UNIVERSIDAD DE CHILE
FACULTAD DE CIENCIAS FÍSICAS Y MATEMÁTICAS
DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

LOCALIZACIÓN DE OBJETOS GUIADA POR BOSQUEJOS

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO

JOAQUÍN EDUARDO CURIMIL CAMPOS

PROFESOR GUÍA:
JOSÉ SAAVEDRA RONDO

MIEMBROS DE LA COMISIÓN:
JUAN BARRIOS NÚÑEZ
IVÁN SIPIRÁN MENDOZA

SANTIAGO DE CHILE
2023

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO
POR: JOAQUÍN EDUARDO CURIMIL CAMPOS
FECHA: 2023
PROF. GUÍA: JOSÉ SAAVEDRA RONDO

LOCALIZACIÓN DE OBJETOS GUIADA POR BOSQUEJOS

Dentro del área de visión por computadora, la detección de objetos a tomado bastante fuerza últimamente, por lo que se plantea un problema poco explorado, como es la localización de objetos mediante bosquejos, buscando buenos resultados tanto para clases vistas como no vistas.

Dentro de las soluciones encontradas se usan modelos que se enfocan en el *few-shot detection*, la cuales modifican la estructura de la *faster r-cnn*, cambiando algunos de sus módulos para mejorar sus resultados, principalmente por mecanismos de atención, y estos mismos modelos son modificados para que también trabajen con bosquejos.

A los modelos iniciales se le aplican diferentes tipos de modificaciones, ya sea dentro de su estructura como a su pre-procesamiento de imágenes, buscando de esta forma la solución que de mejores resultados para bosquejos, ya que no fueron considerados en el uso inicial de estos modelos.

Los resultados del estado de arte (sota) entregan un mAP (AP50) de *0.65* para las clases vistas dentro del conjunto de datos de PASCAL-VOC, mientras que el mejor resultado obtenido dentro de este trabajo para este mismo conjunto es de *0.752*. Y para los resultados de clases no vistas no se pudo hacer una comparación directa, ya que los resultados de sota para el conjunto de datos con el que se trabajo, solamente son revisados para clases vistas, mientras que usa el conjunto de datos COCO para los resultados de clases no vistas, teniendo un mAP de *0.150*, mientras que el modelo obtenido con mejores resultados dentro de PASCAL-VOC solo llegan a *0.131*, siendo una gran diferencia al ser más complicado el conjunto de COCO.

Los modelos con *transformers* entregan pésimos resultados para los dos casos de revisión, pero esto se le atribuye al no tener su estructura completa, por lo que se intuye que deberían superar a los modelos obtenidos con su estructura terminada.

Agradecimientos

Mis principales agradecimientos van dirigidos a mi familia, tanto mi padre, mi madre como mi hermano, que desde mis bases han tenido una importante influencia en mi persona, ya sea buenas o malas, pero sus presencias es lo que me permite y permitió seguir hacia adelante, por lo que puedo decir sin duda alguna que la persona que fui y seré es gracias a ellos.

También aprovechar de agradecer a mi profesor José Saavedra y mi compañero Cristóbal, los cuales solucionaron bastantes dudas que tenía en el transcurso de la escritura de este informe, y me aportaron todas las herramientas que necesite para la realización de este trabajo. Y Finalmente agradecer los comentarios del profesor Juan Barrios, que me ayudaron a mejorar la estructura y redacción de este informe.

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

Capítulo 1

Introducción

Una de las áreas que tomara importancia dentro de este trabajo, es la detección automática de objetos en imagen, donde ya no es necesaria la presencia de una persona para identificar el contenido de una imagen, si no que una computadora será la encargada de esto.

Dentro del área de detección se puede encontrar diferentes tipos de problemas, ya sea localización de todo objeto presente dentro de la imagen, solo los que sean parecido a cierto objeto, segmentación de estos objetos, movimientos de los objetos, etc. Pero el problema al que se le hará hincapié dentro de este trabajo es la detección dentro de una imagen de todos los objetos que contenga ciertas características, o siendo más específicos, que compartan el mismo grupo, ya sea que se quiera buscar todos los gatos dentro de una imagen, o las sillas, lo sillones, etc. Estos objetos serán encontrados mediante un consulta, la cual debe poseer una cierta cantidad de características semejantes al objeto buscado, y al ser lo suficientemente parecidas, se generará un emparejamiento entre el objeto buscado y la consulta, por lo que se entenderá que se hace referencia a lo mismo.

La particularidad dentro de este trabajo es el tipo de consulta que se realizará, ya que generalmente se usa una imagen de cierto objeto para buscar todos sus parecidos dentro de otra imagen, ya que se usarán bosquejos para buscar objetos dentro de la imagen, por lo que será necesario encontrar una forma de poder relacionar los *sketchs* con el objeto al que se le hace referencia.

También se enfocará en los resultados de los objetos que no pertenezcan a la fase de entrenamiento, pertenecientes a clases no vistas, y con solo un ejemplo para poder detectar el objeto deseado.

1.1. Motivación

Si un objeto es extraviado y debe ser encontrado de manera rápida, lo más lógico de hacer es buscarlo en grupo con la mayor cantidad de personas, pero para que esto sea exitoso, se necesita que cada integrante tenga una concepción de lo que se está buscando, por lo que toda referencia del objeto perdido mejoraría las chances de encontrarlo, siendo una imagen la referencia perfecta, pero desgraciadamente no siempre se va a tener una disponible. Aquí es donde entra el bosquejo para reemplazar a la imagen, siendo un garabato que contiene todas las características importantes para reconocer el objeto al que se le hace la referencia, por lo que sería uno de los mejores reemplazos de una imagen dentro de toda tarea de búsqueda. Y lo mismo ocurre en los modelos de detección, que generalmente siempre ocupan como referencia imágenes reales del objeto, las cuales no siempre podrían estar disponibles.

Si se lleva al siguiente nivel esta idea de detección de objetos mediante bosquejo, se podría llegar incluso a tareas de detección al nivel que ocupa la policía, donde al solo tener un bosquejo de un criminal, se logra la detención de este. En la figura 1.1 se puede ver como usando solamente un bosquejo, se podría encontrar el objeto deseado dentro de la imagen.

Figura 1.1: Detección mediante bosquejo de objeto.

1.2. Definición y relevancia del problema

Para la obtención de un bosquejo se requiere la existencia de un objeto a representar, rescatando las características más llamativas de este, y de esta forma, lograr una simplificación de la realidad. Pero, dentro de este trabajo, se busca invertir este flujo, al obtener el objeto mediante el bosquejo, y no al revés. Esta tarea se verá reflejada en la detección de objetos mediante bosquejo, al obtener la posición del objeto de interés mediante un bosquejo.

Como se había mencionado antes, usar bosquejos es la mejor solución al necesitar un reemplazo rápido de una imagen, siendo vital en la ausencia de estas, pero debido al poco uso que se les da para problemas de detección, hay una cantidad limitada de resultados semejantes con los que comparar, al ser un problema poco visto y reciente. También se le dará importancia a la clase del objeto a buscar, ya sea una perteneciente al conjunto de entrenamiento, como no, que dentro de este informe serán conocidos como clases vistas y no vistas, respectivamente.

1.3. Objetivos

Los objetivos se separan en dos partes, siendo el primero el general, donde se da una idea a grandes medidas del problema que se desea solucionar, mientras que los objetivos específicos se centrarán más en todas las cualidades que deba poseer la solución buscada.

1.3.1. Objetivo General

El objetivo general es la obtención de la posición exacta dentro de una imagen, de todos los objetos que fueron representados mediante un bosquejo de referencia.

1.3.2. Objetivos Específicos

Se tendrán 3 objetivos importantes dentro de este trabajo:

1. El modelo por obtener debe entregar buenos resultados con toda entrada, ya sea de clases vistas como no vistas.
2. Superar o igualar puntuaciones mAP de modelos pertenecientes al estado del arte.
3. Que todas las soluciones que se realicen contengan un mecanismo de atención, debido a ser el que entrega mejores resultados en problemas de este tipo.

Capítulo 2

Marco Teórico

A lo largo de este trabajo se verán varios términos que serán usados y deberán ser entendidos, y bien definidos, para evitar confusiones en capítulos futuros. Los términos por ver en es este capítulo pasaran de ser de los más sencillos, pero importantes de especificar, hasta los más específicos dentro del área de inteligencia computacional.

2.1. Definiciones básicas

Al trabajar con términos tan amplios, es necesario delimitar el significado de algunos conceptos, especificando ciertos rasgos importantes que deben tener para ser considerados como tal. Y de igual manera, si no cumple con algunas de las especificaciones que definan al termino, se considera que no pertenece a este.

