

$$O_r = \frac{255 - abs(obtenido_{R1} - marcador_{R1})}{255} \quad M_r = \frac{255 - abs(obtenido_{R2} - marcador_{R2})}{255}$$
$$O_g = \frac{255 - abs(obtenido_{G1} - marcador_{G1})}{255} \quad M_g = \frac{255 - abs(obtenido_{G2} - marcador_{G2})}{255}$$
$$O_b = \frac{255 - abs(obtenido_{B1} - marcador_{B1})}{255} \quad M_b = \frac{255 - abs(obtenido_{B2} - marcador_{B2})}{255}$$

Donde el subíndice de O indica el canal del color obtenido y el subíndice de M indica el canal del marcador almacenado, se divide entre 255 para poder normalizarlo en un valor entre 0 y 1. Posteriormente se promedian los nuevos valores de cada canal como se ve en la siguiente ecuación:

$$P_O = \frac{O_r + O_g + O_b}{3} \qquad P_M = \frac{M_r + M_g + M_b}{3}$$

Este procedimiento se realiza para cada uno de los colores obtenidos, por lo que se tiene un promedio por cada uno de los colores del marcador. Por ultimo se calcula un

[32] Elaboración propia

nuevo promedio para ver que tan cerca se encuentran los valores P_O *y* P_M de los dos colores obtenidos a los dos colores almacenados mediante:

$$P_F = \frac{P_O + P_M}{2} \tag{21}$$

Este promedio final es agregado a una nueva lista temporal en donde se guarda el promedio final junto con el nombre del marcador con el que se comparó. Una vez realizada todas las comparaciones se ordena la lista de mayor promedio a menor, y se extrae el primer elemento de la lista el cual es el que tiene una mayor probabilidad de ser el marcador con el que se comparó. Como se menciono anteriormente estos marcadores deben de ser entrenados dentro de la arquitectura para poder ser identificados de manera general para posteriormente aplicar el método anterior para identificar la ubicación asignada a los colores del marcador.

Una vez terminado el entrenamiento de las arquitecturas Faster R-CNN y SSD con todas las clases requeridas se tendrá a la salida varios archivos con extensiones: "data" y "meta" y se encuentran en formato "chkp" o formato de punto de control. Estos archivos que se pueden observar en la Figura 66 son generados para cada punto de control establecido para el entrenamiento de cada una de las arquitecturas.

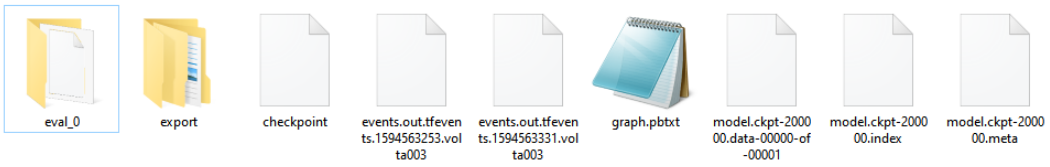


Figura 66: Archivos obtenido del entrenamiento de la arquitectura Faster R-CNN. [33]

El archivo con extensión ".data" contiene los pesos del entrenamiento realizado además de valores de otras variables y por último el archivo con extensión "meta" contiene el grafo que describe el modelo o más explícitamente las capas internas de la arquitectura con el fin de que se puedan usar estos archivos para reentrenar la red nuevamente, se puede ver un ejemplo de lo que contiene este archivo en la Figura 67.

[33] Elaboración propia

```
name: "FeatureExtractor/MobilenetV2/expanded_conv_4/depthwise/BatchNorm/beta/RMSProp_1/Assign"
op: "Assign"
input: "FeatureExtractor/MobilenetV2/expanded_conv_4/depthwise/BatchNorm/beta/RMSProp_1"
input: "FeatureExtractor/MobilenetV2/expanded_conv_4/depthwise/BatchNorm/beta/RMSProp_1/Initializer/zeros"
attr {
  key: "T"
  value {
    type: DT_FLOAT
  }
}
attr {
  key: "_class"
  value {
    list {
      s: "loc:@FeatureExtractor/MobilenetV2/expanded_conv_4/depthwise/BatchNorm/beta"
    }
  }
}
attr {
  key: "use_locking"
  value {
    b: true
  }
}
attr {
  key: "validate_shape"
  value {
    b: true
  }
}
```

Figura 67: Ejemplo de la información contenida del archivo "meta" [34]

Lo anterior funciona perfectamente cuando se piensa seguir modificando la red agregando nueva información al dataset de entrenamiento, pero cuando se quiere usar el modelo en un entorno real o de producción no es conveniente ya que no se requiere de toda la información contenida en estos archivos por lo que es más conveniente tener el modelo y sus pesos en un solo archivo. Esto es posible realizar mediante tensorflow mediante un este proceso se le conoce como "congelar" el modelo, pero más comúnmente encontrado como "freezing", este nos permite combinar los archivos necesarios y remover la información innecesaria. Para poder lograr esto es necesario realizar los siguientes pasos: primero es necesario cargar el grafo obtenido al final del entrenamiento el cual contiene nuestro modelo el cual describe las capas, posteriormente es necesario cargar los pesos obtenidos del entrenamiento, después se remueven todos los metadatos que no son necesarios para el proceso de inferencia del modelo y por último la información ya procesada es serializada en un nuevo formato y exportada en un nuevo archivo con extensión "pb". Una vez realizado este proceso entonces ya solo es necesario volverlo a cargar para ejecutar el proceso de inferencia el cual consiste simplemente en evaluar al modelo con alguna imagen o vídeo de entrada y poder observar la salida.

Posteriormente si uno requiere de ejecutar este modelo en un dispositivo móvil con menores capacidades a las de un equipo de cómputo como un dispositivo móvil o una tarjeta de desarrollo como lo es la Jetson Nano de Nvidia entonces es necesario optimizar el modelo obtenido de tal manera que este pueda ser ejecutado de manera eficiente en el dispositivo.

Para este trabajo de tesis se hace uso de la tarjeta de desarrollo Jetson Nano de Nvidia la cual cuenta con un CPU Quad-core ARM a57 que funciona a 1.43 Ghz, tiene una GPU Maxwell con 128 núcleos CUDA, 4 GB de Ram LPDDR4, almacenamiento mediante una tarjeta microSD, cuenta con un puerto HDMI, 4 puertos USB 3.0, 1 puerto micro USB para alimentación, posee 40 pines GPIO con capacidad de protocolos de

[34] Elaboración propia

comunicación como I2C, SPI y UART. Esta tarjeta de desarrollo tiene unas dimensiones reducidas de 100 x 80 x 29 mm y puede entregar un poder de cómputo de hasta 472 GFLOPS. Este se puede observar a continuación en la Figura 68.

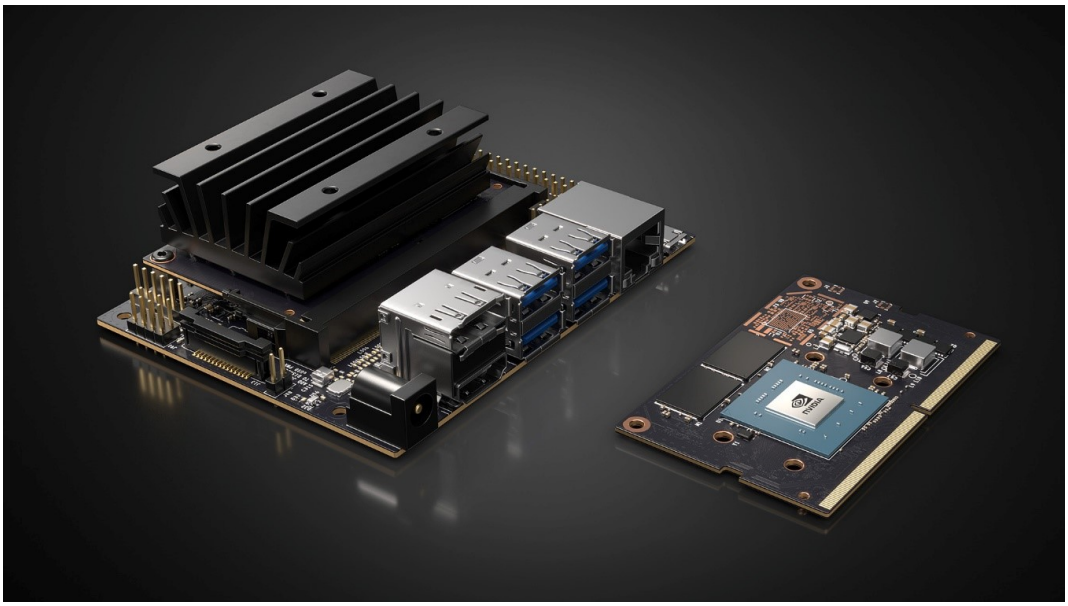


Figura 68: Tarjeta de desarrollo Jetson Nano.[35]

Esta tarjeta, gracias a que cuenta con una GPU la cual posee la capacidad de ejecutar modelos de redes neuronales como son los propuestos anteriormente para el problema de detección y localización de objetos., esto es posible gracias a sus capacidades además de que posee los puertos necesarios para poder acoplar una cámara web. Para que los modelos se ejecuten de manera eficiente es necesario optimizar los pesos mediante el Nvidia TensorRT[34] el cual es un SDK de alto rendimiento para inferencia para Deep Learning. Este incluye un optimizador de inferencia que permite reducir el tiempo de ejecución y aumentar el rendimiento del modelo optimizado. En la Figura 69 se puede observar las optimizaciones que lleva a cabo sobre el modelo entrenado con el fin de mejorar su desempeño.

