Universidad de Concepción
Departamento Ingeniería Informática y Ciencias de la Computación

APLICACIÓN DE MACHINE LEARNING PARA EL PROBLEMA DE ROBOTIC GRASPING

Angelo Fernando Montaño Pedreros
Departamento de Ingeniería Informática
Universidad de Concepción

Julio Erasmo Godoy del Campo

27 de Marzo de 2019

Abstract

El área de robótica, debido a su creciente utilización en un gran número de áreas, desde los hogares hasta grandes industrias, presenta muchos desafíos. Entre estos desafíos, la capacidad de manipular objetos específicamente el agarre robótico o robotic grasping siempre ha sido una tarea compleja para los robots por la gran cantidad de factores que influyen en un ambiente no estructurado y, además, por su falta de capacidad para comprender los datos de percepción recibidos. Técnicas para poder realizar tareas de agarre robótico incluyen métodos de visión por computadora y de inteligencia artificial.

En esta memoria de título se diseñó un sistema que permite al robot Baxter identificar objetos dentro de un área de trabajo, recibir la orden de agarre de un objeto en específico o de manera aleatoria y ejecutar la acción eligiendo un punto de agarre eficiente. Los resultados mostraron un buen desempeño de Baxter al realizar los distintos agarres a pesar de la falta de precisión en algunas acciones.

Índice

1. Introducción

En las últimas décadas, la mayoría de los robots se han vuelto más precisos trabajando en ambientes totalmente controlados, como puede ser una línea de ensamblaje de autos, ya que sabe en todo momento dónde están los objetos y donde debe colocarlos. Por lo que no debe haber ningún análisis ni procesamiento de por medio, solo ejecutar un movimiento predefinido una y otra vez. Todo esto cambia cuando el ambiente es dinámico y no estructurado, donde el robot no conoce la posición y distribución de elementos, siendo necesario realizar un análisis de la situación actual del mundo en base a distintos datos percibidos y decidir de manera autónoma sus próximos movimientos [15]. Esto último resulta bastante complejo y es uno de los grandes desafíos de la robótica moderna [40].

Uno de los campos más estudiados en el área de inteligencia artificial es machine learning o aprendizaje automático, el cual es una rama de la inteligencia artificial basada en la idea de que los sistemas pueden aprender de los datos, de las interacciones con el ambiente, identificar patrones y tomar decisiones con la mínima intervención humana. Este nace del reconocimiento de patrones y la teoría de que las computadoras pueden aprender sin ser programadas para realizar tareas específicas [26].

Las técnicas de machine learning encuentran cada día más aplicaciones, siendo una de las más famosas o conocidas los vehículos autónomos, capaces de conducirse a sí mismos sin necesidad de intervención humana, aunque como todo sistema, estos no están libres de fallos y errores, por lo que aún no son del todo confiables. También están los asistentes virtuales, como Alexa y Siri, los cuales utilizan técnicas de procesamiento del lenguaje natural para “entender” el lenguaje humano y responder de manera acorde. También se aplican técnicas de machine learning a las búsquedas online, publicidad, marketing personalizado, detección de fraudes, entre muchas otras aplicaciones [29].

Grandes empresas usan inteligencia artificial y robótica en los distintos procesos productivos con el fin de mejorar el rendimiento. Empresas como Amazon o Alibaba, por ejemplo, usan miles de robots en sus procesos de despacho y bodegaje donde se puede apreciar la eficiencia que otorgan estos agentes autónomos, creando complejos sistemas multiagente, generando un ambiente totalmente automatizado [6].

Debido al interés en robótica y automatización de la empresa Amazon surge el “Amazon Picking Challenge”, llegando a competir equipos de universidades y empresas de gran prestigio en el campo de la robótica [3]. Esta competencia consiste en desafíos que ponen a prueba la capacidad de los sistemas robóticos para cumplir un pedido ficticio al recoger de manera autónoma los artículos solicitados. El agarre robótico o robotic grasping (términos que se usarán indistintamente en este documento) es una tarea bastante compleja de ejecutar por un robot, ya

que este debe ser capaz de reconocer el objeto y decidir un punto y forma de agarre eficaz. Por lo que este tipo de desafíos ayudan a encontrar nuevas propuestas y soluciones, con el fin de aumentar el conocimiento común y construir a partir de esto.

Actualmente la facultad de Ingeniería de la Universidad de Concepción tiene un robot de la compañía Rethink Robotics llamado Baxter [13], versión de investigación. En esta memoria de título se busca implementar un programa que permita a Baxter identificar una serie de objetos posicionados en un área de trabajo, determinar su posición en este, recibir la petición de agarre de un elemento, computar la forma de agarre eficiente y ejecutar la acción de agarrar y dejar dicho objeto en un depósito preestablecido. En la opinión de quien escribe, una gran ventaja que tienen los proyectos relacionados con el área de robótica es la capacidad de poder ver los resultados de manera tangibles en el mundo real, donde el robot actúa de vínculo o mediador entre lo virtual y la realidad.

1.1. Objetivo General

El objetivo de esta memoria es diseñar e implementar un sistema que utilice técnicas de machine learning y visión de computadoras, utilizando redes neuronales y detección de bordes y contornos, que permitan al robot Baxter identificar y manipular objetos físicos mediante el agarre efectivo, de manera autónoma.

1.2. Objetivos Específicos

- Comprender el estado del arte en métodos de robotic grasping
- Implementar un método de detección de contornos a partir de imágenes de objetos.
- Diseñar y entrenar una red neuronal para la tarea de clasificación de objetos.
- Diseñar e implementar un método de detección de puntos de agarre eficaz.
- Lograr que el robot Baxter sea capaz de identificar los objetos en el área de trabajo y en base a la decisión de un operador, que recoja un objeto solicitado o de manera aleatoria, para luego dejarlo en un depósito.

2. Discusión Bibliográfica

En esta sección se presentan trabajos con objetivos similares que utilizan al robot Baxter o tecnologías similares para el proceso de agarre inteligente robótico, realizados por distintos equipos de trabajo.

2.1. Reconocimiento de imágenes mediante Machine Learning

Se define reconocimiento de imágenes, en contexto de visión computacional, como la capacidad de un sistema de identificar objetos, lugares, personas, escritura y acciones en imágenes [35]. El reconocimiento de imágenes es utilizado para realizar una gran cantidad de tareas visuales en máquinas, como etiquetar contenido en una escena, realizar búsquedas en el contenido de imágenes, guiar robots autónomos, automóviles sin conductor, sistemas de prevención de accidentes, entre otros.

Para llevar a cabo muchas de las tareas descritas, se utilizan *redes neuronales artificiales*, entrenadas en una gran cantidad de imágenes correctamente etiquetadas. Para ello, es necesario encontrar grandes sets de datos con millones de imágenes con muchas categorías. Una de las más grandes y conocidas es ImageNet [17], el cual es una gran base de datos visual diseñada para su uso en la investigación de software de reconocimiento visual, que posee más de 14 millones de imágenes y 1000 categorías/clases de objetos.

Desde 2010, ImageNet anualmente realiza competencias de visión de computadoras llamado "ImageNet Large Scale Visual Recognition Challenge (ILSVRC)" [18]. En este desafío, equipos de investigadores evalúan sus algoritmos de detección de objetos y clasificación de imágenes a gran escala, donde se compite por obtener la mayor precisión en varias tareas de reconocimiento visual.

Desde sus inicios en 2010, se han visto grandes avances en procesamiento de imágenes, donde fueron apareciendo distintas configuraciones, modelos y técnicas para el problema de clasificación de objetos. Pero en 2012 se logró un gran avance en esta materia con el desarrollo de modelos de deep learning para esta tarea. Ese año se publicó el paper ImageNet Classification with Deep Convolutional Networks [22], el cual fue uno de los artículos más influyentes en el área de clasificación de imágenes, ya que disminuye en gran medida la tasa de error alcanzada anteriormente, además de aumentar la precisión de las predicciones. El modelo de red utilizado tenía el nombre de "AlexNet", la cual es considerada como punto de partida para las competiciones de años siguientes. A partir de esta aparecieron nuevos modelos y arquitecturas de red como VGG [37], ResNet [14], GoogLeNet [38], DenseNet [16], entre otras. Actualmente la

arquitectura de la red neuronal propuesta en el paper [22] sigue siendo la base del estado del arte de muchas redes neuronales para procesar imágenes.

2.2. Robótica Colaborativa

Los robots modernos han estado en constante desarrollo desde hace más de 50 años, aunque inicialmente solo eran máquinas que ejecutaban movimientos y acciones preestablecidas, como el robot *Unimate*, el cual es el primer robot industrial programable, fabricado por la empresa Unimation, primera compañía fabricante de robótica. Instalado en una cadena de montaje de la compañía General Motors en el año 1961, *Unimate* consiste en una caja computarizada con un brazo robot, con la tarea de recoger piezas a altas temperaturas y luego soldarlas al chasis de los vehículos, tarea bastante peligrosa para un trabajador. Esta invención llegó a mejorar el proceso de fabricación, además de reducir las tareas peligrosas ejecutadas por humanos.

Tiempo después, los robots pasaron a percibir y comprender el entorno que los rodeaba y en base a eso tomar decisiones y ejecutar movimientos. Se convirtieron en agentes inteligentes. Uno de los primeros robots inteligentes creados fue *Shakey*, desarrollado desde 1967 a 1972 por Stanford Research Institute, como un robot de propósito general que usaba inteligencia artificial para percibir obstáculos en el entorno, planificar una ruta óptima y desplazarse de manera autónoma. Considerado el abuelo de los autos autónomos, *Shakey* marcó un antes y un después en la robótica moderna, debido a la capacidad de "pensar" por su propia cuenta, lo cual hasta ese entonces era ciencia ficción. De este robot surgieron diversos avances en múltiples áreas, robótica, inteligencia artificial, navegación, reconocimiento de imágenes, entre otras [36].

A medida que transcurre el tiempo, el poder de cómputo fue incrementando por parte de las computadoras, traduciéndose en sistemas más complejos y poderosos permitiendo a robots hacer tareas cada vez más complicadas.

Actualmente, los robots están presentes en gran cantidad de ambientes, sobre todo en el sector industrial, donde se tienen grandes ambientes automatizados, trabajando en conjunto con los operadores, mejorando la producción y rendimiento en general, creando "fábricas inteligentes" [6]. De esto surge lo que se conoce como *robótica colaborativa*, la cual, como su nombre lo indica, busca que máquinas, sistemas IT y equipos humanos colaboren y trabajen conjuntamente en red. Los robots colaborativos son sistemas capaces de trabajar junto a operarios humanos, sin tener que utilizar medidas de seguridad tradicionales como jaulas o barreras, incorporando sistemas avanzados de control de fuerzas para evitar colisiones y golpes a los operarios. La robótica colaborativa permite que la tecnología y la automatización robótica, sea accesible para las empresas medianas y pequeñas [2].

Baxter es un robot creado originalmente para tareas de manufactura, como un robot colaborativo para automatización industrial (Figura 1). Por lo comentado anteriormente, Baxter puede desenvolverse sin ningún problema y de manera segura en nuestro espacio de trabajo.

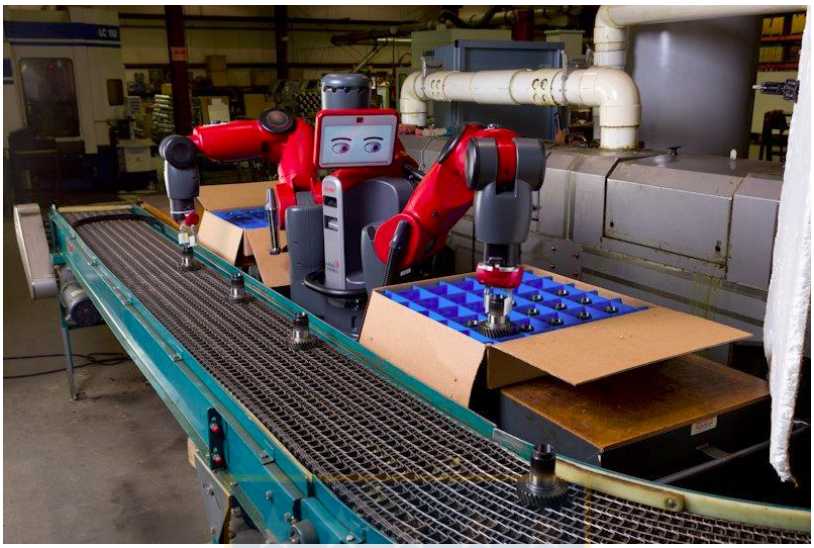

Figura 1: Baxter en empresa de manufactura, obtenida de [2]

2.3. Robotic Grasping y Baxter

La tarea de grasping, en el contexto de la robótica, consiste en hacer que un agente autónomo sea capaz de agarrar objetos de distinto tipo de manera efectiva, detectando posibles puntos de agarre, imitando la manera en que los humanos ejecutamos la acción de agarre. Esto puede resultar muy útil para tareas de organización física de objetos, por ejemplo, agrupando por categorías o tipo de objeto.

Como se menciona anteriormente, Baxter está diseñado para realizar tareas donde utiliza sus brazos para manipular distintos tipos de objetos en procesos de manufactura. Además, muchos equipos de robótica adquieren estos robots para realizar pruebas e investigaciones respecto a este tema.