2.1.1. Imagen real (Foto)

Se considerará como *imagen real*, o *foto*, a la obtención de una representación visual de la realidad, que pueda ser guardada de forma digital y a color (RGB o blanco y negro). Teniendo como principal diferencia al concepto más general que todos conocen, en que al menos uno de los objetos que aparezcan dentro de esta definición de imagen, debe ser lo suficientemente detallado para poder ser fácilmente identificables al objeto que representa en la realidad, para cualquier observador, como en el ejemplo incluido en la figura 2.1.a. Por lo que. si dentro de una imagen no se puede reconocer ninguno de los objetos que aparecen, no se considerara como una imagen válida, como en la figura 2.1.b. Se presenta una comparación visual de ambas situaciones mencionadas en la figura 2.1.

Otra importante característica que debe tener es la presencia de un objeto dentro de la imagen, ya que tanto el suelo, el océano como el cielo no se consideran como un objeto dentro de este trabajo, debido a su gran tamaño, por lo que una imagen de un paisaje, sin ningún objeto, ya sean flores, animales, vehículos, edificaciones, etc, no será considerado una imagen. En la figura 2.2 se muestran dos casos, siendo la figura izquierda un ejemplo de presencia de objetos, mientras que la derecha es un ejemplo de ausencia de estos, a pesar de que haya nubes presentes, pero como se ha mencionado anteriormente, debido a sus dimensiones, no es considerado como tal.

(a) Objetos nítidos (Se considera imagen)

(b) Objetos borrosos (No se considera imagen)

Figura 2.1: Ejemplo de objetos borrosos.

(a) Presenta objetos (Se considera imagen)

(b) No presenta objetos (No se considera imagen)

Figura 2.2: Ejemplo de presencia de objetos.

2.1.2. Bosquejo (Sketch)

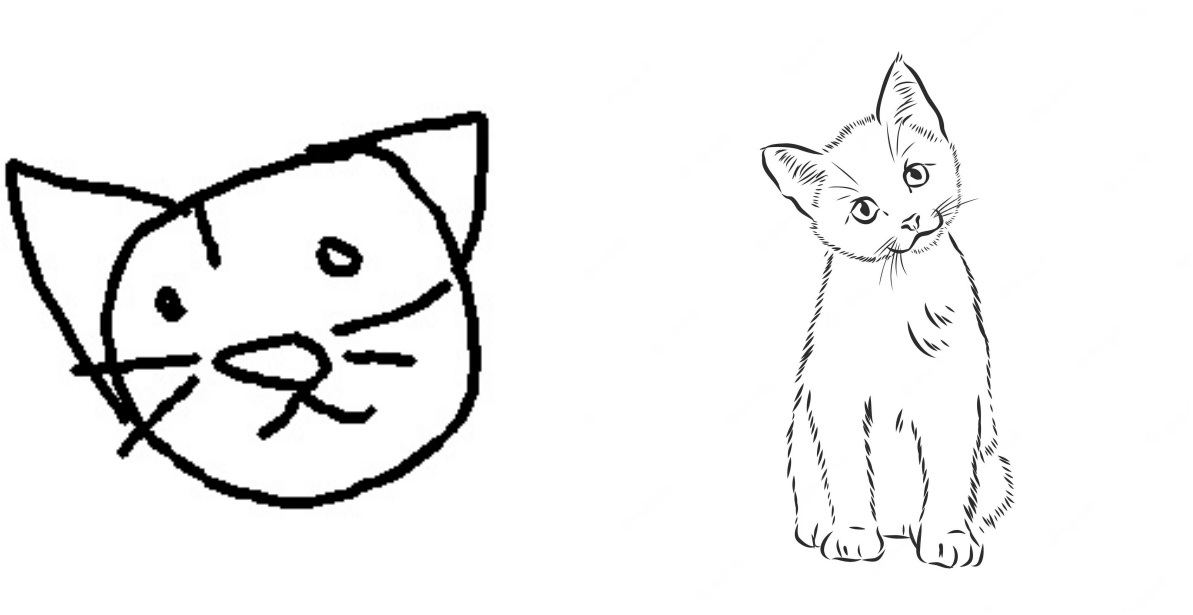
La definición de bosquejo según la RAE es: "*Diseño o proyecto de una obra artística, hecho de manera provisoria, solamente con los elementos esenciales.*", por lo que todo dibujo hecho solo con líneas guías de contorno puede ser considerado como un bosquejo, ya sean líneas sencillas, que den una representación básica del concepto a conseguir, como puede llegar a tener varios trazos que representen de manera detallada y exacta lo deseado a obtener. En la figura 2.3 se puede observar la diferenciación entre ambos casos.

Dentro de este trabajo solo se considerará como un bosquejo válido a su versión simplificada, que deberá cumplir con los siguientes puntos:

1. Tener la menor cantidad de trazos.
2. Contener un único objeto dentro del bosquejo.
3. No tener color aparte del negro y el blanco.

4. Contener las partes necesarias del objeto a representar, para saber a qué se hace referencia dentro del bosquejo, siendo para un gato sus características orejas, colas y cara, por lo que cualquier persona puede reconocer que se está dibujando a un gato.

5. No tener demasiados detalles innecesarios dentro del dibujo, como en el caso del gato puede ser su pelaje, la exactitud de sus patas y la exactitud de su posición estructural.



(a) Bosquejo basico (b) Bosquejo detallado

Figura 2.3: Diferentes tipos de bosquejos.

2.1.3. Clases

Se conocerá como **Clase** al conjunto de diferentes instancias que hacen referencia al mismo objeto, siendo siempre una cantidad limitada de clases la que pueden aparecer dentro de una imagen.

2.1.4. Emparejamiento (*Match*)

Al juntar dos, o más, instancias de una misma clase, se va a considerar como un **emparejamiento**. Teniendo la peculiaridad de que cada instancia puede ser de diferente tipo, con tal que hagan referencia al mismo concepto. Para el desarrollo de este trabajo, siempre se emparejarán una foto junto a un bosquejo, teniendo como única restricción la presencia de la misma clase en ambas situaciones.

2.1.5. Búsqueda

Se limitará la definición de **búsqueda** a la exploración de un conjunto de imágenes (o bosquejos), con la intención de encontrar un elemento de estos que cumpla con ciertos requisitos, siendo en este caso, la presencia de una clase. Esta acción se realizará principalmente para la obtención de una imagen y un bosquejo de la misma clase, y de esta forma, lograr un emparejamiento, pero también habrá ocasiones en las que se busca encontrar la presencia

de una clase dentro de una imagen, la cual podría contener la clase deseada, como no.

2.1.6. Imagen objetivo (*Target*)

Toda vez que se mencione una **imagen objetivo**, o ***target***, se hará referencia a la imagen donde se desea realizar una búsqueda especifica de alguna, o varias, instancias de una única clase. La imagen en cuestión puede contener varias clases distintas de la deseada, e incluso puede no poseer ninguna instancia de la clase buscada.

2.1.7. Entrada de consulta (*Query*)

La **entrada de consulta**, o ***query***, se complementa con la definición de la imagen objetivo, ya que esta sería la foto, o bosquejo, que contiene a la clase que se desea buscar dentro del *target*. Siendo su principal característica la presencia de una sola instancia de una única clase, por lo que en caso de tener dos *Querys*, se deberá realizar dos distintas búsquedas en el *target*.

2.2. Conjunto de datos

Todo trabajo que desea entrenar un algoritmo necesita de una gran cantidad de datos, para que el modelo, mediante iteraciones, pueda aprender los patrones de esta información, y de esta forma, modificar sus parámetros para aprender a realizar una tarea determinada, por lo que, dependiendo del objetivo que se quiere obtener, la naturaleza de esta información puede variar, haciendo que los datos con los que se desean trabajar puedan ser palabras, sonidos, letras, imágenes, etc.

Debido a la naturaleza del problema que se busca solucionar, se van a necesitar dos grandes conjuntos de datos, siendo el primero perteneciente a imágenes reales, que contengan información de diferentes objetos dentro de una misma imagen, y esta información debe contener la clase a la que pertenece el objeto, y su localización precisa dentro de la imagen, ya que estos datos son necesarios para lograr que la red aprenda los patrones necesarios para identificar el objeto buscado dentro de una imagen. Y el segundo conjunto de datos corresponde a los bosquejos de diferentes clases, que contengan solamente información de la imagen y de la clase a la que pertenecen, y de esta forma generar una relación con la clase que se desea encontrar dentro de la imagen real.