[35]Recuperada de `http://cms.ipressroom.com.s3.amazonaws.com/219/files/20192/jetson-nano-family-press-image-hd.jpg`

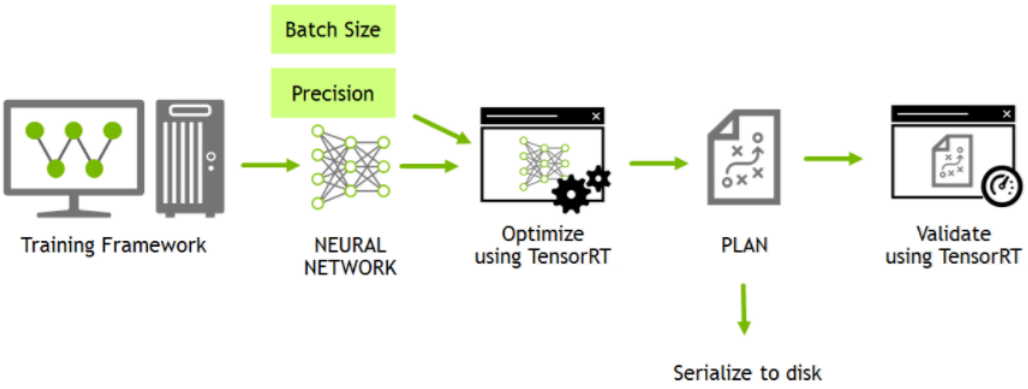
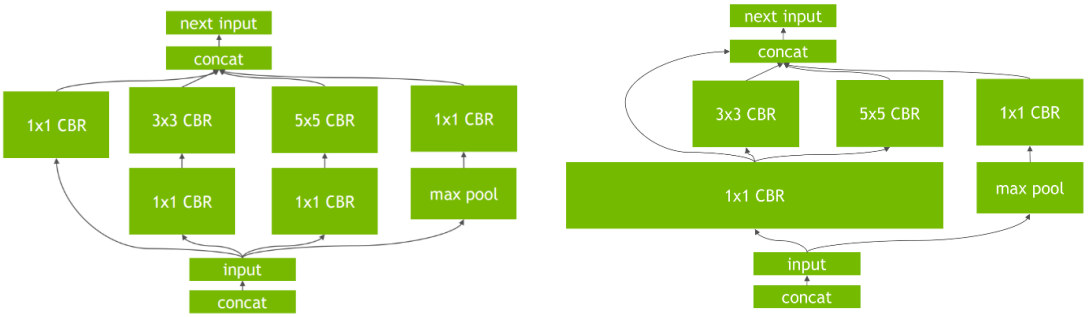


Figura 69: Optimización realizada por TensorRT sobre un modelo entrenado.[36]

De entre las optimizaciones que lleva a cabo TensorRT se encuentran optimizaciones en los grafos del modelo mediante la reducción de nodos. En la Figura 70a se puede observar el grafo original mientras que en la Figura 70b se puede observar el grafo optimizado mediante un proceso llamado fusión vertical el cual combina los nodos ReLU, bias y la convolución de la Figura 70a en un solo nodo CBR de la Figura 70b lo que reduce la ejecución de tres nodos a uno solo.



(a) Grafo sin optimizar **(b)** Grafo optimizado

Figura 70: Optimizaciones realizadas en el grafo[37]

Otra optimización llevada a cabo por TensorRT consiste en disminuir la precisión de los parámetros de la red, originalmente estos parámetros durante el proceso de entrenamiento son del tipo FP32 (Floating Point 32 bit) ya que estos son ideales para ajustar los parámetros de la red en cantidades muy pequeñas pero el problema que se tiene con esto al ejecutarse en dispositivos con menos capacidades es que requiere de más

[36]Recuperada de `https://miro.medium.com/max/700/1*7zvHEeYhiHBqjO_pzlWKww.png`

[37]Recuperadas de `https://developer.nvidia.com/tensorrt`

memoria para almacenar este tipo de valores por lo que TensorRT aborda ese problema al cambiar la precisión de los parámetros de FP32 a INT8 con un impacto mínimo en la precisión de la red, para lograr esto hace uso de la "Cuantificación lineal simétrica" la cual es una operación de escalado desde el rango de FP32 al rango de INT8. Una vez realizadas las optimizaciones del modelo mediante TensorRT este se encuentra listo para ser ejecutado. Con estas optimizaciones se busca garantizar el mayor rendimiento posible en la Jetson Nano y aprovechar la GPU que esta posee.

4.3. Sistema de control para el manipulador de objetos

Para poder tomar los objetos es necesario establecer un algoritmo de control que nos permita posicionar al robot de tal manera que el manipulador pueda tomar el objeto que se le pone de frente, para poder posicionar el robot se realizó de la siguiente manera: Se toma la anchura de la imagen de la cámara y se divide entre 2 para calcular el punto medio como se ve en la ecuación:

$$mid_x = \frac{width}{2} \tag{22}$$

Donde *width* es la anchura de la resolución de la cámara. Posteriormente se calcula la posición media de la anchura del bounding box del objeto que se desea tomar con la siguiente ecuación:

$$object_{mid_x} = \frac{X_{max} - X_{min}}{2} \tag{23}$$

En donde X_{max} es la coordenada en x en donde termina el bounding box y X_{min} es la coordenada en x donde inicia el bounding box.

Una vez realizado esto se procede a determinar de que lado de la imagen se encuentra el bounding box para poder conocer la dirección en que el robot tiene que rotar. Se tiene que:

$$dir = \begin{cases} L & object_{mid_x} < mid_x \\ R & object_{mid_x} > mid_x \end{cases} \tag{24}$$

Una vez determinada la dirección que debe de rotar el robot es necesario calcular que tanto tiene que girar el robot a la izquierda o a la derecha, para hacer esto primero se calculó la proporción que se tienen de grados por pixel en la imagen. Para esto se realizó de esta manera:

$$px_{proportion} = \frac{width}{camera_{fov}} \tag{25}$$

Donde *width* es la anchura de la imagen de la cámara y $camera_{fov}$ son los grados de visión que posee la cámara, para el prototipo la cámara que se utiliza es de 78°.

Con este valor obtenido el siguiente paso es calcular la distancia en pixeles del centro del objeto al centro de la imagen de la cámara mid_x de la siguiente manera:

$$px_{distance} = |mid_x - object_{mid_x}| \tag{26}$$

Se calcula el valor absoluto ya que la dirección la obtuvimos con la ecuación 24 por lo que solo requerimos la distancia sin importar el signo.

Una vez obtenida la distancia necesaria para centrar al objeto con la cámara se procede a calcular cuantos grados debe de girar el prototipo para poder centrarse al objeto, esto se realiza de la siguiente manera:

$$rotation_{degrees} = \frac{px_{distance}}{px_{proportion}} \tag{27}$$

El problema que existe con esto es que el prototipo acepta como parámetros milisegundos en lugar de grados por lo que se tiene que caracterizar el prototipo para conocer en que tiempo se tarda en mover una determinada cantidad de grados como se muestra en la siguiente ecuación:

$$degrees_{proportion} = \frac{rotated_{time}}{rotated_{degrees}} \tag{28}$$

Donde $rotated_{time}$ es el tiempo que se obtuvo del prototipo al moverse la cantidad de grados: $rotated_{degrees}$

Por último, para obtener el tiempo que es necesario que se mueva el prototipo en función a los grados obtenidos se realiza de la siguiente manera:

$$calc_{ms} = rotation_{degrees} * degrees_{proportion} \tag{29}$$

Donde $calc_{ms}$ entrega el valor en milisegundos. Con este dato se le manda al prototipo la dirección en la que debe rotar y el tiempo que debe realizarlo. En la Figura 71 se puede observar una línea azul la cual indica donde se encuentra el centro de la imagen. Se puede observar la bounding box del objeto de color verde y una línea rosa desde la mitad del objeto a la mitad de la imagen, esta es la distancia necesaria para que el objeto se centre.

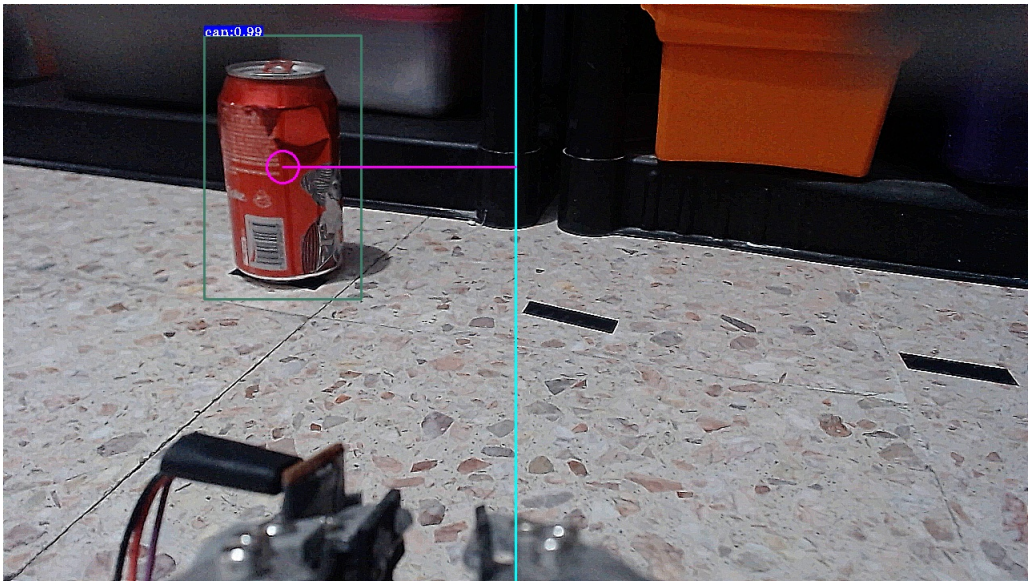


Figura 71: Ejemplo de elementos necesarios para centrarse con respecto al objeto. [38]

Una vez realizado esto el robot se encontrará casi completamente cerca del centro de la imagen de la cámara, pero para cerciorarse que este se encuentre completamente centrado el proceso se repite de nuevo hasta que la distancia del centro del objeto al centro de la imagen sea mínima. Ya que el prototipo se encuentra centrado con respecto al objeto entonces se procede a medir la distancia desde el prototipo al objeto, una vez obtenido este valor el prototipo procederá a avanzar en línea recta la distancia medida, esto permitirá que la garra se encuentre en alcance del objeto, entonces el manipulador procede a cerrarse para poder tomar el objeto, para cerciorarse que el objeto fue tomado el manipulador monta un sensor que se activa cuando se tiene sujeto un objeto.

Así pues, con el objeto tomado el último paso es buscar los marcadores con la cámara para poder obtener las indicaciones de donde llevar el objeto a su destino final.