El proyecto realizado en [32] resulta muy útil para la realización de esta memoria, ya que en éste se plantea una manera de enseñar a un robot Baxter como realizar el agarre de distintos objetos. Primero se plantea la dificultad de obtener un conjunto de datos etiquetados para realizar el entrenamiento en esta tarea. Para solucionar esto se realiza un método para recopilar datos de manera automática utilizando a Baxter durante cerca de 700 horas de trabajo, realizando una sucesión de movimientos de agarre aleatorios donde se chequea si el agarre fue exitoso o no, gracias a los sensores de fuerza, generando una lista de ejemplos positivos y negativos. De esta manera se obtiene un gran dataset para entrenar la red. Una vez entrenada la red con el conjunto

de datos generados realizando agarres aleatorios, se vuelve a recopilar datos, pero ahora el robot usa este modelo como un control previo, mejorando la recopilación de datos. Utiliza la experiencia previa para corregir modalidades de agarre incorrectas y a la vez refuerza las correctas. Este método de entrenamiento se conoce como "aprendizaje por etapas". Se logran recopilar cerca de 50 mil datos de agarres.
En otras investigaciones como en [25] se utilizan "granjas" de robots (Figura 2), donde varios robots trabajan en conjunto, recopilando una gran cantidad de datos. En [25] se utilizan dos etapas de recopilación, donde en la primera se usan 14 robots trabajando en conjunto, recopilando cerca de 800 mil intentos de agarre. Luego, en la segunda etapa, se utilizaron 8 robots Kuka obteniendo más de 900 mil agarres, generando un dataset enorme comparado con [32].

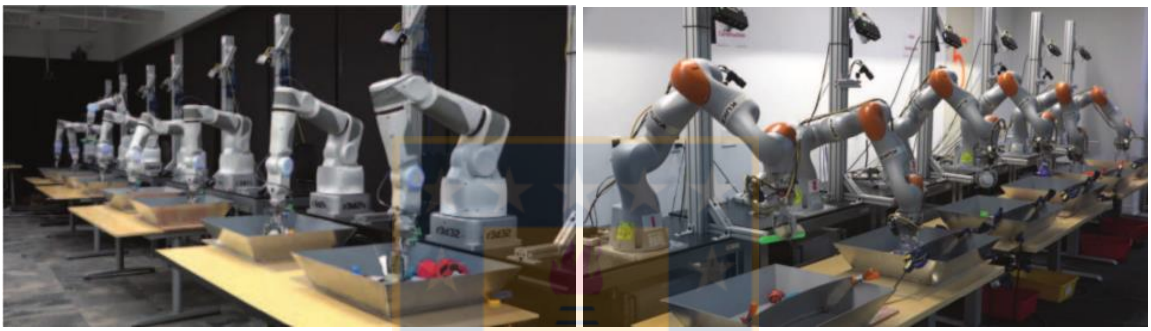

Figura 2: Robots trabajando para recopilar información. primera etapa(izquierda), segunda etapa (derecha) de [25]

La red entrenada en [32] fue utilizada en esta memoria para realizar la detección de puntos de agarre, explicada con más detalle en la sección 3.2.

En la investigación [31] se continúa el trabajo en [32] postulando un proceso mejorado para la detección de puntos de agarre. Inicialmente, no es posible diferenciar un agarre fuerte de uno débil. En base a esto, se propone un framework adversario, donde hay un protagonista y un adversario que compiten entre ellos. El objetivo del protagonista es encontrar un agarre robusto y el del adversario es desestabilizar el agarre del protagonista. Se plantean dos mecanismos adversarios, el primero que ejecuta la acción de sacudir y el segundo trata de arrebatar el objeto. De esta manera, se ejecutan los agarres, se comprueba si fue exitosos para luego ejecutar el agente adversario, el cual sacude o arrebata el objeto, marcando cuando logra desestabilizar el agarre, agregando esta información al entrenamiento. Así el agente que se encarga de agarrar en las siguientes iteraciones busca puntos de agarres más estables y el agente adversario mejora sus acciones de perjudicar al protagonista. Al igual que en el artículo anterior, se utiliza un robot Baxter, donde el agente que sacude es su propio brazo y el agente que arrebata el objeto es su otro brazo. El artículo, afirma que se obtiene una mejora considerable utilizando este framework adversario, donde se alcanza un 82% de efectividad agarrando objetos nuevos para el modelo, contra un 62% de efectividad sin usar adversario.

En [24], al igual que en [32] y [31], se utiliza un robot Baxter para ejecutar el grasping. En este se plantea el problema de detectar posibles agarres en imágenes RGB-D. Se utiliza aprendizaje profundo para resolver esto, diseñando un algoritmo de detección de “dos pasos” con dos redes neuronales, donde la primera red, que es pequeña y liviana, se encarga de detectar los posibles agarres, realizando un filtrado de las opciones. Esta red funciona como filtro inicial, el cual no es muy estricto, pero permite podar una gran cantidad de opciones, además de que funciona rápido. Luego, la siguiente red, la cual es más grande, con más características, se encarga de seleccionar el mejor rectángulo de agarre de los candidatos detectados por la primera red. Para entrenar estas redes, se utilizó el dataset de Cornell [30].

Las principales diferencias que tiene este proyecto respecto a otros trabajos e investigaciones son:

- En este proyecto solo se utilizan las cámaras que posee Baxter, las investigaciones mencionadas anteriormente utilizan cámaras que cuentan con sensores de profundidad, utilizando un sensor Microsoft ® Kinect [21] por ejemplo, lo cual permite detectar distintas condiciones en el conjunto de objetos que permiten ejecutar agarres más precisos. Además de que estas cámaras tienen mejor resolución y calidad de imagen. Cómo es posible detectar profundidad con estas cámaras, esto permite facilitar el proceso para detectar objetos montados, por lo que en algunas investigaciones, el robot es capaz de detectar estas acumulaciones de objetos y comienza a recoger los que están sobre el montón de objetos. Además, en este trabajo solamente se utilizó una perspectiva aérea de la vista del espacio de trabajo, ya que en otros trabajos como en [10], [11] y [12], se utilizan un conjunto de perspectivas para mejorar la precisión del agarre. Resulta bastante interesante lo logrado en [11], donde se utiliza un “structure sensor”, para escanear el ambiente de trabajo y obtener una representación 3D de este, con lo cual es posible detectar los objetos montados unos sobre otros.

- Otra diferencia de este proyecto es que no es capaz de detectar los objetos y puntos de agarre en tiempo real. El sistema saca una imagen y con esta se realizan los análisis de clasificación y detección de agarre, por lo que, si se saca o se mueve el objeto, la acción del robot se llevará a cabo hasta que detecte que no hay nada en el gripper (garra o mano robot). A diferencia de [27], [28] y [34], que son capaces de hacer seguimiento del objeto en tiempo real.

- Otra característica de este proyecto es que todos los agarres se ejecutan desde arriba hacia abajo, posicionándose sobre el objeto para luego bajar en línea recta sobre el punto de agarre y ángulo correspondiente, debido a la perspectiva aérea que tiene este sistema. Al igual que los trabajos desarrollados en [25], [31] y [32]. En cambio, los trabajos que usan varias perspectivas para percibir el ambiente, realizan distintos tipos de movimientos,

como en [39] que realiza algunos movimientos de manera horizontal, posicionándose a la altura del objeto deseado, enfocando con la cámara el costado del objeto y luego ejecuta el movimiento.

Una competencia bastante importante que tiene que ver con robótica y agarre es el "Amazon Picking Challenge" [3], la cual surge el año 2015 y tiene como objetivo aumentar el conocimiento común de robótica. Amazon utiliza robots Kiva [6] para mover sus estantes a través de su almacén (Figura 3), con lo cual reducen el tiempo dedicado a la búsqueda y desplazamiento de los trabajadores. Ahora los empleados se posicionan en lugares estratégicos y los estantes se acercan a ellos, donde el trabajador busca y recoge el objeto necesario.

Figura 3: Robot Kiva utilizado en los almacenes de Amazon, obtenida de [1]

A la empresa le gustaría que esta intervención del trabajador fuera sustituida por un robot, para reducir el tiempo que se tarda en buscar y recoger. Pero el problema es que hay una cantidad enorme de tipos de objetos a recoger en las bodegas, lo cual aumenta demasiado la dificultad de automatizar esta tarea. A partir de esta problemática nace el desafío con el fin de recibir equipos de distintos lugares y que presenten distintas soluciones a este problema. [3]

En el artículo [9] se pueden encontrar las impresiones y recomendaciones de los ganadores del primer Picking Challenge de Amazon. En este nos comentan los aspectos a tener en cuenta en el desarrollo de un sistema robot (cuatro aspectos clave).

Para poder llevar a cabo el desarrollo de este proyecto, se tomaron en cuenta dos aspectos claves:

1. Modularidad v/s integración
2. Cálculo v/s Realización

Del primer aspecto, podemos rescatar la idea de que es bueno realizar el sistema de forma modular, pero sin despreocupar la tarea de integración, ya que el comportamiento de todo el sistema determina el éxito, no el funcionamiento de los módulos por separado. En nuestro caso, se mantienen separados los módulos de procesamiento de imagen, detección de agarre y ejecución de movimiento, pero están estrechamente integrados.

Con respecto al segundo aspecto, se puede rescatar el concepto del equilibrio que debe haber entre el cálculo y la ejecución, ya que la forma de computar debe ser acorde a las capacidades de ejecución disponible. En nuestro sistema, no existe un computador de grandes características, capaz de computar grandes volúmenes de datos, por lo que se reduce la cantidad de información como tamaño de redes, dimensiones del input, entre otros. Esto para lograr un equilibrio entre el tiempo que toma realizar la acción y la calidad de esta. Ya que no nos sirve que el sistema calcule el punto de agarre óptimo y tarde más de una hora en realizarlo. Por lo que siempre hay que tener presente este tipo de trade-off.

Bajo esas dos ideas mencionadas anteriormente, se realiza y se planifica la implementación del sistema.

3. Método Desarrollado

El trabajo desarrollado se divide en tres partes:

1) En primer lugar, se encuentra el procesamiento y reconocimiento de imágenes, para identificar los distintos objetos en el espacio de trabajo.

2) La segunda parte consiste en la detección de un punto y ángulo de agarre eficiente para llevar a cabo la acción de agarre, tarea para la cual se utilizaron técnicas de machine learning.

3) Por último, tenemos la parte de robótica, donde se unen las dos partes anteriores y se implementan en el robot Baxter, además de preparar un espacio de trabajo adecuado para Baxter.

Se utilizó el robot Baxter, ya que cuenta con las capacidades necesarias para ejecutar la tarea de agarre de distinto tipo, gracias a su forma humanoide y sus dos brazos con garras paralelas, lo que permite agarrar una amplia gama de objetos. Además, este tipo de robot está pensado para hacer tareas industriales sencillas como cargar y descargar, ordenar y manejar materiales. Para hacer que Baxter lleve a cabo la tarea de agarrar objetos, se estudiaron diversas técnicas de reconocimiento de imágenes y machine learning, además de las librerías y herramientas que ofrece ROS (Robot Operating System) [33].

Para realizar el robotic grasping el robot primero tiene que realizar una secuencia de tareas:

1) Identificar y reconocer los distintos objetos que hay sobre el espacio de trabajo.
2) Posicionar el gripper sobre el objeto seleccionado.
3) Analizar su forma y contorno, para determinar el punto y ángulo de agarre efectivo.
4) Ejecutar el movimiento de agarre.
5) Dejar el objeto en un depósito establecido para su tipo.

A continuación, se expone de manera detallada los procesos de visión artificial, detección de agarre e implementación en Baxter

3.1. Visión Artificial

La visión artificial, también conocida como visión por computador, es un subcampo de la inteligencia artificial cuyo propósito es programar una máquina para que “entienda” una escena o las características de una imagen. Es decir, permite el análisis y proceso de imágenes para extraer información relevante de ellas. En nuestro caso, es utilizada para ver e identificar los distintos

objetos que puede encontrar el robot Baxter en su espacio de trabajo. Para realizar esta tarea se utilizó OpenCV para el procesamiento de imágenes y una red neuronal profunda encargada de catalogar los objetos.

OpenCV (Open Source Computer Vision Library), es una biblioteca open source que posee métodos de machine learning y reconocimiento de imágenes. Es compatible con C++, MatLab, Python y Java, lo que lo hace idóneo para trabajar con el robot Baxter.

Para diseñar y entrenar la red neuronal encargada de la clasificación se escogió Keras utilizando Tensorflow como backend [20]. Keras es una librería de redes neuronales de código abierto escrita en Python. Tensorflow es una biblioteca de código abierto para aprendizaje automático, desarrollado por Google. En esencia, Keras es una capa que se ejecuta sobre Tensorflow, construyendo los modelos con dicha tecnología.

Esta parte del trabajo se puede dividir en dos secciones: procesamiento de imágenes y clasificación de objetos, las cuales se detallan a continuación.