Las bases de datos usadas para poder cumplir con los requisitos anteriormente mencionados, serán la famosa PASCAL-VOC (*Visual Object Classes*) usada en varios papers al tener una gran cantidad de imágenes, y cada una de estas con su información bien detallada y ordenada. Será usada para la parte de localización de los diferentes objetos dentro de una imagen, mientras que la base de datos *Quick, Draw!* de *Google* nos entrega una gran cantidad de bosquejos de diferentes clases, por lo que solo se usarán las clases que también aparezcan en la otra base de datos.

Se procederá a detallar las dos bases de datos que se usarán, las clases que cada una contiene, la cantidad de datos que poseen, la forma en que se entrega la información y los datos que serán usados dentro de este trabajo:

2.2.1. PASCAL-VOC

Comenzando con los significados de su nombre, las siglas de PASCAL provienen de su origen en ingles de *Pattern Analysis, Statistical Modelling, and Computational Learning*, que en español podría traducirse a *Analisis de Patrones, Modelo estadístico y aprendizaje computacional*.

PASCAL es una Red de Excelencia financiada por la Unión Europea. Ha establecido un instituto distribuido que reúne a investigadores y estudiantes de toda Europa, llegando actualmente a nuevos países de todo el mundo. PASCAL está desarrollando la experiencia y los resultados científicos que ayudaran a crear nuevas tecnologías, como interfaces inteligentes y sistemas cognitivos adaptativos. Para lograr esto, apoya y fomenta la colaboración entre expertos en *machine learning*, estadísticas y optimización. También promueve el uso del *machine learning* en muchos dominios de aplicaciones relevantes, como la visión artificial, habla, háptica, interfaz cerebro-computadora, modelado de usuario para interacción computadora-humano, integración multimodal, procesamiento de lenguaje natural, recuperación de información y acceso a información textual.

El objetivo de PASCAL es construir un Instituto Distribuido en toda Europa que sea pionero en métodos basados en principios de análisis de patrones, modelado estadístico y aprendizaje computacional como tecnologías habilitadoras centrales para interfaces multimodales que sean capaces de una interacción natural y fluida con, y entre, usuarios humanos individuales.[23]

Figura 2.4: Logo de PASCAL.

PASCAL realizó una serie de desafíos para la detección de objetos entre 2005 y 2012 siguiendo una estructura de archivos estandarizada para almacenar estas anotaciones de imágenes. El desafío PASCAL Visual Object Classes (VOC) tenía dos componentes principales:

1. Un conjunto de datos disponible públicamente con un software de evaluación estandarizado
2. Un concurso anual y un taller.

Los objetivos principales de este desafío eran averiguar la capacidad de los modelos para realizar:

- **Clasificación** - Comprobar si un objeto forma parte de la imagen.
- **Detección** - Localiza la posición de los objetos presentes en la imagen.

Esta serie de desafíos llegó a su fin en 2012 con mejoras importantes y mejoras en el conjunto de datos. PASCAL-VOC proporciona conjuntos de datos de imágenes estandarizados para más de 20 clases diferentes que se usan comúnmente para tareas como detección de objetos, segmentación semántica y otras tareas de clasificación.[24]

Entrando a detalle en el conjunto de datos de imágenes que proporciona PASCAL-VOC, se verá cuáles son las 20 diferentes clases que posee, las cuales se pueden clasificar de la siguiente manera:

- ***Person****: Person*
- ***Animal****: bird, cat, cow, dog, horse, sheep*
- ***Vehicle****: aeroplane, bicycle, boat, bus, car, motorbike, train*
- ***Indoor****: bottle, chair, dining table, potted plant, television, sofa*

Aunque en el paper *The PASCAL Visual Object Classes (VOC) Challenge* donde se explica a detalle cómo funciona este desafío, nos presenta de mejor manera las clases presentes mediante un diagrama, el cual puede ser observado en la figura 2.5.

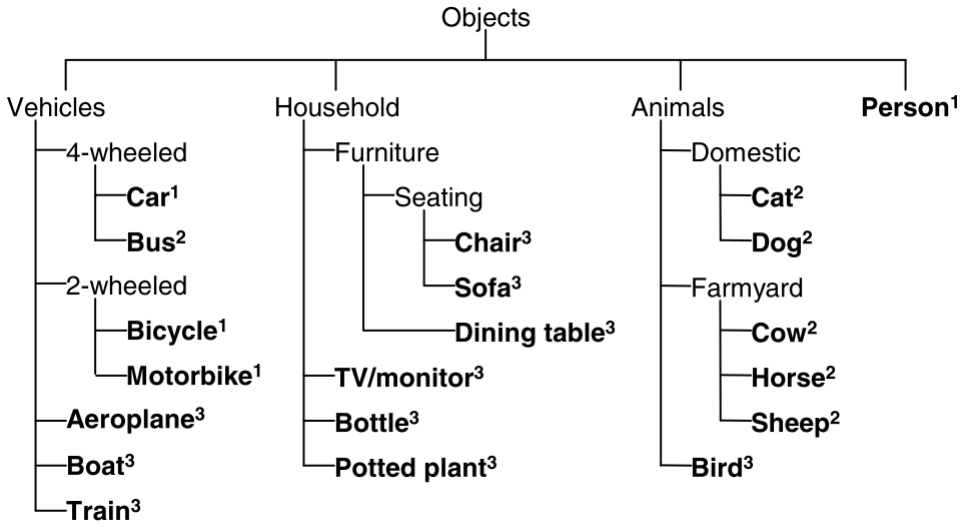


Figura 2.5: Clases del conjunto de datos de PASCAL-VOC.

Debido a la importancia que tiene este conjunto de datos en lo desarrollado, se verá a detalle que datos posee, como esta información puede ser usada para la solución del problema, y como están organizadas dentro de cada carpeta, siendo esta última de la siguiente forma:

- **Annotations**
- **ImageSets**
  - **Layout**
  - **Main**
  - **Segmentation**

- **JPEGImages**
- **SegmentationClass**
- **SegmentationObject**

En las siguientes secciones se verá que tipo de información posee cada carpeta, para finalmente elegir solo la información de algunas de estas, ya que no se necesita la totalidad de los archivos para realizar el objetivo principal de este trabajo.

2.2.1.1. División de los datos

Antes de analizar cada carpeta, es importante mencionar como se obtuvo el conjunto de datos final, siendo necesario para esto, la descargas de tres archivos diferentes, siendo los dos primeros pertenecientes a los datos que servirán para el entrenamiento del modelo, siendo obtenidos mediante la página oficial de PASCAL, de sus *datasets* enfocados en entrenamiento y validación de los concursos de 2007 y 2012, en sus respectivos links: VOCtrainval2007 y VOCtrainval2012, mientras que el conjunto de prueba será obtenido del *dataset* de *test* del concurso de 2007, mediante el link: VOCtest2007.

Una vez conseguido los datos necesarios para el entrenamiento, validación y testeo, es necesario verificar la similitud entre los diferentes archivos. Principalmente para no tener problemas al momento de ingresar estos datos al código, pero verificando la información de cada archivo descargado, se pudo confirmar que todo los *datasets* presentaban la misma distribución en sus carpetas, con la diferencia de que cada una tenían diferentes imágenes, y con la semejanza de que había exactamente el mismo tipo de información para cada imagen en los diferentes archivos, los cuales serán vistos en los siguientes puntos.

2.2.1.2. *Annotations*

La primera carpeta que cada conjunto de datos posee, es la de *Annotations*, donde se encuentran archivos *xml* de cada imagen, teniendo incluso los mismos nombres de la imagen a la que le hace referencia, pero con la única diferencia de que su extensión en vez de ser *jpg*, son del tipo *xml*.

Cada archivo *xml* posee el mismo tipo de información, con la diferencia del contenido de esta, ya que cada imagen contiene diferentes cantidades y tipos de objetos. Por lo que mediante este archivo se puede conocer todo lo importante que hay dentro de la imagen, como puede ser la clase de los objetos dentro de esta, sus ubicaciones, el medio de donde se obtuvo, el dueño de la fotografía, y si presenta cierto tipo de datos disponibles.