[38] Elaboración propia

Capítulo 5

Pruebas y resultados

En este capítulo presentaremos las pruebas realizadas en los modelos de visión artificial para las arquitecturas propuestas. Con cada prueba realizada se mostrarán los resultados obtenidos junto con un análisis de esa prueba con el fin de identificar los puntos de mejora para perfeccionar los resultados en las pruebas consecuentes. Posteriormente se mostrarán los resultados obtenidos con la integración en el prototipo móvil.

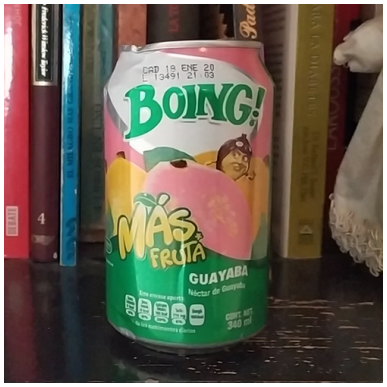
5.1. Pruebas

Para poder realizar estas pruebas se partió de la creación del dataset de imágenes, originalmente se establecieron las clases de objetos a detectar además de clases de excepción la cual nos permitirá descartar ciertos objetos con el fin de evitar confundir los objetos principales. Las clases principales que se establecieron originalmente son:

- **Tetra:** Objetos como envases de jugo, leche, crema, etc.
- **Bottle:** Botellas de plástico/vidrio de refresco, jugo, etc.
- **Can:** Latas de diversos tamaños de jugo y refresco
- **Bottle-cap:** Taparroscas de botellas de plástico
- **Paper:** Hojas de papel de distintos tamaños, colores y formas.

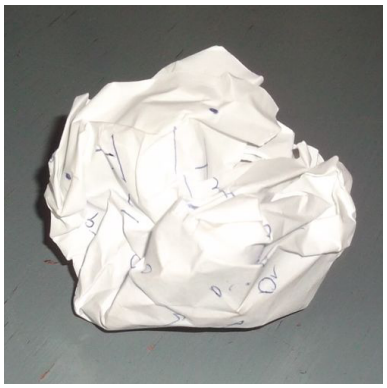
En la Figura 72 se puede observar algunos ejemplos de los objetos de las clases principales:

(a) Tetra



(b) Can

(c) Bottle



(d) Paper

(e) Bottle-cap

Figura 72: Ejemplos de las distintas clases.[2]

Para las clases de excepción se establecieron:

[2]Elaboración propia.

- **Dc-adapter:** Adaptadores de carga de dispositivos electrónicos.
- **Box:** Toda clase de cajas excepto las que entran en la clase Tetra.
- **Trash-can:** Botes de basura de distintos tamaños, formas y colores.

En la Figura 73 se puede observar algunos ejemplos de las clases de exclusión.

(a) Box

(b) Dc-adapter

(c) Trash-can

Figura 73: Ejemplos de las clases de exclusión.[3]

Una vez definidas las distintas clases a utilizar se procedió a diseñar el dataset a utilizar para entrenar el sistema de visión artificial, para esto se tomaron en cuenta ciertas características con las cuales se busca tener los mejores resultados posibles. Es importante recalcar que este proceso es iterativo e incremental ya que busca construir el mejor dataset posible para detectar objetos por lo que esta ira incrementando y cambiando en función a los resultados que se vayan presentando, con esto se busca poder

[3]Elaboración propia.

identificar que objetos funcionan mejor, desde que distancia pueden ser detectados, desde qué posiciones y colores para así poder ser ajustado.

Se tomaron los siguientes criterios para las imágenes que forman parte del dataset:

- Resolución de imagen de 1920x1080 px
- El objeto debe encontrarse en distintas posiciones
- El objeto debe encontrarse a distintas distancias de la cámara para variar su tamaño en la fotografía.
- El objeto debe tener fondos variados ya que esto permitirá extraer más características de cada uno de los objetos para cada clase.
- El objeto puede ser parcialmente visible
- Pueden aparecer varios objetos en una misma imagen, pero sin sobreponerse uno a otro.

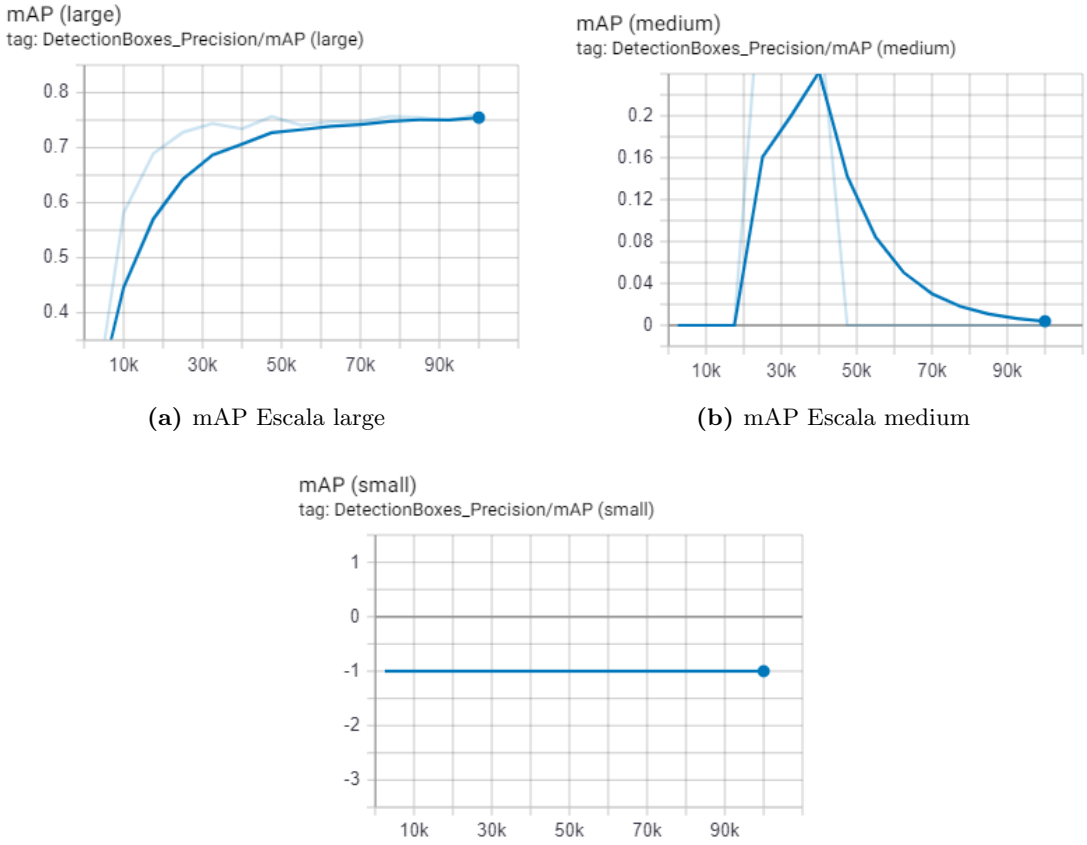
Para las primeras pruebas se procedió de manera manual a tomar fotografías de varios objetos en distintas posiciones y fondos, con esto se obtuvo un total de 2450 imágenes las cuales posteriormente fueron etiquetadas de manera manual. El proceso de etiquetado consiste en seleccionar en una imagen en donde se encuentra el objeto de interés e indicar a que clase pertenece mediante una etiqueta previamente definida, en este caso son: tetra, bottle, can, bottle-cap, paper, dc-adapter, box y trash-can.

Una vez realizado esto se procedió a dividir el dataset en training y test, esto se realiza con el fin de evitar un sobre ajuste, se dividió en un porcentaje de 70/30 para entrenamiento y validación. Como se mencionó anteriormente se hace uso de la técnica conocida como Transfer Learning la cual nos permite partir de un modelo ya entrenado en una tarea similar, en este caso la detección de objetos, para esto se utilizaron modelos entrenados en el dataset COCO y se aplicó esta técnica. Se hizo uso de los modelos faster_rcnn_inception_v2_coco[39] y ssd_mobilenet_v2_coco[39], los cuales usan la arquitectura Faster R-CNN y SSD respectivamente. El entrenamiento se llevó a cabo haciendo el uso del clúster IBM Power9 del LNS[41] el cual cuenta con 2 procesadores Power9 de 20 núcleos, 1024GB de memoria RAM, 2 discos duros SSD de 1.9 TB y 4 GPUs Nvidia V100 con 16Gb de VRAM las cuales fueron utilizadas con la versión 1.14 de Tensorflow GPU para aprovechar su potencia de cálculo. A continuación, se presentan los resultados para esta primera prueba. En la Tabla 2 se pueden observar los tiempos de entrenamiento de cada uno de los modelos y arquitecturas.

En la Figura 74 se pueden observar las gráficas mAP del entrenamiento de Faster R-CNN.

Modelo	Tiempo de entrenamiento
Faster R-CNN	2h 41m
SSD	5h 44m

Tabla 2: Tiempo de entrenamiento para la Prueba 1



(a) mAP Escala large

(b) mAP Escala medium

(c) mAP Escala small

Figura 74: Primer dataset con 2450 imágenes.[4]

El mAP o también conocido como mean average precisión es una métrica utilizada para medir el desempeño de un modelo, para medir este existen diversos métodos, pero nos enfocaremos principalmente en dos métodos: en función a escalas y en función al IoU de los objetos. El primero es utilizado por Faster R-CNN y hace el cálculo del mAP a través de escalas las cuales representan el área que ocupa el objeto con respecto a la

[4]Elaboración propia.