3.1.1. Procesamiento de imágenes

La imagen del espacio de trabajo proviene de la cámara ubicada en el brazo izquierdo de Baxter, con la que se captura una imagen RGB con una dimensión de 1280 x 800 píxeles.
En esta parte del sistema se ejecutan dos funciones, img_process (ver Algoritmo 1) y recortes_img (ver Algoritmo 2).
La primera función, img_process, como su nombre lo indica, se encarga de realizar el procesamiento de las imágenes obtenidas de la cámara.

Algoritmo 1: Procesamiento

```
Función img_process(img)
        gray ←cvtColor(img)
        blur ← GaussianBlur(gray)
        canny ← Canny(blur)
        kernel ← getStructuringElement(MORPH_ELLIPSE)
        dilated ← dilate(canny, kernel)
        contornos [ ] ← findContours(dilated)
        for c in contornos do
                x, y, w, h ← boundingRect(c)
                área ← w*h
                if área <= 1000 or y < 300 then
                        continue
                cont_validos.append(c)
return cont_validos
```

Una vez obtenida la imagen, se procede a transformar a escala de grises para reducir su dimensión y facilitar su procesamiento. Además, se aplica un Gaussian blur o desenfoque Gaussiano, el cual es un filtro que ayuda a eliminar el ruido de la imagen, suavizando y eliminando detalles de esta. Luego del procesamiento anterior, la imagen resultante está lista para aplicar el detector de bordes Canny [4], el cual nos entrega todos los bordes presentes en la imagen. Después de obtener todos los bordes, se aplica una transformación morfológica de dilatación la cual se encarga de agrandar el borde detectado por la función Canny y de esta manera unir bordes que pudieran estar separados.

Figura 4: Diferencias entre los bordes detectados: Canny (izquierda), Canny con bordes aumentados (centro), imagen original (derecha)

Con la imagen de bordes dilatados, procedemos a buscar los contornos presentes en la imagen, que corresponden a curvas cerradas de puntos sin saltos, los que corresponden a los objetos presentes en el área de trabajo.

En la Figura 4 se puede apreciar cómo se agrandan los bordes detectados por la función Canny, lo cual facilita y mejora la detección de contornos, ya que en la imagen del martillo, el detector de contornos encuentra dos contornos independientes que serían dos objetos, mientras que al aplicarlo a la imagen dilatada solo encuentra uno.

Debido a la posición de la cámara, en todas las imágenes captadas está presente el gripper del robot, lo cual dificulta el procesamiento, debido a que se encuentran todos los contornos en la imagen, incluidos los del gripper. Por lo que para ignorar esta parte de las imágenes, se define un área de interés limitando la detección de contornos. Se ignora cualquier contorno detectado con una coordenada de altura menor a 300 píxeles ($y<300$). Esto se puede apreciar en la Figura 5. También se omiten todos los contornos que tengan un rectángulo con área menor a 1000 pixeles^2, para evitar detectar objetos muy pequeños o pequeños detalles en las imágenes, ya que en varias pruebas, por problemas de iluminación generalmente, se detectaban contornos pequeños dentro de los objetos, perjudicando la ejecución del programa. Los contornos que cumplan con lo anterior son almacenados en una lista como *contornos válidos*.

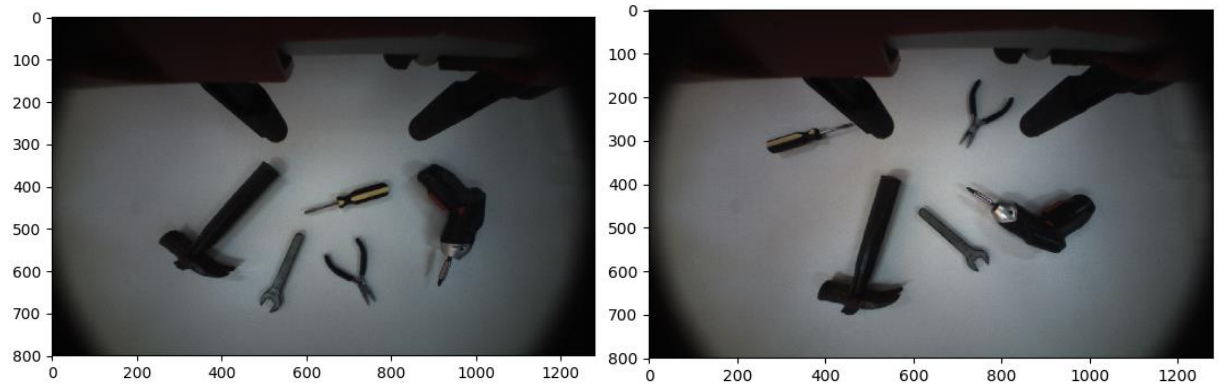

Figura 5: Desde la perspectiva de Baxter, donde puede ver todos los objetos (izq) y donde solo ve tres (der).

Algoritmo 2: Recortes de Imagen

Función recorte_img(imagen, contornos)
    for c in contornos **do**
        x, y, w, h ← boundingRect(c)
        xcentro, ycentro ← x+w/2, y+h/2
        global xi, yi
        yi, yf ← y-margen, y+h+margen
        xi, xf ← x-margen, x+w+margen
        ptos_corte.append((xi, yi))
        crop ← imagen[yi:yf, xi:xf]
        recortes.append(crop)
return recortes

Después de obtener los contornos válidos, se procede a obtener los recortes de cada contorno, que corresponden al área delimitada por los puntos de contorno, para ello se recorre la lista de contornos, se obtiene el rectángulo que lo contiene, se aplica un margen de corte para aumentar el tamaño del corte realizado y así obtener el contexto del objeto en el espacio, luego se guardan los puntos de corte de cada uno en una lista global y se procede a cortar la imagen, guardando cada sección de imagen en una lista. De esta manera tenemos cada objeto de interés por separado en una imagen cada uno.

Ahora que ya tenemos las imágenes aisladas de cada objeto, se procede a identificar a qué clase de objeto corresponde.

3.1.2. Clasificación de objetos

Para realizar la tarea de clasificación es necesario tener un sistema inteligente que sea capaz de reconocer patrones, formas, colores y otras características, con el fin de poder hacer una predicción con esta información. Para realizar esto se utilizó un método de Deep learning [23] o aprendizaje profundo, diseñando una red neuronal de tipo convolucional, la más indicada para trabajar con imágenes debido a la cantidad de parámetros que se manejan en este tipo de datos [22]. Estas son muy similares a las redes neuronales comunes como el perceptrón multicapa, pero se diferencian en la arquitectura, ya que se compone de tres tipos de capas: capa convolucional, capa de reducción o pooling y capa densamente conectada (Figura 7).

La capa convolucional es la capa encargada de extraer información y características de la imagen. Esta capa recibe la imagen y aplica sobre ella un filtro que genera un mapa de características de la imagen original.

La capa de reducción se coloca generalmente después de la capa convolucional y como su nombre lo indica, reduce la cantidad de parámetros al guardar las características más comunes de la imagen (Figura 6).

Por último, tenemos la capa densamente conectada, donde cada píxel de imagen se considera como una neurona separada al igual que en un perceptrón multicapa.

Gracias a estas capas es posible codificar y procesar ciertas propiedades, ganando eficiencia y reducción en la cantidad de parámetros en la red [22].

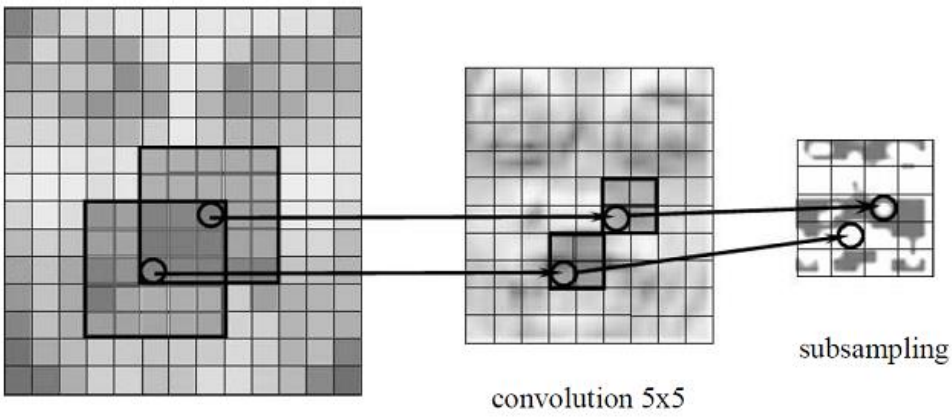

Figura 6: Operación de convolución y pooling o agrupación, obtenida de [19]

Para el proceso de entrenamiento se utilizó un dataset etiquetado con 11 objetos: alicate, control remoto, cuchillo, destornillador, gamepad, llave, martillo, mouse, reloj, taladro y tijera.

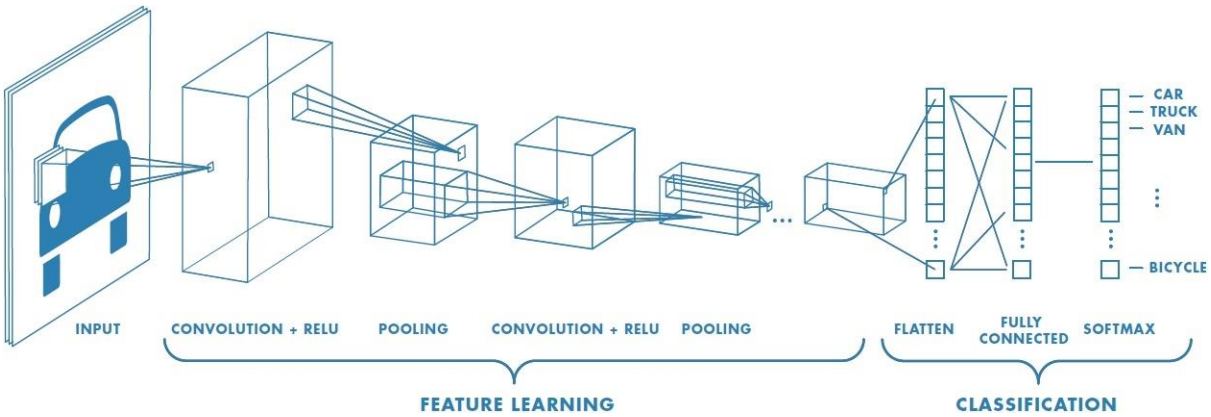

Figura 7: Arquitectura de red neuronal convolucional estándar, obtenida de [7]

La arquitectura de la red convolucional utilizada es una versión simplificada de la red vgg-16 [37], compuesta por 6 capas convolucionales, 3 capas de agrupación (max pooling, Figura 8), agrupadas en conjuntos de dos capas convolucionales seguidas de una capa de agrupación, una capa densamente conectada y una capa de salida. La capa de entrada recibe una imagen de dimensión 100x100x3, ya que un tamaño de entrada mayor presenta más tiempo y un mayor uso de recursos, lo cual afecta negativamente a la ejecución del programa en el robot.

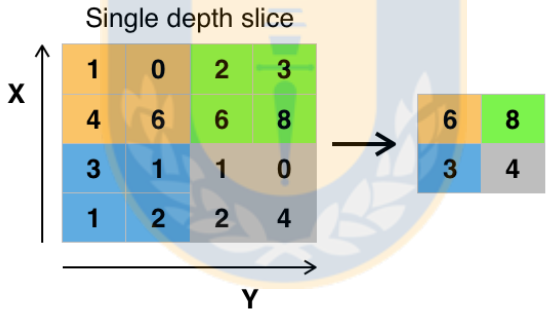

Figura 8: Operación de Max Pooling, obtenida de [8]

Para realizar el proceso de entrenamiento, inicialmente se pensó utilizar el dataset de ImageNet [17], debido a la gran cantidad de categorías e imágenes que posee, siendo uno de los datasets más grandes que hay actualmente. Sin embargo, sólo se pudo acceder a un conjunto reducido de imágenes, ya que la autorización solicitada para acceder al dataset completo no fue respondida. Se utilizó este conjunto reducido de 8000 imágenes aproximadamente, para realizar los primeros entrenamientos, sin obtener buenos resultados debido a que muchas de las imágenes utilizadas tienen demasiada información, como objetos ajenos a la categoría de la imagen. Por ejemplo, en la categoría martillo, se podían encontrar imágenes con personas utilizando el martillo o martillos junto a otras herramientas, esto se puede apreciar en la Figura 9. Esto supone un problema para una red relativamente pequeña y un entrenamiento con una máquina con capacidades y poder de cómputo limitado.

Figura 9: Ejemplos de imágenes presentes en el dataset de ImageNet con demasiada información, obtenidas de [17]

Finalmente, se decidió crear un dataset personalizado, donde las imágenes utilizadas fuesen lo más "limpias" posible, presentando únicamente el objeto en cuestión (la gran mayoría con fondo blanco) para facilitar y reducir el tiempo de entrenamiento de la red. Se utilizaron cerca de 2.200 imágenes para entrenar la red. Además, para mejorar el proceso de aprendizaje de la red, se realizó un proceso de *data augmentation*, aumentando de manera artificial el tamaño de nuestro dataset mediante la aplicación de distintos tipos de perturbaciones a las imágenes originales como podrían ser rotaciones, invirtiendo ejes, descentrando la imagen, entre otras. De esta forma, es posible obtener varias imágenes a partir de una sola (Figura 10).