En el fragmento de código 2.1, se puede observar un ejemplo de este tipo de archivo, donde se puede observar todo lo antes mencionado:

- Teniendo en la línea 3 el nombre de la imagen a la que se hace referencia, teniendo solo diferente su tipo de archivo.
- Desde la línea 4 a la 9 se puede observar la fuente de donde se obtuvo la imagen.
- Desde la línea 10 a la 13 se puede conocer el dueño de la imagen

- de la línea 14 a la 18 se entrega el tamaño de la imagen
- La línea 19 avisa sobre la disponibilidad de la información segmentada de la imagen, siendo en este caso un cero, y, por ende, negativa la presencia de esta.
- Desde la línea 20 hasta la 31 se presenta la información más importante dentro de este trabajo, siendo esta la clase y localización de los objetos dentro de la imagen, específicamente para este caso solo un objeto de clase *dog*, el cual se encuentra dentro de un rectángulo, también conocido como *bounding box*, de coordenadas: A:(56,63) y B:(284,290), siendo el primer punto (A) el mínimo del rectángulo, y el segundo (B) el máximo, comenzando desde el origen superior izquierdo.

Debido a la importancia que tiene dentro de este trabajo la localización y clase del objeto dentro de la imagen, se va a hacer énfasis en la última parte de los datos obtenidos del ejemplo del archivo *xml*, perteneciente al fragmento de código 2.1, donde rescatando el *bounding box* y la clase del objeto, se puede ubicar de forma concisa dentro de la imagen al objeto etiquetado, como se puede ver en la figura 2.6, especificando de forma clara el punto mínimo (A) y máximo (B) del rectángulo anteriormente mencionado.

Es necesario mencionar que en este ejemplo solo se tenía un objeto dentro de la imagen, pero esto no es una restricción, ya que existen los casos en que poseen más de un objeto, incluso llegando a tener múltiples objetos de diferentes clases. Pero a pesar de esto, siempre se mantiene la misma estructura, pudiendo ser más largo el archivo dependiendo de la cantidad de objetos clasificados que haya dentro de la imagen

Código 2.1: Contenido del archivo 000029.xml

```
<annotation>
  <folder>VOC2007</folder>
  <filename>000029.jpg</filename>
  <source>
    <database>The VOC2007 Database</database>
    <annotation>PASCAL VOC2007</annotation>
    <image>flickr</image>
    <flickrid>263839604</flickrid>
  </source>
  <owner>
    <flickrid>Andy(Shotage)</flickrid>
    <name>Andy Newson</name>
  </owner>
  <size>
    <width>500</width>
    <height>331</height>
    <depth>3</depth>
  </size>
  <segmented>0</segmented>
  <object>
    <name>dog</name>
    <pose>Unspecified</pose>
    <truncated>0</truncated>
    <difficult>0</difficult>
    <bndbox>
```

```
      <xmin>56</xmin>
      <ymin>63</ymin>
      <xmax>284</xmax>
      <ymax>290</ymax>
    </bndbox>
  </object>
</annotation>
```

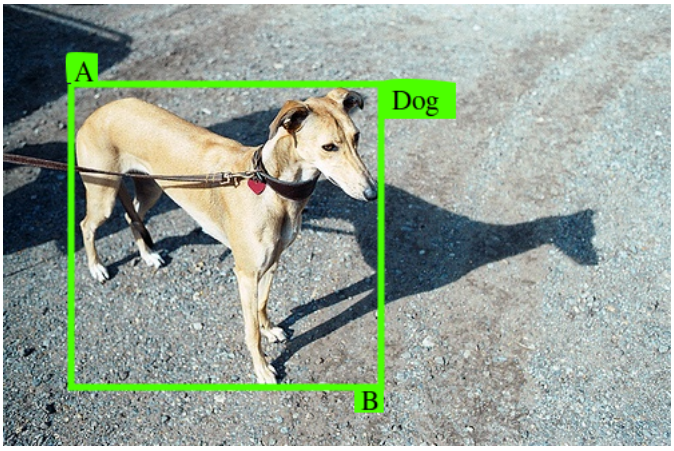


Figura 2.6: Imagen 000029.jpg con su *bounding box* obtenida del archivo *xml*.

2.2.1.3. *ImageSets*

Esta carpeta contiene diferente tipo de información sobre cada imagen, por lo que es necesario separarlas en tres sub-carpetas diferentes, donde todas tendrán archivos del tipo *txt*, que contiene un listado de nombres de imágenes, pero siendo todos diferentes y tratados de forma específica.

La primera sub-carpeta llamada *Layout* contiene un listado del nombre de cada imagen que pertenezca al conjunto de dato correspondiente (trainval o test), y esto es debido a que en ocasiones puede haber dos *datasets* diferentes mezclados dentro de una misma carpeta, como ocurre con los conjuntos de los mismos años, y en esta ocasión, con los archivos *VOCtrainval2007* y *VOCtest2007*, por lo que para la carpeta que contiene los archivos VOC del año 2007, contendrá dos listados semejantes, uno perteneciente para el conjunto de entrenamiento y validación, y otro para el conjunto de testeo, mientras que para la carpeta de los archivos VOC del 2012 solo contendrá un listado de los conjunto de entrenamiento y validación.

La segunda sub-carpeta se llama *Main*, y contiene un listado por cada clase que existe (20 clases en total) y el tipo de conjunto al que pertenece, siendo del estilo *{Clase}_{conjunto}.txt*. Donde cada lista posee los nombres de todas las imágenes disponibles dentro de su conjunto, con la peculiaridad de que cada nombre posee a su costado un dígito, el cual indica la presencia de la clase a la cual se hace referencia dentro del nombre del listado correspondiente. Por ejemplo, trabajando dentro del conjunto de testeo, se quiere saber si la imagen *000067.jpg* posee un avión dentro de sus objetos, por lo que bastaría revisar el archivo *aeroplane_test.txt* y ver el dígito que acompaña al nombre 000067, si es un 1, hay un avión dentro de la imagen,

y en caso de haber un -1, no hay ningún avión dentro de la imagen.

Finalmente, la última sub-carpeta *segmentation* posee un listado de todas las imágenes del conjunto correspondiente que contengan información sobre su segmentación. Ya que como se había visto en el fragmento de código 2.1, no todas las imágenes poseen la presencia de este tipo de información, la cual será explicada más adelante a detalle.

2.2.1.4. JPEGImages

Esta carpeta contiene los datos más importantes del conjunto de datos, siendo estos las imágenes con las que se trabajaran (fotos de diferente tipo), solo teniendo archivos tipo *jpg*, ya que toda la otra información necesaria, ya sea, clases de los objetos, localizaciones, tamaños, etc, están siendo almacenadas en todas las carpetas anteriormente explicadas.

Como se puede ver en la figura 2.7, todos los elementos pertenecientes a los *datasets* de imágenes son diferentes en varios aspectos, ya pueda ser en su tamaño, contenido, resolución, lugar u objetos. Pero todas estas imágenes cumplen con las definiciones básicas impuestas en la sección 2.1, por lo que se puede tener la seguridad que siempre aparecerá como mínimo un objeto de alguna de las 20 clases antes mencionadas, fácilmente identificable por el ojo humano y fáciles de trabajar.

Figura 2.7: Ejemplos de las imágenes del conjunto de datos.

2.2.1.5. Segmentación de clases

Esta carpeta entrega información de la localización exacta de diferentes clases dentro de una imagen, y debido a la naturaleza especifica que se necesita de los datos que deben ser entregados, tiene la peculiaridad de ser un archivo *jpg* a diferencia de las otras carpetas que solo se limitaban a entregar información mediante texto.

El tipo de archivo que contiene esta carpeta es casi igual al que aparece en la de *JPEGImages*, pero en vez de contener la totalidad de la imagen, solo posee una máscara en la que se diferencia cada clase con un color diferente. Por lo que todo objeto que comparta la misma clase será considerado como una misma entidad.

Lo anteriormente mencionado puede ser mejor entendido mediante la figura 2.8, donde se pueden diferenciar tres sub-figuras, siendo la primera (2.8.a) la imagen original perteneciente a la carpeta *JPEGImage*, mientras que la segunda (2.8.b) se puede ver la segmentación de clases mediante una imagen con cuatro colores diferentes, siendo el rosado correspondiente a todos los objetos que pertenecen a la clase *persona*, en morado a todos los objetos correspondientes a la clase *caballo*, en rojo puede verse una única instancia del objeto perteneciente a la clase *silla*, y en negro esta todo lo correspondiente al *background*, o mejor dicho, a todo pixel que no esté relacionado con ninguna clase.

2.2.1.6. Segmentación de objetos

Esta carpeta es semejante a la de la sección anterior, pero con la diferencia que no se separa a la imagen por clases, si no por objetos, por lo que, en caso de tener dos objetos de la misma clase, sera segmentados como dos entidades diferentes.