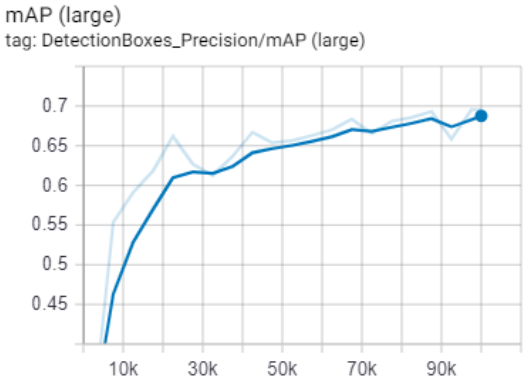
imagen total, estas son:

- mAP_{small} El cual es utilizado para objetos pequeños en donde: $area < 32^2 px$
- mAP_{medium} El cual es utilizado para objetos de tamaño medio en donde: $area > 32^2 px$ y $area < 96^2 px$
- mAP_{large} El cual es utilizado para objetos de tamaño grande en donde: $area > 96^2 px$

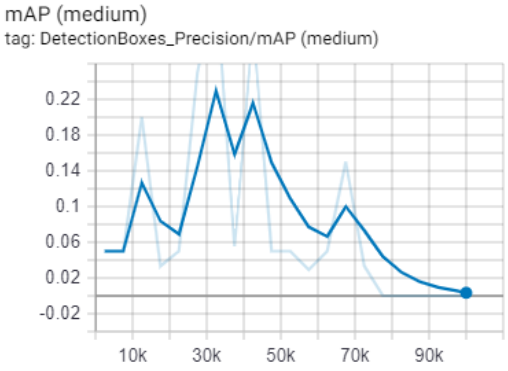
Esto permite una mejor diferenciación en el modelo en función al tamaño de los objetos presentes en el dataset. El mAP aumentará mientras más clases tengan más detecciones correctas ya que todas estas se promedian por lo que si algunas clases se desempeñan muy bien, pero otras no tanto afectarán al mAP de la escala y este no aumentará.

Retomando las gráficas de la Figura 74, en la Figura 74a se muestra el mAP para objetos grandes, al final del entrenamiento alcanza un 0.76 o 76 % de precisión, en la figura 74b se muestra el mAP para objetos de tamaño medio con 3.8 e-3 lo que indica que tiene dificultad para identificar objetos de tamaño medio, y en la Figura 74c se muestra el mAP para objetos pequeños en donde se ve un -1, esto indica que no puede identificar objetos pequeños.

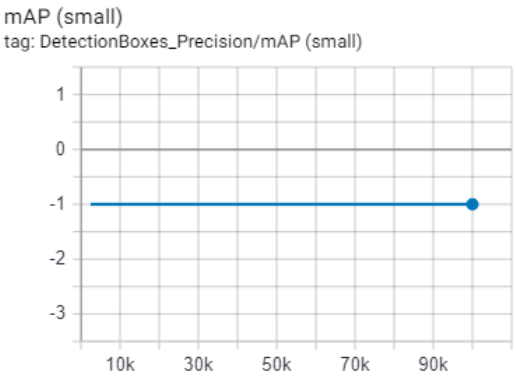
En la Figura 75 se muestra el mAP obtenido para SSD para cada una de las escalas posibles, en la Figura 75a se muestra el mAP para objetos grandes con un valor de 0.6952, en la Figura 75b se muestra el mAP para objetos de tamaño medio con un valor de 3.41e-3 y en la Figura 75c el mAP para objetos pequeños con un valor de -1.



(a) mAP Escala large



(b) mAP Escala medium



(c) mAP Escala small

Figura 75: SSD Prueba 1.[5]

Después de realizada la primera prueba se detectó que un problema importante para realizar el entrenamiento es la necesidad de incrementar el dataset y hacerlo de manera manual conlleva mucho tiempo por lo que se optó por tomar secuencias de vídeo de los objetos para posteriormente realizar un programa en python el cual se encarga de extraer los cuadros del vídeo. Con esto se logró incrementar el dataset en alrededor de 1000 imágenes que tuvieron que ser etiquetadas de manera manual.

Para la segunda prueba el dataset pasó a tener 3450 imágenes las cuales fueron divididas en 70/30 para training y test. En la Tabla 3 se pueden observar los tiempos de entrenamiento para cada una de estas.

En la Tabla 4 se pueden observar los resultados obtenidos tanto para Faster R-CNN como para SSD en sus tres escalas.

Con la segunda prueba se procedió a hacer uso de una cámara de video para poder probar los modelos entrenados para poder observar su desempeño de manera real y no

[5]Elaboración propia.

Modelo	**Tiempo de entrenamiento**
Faster R-CNN	2h 49m
SSD	5h 56m

Tabla 3: Tiempo de entrenamiento para la Prueba 2

Arquitectura	**Escala**	**mAP**
Faster R-CNN	Large	0.7677
Faster R-CNN	Medium	0.1715
Faster R-CNN	Small	-1
SSD	Large	0.6862
SSD	Medium	0.2626
SSD	Small	-1

Tabla 4: Resultados obtenidos para la Prueba 2

solo mediante las gráficas obtenidas de estos. Faster R-CNN mostró mejora al detectar objetos que se le mostraban, el principal problema que se detecto es que tenía dificultad para identificar objetos que se encontrarán muy lejos o que tuvieran un fondo con todos oscuros. Con respecto a SSD este mostró mejoría con respecto a la primera prueba al ya poder detectar objetos. Para poder mejorar los resultados con lo observado anteriormente se concluyó que era necesario incrementar el dataset con imágenes de objetos a distancias más lejanas y con más variaciones de fondos.

Gracias a los resultados obtenidos en la segunda prueba al tener un incremento en el mAP en la escala large y medium en ambas arquitecturas se procedió a crear un programa de auto etiquetado de video, este recibe una secuencia de video de entrada y esta es pasada al modelo ya entrenado para realizar las detecciones y posteriormente crear un archivo XML el cual contiene los bounding box de los objetos detectados y su etiqueta. Posteriormente se grabaron un total de 26 Gb de video los cuales fueron pasados a este programa y esto permitió obtener 2500 imágenes más para agregar al dataset. Es importante mencionar que, aunque ya se realizó el proceso de etiquetado de manera automática se verificaron estas de manera manual para cerciorarse que fueran correctas.

Además de esto otro problema que se presentaba en este punto es que las imágenes eran separadas manualmente en el dataset de train y de test, por lo que se procedió a crear un programa el cual de manera automática, aleatoria y reproducible se encarga de separar las imágenes solo indicando el porcentaje del dataset de train y el restante

es asignado a test.

Una vez implementados estos cambios y con un nuevo dataset de 5950 imágenes se procedió a realizar la tercera prueba. En la Tabla 5 se pueden observar los tiempos de entrenamiento para cada arquitectura.

Modelo	**Tiempo de entrenamiento**
Faster R-CNN	3h 0m
SSD	7h 10m

Tabla 5: Tiempo de entrenamiento para la Prueba 3

Como se puede observar el tiempo de entrenamiento de Faster con respecto de la segunda prueba aumentó muy poco a pesar de casi doblar la cantidad de imágenes, con respecto a SSD se ve incremento apreciable en el tiempo de entrenamiento, es importante mencionar que el entrenamiento de SSD no se encuentra tan optimizado, por eso el tiempo de entrenamiento mayor. En la Tabla 6 se puede observar los resultados obtenidos para la tercera prueba.

Con esta tercera prueba haciendo uso de la webcam se observó que Faster-RCNN se presentaban unos problemas, en ocasiones algunos objetos eran confundidos con otra clase como lo son las latas, estas eran confundidas con las cajas de jugo de la misma marca, esto se debió principalmente a que el extractor de características tomo el logo como una característica predominante por lo que es más propenso a confundir los objetos cuando este logo se encuentra presente.

Con respecto al nulo incremento del mAP para la escala small se procedió a tomar fotografías de manera manual desde una distancia más lejana (3 metros) para buscar que el objeto ocupara la menor área de la imagen.

Otro cambio más realizado después de esta tercer prueba es delimitar mas los objetivos que se tienen con la clase paper, esto se realizó ya que se presentaron complicaciones

Arquitectura	**Escala**	**mAP**
Faster R-CNN	Large	0.7546
Faster R-CNN	Medium	0.3645
Faster R-CNN	Small	-1
SSD	Large	0.6231
SSD	Medium	0.1339
SSD	Small	-1

Tabla 6: Resultados obtenidos para la Prueba 3

Prueba	**Arquitectura**	**Tiempo de entrenamiento**
Prueba 4 (9,024 imágenes)	Faster R-CNN	5h 39m
	SSD	12h 53m
Prueba 5(10,215 imágenes)	Faster R-CNN	7h 43m
	SSD	18h 5m

Tabla 7: Tiempo de entrenamiento para la Prueba 4 y Prueba 5

al confundir elementos como telas o bolsas de plástico con papel, esto por los dobleces que presentaban, por lo que se decidió limitar esta clase a solo detectar pelotas de papel ya que estas poseen características mas distintivas, este cambio también se realizó por que es más fácil manipular una bola de papel que una hoja con pocos dobleces y poco espesor. Para finalizar, el último cambio propuesto fue pasar de los 150k steps a los 200k steps, esto se realizó con el fin de darle la oportunidad al entrenamiento de estabilizarse en un mAP y evitar que este oscile.

Con esto se realizaron mas pruebas para ir incrementando el dataset para la prueba 4 con un total de 9,024 imágenes y la prueba 5 con 10,215 imágenes, solo se reentreno cada vez con más imágenes tomando en cuenta los cambios anteriores y esto implica remover casi todas las imágenes de papel a excepción de las que cumplen las condiciones además de agregar nuevas tomando en cuenta los cambios. En la Tabla 7 se pueden observar los tiempos de entrenamiento para estas dos pruebas, como se puede notar, al incrementar el tamaño del dataset en gran medida también lo ha hecho el tiempo de entrenamiento necesario para las pruebas 4 y 5 respectivamente.

En la tabla 8 se puede observar los resultados obtenidos para la cuarta y quinta prueba de los modelos entrenados para cada arquitectura.

Con esta cuarta y quinta prueba se observó que Faster R-CNN ya detectaba más objetos de manera real, es importante tomar en cuenta que como se redujo la cantidad de imágenes en la categoría de paper esto afecta negativamente al AP de su categoría y esto se ve reflejado en la disminución del mAP para cada escala.

En cuanto a SSD, se notó que era necesario incrementar el tamaño del dataset ya que a pesar de realizar pruebas de manera real y aunque este ya detectaba más objetos, estos eran asignados a una clase incorrecta por lo que se modificaron unos parámetros de la arquitectura para la siguiente prueba. Para esto en la siguiente prueba se cambio el resize de los hiperparámetros de SSD de 300x300 a 500x500, esto con el fin de que al usar una imagen de mayor tamaño se incrementará el área de detección de los objetos de manera general para cada escala, además de que para la prueba 6 se incrementó el dataset a la cantidad de 11,449 imágenes. Este entrenamiento también se aplicó para Faster R-CNN para aprovechar el incremento del dataset y así continuar mejorando sus resultados.