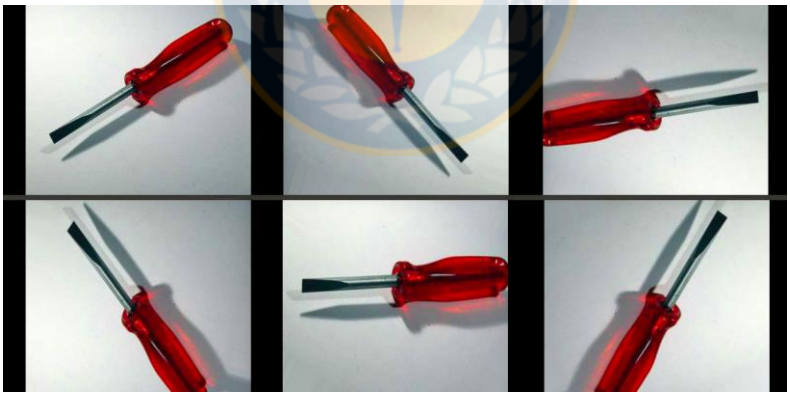

Figura 10: Ejemplo de aplicación de data augmentation a una imagen del dataset

Un problema que surgió con respecto al data augmentation, es que en algunas imágenes con forma alargada, al ejecutar algunas transformaciones y perturbaciones, las imágenes resultantes no eran representativas con la categoría correspondiente (Figura 11), por lo que, luego de hacer pruebas con las imágenes con estas características, se realizan las siguientes operaciones para mantener la integridad de la información de la imagen.

Se realiza un zoom aleatorio entre el rango de 0% y 20%, giro horizontal y vertical aleatorio sobre el eje correspondiente y una rotación aleatoria en un rango de 0 a 40 grados, siendo esta

última operación la que nos asegura que las imágenes obtenidas sean representativas con respecto a su clase.

Figura 11: Ejemplo de error al hacer data augmentation

Siguiendo con el procesamiento de imágenes, una vez obtenidos los recortes de cada objeto encontrado, cada recorte es reescalado a una dimensión de 100x100 píxeles. Cómo es posible que algunos recortes sean alargados, al momento de redimensionar la imagen estos quedarían totalmente deformados, por lo que para preservar el ratio de aspecto se agregan bordes a cada imagen (Figura 12).

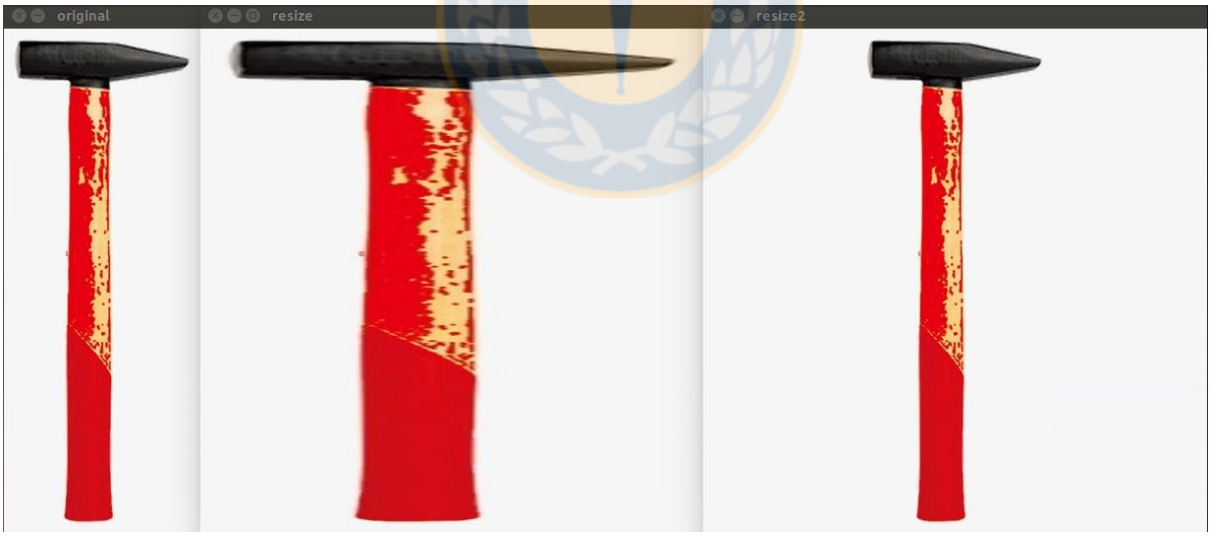

Figura 12: Imagen original (izquierda), redimensión incorrecta (centro), redimensión correcta(derecha)

Para realizar esta corrección, se define en base a las dimensiones originales de la imagen, donde es necesario agregar bordes, para rellenar el espacio vacío y obtener el tamaño deseado sin perder el aspecto de esta. Para generar este borde, se utiliza un tipo de borde que toma el último píxel del lado correspondiente de la imagen original y lo replica para generar el resto de imagen. Una vez reescaladas las imágenes, están listas para pasar por la red neuronal que realiza la

predicción (Ver Algoritmo 3), la cual nos entrega la clasificación de cada objeto encontrado. Ver Figura 13.

Algoritmo 3: Clasificador

Función predicción_obj(recortes[ ])
    For i en recortes **do**
        img ← reescalado(i)
        pred ← prediccion(img)
        nombres_obj.append(pred)
return nombres_obj[]

Esta función recibe una lista de recortes que contiene las imágenes de cada objeto en el espacio de trabajo. Cada recorte es reescalado a una dimensión de 100x100 pixeles y se realiza la predicción de este. Luego la categoría es almacenada en una lista. Finalmente se retorna la lista de categorías de los objetos.

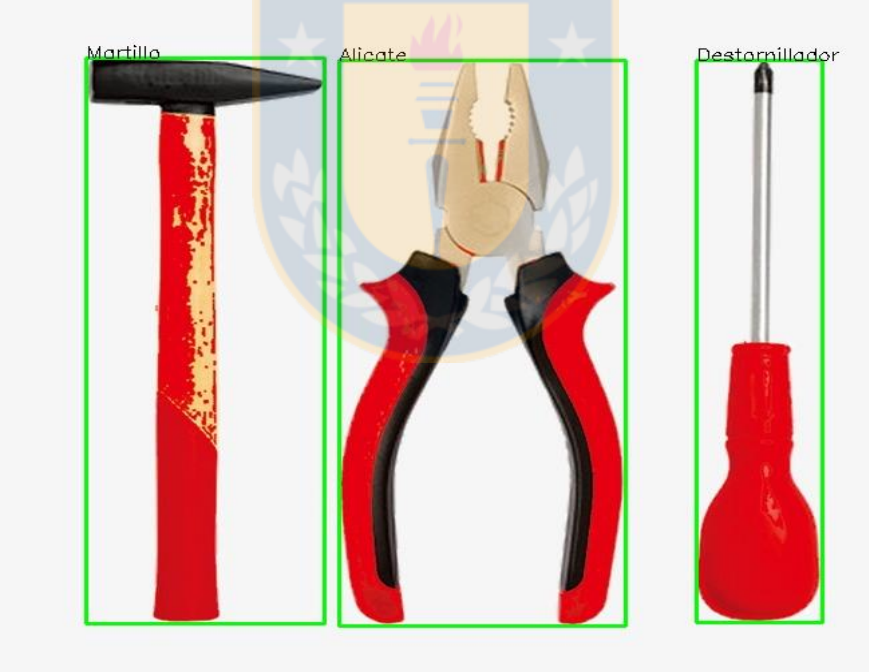

Figura 13: Predicción correcta del clasificador

3.2. Detección de Agarre

Una vez que ya hemos identificado todos los objetos ubicados en la mesa de trabajo, es necesario agarrar cada uno de ellos y dejarlos en el depósito correspondiente a la categoría del objeto. Para

realizar esta tarea es necesario detectar el punto de agarre efectivo y el ángulo necesario para llevar a cabo este agarre.

Para realizar esta detección se utilizó una red neuronal convolucional pre entrenada, la cual es capaz de detectar el rectángulo de grasping prácticamente de cualquier objeto [32].

La red utilizada tiene una arquitectura basada en la red AlexNet [22], la única diferencia respecto a esta es la cantidad de neuronas en la séptima capa. Los pesos convolucionales de la red son inicializados con los pesos pre entrenados de ImageNet. En la primera capa convolucional se usa un filtro de tamaño 11x11 píxeles, con un desplazamiento de 4 pixeles, en la segunda se usa un filtro de tamaño 5x5 píxeles con un desplazamiento de 1 píxel y para la tercera, cuarta y quinta capa se usa un filtro de tamaño 3x3 píxeles. En la Figura 14 se muestra la arquitectura de la red.

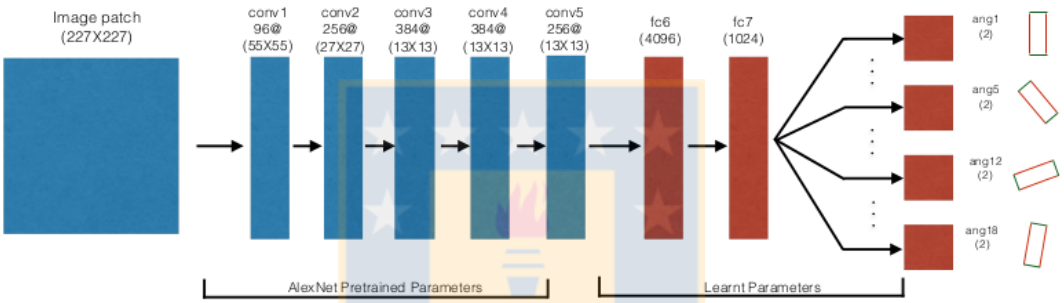

Figura 14: Arquitectura de red neuronal utilizada en [32]

La metodología que utilizaron para obtener los datos necesarios para el entrenamiento de la red, fue mediante un robot Baxter, configurando un espacio de trabajo con varios objetos con distintas dificultades de agarre. Se ejecuta un análisis de imágenes para detectar regiones de interés mediante sustracción de fondo. Luego se elige un punto en la región de interés de manera aleatoria y un ángulo entre 0º y 180º, también de manera aleatoria, luego se ejecuta la acción de agarrar el objeto, el cual es elevado a una altura de 20 centímetros y se anota si fue un agarre válido o no. De esta manera se ejecutan múltiples movimientos para generar el dataset. En el paper [32] se obtienen cerca de cincuenta mil puntos de agarre, válidos y no válidos, con cerca de 700 horas de trabajo robot (Figura 15).

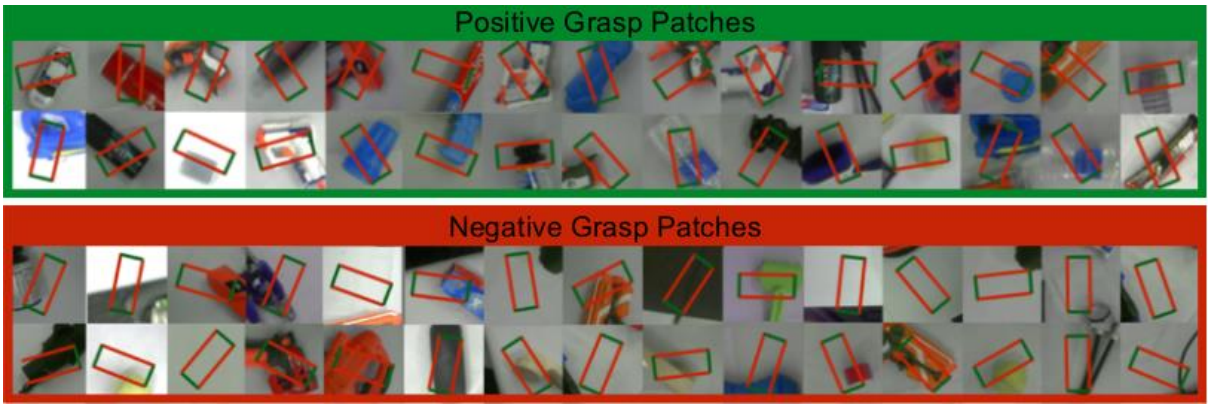

Figura 15: Parches positivos y negativos utilizados para entrenar la red neuronal [32]

La manera en que trabaja el detector de grasp es tomar una imagen del objeto en cuestión, luego esa imagen se divide en múltiples parches, con el fin de obtener distintas partes de la imagen, los cuales son procesados y el predictor nos dice la posibilidad de éxito de agarre en 18 ángulos diferentes (0º, 10º, ..., 170º), tomando el centro del parche. Lo que da como resultado un arreglo binario de largo 18, que nos indica si fue posible calcular el rectángulo de agarre efectivo con cada uno de los 18 ángulos, representando una puntuación con respecto a la posibilidad de agarre. Luego se elige el parche con mayor puntuación respecto a los ángulos posibles de agarre y se define la posición y ángulo de agarre eficiente.

En la Figura 16 se pueden ver algunos ejemplos de cajas de agarre calculadas en ejemplos de prueba realizadas en nuestro sistema.