En la sub-figura 2.8.c se puede ver la segmentación de objetos de la imagen de la figura 2.8.a, siendo semejante a lo obtenido de la segmentación de clases, pero teniendo la gran diferencia de que cada objeto identificable dentro de la imagen es marcado con un color diferente, y por ende, diferenciándose entre ellos.

Es importante mencionar que la diferencia entre la segmentación de objetos y la de clases solo se puede notar cuando hay más de un objeto de una clase por imagen, ya que que hay casos, como en lo observado en la figura 2.9, que al solo tener un objeto cada clase, ambas segmentaciones son idénticas e indiferenciables.

Para los alcances de este trabajo, no se usarán estos archivos, debido al tipo de problema que se desea resolver, siendo suficiente y necesario los datos mostrados en la figura 2.6, siendo estos la clase del objeto, y su ubicación dentro de la imagen.

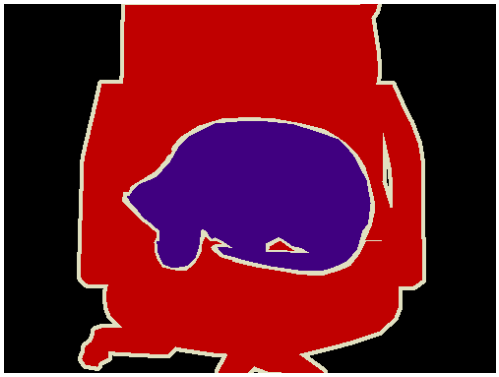
(a) Imagen Original

(b) Segmentación de Clases

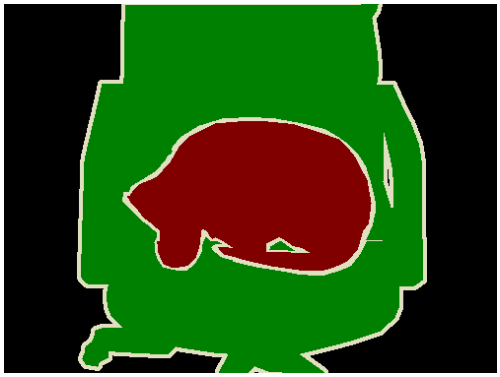
(c) Segmentación de Objetos

Figura 2.8: Segmentación de clases y objetos de la imagen 003889.jpg.

(a) Imagen Original



(b) Segmentación de Clases



(c) Segmentación de Objetos

Figura 2.9: Segmentación de clases y objetos de la imagen 000063.jpg.

2.2.2. Quick, Draw!

El otro conjunto de datos necesario para el desarrollo de este trabajo es el obtenido de la página web: Quick, Draw!. Esta página tiene como principal objetivo el entrenamiento de una red neuronal que reconozca bosquejos, y esto lo hace mediante un juego, el cual consiste en que un usuario debe dibujar un objeto en menos de 20 segundos, y la red neuronal tiene que adivinar lo que se está dibujando dentro de este periodo de tiempo.

Esta red neuronal comparte algunas de las mismas tecnologías que usa el *Handwriting Recognition* de *Google Translation*, aplicación que se usa para poder escribir a mano en la pantalla táctil de lo que se desea ingresar. El sistema de reconocimiento de esta aplicación no se centra solamente en la letra que se dibuja, sino también como esta siendo dibujado, fijándose en las primeras lineas que se hicieron y sus direcciones, por lo que cuanto más se usa esta aplicación de Google, más personalizada se hace para cada usuario, pudiendo llegar a reconocer más fácilmente lo que se escribe en la pantalla con el paso del tiempo. Por lo que, usando este mismo principio dentro de bosquejos, se obtiene el conjunto de datos de *Quick, Draw!*, siendo que cuanto más se use, aparte de aumentar su tamaño, mejora la red neuronal de identificación de bosquejos. [25]

A diferencia del conjunto de datos original, en el cual se guarda la información de los trazos de cada bosquejo, se usará una versión donde se almacena la totalidad del dibujo del *sketch*, por lo que realmente se estará trabajando con el mismo tipo de archivos que las imágenes de PASCAL-VOC, siendo estos archivos tipo *jpg*. Aunque a diferencia de los conjunto de datos de PASCAL-VOC, este conjunto contiene menos cantidad de información por imagen, siendo los únicos datos entregados el bosquejo del objeto y la clase a la que pertenece.

Otras características importantes de este conjunto de datos es la gran cantidad de clases e imágenes que posee si se compara con PASCAL-VOC, teniendo aproximadamente 325 clases, y llegando a los superar las 100.000 de elementos en algunas de estas, siendo todos del mismo tamaño. Pero por desgracia no se incluye la totalidad de clases deseadas, ya que de las 20 clases que posee el otro conjunto, solo 17 se repiten en este, por lo que solamente se trabajará con las clases que pertenezcan a ambos conjuntos, siendo estas:

- ~~***Person**: Person*~~
- ***Animal**: bird, cat, cow, dog, horse, sheep*
- ***Vehicle**: aeroplane, bicycle, boat, bus, car, motorbike, train*
- ***Indoor**: ~~bottle~~, chair, dining table, ~~potted plant~~, television, sofa*

Las clases de *person*, *potted plant* y *bottle* fueron eliminadas por no pertenecer a este conjunto de datos, pero todas las demás si aparecían con algunos pequeños cambios, siendo el más notorio el de *boat*, ya que esta clase tampoco aparecía en este conjunto, pero si su simil de *sail boat*, o que en español sería bote de vela, por lo que en este único caso se especificó la clase de botes a solo los botes de vela.

Finalmente, uno de los más importantes rasgos de este *dataset*, es la información con la que se está trabajando, siendo estos bosquejos, que pueden variar significativamente depen-

diendo de la persona que lo realice. Esto puede ser observado de mejor manera en la figura 2.10, donde al dibujar a un gato en un *sketch*, el resultado final puede variar demasiado, al centrarse en diferentes rasgos y no agregando otros. Pero debido a la gran cantidad de ejemplos que posee, se consigue una buena generalización del objeto representado.

Figura 2.10: Ejemplos de los bosquejos de la clase gato.

Otro problema de trabajar con bosquejos es la similitud que pueden llegar a tener, ya que dos personas diferentes pueden dibujar de manera similar a objetos de diferentes clases, como se puede observar en la figura 2.11, donde la moto y la bicicleta son casi idénticas.

El principal problema de tener similitud entre clases, generalmente hablando, es la pérdida de especificación de estas, pudiendo llegar a no tener diferencias si estos elementos son tratados en sus respectivas clases o llegarlos a unir en solo una. Lo cual podría traer problemas a futuro y cierto *bias* en algunos resultados.

(a) Moto (b) Bicicleta

Figura 2.11: Similitud entre bosquejos de diferentes clases.

2.3. *Inteligencia Artificial*

La inteligencia artificial es una de las áreas que más interés ha generado este ultimo tiempo, debido a sus buenos resultados y el alcance que puede llegar a tomar en diferentes áreas, pero para poder adentrarse en este tema, es necesario tener una noción a lo que se refiere con este termino, siendo este un campo de la ciencia informática dedicado a la resolución de problemas cognitivos asociados comúnmente a la inteligencia humana, como el aprendizaje, la resolución de problemas y el reconocimiento de patrones [26]. En la figura 2.12, donde se observa como la inteligencia artificial es el grupo más grande que incluye a todos los demás, habiendo más subgrupos dentro de sus elementos.

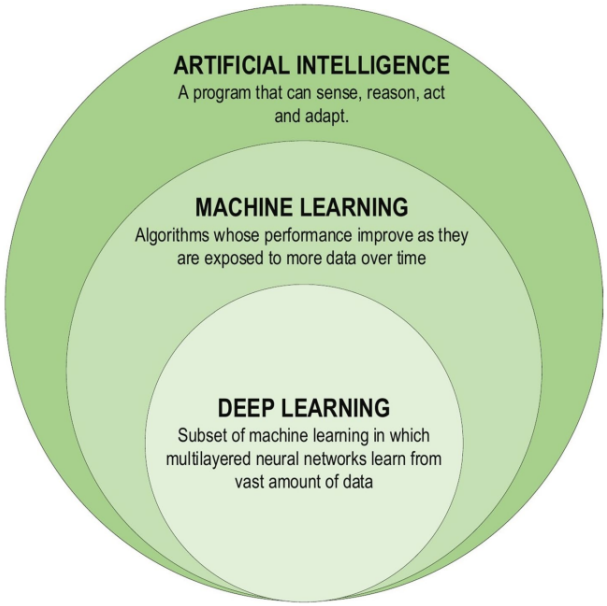


Figura 2.12: Diferentes áreas dentro de la inteligencia artificial [17].