En la Tabla 9 se pueden observar los tiempos de entrenamiento para la prueba 6.

Prueba	Arquitectura	Escala	mAP
Prueba 4	Faster R-CNN	Large	0.6343
	Faster R-CNN	Medium	0.2251
	Faster R-CNN	Small	-1
	SSD	Large	0.5648
	SSD	Medium	0.1354
	SSD	Small	-1
Prueba 5	Faster R-CNN	Large	0.591
	Faster R-CNN	Medium	0.251
	Faster R-CNN	Small	-1
	SSD	Large	0.5629
	SSD	Medium	0.1721
	SSD	Small	-1

Tabla 8: Resultados obtenidos para la Prueba 4 y Prueba 5

Modelo	Tiempo de entrenamiento
Faster R-CNN	8h 12m
SSD	1d 15h 1m

Tabla 9: Tiempo de entrenamiento para la Prueba 6

En la Tabla 10 se pueden observar los resultados del entrenamiento para la prueba 6 de Faster R-CNN y SSD.

Como se puede observar, el incremento del dataset y la continua adición de imágenes de objetos cada vez mas lejanos nos permitieron tener un incremento en el mAP para ambas arquitecturas en la escala small.

Con el cambio de los hiperparámetros de SSD para pasar de una resolución de 300x300 a una de 500x500 realizando pruebas reales con la cámara web permitió observar un mayor incremento en las detecciones de objetos en distancias cercanas e intermedias (30-50 cm). La desventaja de haber realizado este cambio se presenta en el tiempo de entrenamiento, ya que se pasó de 18 horas a 39 horas, este simple cambio junto con el incremento del dataset duplicó el tiempo de entrenamiento.

Con estos buenos resultados obtenidos con la prueba 6 se procedió a realizar benchmarks en distintos equipos además de usar la tarjeta Nvidia Jetson Nano, para el caso

Arquitectura	**Escala**	**mAP**
Faster R-CNN	Large	0.6462
Faster R-CNN	Medium	0.2661
Faster R-CNN	Small	0.075
SSD	Large	0.6005
SSD	Medium	0.1998
SSD	Small	0.024

Tabla 10: Resultados obtenidos para la Prueba 6

	GTX 1060	**GTX 1050 Móvil**	**Jetson Nano**
Cantidad de imágenes	1000	1000	1000
Resolución	1024x600	1024x600	1024x600
FPS	12.46	6.73	1.47
Tiempo de inferencia	0.08 segundos	0.148 segundos	0.67 segundos
Tiempo total	82.61 segundos	150.35 segundos	681.93 segundos
Características	1280 CUDA, 6 Gb VRAM	640 CUDA, 4 Gb VRAM	128 CUDA, 4 Gb VRAM
Costo	200 USD	$100 USD	$99 USD

Tabla 11: Comparativa del modelo Faster R-CNN en distintas GPUs.

de la Jetson se realizó la optimización mediante TensorRT de los modelos previamente entrenados (prueba 6). En la Tabla 11 se puede observar los resultados del benchmark de Faster R-CNN en distintas configuraciones y en la Tabla K se puede observar los resultados de benchmark para SSD.

Como se puede observar en las comparaciones, tarjetas gráficas como la GTX 1060 o GTX 1050 ofrecen buenos resultados, el problema que radica en estas es que no es una solución completamente integrada, para poder hacer funcionar estas tarjetas se requiere un equipo de grandes dimensiones y gran costo por lo que no es posible montarlo en un prototipo robótico móvil, por esto mismo se optó por hacer uso de la tarjeta Jetson Nano, a pesar de no ofrecer una gran cantidad de FPS la cantidad dada es suficiente para ejecutar el sistema de visión artificial en el prototipo móvil.

Posteriormente se procedió a agregar imágenes de los marcadores desde distintas

	GTX 1060	**GTX 1050 Móvil**	**Jetson Nano**
Imágenes	1000	1000	1000
Resolución	1024x600	1024x600	1024x600
FPS	29.49	15.93	3.48
Tiempo inferencia	0.05225 segundos	0.0967 segundos	0.437 segundos
Tiempo total	52.25 segundos	96.75 segundos	437.95 segundos
Características	1280 CUDA, 6 Gb VRAM	640 CUDA, 4 Gb VRAM	128 CUDA, 4 Gb VRAM
Costo	200 USD	$100 USD	$99 USD

Tabla 12: Comparativa del modelo SSD en distintas GPUs.

posiciones en las cuales serán vistas por el prototipo móvil, con estas nuevas imágenes el dataset pasó a tener un tamaño de 12,111 imágenes. Una vez agregadas estas imágenes se procedió a realizar el entrenamiento de SSD en su versión 500x500 y 300x300 para hacer una comparativa de ambas además de entrenar Faster R-CNN. En la Tabla 13 se puede observar el tiempo de entrenamiento.

Modelo	**Tiempo de entrenamiento**
Faster R-CNN	10h 16m
SSD 300x300	1d 19h 43m
SSD 500x500	2d 0h 37m

Tabla 13: Tiempo de entrenamiento para la Prueba 7

Con los tiempos anteriores se puede observar que con solo el cambio de resolución el entrenamiento aumentó en más de 4 horas. En la Tabla 14 se puede observar los resultados obtenidos para cada versión del modelo en la prueba 7.

Como se puede observar Faster R-CNN mostró un incremento considerable en su mAP en la escala Large y Medium. Con respecto a ambas versiones de SSD se puede observar que la variación entre ambos mAP es poca entre las distintas versiones. Realizando pruebas reales de detectó que todas las arquitecturas y sus variantes presentaban problemas para identificar los marcadores por lo que se procedió a agregar más imágenes de marcadores para posteriormente entrenar y ver los resultados obtenidos. Alternamente a estas pruebas se realizó una en SSD con resolución de 1000x1000 la cual no fue tomada en cuenta para posteriores pruebas ya que esta tomaba una excesiva

Arquitectura	**Escala**	**mAP**
Faster R-CNN	Large	0.7408
Faster R-CNN	Medium	0.5096
Faster R-CNN	Small	0.064
SSD 300x300	Large	0.6508
SSD 300x300	Medium	0.1916
SSD 300x300	Small	0.077
SSD 500x500	Large	0.6657
SSD 500x500	Medium	0.2305
SSD 500x500	Small	0.069

Tabla 14: Resultados obtenidos para la Prueba 7

cantidad de tiempo en entrenar: 4 días con 7 horas y esta presentaba mAP inferiores a los obtenidos con la versión de 300x300 y 500x500 por lo que se descartó incrementar más la resolución.

Posteriormente se realizaron las pruebas de los modelos en la tarjeta Jetson Nano que irá montada en el prototipo, se optó por descartar el uso de la arquitectura Faster R-CNN ya que el modelo a pesar de haber sido optimizado mediante TensorRT no siempre era cargado correctamente en la tarjeta debido a su tamaño, por esto mismo se procedió a hacer uso solo de SSD para el desarrollo del prototipo ya que este no presento ningún problema al ser cargado en la tarjeta. Igualmente, también se optó por solo hacer uso de la versión 300x300 de SSD ya que las mejoras entre esta y la versión de 500x500 fue mínimo al ya haber aumentado el dataset a un tamaño suficiente para que este pueda aprender mejor las características de los objetos.

Asimismo, se decidió mejorar más el dataset con el fin de obtener mejores resultados en la detección, para esto se usó la técnica "Data Augmentation" la cual consiste en generar de manera artificial nuevos datos para el dataset mediante la introducción de perturbaciones y variaciones en las imágenes con el fin de obtener mayor diversidad. Para esto se introdujeron 3 modificaciones a las imágenes existentes, la primera de estas es la disminución del brillo de las imágenes como se puede observar en la Figura 76a, la segunda consiste en mejorar la nitidez de las imágenes como se puede observar en la Figura 76b y por último se generó un desenfoque de movimiento en las imágenes como se puede observar en la Figura 76c. Gracias a el uso de esta técnica el dataset pasó a tener un tamaño de 48,444 imágenes.

(a) Disminución de brillo.

(b) Mejora de nitidez.

(c) Desenfoque de movimiento.

Figura 76: Data augmentation.[6]

Con este incremento del dataset se procedió a realizar la prueba 8 solo para la arquitectura SSD de 300x300, en la Tabla 15 se puede observar el tiempo de entrenamiento de esta prueba.

Como se puede observar el incremento del tiempo fue mínimo, esto se debe a que

[6]Elaboración propia.

Modelo	**Tiempo de entrenamiento**
SSD 300x300	1d 21h 36m

Tabla 15: Tiempo de entrenamiento para la Prueba 8

Arquitectura	**Escala**	**mAP**
SSD 300x300	Large	0.6045
SSD 300x300	Medium	0.1276
SSD 300x300	Small	0.02416

Tabla 16: Resultados obtenidos para la Prueba 8

las nuevas imágenes son solo modificaciones pequeñas con respecto de las originales, por lo que le es más fácil al extractor de características abstraer los objetos dentro de las imágenes. En la Tabla 16 se puede observar los resultados obtenidos para la prueba 8.