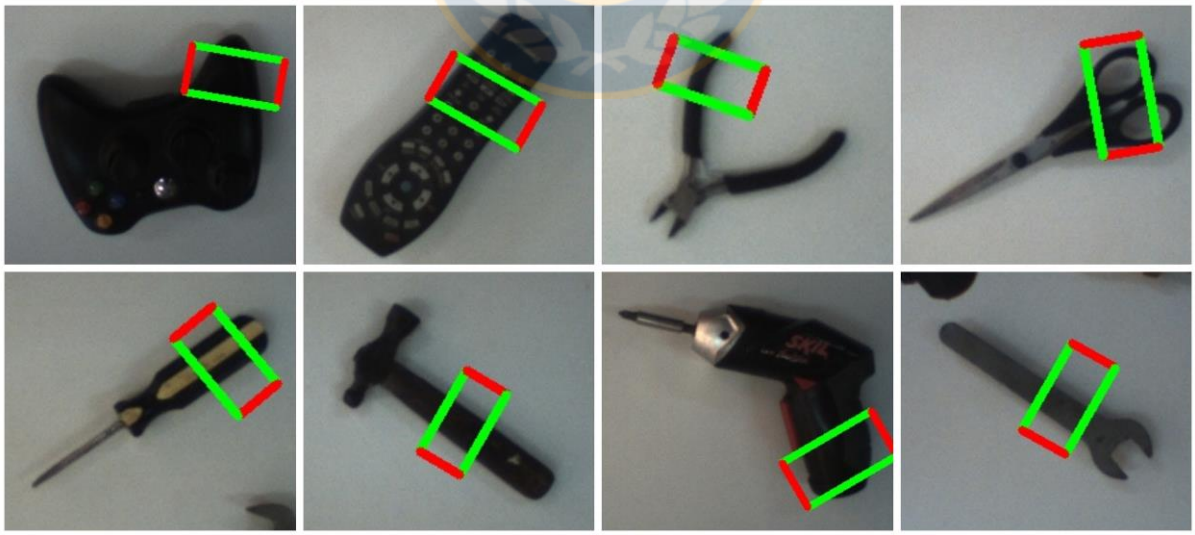

Figura 16: Ejemplos de rectángulos de agarre calculados en ejecuciones

3.3. Robótica

Para iniciar la implementación, es necesario preparar y tener configurado de manera adecuada el espacio de trabajo de Baxter. Para ello se utiliza una mesa cubierta con una plancha de madera color blanco opaco, lo cual permite tener un plano donde se obtiene un gran contraste con respecto a los objetos sobre él, facilitando la tarea de detección de bordes y contornos. Además, la opacidad ayuda a reducir el brillo y reflejo que pueden ocasionar la luz solar y artificial del mismo laboratorio. Este laboratorio cuenta con grandes ventanales a un costado, lo que implica una gran cantidad de luz solar entrante, sobre todo en las tardes, lo que dificulta bastante obtener buenos resultados en la detección. Para mitigar esto se utilizan cortinas gruesas que ayudan a reducir la luz y hacer que sea más homogénea, además la mayoría de pruebas se realizan en la mañana.

Figura 17: Espacio de trabajo de Baxter

Una vez ejecutado el programa, el robot Baxter, ubicado frente a la mesa, posiciona su brazo izquierdo a una altura de 30 centímetros, aproximadamente 47 centímetros hacia el frente y desplazado 10 centímetros hacia la izquierda respecto al centro del torso del robot. Se utilizan estas distancias, principalmente para aprovechar el espacio de trabajo, ya que de esta manera el brazo se ubica casi al centro de la mesa, facilitando el acceso a toda el área de trabajo, lo que nos permite tener un buen ángulo de visión y espacio de acción. Ésta corresponde a su posición inicial de búsqueda.

Además, en el espacio de trabajo se encuentran tres depósitos, como se puede apreciar en la Figura 17, donde en el primero se almacenan herramientas pequeñas como alicates, cuchillos, destornilladores y tijeras. En el segundo se guardan elementos más delicados como controles, gamepad, mouse y relojes. En el último depósito se almacenan herramientas pesadas como martillos, taladros y llaves.

En primer lugar, se realiza un análisis para extraer la imagen que captura la cámara ubicada en el brazo, para identificar los distintos objetos encontrados, mostrando por consola la lista de elementos encontrados y la cantidad de cada uno. La Figura 18 muestra el espacio de trabajo desde el punto de vista de Baxter.

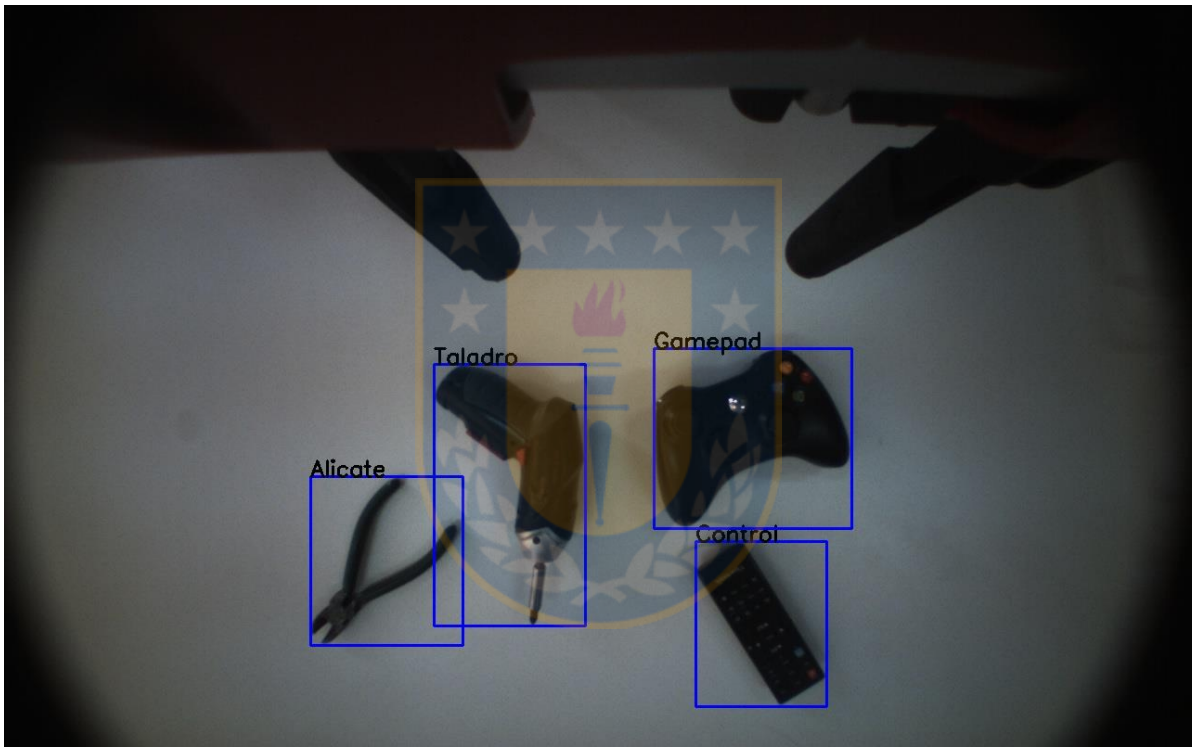

Figura 18: Punto de vista de Baxter

Luego se le da a escoger al usuario como desea recogerlos, teniendo 3 modos de ejecución disponibles:

- Búsqueda por selección: Se ingresa por teclado el nombre del objeto que se quiere recoger y se ejecuta un agarre “inteligente”.
- Búsqueda automática: En este modo se recogen todos los elementos ubicados en la mesa, de manera aleatoria, con agarres “inteligentes”.
- Búsqueda básica: Se recogen todos los elementos de la mesa, pero de manera simple, o sea encontrando el centro del objeto y ejecutando el agarre.

Entiéndase por agarre “inteligente”, el proceso de calcular un punto de agarre efectivo y ejecutar la acción.

Antes de describir cada modo, se detallan a continuación los aspectos comunes al proceso de agarre de los tres modos disponibles.

La ejecución de agarre se realiza en base a los parámetros: punto de agarre, ángulo de agarre y nombre de categoría de objeto. Con el nombre del objeto, se define a cuál depósito se llevará el objeto una vez agarrado. En el Algoritmo 4 se puede ver el funcionamiento desde un punto de vista técnico.

Una vez definido a qué depósito va el objeto en cuestión, se procede a realizar la transformación de píxeles a coordenadas de Baxter, donde se le entrega el punto y la altura a la que se encuentra el brazo. Luego se procede a ejecutar la sucesión de movimientos:

1. Se desplaza a la posición donde se encuentra el punto de agarre con su ángulo.
2. Sobre la posición anterior, se procede a bajar hasta llegar a unos 8 centímetros sobre el objeto
3. Se sigue bajando hasta llegar a unos 2 centímetros entre el gripper y la mesa
4. Se cierra el gripper y se espera medio segundo para asegurar el agarre
5. Se levanta a unos 30 centímetros, altura predeterminada de Baxter con respecto a la mesa
6. Se desplaza a la posición del depósito correspondiente
7. Se baja hasta unos 5 centímetros de altura con respecto a la mesa
8. Se abre el gripper
9. Se sube a unos 40 centímetros
10. Se vuelve a la posición inicial

En los pasos 2, 5 y 9, se utilizan esas alturas para evitar movimientos erróneos por parte de Baxter. En el paso 2, se llega a una altura de 8 centímetros sobre el objeto, para corregir su trayectoria al punto calculado de agarre, ya que de esta manera, se logra tener una pausa y un acercamiento al punto deseado. Esto debido a imprecisiones que posee Baxter en sus brazos. En el paso 9, se sube a 40 centímetros de altura, ya que de esta forma se evita que, al momento de volver a la posición inicial, no se ejecute un movimiento extraño que altere el espacio de trabajo, ya que en algunas pruebas realizadas, al momento de dejar el objeto en un depósito, el brazo queda en una posición “incómoda”, ejecutando una corrección que golpea los depósitos de objetos. Al aumentar la altura, este problema se reduce.

En caso de que en el paso 4 no se pueda realizar el agarre o se caiga el objeto, el gripper tiene un sensor de fuerza, por lo que es posible detectar que la garra está vacía, por lo que en ese caso se vuelve a la posición inicial.

Algoritmo 4: Movimiento Baxter

```
Función move(punto, angulo, nombre)
    if nombre='Alicate' or 'Cuchillo' or 'Destornillador' or 'Tijera' then
        deposito ← (x1, y1)
    if nombre='Control' or 'Gamepad' or 'Mouse' or 'Reloj' then
        deposito ← (x2, y2)
    if nombre='Martillo' or 'Taladro' or 'Llave' then
        deposito ← (x3, y3)
    (px, py) ← pixel_to_baxter(punto)
    mover_baxter([px, py, 0], angulo)
    mover_baxter([px, py, -0.18], angulo)
    mover_baxter([px, py, -0.22], angulo)
    gripper.close()
    mover_baxter([px, py, 0], angulo)
    if gripper.force()=0 then
        gripper.open()
        mover_baxter([Xinicial, Yinicial,0], 0)
    else
        mover_baxter([deposito[0],deposito[1], 0], angulo)
        mover_baxter([deposito[0],deposito[1], -0.17], angulo)
        gripper.open()
        mover_baxter([deposito[0],deposito[1], 0.10], angulo)
        mover_baxter([Xinicial, Yinicial, 0.10], 0)
        mover_baxter([Xinicial, Yinicial, 0], 0)
```

A continuación, se presentan los tres modos de ejecución, mencionados anteriormente, de manera más detallada.

3.3.1. Búsqueda por selección

En este modo se presenta al usuario la opción de elegir qué objeto quiere recoger según su categoría. Se recogen todos los objetos que cumplan con la categoría solicitada, por lo que si se solicita recoger un taladro, el sistema recogerá todos los objetos detectados como taladros. Una vez que se entrega el nombre de la clase de objeto a recoger, el programa marca el objeto o los objetos presentes en el espacio, seleccionados de la lista de objetos, y accede a los puntos de recorte con respecto a la imagen global, para desplazarse hasta esos puntos. Una vez ubicado sobre el punto, se realiza un análisis de contornos para confirmar que el objeto está ahí. Como pueden haber más objetos cerca del objetivo deseado, el detector de contornos podría detectar más de un contorno, por lo que se aplican dos filtros. El primer filtro limita la imagen definiendo

un área de interés y luego guarda los centros de los contornos en una lista. En el caso de que se encuentre más de un centro, se aplica el segundo filtro que define el centro donde se ubica el gripper y se calcula la distancia a los centros de los contornos, eligiendo el que esté a una distancia menor con respecto al gripper.

En la Figura 19 se puede apreciar cómo funciona este filtro, donde se solicitó recoger la llave, se desplazó a su posición y se realizó el cálculo de las distancias entre los objetos cercanos, para elegir el objeto más cercano.