Una de las ramas más conocidas dentro de la inteligencia artificial es el *Machine Learning*, o su traducción a español que sería aprendizaje de maquinas, ciencia de desarrollo de algoritmos y modelos estadísticos que utilizan los sistemas de computación con el fin de llevar a cabo tareas sin instrucciones explícitas, en vez de basarse en patrones e inferencias.

Y una sub-rama del *machine learning* es la actualmente famosa *Deep Learnig*, que hace todo los procesos con una sola estructura. Y una buena manera de ver esto es mediante el ejemplo de la figura 2.13, donde se realiza una tarea de clasificación de imágenes de gatos y perros, pero lo importante es la forma en que cada uno lo realiza, ya que en el primer caso, *machine learning*, se puede observar que las imágenes deben pasar por varios procesos para cumplir la tarea final, mientras que en *deep learning* todo estos pasos son realizados por un solo elemento, que desde este momento se conocerá como **modelo**, el cual recibirá una entrada y entregara un resultado mediante una única estructura, siendo esta la responsable de todo las actividades necesarias para la obtención de objetivo final.

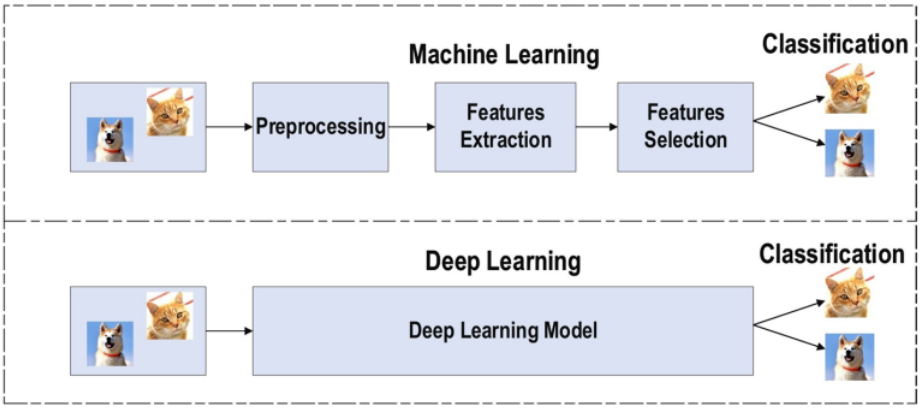


Figura 2.13: Diferentes entre *machine learnig* y *deep learning* [17].

Debido a las necesidades de este trabajo, de detección de objetos mediante bosquejos, se considera al *deep learning* una herramienta fundamental para la resolución de este problema, al tratar de imitar una actividad intrínseca de los humanos, que sería la detección de objetos mediante la vista, y la necesidad de un modelo fácilmente modificable. Pero para realizar esta actividad se usarán modelos de *deep learning*, que, recibiendo una entrada, siendo en este caso una foto, se pueda detectar lo deseado dentro de esta. Es importante mencionar que la parte de aprendizaje dentro de la solución presentada se encuentra dentro de los modelos, ya que estos códigos poseen la habilidad de corregir sus valores mediante un entrenamiento, llegando a mejorar sus resultados cuanto más datos se tengan para aprender.

Si bien se mencionó la detección de objetos, también es necesaria la incorporación de sus clasificaciones, ya que cada objeto observado pertenece a cierto grupo, o como se conocerá dentro de este trabajo, a cierta clase, y al igual que los humanos aprenden a generalizar y diferenciar cada objeto en diferentes clases, lo modelos también tienen que hacerlo, con la particularidad para este caso, de no solamente clasificarlos, si no también relacionarlos con un *sketch* que pertenezca a la misma clase.

Estas dos tareas, de detección y clasificación, mejoran cuanto más datos se posean, ya sea en humanos o en máquinas, por lo que se prioriza tener un conjunto de datos bien amplio y diverso, siendo necesario estas dos características para un aprendizaje correcto. Si solo se tiene una gran cantidad de datos, semejantes entre estos, puede ocasionar que se obtengan malos resultados en un ambiente mínimamente diferente, y si solo se tienen un pequeña cantidad de datos bastante diferentes entre estos, no se lograra un aprendizaje real, al no encontrar una relación entre los elementos que deberían parecerse, al ser estos muy escasos. Por lo que elegir un buen conjunto de datos, también es una parte fundamental en el proceso de aprendizaje.

En el problema propuesto de detección de objetos mediante *sketch*, se pueden identificar tres partes importantes, siendo la primera la obtención de la clase buscada mediante un bosquejo, la segunda parte consistirá en la detección de todos los objetos que pertenezcan a una imagen, para que la última parte, con cada objeto detectado, y la clase obtenida del *sketch*, se puedan detectar cual de todos estos objetos compartan la misma clase que el bosquejo inicial, y de esta forma, poder localizar solo la clase deseada. En la figura 2.14 se puede ob-

servar como funcionara lo anteriormente mencionado, teniendo solo un objeto en la imagen objetivo, y teniendo la misma clase que el *sketch*, siendo todas las actividades anteriormente mencionadas responsabilidad del modelo a elegir.

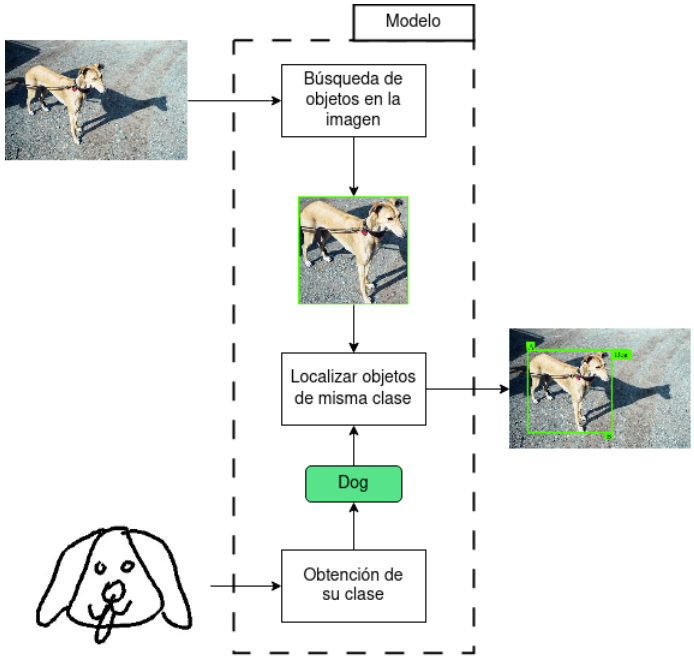


Figura 2.14: Procesos del modelo para este trabajo

La manera de lograr lo anteriormente mencionado, es mediante el uso de diferentes métodos pertenecientes al *machine learning*, ya que las diferentes soluciones mostradas en la sección 5 de resultados, poseen términos importantes de comprender, si se quiere entender lo que se está haciendo. Por lo que se hará un pequeño recorrido por los métodos y términos más importantes dentro de esta área, y los que fueron mayormente usados dentro de las soluciones propuestas.

2.3.1. Separación de datos

Debido a la naturaleza del *machine learning*, se requiere de un periodo de entrenamiento, donde los algoritmos que son usados corrigen sus valores para que la salida sea la esperada. Por lo que siempre se requerirá de una gran cantidad de datos para un mejor resultado, pero no la totalidad de los disponibles, ya que este no es el único proceso que necesita el uso de datos, ya que hay que verificar el funcionamiento de los algoritmos después de su entrenamiento, por lo que los datos que serán usado serán separados en tres conjuntos:

- **Conjunto de entrenamiento**: Este conjunto contiene todos los datos que serán usados en el entrenamiento, teniendo como importancia el tamaño de este grupo, ya que estos corresponden del 60 % al 80 % de la totalidad de los datos.
- **Conjunto de validación**: Este conjunto de datos no siempre es usado, debido a que se usa principalmente para ajustar los parámetros de los algoritmos (o modelos), ya que,

al ser datos externos del entrenamiento, dan una mejor visión de que parámetros dan mejores resultados en un ambiente diferente al de aprendizaje, y por ende, generaliza lo modelos. Naturalmente corresponden del 10 % al 20 % de la totalidad de los datos.

- **Conjunto de testeo**: Este conjunto es usado para verificar que tan bien funciona un modelo en un entorno externo, por lo que se usan principalmente para calificar la funcionalidad del modelo resultante, y de esta forma compararlo con otros. Naturalmente corresponden del 10 % al 20 % de la totalidad de los datos.