5.1.1. Técnica de mejora de detección basada en recortes

Ante los resultados observados en la implementación del robot se observó que ante distintas condiciones de iluminación el robot presenta dificultad para detectar los objetos a distancias mayores a 80 cm lo cual se puede corroborar con la Tabla 16 en donde el modelo todavía encuentra dificultad para detectar objetos pequeños y por ende más lejanos. Debido a esto se diseñó un nuevo enfoque de detección basado en el recorte del área de observación. En la red neuronal ingresa una imagen de tamaño m x n pero la red se encarga de disminuir la resolución de esta y ajustarla para poder procesarla por lo que esta no depende de una resolución fija. Tomando esto en cuenta entonces es posible ingresar regiones de la imagen a la red en lugar de la imagen completa y esta será evaluada para ver si se encuentra un objeto dentro de esta. Partiendo de esto se propone que partiendo de una imagen de tamaño m x n es posible extraer regiones de esta y posteriormente ingresarlas a la red, la justificación de hacer esto es que si se encuentra un objeto dentro de la imagen y esta se recorta el objeto pasará a tomar una mayor área dentro de la imagen y por ende el objeto se encuentre en una mayor escala de la arquitectura, por ejemplo, si se encuentra un objeto en la imagen original con escala “small” pero se recorta esta imagen, al tomar mayor área dentro del recorte ahora el objeto se encontrará en la escala “médium”. Gracias a esta técnica es posible ver objetos desde una distancia más lejana al aumentar el área que ocupan estos dentro e la imagen, pero además de esto, con objetos más cercanos se tiene la posibilidad de mejorar la calidad de la detección del objeto. El algoritmo disminuye el largo de

la imagen en 160 pixeles en cada extremo, este es utilizado para obtener las regiones ocupadas por estos recortes y es el siguiente:

```
i = 3
resolucion_original = (1920, 1080)
coordenadas_recortes = []
for i in range(i):
    inicio_x = (160 / 2) * (i)
    inicio_y = 0
    coordenadas_inicio = (int(inicio_x), int(inicio_y))
    fin_x = resolucion_original[0] - ((160 / 2) * (i))
    fin_y = fin_x / m
    coordenadas_fin = (int(fin_x), int(fin_y))
    recorte = [coordenadas_inicio, coordenadas_fin]
    coordenadas_recortes.append(recorte)
```

Al finalizar la ejecución de este algoritmo este devuelve una lista con las coordenadas que ocupa cada una de las regiones a utilizar la cual posteriormente es utilizada para seleccionar y evaluar las regiones de la imagen original.

Esta técnica fue implementada en el prototipo para mejorar el rango de detección de objetos, para esto de la imagen original tomada se hacen 3 recortes cada vez más pequeños con el fin de hacer que el objeto ocupe una mayor área con respecto de la imagen. Por lo que el prototipo al solicitar la detección de objetos evalúa 4 imágenes en lugar de 1, evalúa la imagen original más las 3 regiones recortadas. En la Figura I se puede observar las regiones que son recortadas de la imagen.

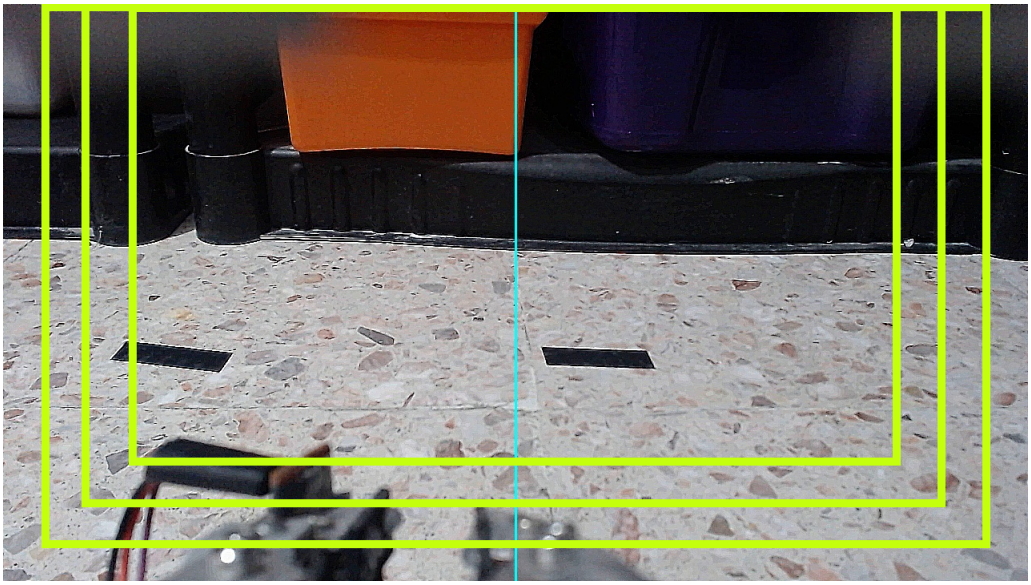


Figura 77: Zonas de recorte de la imagen.[7]

5.2. Resultados

A continuación, se presentan los resultados obtenidos del trabajo de tesis presentado, se mostrarán las especificaciones finales del prototipo junto con diversas pruebas realizadas de manera real con el prototipo y en un entorno controlado para evaluar su desempeño en las tareas a cumplir.

En la Figura 78 se muestra una fotografía del prototipo final, este cuenta con unas dimensiones de 30 cm de largo por 24 cm de ancho por 16.5 cm de alto, es importante mencionar que el alto y el largo del prototipo puede variar en función de la posición en la que se encuentre el brazo manipulador.

[7]Elaboración propia.

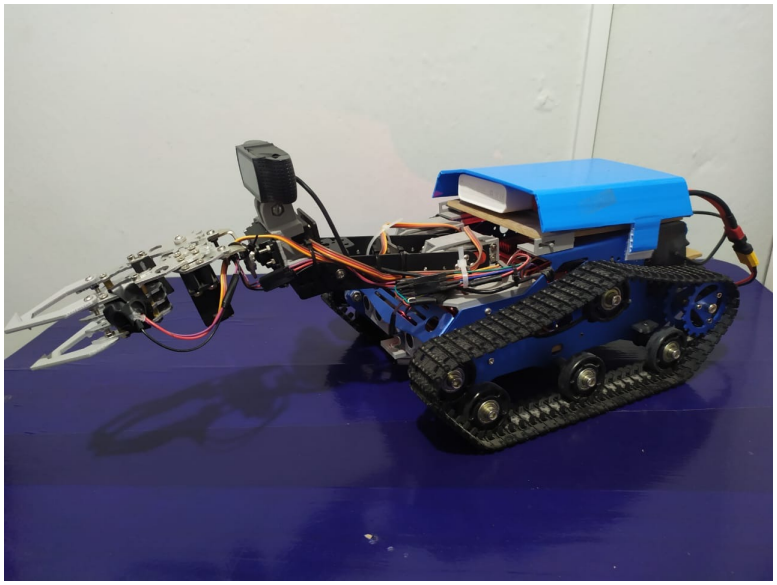


Figura 78: Fotografía del prototipo final.[8]

El prototipo cuenta con dos baterías, una de 2200 mAh que puede proporcionar hasta 44 A y otra batería de 20000 mAh que puede proporcionar hasta 2.4 A, gracias a estas baterías el prototipo puede funcionar de manera ininterrumpida durante 3 horas aproximadamente, posteriormente se requieren de 3 horas para poder recargar las baterías completamente.

El prototipo final hace uso de la arquitectura SSD ya que esta le permite tener una mayor tasa de cuadros por segundo además de requerir una menor cantidad de memoria con respecto a Faster R-CNN ya que esta muchas veces no era cargada correctamente por falta de memoria. Es importante mencionar que solo fueron utilizadas las clases bottle y can ya que la clase tetra la que contenía cajas de juego, leche, etc. Las cuales eran fácilmente confundidas con otros objetos que poseen la misma forma rectangular, se descartó la clase "bottle-cap" a la cual pertenecen las taparroscas de botellas de plástico ya que estas no se encontraban en el rango de manipulación del brazo robot además de que al ser objetos tan pequeños estos muchas veces no eran detectados mas que a distancias muy cercanas y por último se descartó la categoría paper donde primero se establecieron hojas de papel y posteriormente se delimitó a pelotas de papel ya que estas tampoco entraban en rango de manipulación del brazo robot además de que al tener infinitas posibilidades de ser plegadas provocaron que estas no siempre fueran detectadas por el detector de objetos además de ser confundidas por otros objetos por sus características.

Para poder medir el desempeño del prototipo completo se establecieron una serie de pruebas para poder determinar su rango de funcionalidad, la primera prueba consistió en colocar el prototipo en una línea de visión directa a diversos objetos y a distintas

[8]Elaboración propia.

distancias para poder determinar desde donde es posible detectar objetos, en la Figura 79 se puede observar un ejemplo de la prueba desde el prototipo.

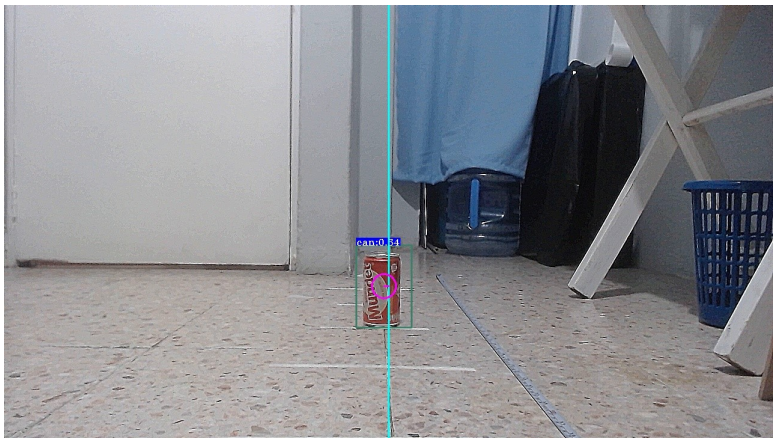


Figura 79: Ejemplo de lata a 80 cm de distancia del prototipo.[9]

En la Tabla 17 se muestra los resultados obtenidos para un rango de distancia entre 20 a 140 cm para la clase “bottle”, en la Tabla18 se muestran los resultados obtenidos para la clase “can”.

Distancia(cm)	**Detección**
20	Si
40	Si
60	Si
80	Si
100	Si
120	Si
140	No

Tabla 17: Detecciones clase **bottle** rango de 20 - 140 cm

Con lo obtenido anteriormente se puede observar que el prototipo puede detectar objetos en un rango de 20 a 100 cm en una línea de visión directa. Posteriormente se realizó la misma prueba, pero para los marcadores utilizados para determinar la ubicación de los contenedores, estos se encuentran colocados a una altura de 30 cm del suelo, en la Figura 80 se puede observar un ejemplo.

[9]Elaboración propia.