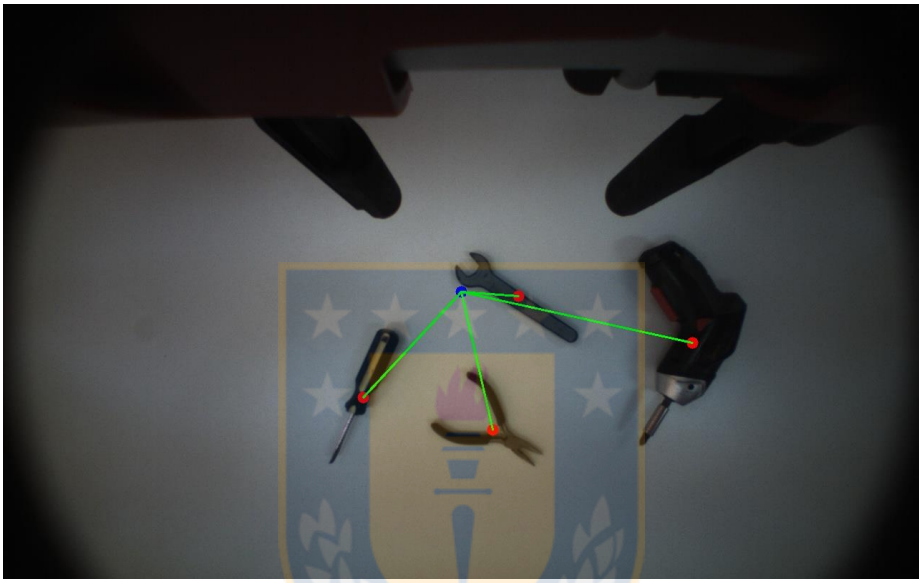

Figura 19: Representación del filtro, en base a distancias desde el centro (punto azul) hacia el centro de objetos (puntos rojos).

Luego de encontrar el contorno del objeto solicitado, se procede a realizar un recorte de la imagen captada por la cámara, para luego pasar el recorte por la red neuronal encargada de detectar el punto y ángulo de agarre adecuado. Una vez calculado, nos retorna el rectángulo de agarre y el ángulo, por lo que solo nos falta calcular el punto central de la caja de agarre. Se realiza la transformación de píxeles a coordenadas de baxter y se actualiza la pose actual del brazo para realizar la secuencia de movimiento.

3.3.2. Búsqueda automática

En este modo, el brazo se desplaza a cada recorte detectado en el análisis inicial, para realizar el agarre. Para ello sigue una secuencia bastante similar al modo anterior, donde en cada desplazamiento, se posiciona sobre el objeto, captura la imagen para realizar un análisis de contorno, luego se realiza el recorte de la imagen para obtener el objeto a analizar. Al obtener el recorte, se pasa por la red para detectar el rectángulo de agarre y se calcula su centro de agarre

para luego ejecutar la secuencia de movimientos, realizándose para cada recorte. En el caso de que no pueda recoger un objeto o se le caiga, este modo seguirá con el resto de objetos hasta que la lista de recortes se vacíe. Una vez terminada, se realizan dos búsquedas extras para comprobar de que no quede ningún objeto en la mesa.

Por ejemplo, en la mesa inicialmente hay 3 objetos, se procede a recogerlos, pero al momento de recoger el segundo, este se le cae, por lo que el brazo vuelve a la posición inicial y se desplaza al tercer objeto, ignorando el que se le acaba de caer. Una vez termina de depositar el tercer objeto, procede a realizar las búsquedas extras para recoger lo que se le pudiera haber caído, encontrando el objeto que le faltaba y dejándolo en su depósito. Se realizan dos búsquedas después de recorrer la lista de recortes por completo.

3.3.3. Búsqueda Básica

En este método, similar al método anterior, recoge todos los objetos que encuentre en la mesa, por lo que hace un proceso similar a los otros. En este caso se desplaza a los puntos de corte detectados en la imagen inicial, capturando la imagen sobre los objetos y analizando los contornos presentes en la imagen. Como en este modo no hay que utilizar la red para detectar los puntos de agarre eficientes, no es necesario volver a recortar la imagen, por lo que se procede a encontrar el rectángulo de área mínima que encierra al contorno, de esta manera obtener su centro y ángulo de grasp, calculado en base a la inclinación del rectángulo para luego sumar 90 grados y así obtener la posición correcta de las garras, para desplazarse sobre él y ejecutar la secuencia de agarre.

Cabe mencionar que este es el modo más rápido de los tres, ya que solo realiza un análisis por recorte, a diferencia de los otros que realizan dos.
Una vez se ejecuta el modo seleccionado, se vuelve a tomar una foto global y se realiza el análisis general inicial.

4. Experimentos y Resultados

En este apartado se describen los experimentos y pruebas realizados al sistema implementado y los resultados obtenidos.

Las pruebas realizadas se pueden dividir en tres partes. La primera parte consiste en analizar la red neuronal utilizada para clasificar los objetos. La segunda parte consiste en comprobar el funcionamiento del clasificador durante la ejecución en Baxter. Finalmente, la tercera y última parte de las pruebas evalúan la ejecución de los agarres por Baxter.

4.1. Clasificador

Antes de hablar de las pruebas realizadas en este apartado, se procede a explicar cómo está formulado el dataset de entrenamiento. Como se comentó en la sección 3.1, para realizar el proceso de entrenamiento se tenía la idea de utilizar un dataset público, como lo son ImageNet o CIFAR [5], los cuales tienen una gran cantidad de imágenes y categorías. El principal problema con estos datasets es que en muchas de sus imágenes hay otros objetos, lo que dificulta "entender" la imagen por parte de la red, además de que el número de imágenes impacta directamente en la cantidad de tiempo necesario para llevar a cabo el entrenamiento. Por esto, se tomó la decisión de diseñar un dataset más pequeño, con cerca de 2200 imágenes para el entrenamiento y 500 para la validación. Las imágenes fueron extraídas directamente de Google y la mayoría consiste en el objeto en cuestión con un fondo uniforme, lo cual nos permite tener buenos resultados en poco tiempo.

Para medir el rendimiento del clasificador, se realizaron dos pruebas, la primera consistió en comparar los distintos niveles de exactitud y pérdida alcanzados por los entrenamientos realizados con distintas configuraciones de redes. La segunda prueba consistió en probar la exactitud obtenida por las redes en un dataset de test, con imágenes nunca antes vistas por la red.

En la primera prueba se consideraron las siguientes configuraciones de redes neuronales:

- AlexNet modificada: similar a la original [22], con la diferencia en el tamaño de entrada (100x100) y el tamaño de filtro en la primera capa de convolución (3x3).
- MiniVGG7: versión reducida de la arquitectura VGG [37], con cuatro capas de convolución, dos capas densamente conectadas y una capa de salida.
- MiniVGG8: similar a la red anterior, con la diferencia en el número de capas, con 6 capas de convolución, una densamente conectada y una de salida.
- Red convolucional básica: con una arquitectura de dos capas de convolución, una densamente conectada y una de salida.

Las topologías de las redes se pueden ver con mayor detalle en Anexos 7.1.

Redes	Exactitud	Pérdida	Exactitud Validación	Pérdida Validación
AlexNet	81.03%	0.5770	76.77%	0.7872
MiniVGG-7	83.84%	0.5332	80.18%	0.7803
MiniVGG-8	86.09%	0.4673	85.90%	0.5399
Red Básica	69.94%	0.9476	71.32%	0.9627

Tabla 1: Resultados de entrenamiento

La Tabla 1 muestra los resultados obtenidos durante el proceso de entrenamiento, con la siguiente nomenclatura:

- Exactitud: Corresponde al porcentaje de predicciones correctas realizadas sobre el conjunto de entrenamiento. Mayor exactitud es mejor.
- Pérdida: Métrica que indica la inconsistencia o error entre las predicciones y las etiquetas correspondientes. Menor pérdida es mejor.
- Exactitud y Pérdida Validación: Corresponde a la exactitud y error obtenido en el conjunto de validación, el cual indica la capacidad de generalizar por parte de la red, ya que el conjunto de validación es nuevo para la red.

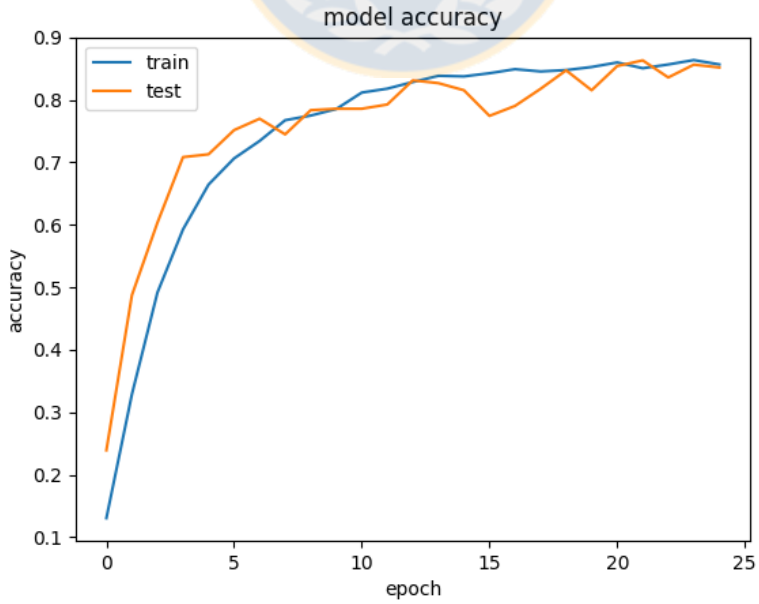

Figura 20: Gráfico exactitud (accuracy) MiniVGG-8

En la Figura 20, se puede apreciar la exactitud (accuracy) alcanzada en el proceso de entrenamiento (train) y validación (test) de la red, utilizando 2200 y 500 imágenes respectivamente. La forma del gráfico resultante indica que se realizó un entrenamiento correcto, ya que a medida que se avanza en épocas, y con cierta variabilidad, la exactitud va mejorando con respecto a la época anterior. Los gráficos de las otras redes utilizadas se pueden ver en Anexos 7.2.

Para realizar la segunda prueba, se utilizó un conjunto de imágenes nuevas para la red, con un total de 66 imágenes, donde cada una posee un solo objeto de las 11 categorías. Para ello se implementó un script en Python que recibe como input todas las imágenes de testing y realiza las predicciones correspondientes, simulando la operación explicada en la sección 3.2. Esta prueba nos entrega los resultados expuestos en la Tabla 2.

Redes	Correctas	Incorrectas	% de acierto
AlexNet	40	26	60.6%
MiniVGG-7	54	12	81.81%
MiniVGG-8	55	11	83.33%
Red Básica	27	39	40.9%

Tabla 2: Resultado de predicción en el set de prueba

Con los resultados obtenidos, se puede observar que la mejor red diseñada para nuestro caso es la red MiniVGG-8, la cual entrega un porcentaje de predicciones correctas superior a 80%, un resultado bastante satisfactorio para la tarea de clasificación. Esto se debe principalmente a que es la red más profunda de las 4 diseñadas para la experimentación. Además de que la arquitectura de VGG es más reciente que AlexNet, lo que significa que es un avance en la tarea de clasificación, mejorando el rendimiento.

La red MiniVGG-8 es utilizada tanto en las pruebas siguientes como en la versión final del sistema implementado.

4.2. Clasificación durante ejecución

En esta parte, se realizan pruebas directamente con el robot Baxter, utilizando sus cámaras para comprobar la precisión del clasificador, durante el funcionamiento del sistema. Esto se realiza, principalmente para comparar el rendimiento y ver cómo afecta la calidad de imagen obtenida de las cámaras integradas, además de otros factores externos, como la iluminación, sombras producidas por los objetos, entre otras. Esto debido a que, en las pruebas anteriores, se obtienen

los resultados en condiciones óptimas, con imágenes de alta calidad, con poco ruido y poca interferencia.

Para realizar esta prueba, se reunieron un total de 18 objetos (Figura 21) de las distintas categorías que es capaz de reconocer la red. Cada objeto es puesto bajo la cámara del Baxter en posición normal, invertida y rotada. Por lo tanto, se consideró una muestra de 54 imágenes con su predicción correspondiente. Los resultados se pueden ver en la Tabla 3.

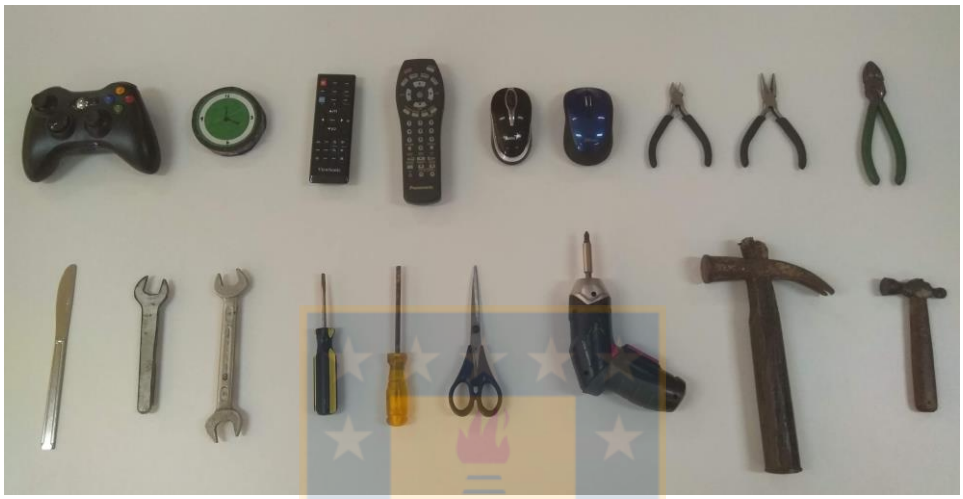

Figura 21: Objetos utilizados para pruebas

Input para Red Mini VGG-8	Correctas	Incorrectas	% de acierto
Imágenes de Alta calidad (sección 4.1)	55	11	83.33%
Imágenes de cámara Baxter	38	16	70.37%

Tabla 3: Comparación de resultados obtenidos en 4.1 y 4.2

Los resultados obtenidos en esta prueba nos muestran cómo funcionará el clasificador en la práctica y cómo impacta la calidad de las imágenes utilizadas para realizar la predicción, donde se ve una disminución en la cantidad de predicciones correctas, bajando más de un 10% de precisión. Esta reducción de predicciones correctas se debe más que nada a la calidad y definición de las imágenes obtenidas de las cámaras de Baxter, las cuales son bastante inferiores respecto a las usadas en las pruebas realizadas en 4.1. Esta diferencia se puede apreciar mejor en la Figura 22.