Algo importante a tener en cuenta de estos conjuntos, es que ningún dato puede pertenecer a dos de estos al mismo tiempo, por lo que los tres son excluyentes entre ellos.

2.3.2. *One-Shot*

Como se mencionó en el apartado anterior, existe un conjunto de entrenamiento y otro donde se comprueba la eficacia del modelo, pero en la mayoría de los casos estos dos conjuntos comparten las mismas clases, por lo que al final solo se aprende y verifica un tipo de datos específicos. Pero hay un área de inteligencia artificial que tiene como principal objetivo hacer predicciones correctas dado solo un único ejemplo para cada nueva clase no vista [18], esta área es conocida como *one-shot*.

En *One-Shot* los conjuntos de entrenamiento pueden compartir algunas clases del conjunto de testeo, pero este último tiene la necesidad de poseer clases que no aparezcan en el conjunto de entrenamiento, por lo que, al momento de evaluar un modelo, puede verse su efectividad en las clases vistas (pertenecientes al conjunto entrenamiento) como en las clases no vistas (no pertenecientes al conjunto entrenamiento).

Al buscar buenos resultados en datos no vistos, se desea que el modelo tenga una buena generalización en vez de buenos resultados en casos puntuales, por lo que es importante que el modelo sea bueno distinguiendo rasgos pertenecientes a una clase, y de esta forma, entregar buenos resultados con clases no vistas que tengan rasgos similares.

Este apartado es importante dentro de este trabajo, debido a que pertenece a uno de los objetivos principales de este, al querer conseguir buenos resultados en las clases no vistas.

2.3.3. *Feature extraction*

El *Feature extraction*, o su traducción al español, extracción de características, es uno de los campos más importantes de la inteligencia artificial. Consiste en extraer las características más relevantes de una imagen y asignarlas a un *label*. Tomando una importancia notable en el área de clasificación de imágenes, siendo que uno de sus pasos cruciales es el análisis de las propiedades de estas *features*, y organizarlas en clases. [19].

En el área de la clasificación, al utilizar técnicas de *machine learning* se requieran varios pasos, como pueden ser el pre-procesamiento, *Feature extraction*, *wise feature selection*, aprendizaje y clasificación. Mientras que al usar *deep leaning* se puede lograr todos los pasos

anteriormente mencionados en un solo bloque.

Además, la selección de características tiene un gran impacto en el rendimiento de las técnicas de *machine learning*, por lo que una selección de características sesgada puede conducir a una discriminación incorrecta entre clases. Por el contrario, al usar *deep learning* se automatiza el aprendizaje de la extracción de estas características para diferentes tareas, disminuyendo las posibilidades de cometer errores [17].

Si bien se ha mencionado la inclusión de este proceso en una única estructura dentro del *deep learning*, también se presta importancia a la elección de este, ya que ciertas estructuras (redes) usadas entregan mejores resultados dependiendo de la tarea que se realice, pero este punto se verá más adelante.

2.3.4. *Perceptron*

Uno de los más grandes desafíos del hombre ha sido poder entenderse a sí mismo, desde el funcionamiento de sus cuerpos hasta el de sus cerebros, pero en este último punto es donde menos avance se han logrado, debido a la compleja estructura que este posee. Pero esto no significa que no se han hecho avances al respecto, es verdad que aún no se ha llegado a comprender el proceso de este en su totalidad, si se ha llegado a entender sus partes más fundamentales, siendo estas las **neuronas**, las cuales son las encargadas de recibir, procesar y transmitir información.

El objetivo de la inteligencia artificial es lograr tareas humanas mediante computadoras, y que mejor manera de partir que empezando con el responsable de la ejecución de estas tareas, el cerebro, el cual se puede considerar como un gran rompecabezas, con más de mil millones de piezas las cuales no toman un real significado si solo se consideran individualmente, ya que su verdadero valor se encuentra al juntarse con otras piezas, llegando a formar hermosos paisajes de diferentes climas, y todas juntas forman un gran ecosistema. El gran ecosistema sería una referencia al cerebro, los diferentes paisajes son sus diferentes lóbulos encargados del movimiento, visión, audición, tacto, etc. y las piezas son las neuronas, las cuales individualmente no aportan demasiado, pero en conjunto pueden lograr importantes tareas.

La estructura de la neurona es importante a conocer, ya que de esta manera se entenderá cómo funcionan individualmente y como se comunican entre ellas, y al tratar de replicarlas, se puede llegar a imitar estas estructuras mediante código de computadora. En la figura 2.15 se puede observar una simplificación básica de una neurona real, con tres partes principales marcadas con números, siendo el n^o1 las dendritas, receptores de impulsos nerviosos provenientes desde otras neuronas, el n^o2 es el núcleo, lugar donde se guarda todo lo importante de la neurona y el n^o3 corresponde a los telodendrones, ramificaciones extremas de los axones y encargadas de conectarse a otras neuronas mediante la sinapsis. Como se puede ver, estas las tareas de las diferentes partes se pueden resumir correspondientemente como, recibir información de otras neuronas, procesamiento de la información y envío de esta información a otras neuronas.

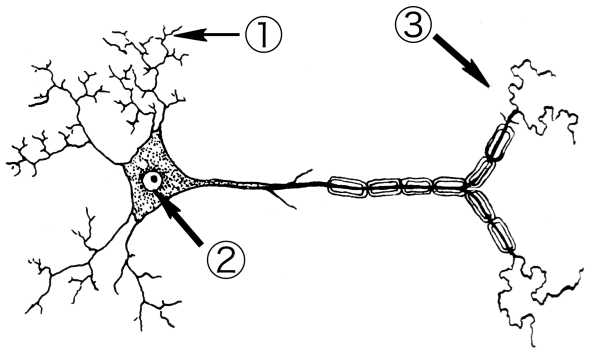


Figura 2.15: Neurona y sus partes simplificadas

Ha habido varios trabajos que han intentado pasar estas estructuras al mundo digital, pero el primero en mostrar resultados fue el trabajo de *McCulloch y Pitts* [20], el cual seguía los mismos principios de las neuronas, recibiendo información mediante *inputs*, los cuales eran multiplicados por un peso, para luego ser sumados (procesados), y de esta forma, entregar un resultado. En la fórmula 2.1 se pude observar los cálculos realizados para la obtención de la salida de esta neurona, con la particularidad de ser usada principalmente para imitar funciones lógicas, al igualar los pesos (W) a 1, solo se necesitaría variar el valor del θ, pudiendo de esta manera imitar estos resultados, por lo que teniendo $\theta = 1$ se obtiene una compuerta *OR*, al tener $\theta =$ *números de entradas* imita una compuerta *AND*, con $\theta = 0$ se obtiene una compuerta *NOR* y si solo recibe una única entrada, y se fija $\theta = 0$ imitaría una compuerta *NOT*.

$$\begin{gathered} red = \sum_{i=1}^{n=4} X_i \cdot W_i + X_0 = X_0 + X_1W_1 + X_2W_2 + X_3W_3 + X_4W_4 \\ Output = \begin{cases} 1 & \text{Si red} > \theta \\ 0 & \text{Si red} < \theta \end{cases} \end{gathered} \tag{2.1}$$

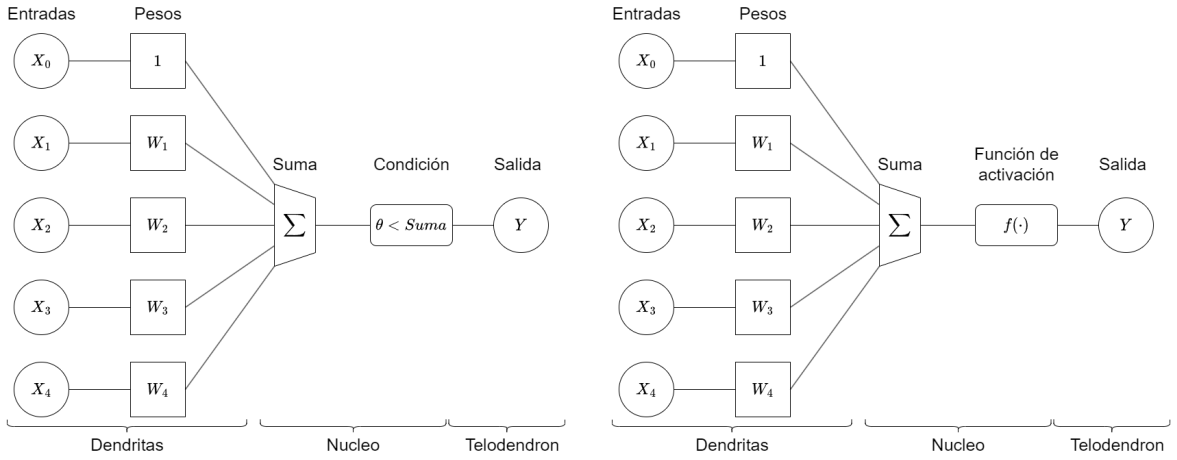
Una de las evoluciones de las neuronas de *McCulloch y Pitts* fue la obtenida por Frank J. Rosenblatt [21], la cual en si imitaba su misma estructura, la mejoraba de distintas maneras, reemplazando la necesidad de tener solo entradas digitales, pudiendo ser cualquier número entre 0 y 1, y la significativa eliminación del θ, siendo reemplazado por una función de activación, la que dependiendo de su elección, puede entregar diferentes resultados. En la formula 2.2 se puede observar que solo se modifica la obtención del *output*, pero siendo ahora importante la elección de la función de activación (FuncAct), siendo *Relu* la más usada actualmente.