Distancia(cm)	Detección
20	Si
40	Si
60	Si
80	Si
100	Si
120	No
140	No

Tabla 18: Detecciones clase **can** rango de 20 - 140 cm

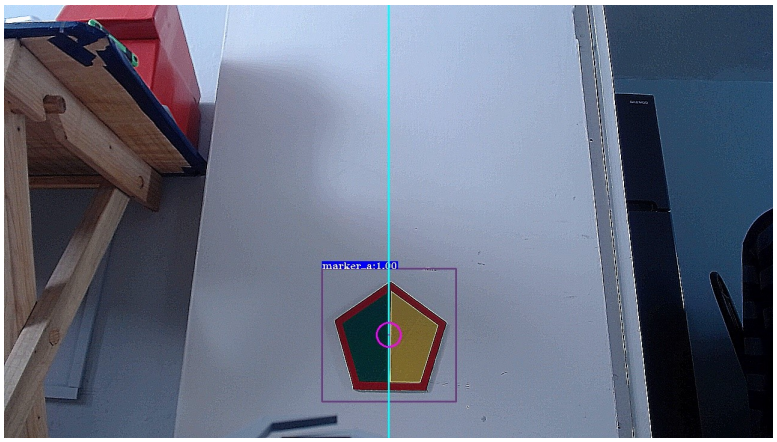


Figura 80: Ejemplo de marcador a 100 cm de distancia del prototipo.[10]

En la Tabla 19 se puede observar los resultados obtenidos para una línea de visión directa de los marcadores.

Además de esta prueba, para los marcadores se realizó una prueba más teniendo un ángulo de línea de visión de 45° ya que a diferencia de las latas y las botellas el marcador cambia su forma y sus proporciones dependiendo desde donde se vea, en la Figura 81 se muestra un ejemplo de esto.

[10]Elaboración propia.

Distancia(cm)	**Detección**
20	Si
40	Si
60	Si
80	Si
100	Si
120	Si
140	No

Tabla 19: Detecciones de marcadores en rango de 20 - 140 cm

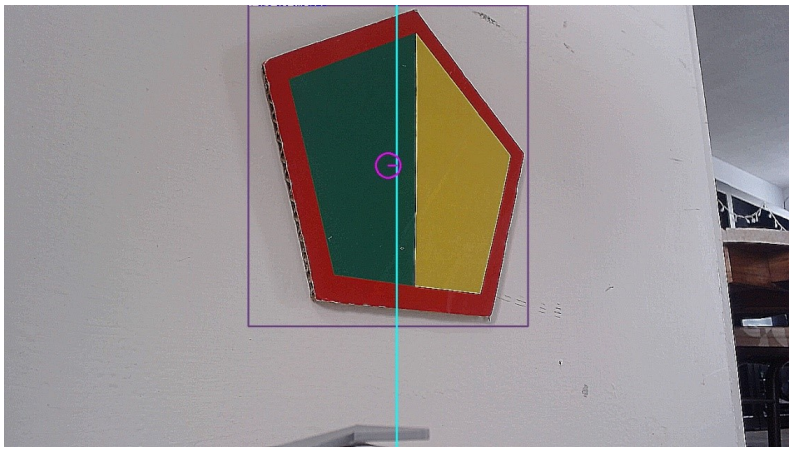


Figura 81: Ejemplo de marcador con un ángulo de línea de visión de 45°.[11]

En la Tabla 20 se pueden observar los resultados obtenidos para la detección de los marcadores en un ángulo de línea de visión de 45°. Como se puede observar el detector de objetos no tiene problema en manejar estas variaciones en la forma del objeto ya que el dataset incluyo esta clase de casos así el prototipo no necesariamente necesita estar situado frente al marcador para detectarlo y puede realizar una detección de los marcadores en un rango de 20 a 120 cm.

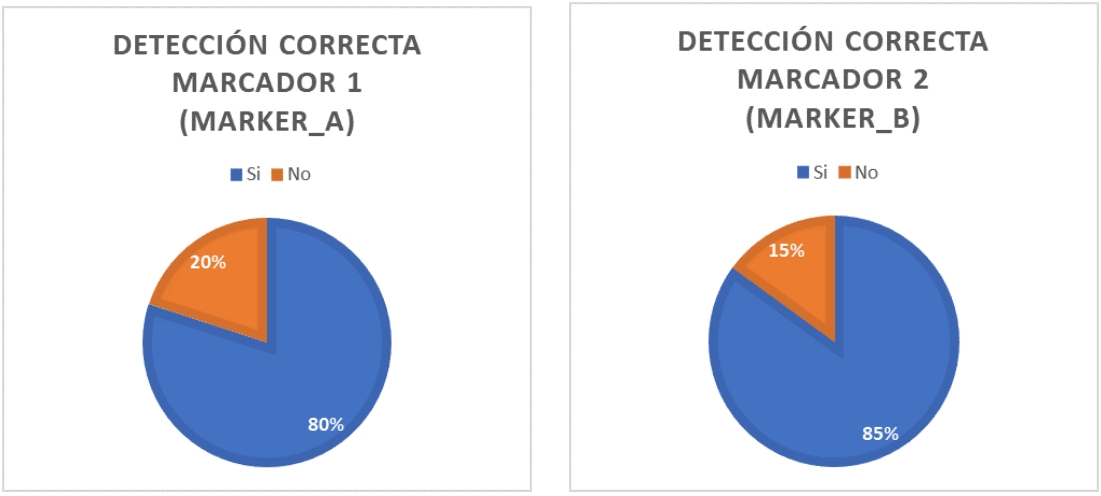
Los resultados presentados anteriormente para los marcadores solo corresponden para saber si hay marcador o no, pero además de eso es necesario ejecutar un proceso extra el cual nos permite identificar los colores dentro del marcador y poder distinguir uno de otro. Se hacen uso de dos marcadores, marker_a y marker_b, cada uno indica en donde se encuentra un depósito de basura para una clase especifica. Para evaluar esto se

[11] Elaboración propia.

Distancia(cm)	Detección
20	Si
40	Si
60	Si
80	Si
100	Si
120	Si
140	No

Tabla 20: Detecciones de marcadores en rango de 20 - 140 cm

realizaron otras pruebas para determinar con 20 intentos, cuantas veces era distinguido correctamente. Este proceso de extracción, detección de color y determinar que marcador es toma aproximadamente 150 ms. En la Figura 82a se puede observar el porcentaje de aciertos para el marcador 1 denominado marker_a mientras que en la Figura 82b se puede observar el porcentaje de aciertos y fallos del marcador 2 denominado marker_b.



(a) Detecciones marcador 1

(b) Detecciones marcador 2

Figura 82: Detecciones para marcadores[12]

Con lo anterior se puede notar que el proceso propuesto para extraer y determinar

[12]Elaboración propia.

colores funciona en casi el 80 % de los casos al diferenciar correctamente los marcadores.

Además de esto se realizó una serie de pruebas en un espacio controlado en donde se establecieron los siguientes objetivos: el robot inicia en una habitación en donde primero tiene que buscar una botella, ir hacia ella, recolectarla. Posteriormente debe de buscar el marcador que se encuentra en la habitación para así poder ir al contenedor de botellas y depositar la botella que lleva. En la Figura 83 se puede observar el porcentaje de aciertos y fracasos al intentar cumplir estos objetivos en 20 distintos intentos.

Figura 83: Objetivos completados primer prueba.[13]

Con lo anterior se puede observar que el robot logró los objetivos establecidos el 75 % de las veces que lo intentó, dentro del otro 25 % sucedieron diversos casos de entre los cuales fueron: el robot detectó el objeto y se acercó a él, pero no pudo tomarlo, el robot tomó el objeto correctamente y detectó el marcador, pero no pudo acercarse a la distancia necesaria para depositar el objeto y lo soltó antes debido a mediciones incorrectas de los sensores utilizados para determinar la distancia.

Por último se estableció una serie de pruebas en donde se establecieron los objetivos anteriores, pero además de estos una vez dejada la botella en su contenedor el robot debe salir del cuarto e ir a la siguiente habitación en donde debe de buscar una lata, ir a ella, recolectarla. Posteriormente buscar el marcador en esa habitación para así ir a él y depositar la lata en el contenedor de latas. En la Figura 84 se puede observar el porcentaje de aciertos y fracasos al intentar cumplir estos objetivos en 20 distintos intentos.

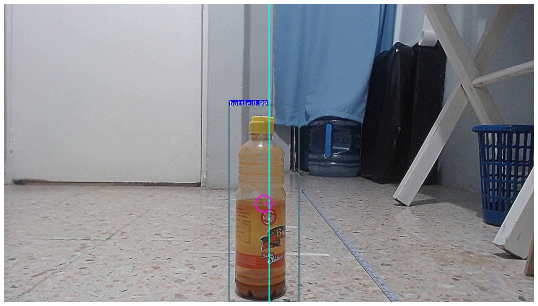
[13]Elaboración propia.

Figura 84: Objetivos completados segunda prueba.[14]

Con la gráfica anterior se puede observar que el robot logró completar los objetivos el 70 % de las veces que lo intentó, dentro del 30 % se encuentran éxitos parciales ya que el robot pudo completar la primer parte que consistió en la recolección de una botella pero este falló en la segunda etapa al tomar la lata y llevarla a su contenedor, sea por que no pudo localizar el marcador o por que no logró acercarse lo suficiente al contenedor debido a medidas inconsistentes de los sensores utilizados para medir la distancia.

Como un añadido se decidió probar la capacidad de generalización del detector de objetos para identificar objetos que se encontrarán dentro de la clase “can” y “bottle” pero que no hayan sido añadidos al dataset. Gracias a esto se pudo observar que es capaz de detectar latas con diseños diferentes a los incluidos en el dataset, también le es posible detectar botellas de mayor capacidad, o con diseños y formas similares, como se puede observar en las subfiguras de la Figura 85, en estos ejemplos se puede observar que el modelo entrenado tiene la capacidad de identificar de manera correcta objetos no incluidos en el dataset.

[14]Elaboración propia.



(a) Botella de salsa picante

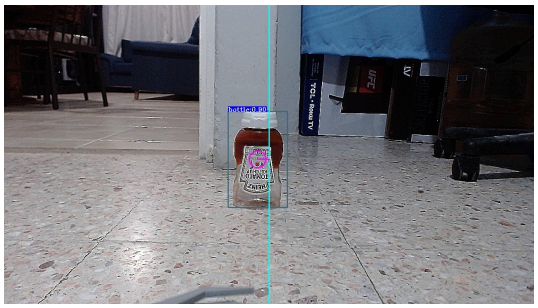
(b) Botella de agua de 3L

(c) Lata de bebida energetica

(d) Extintor en lata

(e) Botella de Aceite

(f) Botella de gel antibacterial



(g) Botella de ketchup

(h) Botella de medicamento

Figura 85: Detecciones generalizadas.[15]

Con lo anterior se puede observar la buena capacidad de generalización de la red entrenada para el sistema de visión artificial además de que también le fue beneficioso la aplicación de la técnica de Data Augmentation para introducir mas variaciones de los objetos ya incluidos en el dataset y así reforzar la extracción de características.