A pesar de esto, es un resultado bastante satisfactorio, y permite obtener buenas predicciones durante la ejecución del sistema.


Figura 22: Comparación de calidad entre las imágenes obtenidas por Baxter(izq) e imágenes de alta calidad utilizadas en 4.1(der)

4.3. Ejecución de agarre

En esta prueba, se somete al robot a realizar un número determinado de agarres de distintos objetos, en distintas orientaciones, para comprobar la capacidad de Baxter para ejecutar la acción de agarre. Para ello se posicionan los distintos objetos sobre el espacio de trabajo y se procede a ejecutar el agarre, registrando la acción como correcta o incorrecta.

Se utilizaron los mismos objetos de la prueba anterior (Figura 17), realizando tres intentos de agarre por objeto, con posiciones y orientaciones aleatorias, dando un total de 54 intentos de agarre, registrando cuando el agarre fue correcto o incorrecto. Para esta prueba, se comparó la efectividad del modo de búsqueda automática frente al modo de búsqueda básica, explicados en la sección 3.3.

Modo de Búsqueda	Correctas	Regular	Incorrectas
Básica	16	20	18
Automática	26	14	14
“Mejora obtenida”	62.5%	-30%	-22.22%

Tabla 4: Resultados de los modos de agarre

Donde:

- Correctas: Indica que el agarre fue realizado satisfactoriamente y dejado en su depósito sin inconvenientes.
- Regular: Corresponde a la acción de agarre donde falla un poco la precisión de los actuadores de Baxter, ya sea por unos centímetros o se le cae el objeto antes de llegar al depósito.
- Incorrectas: Indica que el agarre calculado y ejecutado es completamente erróneo.

- Mejora obtenida: Corresponde al porcentaje de mejora de la búsqueda automática respecto a la búsqueda básica.

Los resultados expuestos en la Tabla 4, nos muestra la cantidad de agarres satisfactorios realizados por los modos de ejecución básica y automática. Donde el modo más efectivo es el automático, ya que este aumenta un 62.5% la cantidad de agarres correctos, disminuye la cantidad de agarres regulares en un 30% y reduce el número de ejecuciones incorrectas en un 22.22%, respecto al modo Básico.

Estos resultados nos muestran que el robot es capaz de ejecutar los agarres, a pesar de los fallos observados, lo cual se debe principalmente a la imprecisión que poseen los brazos de Baxter, además de la calidad de las imágenes obtenidas de las cámaras del robot que impactan en la precisión.

En esta prueba nos damos cuenta que la precisión en el caso de la búsqueda básica es bastante inferior a la búsqueda automática, y esto se debe más que nada a la forma de los distintos objetos. Principalmente los objetos con formas extrañas o poco simétricas, como joysticks, taladros o alicates, resultaron ser los objetos más complicados de agarrar, ya que en el modo inteligente, por lo general el agarre se realiza por el mango, a diferencia del modo básico que va al centro del objeto, lo cual en varios casos complica bastante el agarre. Además, otro problema con este punto, es que en la búsqueda básica, la efectividad del agarre depende mucho del tamaño del objeto a recoger, lo cual significa que es necesario realizar ajustes a la apertura de los grippers del robot.

4.4. Análisis de resultados

Los resultados obtenidos en las distintas pruebas son satisfactorios, se logró obtener un buen funcionamiento del sistema en general como de los módulos por separado, aunque existen aspectos que se pueden mejorar, como el clasificador o la ejecución de los agarres.

En la parte de procesamiento de imágenes, se obtuvieron resultados satisfactorios, donde se realizaron pruebas entre varias redes para decidir la mejor opción a utilizar. De estas pruebas se obtuvo que la mejor configuración de red fue la MiniVGG-8, obteniendo un porcentaje de acierto de 83% con imágenes de alta calidad y un 70% con imágenes de la cámara de Baxter, superando al resto de configuraciones. A pesar de que el porcentaje de acierto no es el más alto, este se puede mejorar realizando sesiones de entrenamiento más extensas y utilizando mayor poder de procesamiento, por ejemplo, mediante GPU, las cuales pueden reducir mucho el tiempo de entrenamiento comparado con el CPU.

En la parte de ejecución de agarres, los resultados obtenidos fueron los esperados: Se realizó la comparación entre los modos de ejecución del sistema, donde se muestra la diferencia entre la

calidad y cantidad de ejecuciones de agarres realizados, dando como resultado que el mejor modo es el automático, pero también es el más lento. Esto indica que Baxter es capaz de recoger los objetos de manera correcta (la mayoría de las veces), pese a que hubo varias ejecuciones de grasp con pequeños errores que perjudican el funcionamiento y ejecución de la acción. Por ejemplo, en algunas ocasiones el gripper llegó con unos centímetros de desviación con respecto al agarre calculado, evitando un agarre efectivo. En otras ocasiones, no se realizó un agarre con la suficiente fuerza, provocando que el objeto cayera a la mesa antes de llegar al depósito. La mayoría de estos detalles son producto de pequeñas imprecisiones inherentes de los brazos de Baxter, lo cual es difícil de corregir.

La comparación realizada entre los modos de ejecución del sistema, nos muestra la diferencia entre la calidad y cantidad de ejecuciones de agarres realizados, donde el mejor modo es el automático, pero también el más lento.

Tomando en consideración los resultados obtenidos, el desempeño del sistema fue el esperado, mostrando una serie de acciones y movimientos competentes para el problema de agarre robótico.

5. Conclusiones

En esta memoria de título se propuso diseñar un sistema que permitiera a Baxter ver, identificar y recoger objetos sobre un espacio de trabajo para dejar en un espacio determinado, dependiendo de la categoría de este, imitando una tarea de organización de elementos. Para ello se utilizó procesamiento digital de imágenes e inteligencia artificial, precisamente deep learning, para realizar esta propuesta. Surgieron algunos problemas durante el desarrollo del proyecto, pero se logró cumplir la propuesta. Los principales problemas tuvieron que ver con la precisión de los actuadores (grippers) de Baxter, ya que estos tienen un margen de error que en ocasiones perjudica la ejecución de los agarres. Por lo que agarrar objetos que requieran movimientos de alta precisión, resulta complicado con Baxter. A pesar de esto, de los objetos utilizados en las pruebas (Sección 4) es posible agarrarlos en la mayoría de casos.

Un trabajo anterior, realizado con el Baxter en la Universidad de Concepción, resultó en la implementación de un sistema que era capaz de ver herramientas quirúrgicas y recogerlas para dejarlas en cajas. Estas herramientas tienen códigos de colores que indican que herramienta eran y de donde había que recogerlas, por lo que el sistema no tenía que "pensar" para ejecutar los movimientos, solo tenía que ver los códigos y ejecutar una acción predeterminada. Por lo que el proyecto realizado en esta memoria es un paso más allá con respecto a la implementación anterior, ya que el sistema necesita hacer un análisis del ambiente y de cada objeto de manera individual, generando una representación del estado actual del ambiente y en base a eso tomar decisiones y actuar. Esta implementación podría servir como base para construir nuevos sistemas con mayor complejidad perfeccionando el robotic grasping u otro tipo de sistema robot, ya que la implementación está separada en módulos, los cuales se podrían utilizar en otras implementaciones.

5.1. Trabajo Futuro

Existen varios posibles caminos para trabajos futuros, entre ellos podemos mencionar:

Mejorar Precisión

- Mejorar la precisión general del sistema, donde es el mayor problema para la ejecución de los agarres. Para solucionar esto se podrían utilizar cámaras con mejor calidad de imagen o utilizar sensores extras que nos permitan obtener una representación más realista y precisa del ambiente.
- Mejorar y optimizar el sistema, para obtener tiempos de cálculo y respuesta reducidos.

Mejorar el sistema de visión artificial

- Hacer reconocimiento y seguimiento de los objetos en tiempo real, lo cual es una mejora que requiere bastantes recursos computacionales, pero podría mejorar enormemente la precisión del sistema, ya que sería capaz de detectar posibles desviaciones al momento de recoger objetos y de esta manera efectuar correcciones a la trayectoria.
- Se podría aumentar el tamaño del área de trabajo realizando un barrido de búsqueda más extenso, ya que actualmente el espacio de trabajo se define en base a una foto tomada en una posición estática.

Mejorar clasificador

- Aumentar el número de categorías que el sistema es capaz de reconocer
- Capacidad de detectar objetos montados, utilizando cámaras con profundidad o alguna otra tecnología que permita facilitar esta tarea.
- Mejorar porcentaje de aciertos, el cual se puede lograr utilizando un set de datos más grande, realizando entrenamientos más largos y modificando la red ya sea haciéndola más profunda o utilizando modelos con un mejor desempeño

6. Bibliografía

[1] Amazon paying $775M for Kiva warehouse robotics company. GeekWire. 2012. Retrieved from: https://www.geekwire.com/2012/amazon-buying-kiva/

[2] Baxter-making Rethink Robotics closes its doors. New Atlas. 2018. Retrieved from: https://newatlas.com/rethink-robotics-closing-down/56664/

[3] M. Bélanger-Barrette. (2015). First Amazon Picking Challenge. USA: Robotiq. Retrieved from https://blog.robotiq.com/amazon-picking-challenge

[4] J. Canny. A Computational Approach to Edge Detection. November 1986.

[5] CIFAR (Canadian Institute for Advanced Research). Retrieved from: https://www.cs.toronto.edu/~kriz/cifar.html

[6] CNET. (2014, November 30). CNET News- Meet the robots making Amazon even faster. Retrieved from https://www.youtube.com/watch?v=UtBa9yVZBJM

[7] Convolutional Neural Network. MathWorks. Retrieved from: https://www.mathworks.com/solutions/deep-learning/convolutional-neural-network.html

[8] Convolutional neural network. Wikipedia. Retrieved from: https://en.wikipedia.org/wiki/Convolutional_neural_network

[9] C. Eppner, S. Hofer, R. Jonschkowski, R. Martin-Martin, A. Sieverling, V. Wall and O. Brock. Lessons from the Amazon Picking Challenge: Four Aspects of Building Robotic Systems. 2016.

[10] M. Gualtieri, J. Kuczynski, A. M. Shultz, A. t. Pas, H. Yanco and R. Platt. Open World Assistive Grasping Using Laser Selection. ICRA, 2017.

[11] M. Gualtieri, A. Pas, K. Saenko and R. Platt. High precision grasp pose detection in dense clutter. 2016.

[12] M. Gualtieri and R. Platt. Viewpoint Selection for Grasp Detection. IROS, 2017.

[13] E. Guizzo. (6 May 2013). Rethink Robotics Opens Up Baxter Robot For Researchers. IEEE Spectrum. Retrieved from: https://spectrum.ieee.org/automaton/robotics/humanoids/rethink-robotics-baxter-research-robot.

[14] K. He, X. Zhang, S. Ren and J Sun. Deep Residual Learning for Image Recognition. 2015.

[15] C. Hodson. (2018). How robots are grasping the art of gripping. Nature: International journal of science. Retrieved from https://www.nature.com/articles/d41586-018-05093-1

[16] G. Huang, Z. Liu, L. van der Maaten and K. Q. Weinberger. Densely Connected Convolutional Networks. 2016.

[17] ImageNet. 2016. Retrieved from: http://www.image-net.org/

[18] ImageNet. Large Scale Visual Recognition Challenge(ILSVRC). Retrieved from: http://www.image-net.org/challenges/LSVRC/

[19] Intriguing properties of neural networks?. 2014. Retrieved from: https://dmm613.wordpress.com/2014/10/15/intriguing-properties-of-neural-networks/

[20] Keras: The Python Deep Learning library. Retrieved from: https://keras.io/

[21] Kinect for Windows. Microsoft: Windows Dev Center. Retieved from: https://developer.microsoft.com/en-us/windows/kinect

[22] A. Krizhevsky, I. Sutskever and G. E. Hinton. ImageNet Classification with Deep Convolutional Neural Networks. 2012.

[23] Y. LeCun, Y. Bengio and G. Hinton. Deep Learning. Nature. 2015

[24] I. Lenz, H. Lee and A. Saxena. Deep learning for detecting robotic grasps. The International Journal of Robotics Research, 2015.

[25] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz and D. Quillen. Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection. 2016.