Por último, en el siguiente enlace se puede consultar un vídeo de una de las pruebas realizadas por el prototipo final: https://youtu.be/yEdNIJ8DXwU

[15]Elaboración propia.

Capítulo 6

Conclusiones

En este trabajo de tesis se propuso diseñar e implementar un prototipo de robot el cual hace uso de algoritmos de control además de aplicar redes neuronales artificiales como un sistema de visión artificial con el fin de identificar, clasificar, recolectar basura y llevarla a un área predefinida de manera autónoma.

El prototipo realizado es diferente a otros existentes ya que este hace uso de inteligencia artificial combinado con elementos electromecánicos para lograr su objetivo mediante el uso de redes neuronales artificiales. El prototipo mostró obtener hasta un 75 % de éxito al poder identificar la basura, moverse hacia ella, recolectarla y manejarse de manera autónoma mediante un sistema de marcadores para ubicarse dentro de un espacio controlado para así posteriormente depositar la basura en un contenedor. Para lograr esto se entrenó e implementó un sistema de visión artificial basado en redes neuronales convolucionales los cuales dotan al prototipo de un sistema inteligente el cual es capaz de aprender a través de experiencia haciendo uso de un conjunto de imágenes, este sistema a su vez le otorga la capacidad de generalizar para poder identificar objetos similares.

En comparación con otros sistemas de visión artificial que requieren equipos de gran costo para poder ser ejecutados, el sistema de visión propuesto fue implementado en una tarjeta de desarrollo de bajo costo la cual le permite disminuir sus costos y dimensiones.

Aunado a esto el sistema de marcadores demostró funcionar con hasta un 85 % de éxito al detectarlos y diferenciar unos de otros permitiendo así al prototipo identificar los lugares relevantes como lo son los contenedores y dentro del espacio controlado donde opera.

Como trabajo a futuro se proponen varias líneas de mejora para el prototipo propuesto, la primera consiste en incrementar la base de datos de imágenes para mejorar la detección de objetos además de la posibilidad de agregar objetos de otras clases además de la adición de una cámara con un enfoque más rápido para acelerar la ejecución del prototipo. Posterior a esto se podría mejorar la parte electrónica del prototipo implementando mejores sensores para las mediciones de distancia y como adición un sensor GPS si se deseara que este funcionara en espacios más grandes además del uso de

motores que permitan un control más fino para así tener un mejor posicionamiento de este para la toma del objeto. Y por último la mejora del manipulador el cual requiera de un control menos fino con el fin de poder aproximarse más rápidamente a los objetos para tomarlos además de la adición de un contenedor para poder llevar varios objetos a vez.

Bibliografía

[1] Kaza, S., Yao, L., Bhada-Tata, P., & Van Woerden, F. (2018). What a waste 2.0: a global snapshot of solid waste management to 2050. 6

[2] Ziraba, A. K., Haregu, T. N., & Mberu, B. (2016). A review and framework for understanding the potential impact of poor solid waste management on health in developing countries. Archives of Public Health, 74(1), 55. 8

[3] Tello Espinoza, P., Martínez Arce, E., Daza, D., Soulier Faure, M., & Terraza, H. (2010). Regional evaluation on urban solid waste management in Latin America and the Caribbean: 2010 report. Pan American Health Organization: Washington, DC, USA. 9

[4] Protrash. (2017, August 27). Mexico's Massive 24 Billion Dollar Recycling Opportunity. Disponible en: https://www.huffpost.com/entry/mexicos-massive-24-billio_b_11730898 9

[5] Viola, P., & Jones, M. (2001, December). Rapid object detection using a boosted cascade of simple features. In Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition. CVPR 2001 (Vol. 1, pp. I-I). IEEE. 12

[6] Dalal, N., & Triggs, B. (2005, June). Histograms of oriented gradients for human detection. In 2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05) (Vol. 1, pp. 886-893). IEEE. 12

[7] Deng, J., Dong, W., Socher, R., Li, L. J., Li, K., & Fei-Fei, L. (2009, June). Imagenet: A large-scale hierarchical image database. In 2009 IEEE conference on computer vision and pattern recognition (pp. 248-255). Ieee. 12

[8] Zhao, Z. Q., Zheng, P., Xu, S. T., & Wu, X. (2019). Object detection with deep learning: A review. IEEE transactions on neural networks and learning systems, 30(11), 3212-3232. 12

[9] Hadush, S., Girmay, Y., Sinamo, A., & Hagos, G. (2020). Breast Cancer Detection Using Convolutional Neural Networks. arXiv preprint arXiv:2003.07911. 13

[10] Nasser, I. M., & Abu-Naser, S. S. (2019). Lung Cancer Detection Using Artificial Neural Network. International Journal of Engineering and Information Systems (IJEAIS), 3(3), 17-23. 13

[11] Li, H., Wang, P., You, M., & Shen, C. (2018). Reading car license plates using deep neural networks. Image and Vision Computing, 72, 14-23. 13

[12] Tiron, G. Z., & Poboroniuc, M. S. (2019, October). Neural Network Based Traffic Sign Recognition for Autonomous Driving. In 2019 International Conference on Electromechanical and Energy Systems (SIELMEN) (pp. 1-5). IEEE. 13

[13] Mittal, G., Yagnik, K. B., Garg, M., & Krishnan, N. C. (2016, September). Spotgarbage: smartphone app to detect garbage using deep learning. In Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing (pp. 940-945). 13

[14] Sivasankar, G., Durgalakshmi, B., & Seyatha, K. (2017). Autonomous Trash Collecting Robot. Published by International Journal of Engineering Research & Technology (IJERT). 13

[15] Zhihong, C., Hebin, Z., Yanbo, W., Binyan, L., & Yu, L. (2017, July). A vision-based robotic grasping system using deep learning for garbage sorting. In 2017 36th Chinese Control Conference (CCC) (pp. 11223-11226). IEEE. 13

[16] Khandare, S., Badak, S., Sawant, Y., & Solkar, S. (2018). Object Detection Based Garbage Collection Robot (E-Swachh). SYSTEM, 5(03). 13

[17] Lindgren, B., & Kuosmanen, G. (2018). An Autonomous Robot for Collecting Waste Bins in an Office Environment. 13

[18] Bai, J., Lian, S., Liu, Z., Wang, K., & Liu, D. (2018). Deep learning based robot for automatically picking up garbage on the grass. IEEE Transactions on Consumer Electronics, 64(3), 382-389. 14

[19] Kokoulin, A. N., Tur, A. I., & Yuzhakov, A. A. (2018). Convolutional neural networks application in plastic waste recognition and sorting. In 2018 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus) (pp. 1094-1098). IEEE. 14

[20] Kong, S., Tian, M., Qiu, C., Wu, Z., & Yu, J. (2020). IWSCR: An Intelligent Water Surface Cleaner Robot for Collecting Floating Garbage. IEEE Transactions on Systems, Man, and Cybernetics: Systems. 14

[21] Cruz, P. P. (2011). Inteligencia artificial con aplicaciones a la ingeniería. Alfaomega. 19, 23

[22] Berzal, F. (2018). Redes neuronales & deep learning. Independently published. 24, 25, 26, 28

[23] Khan, S., Rahmani, H., Shah, S. A. A., & Bennamoun, M. (2018). A guide to convolutional neural networks for computer vision. Synthesis Lectures on Computer Vision, 8(1), 1-207. 27, 29

[24] Ramsundar, B. and Zadeh, R., n.d. Tensorflow For Deep Learning.

[25] ImageNet. (2020). Disponible en: http://www.image-net.org/ 47

[26] COCO - Common Objects in Context. (2020). Disponible en: https://cocodataset.org/ 47

[27] Hoeser, T., & Kuenzer, C. (2020). Object Detection and Image Segmentation with Deep Learning on Earth Observation Data: A Review-Part I: Evolution and Recent Trends. Remote Sensing, 12(10), 1667.

[28] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015). Going deeper with convolutions. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 1-9). 47

[29] Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. In Advances in neural information processing systems (pp. 91-99). 47

[30] Szegedy, C., Vanhoucke, V., Ioffe, S., Shlens, J., & Wojna, Z. (2016). Rethinking the inception architecture for computer vision. In Proceedings of the IEEE conference on computer vision and pattern recognition (pp. 2818-2826). 52

[31] Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster r-cnn: Towards real-time object detection with region proposal networks. In Advances in neural information processing systems (pp. 91-99). 56

[32] Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. Y., & Berg, A. C. (2016, October). Ssd: Single shot multibox detector. In European conference on computer vision (pp. 21-37). Springer, Cham. 61

[33] The PASCAL Visual Object Classes Challenge 2007 (VOC2007). (2020). Disponible en: http://host.robots.ox.ac.uk/pascal/VOC/voc2007/ 62

[34] NVIDIA TensorRT. (2020). NVIDIA Developer. Disponible en: https://developer.nvidia.com/tensorrt 79

[35] Jetson Nano. (2020). Disponible en: https://developer.nvidia.com/embedded/jetson-nano 41

[36] Database, G., & Specs, J. (2020). NVIDIA Jetson Nano GPU Specs. Disponible en: https://www.techpowerup.com/gpu-specs/jetson-nano-gpu.c3643 40

[37] OpenCV. Disponible en: https://opencv.org/

[38] TensorFlow. (2020). Disponible en: https://www.tensorflow.org/ 42

[39] tensorflow/models. (2020). Disponible en: https://github.com/tensorflow/models/blob/master/research/ol
43, 88

[40] pyserial/pyserial. (2020). Disponible en: https://github.com/pyserial/pyserial 43

[41] Laboratorio Nacional de Supercómputo del Sureste de México. (2020). Disponible en http://lns.org.mx/ 88

[42] A. Pal and D. K. Dutta Majumder, "Approaches to supervised learning for pattern recognition," Jan-1994. Disponible en: http://www.dli.gov.in/rawdataupload/upload/insa/insa_2/20005a1c_1.pdf 16, 17