[26] Machine Learning: What it is and why it matters. Cary, NC USA. Retrieved from https://www.sas.com/en_us/insights/analytics/machine-learning.html

[27] N. Marturi, M. Kopicki, A. Rastegarpanah, V. Rajasekaran, M. Adjigble, R. Stolkin, A. Leonardis and Y. Bekiroglu. Dynamic grasp and trajectory planning for moving objects. 2017.

[28] D. Morrison, P. Corke and J. Leither. Closing the Loop for Robotic Grasping: A Real-time, Generative Grasp Synthesis Approach. 2018.

[29] D. Pena. Common Applications of ML Algorithms. 2018. Retrieved form: https://towardsdatascience.com/common-applications-of-ml-algorithms-e16f21c85773

[30] Personal Robotics: Grasping. Cornell University. Retrieved from: http://pr.cs.cornell.edu/grasping/rect_data/data.php

[31] L. Pinto, J. Davidson and A. Gupta. Supervision via Competition: Robot Adversaries for Learning Tasks. 2016.

[32] L. Pinto and A.Gupta. Supersizing self-supervision: Learning to grasp from 50k tries and 700 robot hours. IEEE International Conference on Robotics and Automation (ICRA), 2015.

[33] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger, R. Wheeler and A. Ng. *ROS: an open-source Robot Operating System*. 2009.

[34] J. Redmon and A. Angelova. Real-Time Grasp Detection Using Convolutional Neural Networks. 2015.

[35] M. Rouse and M. Haughn. What is image recognition. SearchEnterpriseAI. Retrieved from: https://searchenterpriseai.techtarget.com/definition/image-recognition

[36] C. Sanchez. Shakey, el primer robot inteligente de la historia y el abuelo del coche autónomo. 2017. Retrieved from: https://www.eldiario.es/hojaderouter/tecnologia/shakey-robot-inteligencia_artificial-coche_autonomo_0_632037257.html

[37] K. Simonyan and A. Zisserman. Very Deep Convolutional Networks for Large-Scale Visual Recognition. ICLR, 2015.

[38] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke and A. Rabinovich. Going Deeper with Convolutions. 2014.

[39] J. Tremblay, T. To, B. Sundaralingam, Y. Xiang, D. Fox and S. Birchfield. Deep Object Pose Estimation for Semantic Robotic Grasping of Household Objects. Conference on Robot Learning (CoRL), 2018.

[40] Understanding Your Environment: An Important Step in Robot Navigation. Waypoints Robotics. Retrieved from: https://waypointrobotics.com/blog/robot-navigation-importance-of-environment/

7. Anexos

7.1. Redes neuronales utilizadas en experimentos

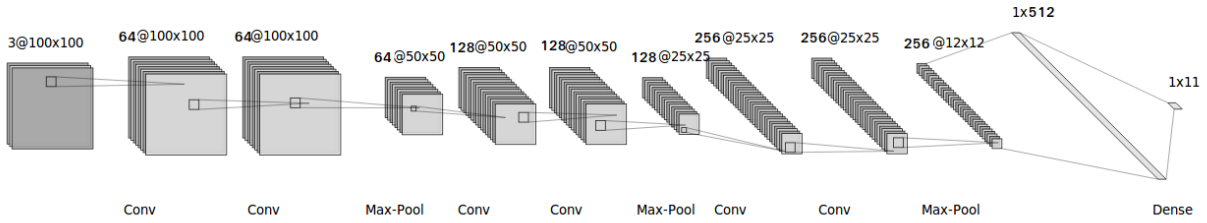

Figura 23: MiniVGG-8 usada para pruebas en 4.1, 4.2 y versión final del sistema

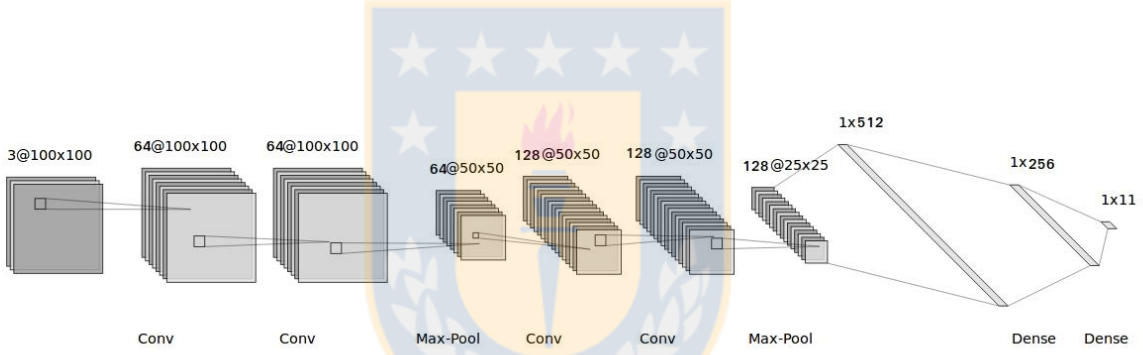

Figura 24: MiniVGG-7 usada para pruebas realizadas en 4.1

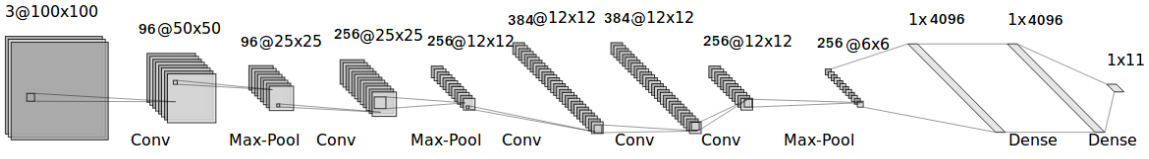

Figura 25: AlexNet usada para pruebas realizadas en 4.1

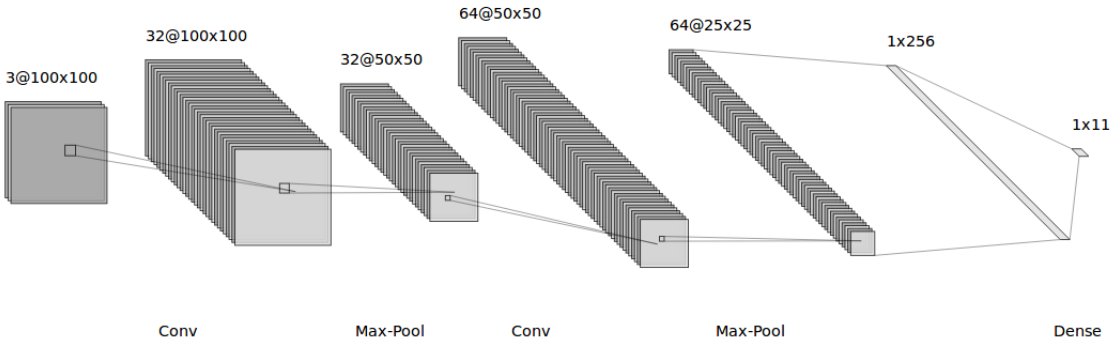

Figura 26: Red Básica usada para pruebas realizadas en 4.1

7.2. Gráficos de exactitud de las redes entrenadas

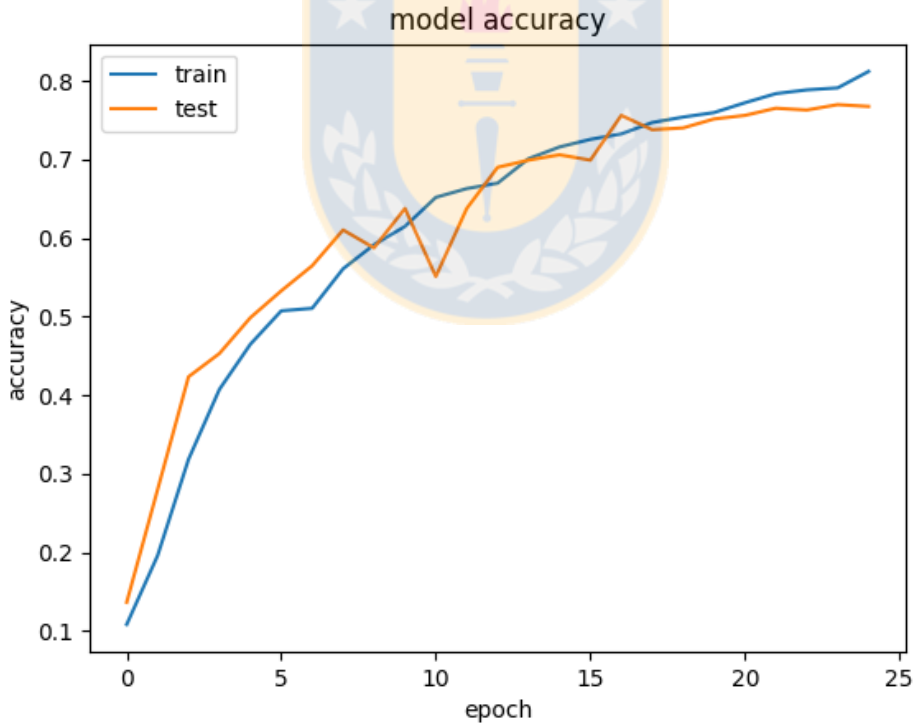

Figura 27: Gráfico accuracy AlexNet

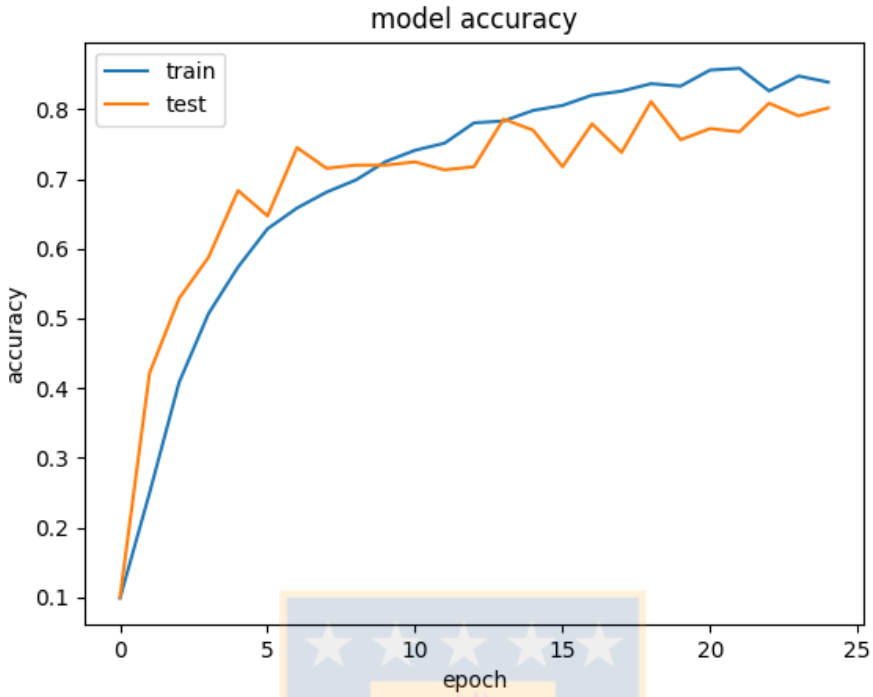

Figura 28: Gráfico accuracy MiniVGG-7

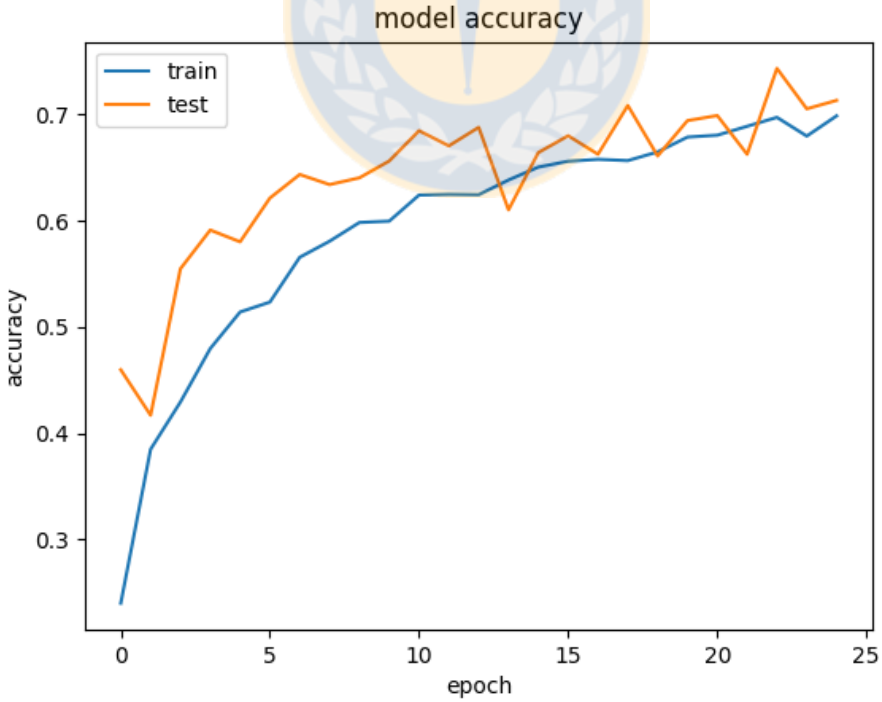

Figura 29: Gráfico accuracy Red Básica