$$\begin{gathered} red = \sum_{i=1}^{n=4} X_i \cdot W_i + X_0 = X_0 + X_1W_1 + X_2W_2 + X_3W_3 + X_4W_4 \\ Output = FuncAct(red) \\ \textit{Teniendo como posible } FuncAct(x) = \begin{cases} Logistic & = \frac{1}{1+e^x} \\ Tanh & = \frac{e^x - e^{-x}}{e^x + e^{-x}} \\ Relu & = max(x, 0) \end{cases} \end{gathered} \tag{2.2}$$

En la figura 2.16 se puede observar de manera estructural como están diseñadas estos dos tipos de neuronas, viendo que en la salida de la sumatoria (el valor red) son procesadas

por diferentes tipos de bloques, anteriormente explicados, pero ambas neuronas comparten la similitud con una neurona biológica, ya sea en su forma o funcionamiento, con la diferencia de solo tener una única salida.

Es necesario saber en qué parte de esta estructura se realiza el aprendizaje, lo cual es fácilmente identificable, teniendo en cuenta que, las entradas son valores externos y no directamente pertenecientes a la neurona en sí, y la función de activación es fija, por lo que los únicos valores que pueden ser modificados son los pesos W_i que acompañan a cada entrada. Lo que ocasionaría que, dependiendo de sus valores, se moldeara el resultado final.



(a) Neurona de McCulloch-Pitts (b) Perceptron

Figura 2.16: Diferencia entre ambos tipos de neuronas.

2.3.5. Redes neuronales

Uno de los principales problemas del *perceptron* es la imposibilidad de imitar la compuerta lógica *XOR*, siendo una tarea que su semejante si puede imitar. Pero esto es debido a que solo se está trabajando con un único elemento, siendo que las neuronas, como se ha mencionado anteriormente no están hechas para funcionar de manera solitaria, si no de manera grupal, logrando de esta manera resolver problemas muchos más complejos, como puede llegar a ser clasificación de objetos.

Al tener un grupo de más de una neurona (*perceptron*) ya se puede considerar como una red neuronal, las que al conectarse entre ellas crean una red robusta que puede imitar tareas complejas. En la figura 2.17 se puede observar dos capas de neuronas conectadas entre ellas, siendo todos los pasos de las neuronas comprimidas a un círculo que las representa, pudiéndose observar que todas las entradas son consideradas en cada neurona de la primera capa (N_i), mientras que todas las salidas de esta capa son tomadas por cada neurona de la segunda capa (O_i), la cual entrega las salidas de esta red.

Es importante mencionar que si bien, se conectan todos los elementos de una capa a la otra, esto no es exclusivamente necesario, ya que existen diferentes tipos de técnicas para mejorar el aprendizaje de los pesos, siendo uno de estos el famoso *dropout*, que de vez en

cuanto desconecta el camino que unía dos neuronas de diferentes capas.

Esta red puede tener una o varias salidas dependiendo de la tarea que se desea completar, teniendo más de una en caso de querer clasificar dependiendo de las diferentes entradas, o pudiendo obtener una salida digital de una solo neurona, donde solo un resultado 0 o 1.

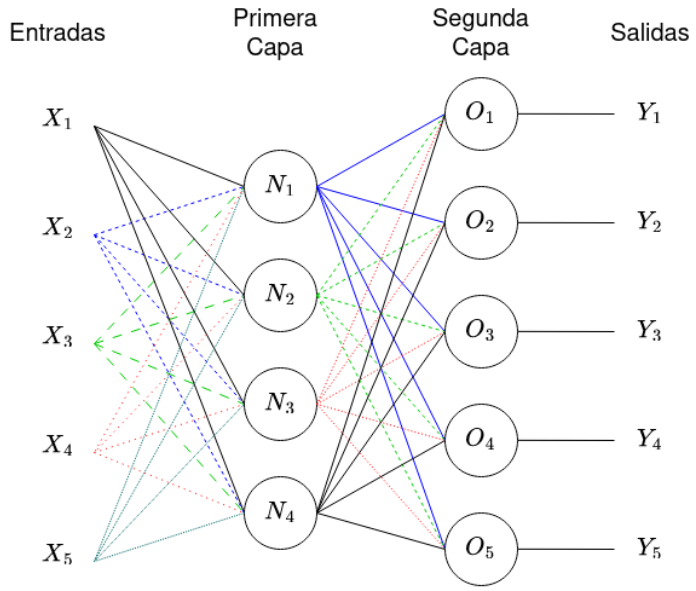


Figura 2.17: Red neuronal de dos capas

2.3.6. Redes convoluciones

Se menciono anteriormente que con la inteligencia artificial se desea imitar tareas humanas mediante maquinas, viendo un claro ejemplo con las redes neuronales, que imitaban de forma básica al funcionamiento del cerebro. Pero resulta que este no es el único caso donde se han visto estos contrastes, ya que de igual forma se ha tratado de imitar la visión humana, siendo una tarea fundamental para que las maquinas puedan entender el mundo exterior mediante imágenes, lo cual podría ser resuelto mediante redes neuronales, pero con paupérrimos resultados, debido a que las imágenes son más complejas que los datos generales, ya que en sí, podrían ser ingresadas píxel a píxel en una red, perdiendo todo sentido de la actividad en si, ya que lo importante de las imágenes no son los píxeles en sí, si no el conjunto de estos, sobre todo las posiciones que poseen con respecto a otros píxeles, y de esta forma, dándole forma a los objetos dentro de las imágenes.

Debido a la necesidad de un método más especializado en el proceso de imágenes, es donde aparece las famosas *Convolutional neural networks (CNN)*, o por su traducción al español, redes neuronales convolucionales, que, usando la operación matemática de la convolución, vista en la fórmula 2.3, consigue relacionar los píxeles con sus vecinos, y de esta forma, obtener información importante de las imágenes.

$$(f * g)(t) = \int_{-\infty}^{\infty} f(\tau)g(t-\tau)d\tau \tag{2.3}$$

Su aplicación dentro de imágenes seguía el principio de su operación matemática, pero aplicada de una manera diferente, en un ambiente 2-D y discreto, teniendo dos partes principales en su estructura, siendo estas: [17]

- **La Capa Convolucional**: Esta es la principal estructura dentro de esta red, siendo la principal razón por el nombre de las CNN, debido a la importancia que tienen dentro de estas. Dentro de esta capa se encuentran términos necesarios de conocer, por lo que se necesita hacer un pequeño repaso de cómo es su funcionamiento de forma individual y total:
  - Antes de cualquier definición, es necesario dejar una cosa clara, siendo esta el procesamiento de las imágenes a nivel computacional, llegando a ser tratadas como matrices de 3 dimensiones, siendo estas los Canales RGB de la imagen (C), la altura de esta (H) y su ancho (W). Por lo que siempre que se mencione que se van a trabajar con imágenes, se estará trabajando con una matriz de tamaño $C \cdot H \cdot W$.
  - El kernel es el encargado de realizar la convolución, siendo este una matriz de un tamaño mucho menor que el de la imagen, que por similitud con el ejemplo a dar se le asignara un tamaño de 2x2 al kernel y 4x4 a la imagen. Lo importante de esta matriz es lo que contiene dentro, ya que al crear una CNN los valores que contiene el kernel son elegidos de forma aleatoria, y mediante iteraciones va aprendiendo los valores que debe tomar, de manera semejante que las redes neuronales, por lo que al hacer los contrastes con esa red, los kernel contiene los pesos de esta red, lo que implica que el verdadero aprendizaje ocurre dentro de estas matrices.
  - Las operaciones convolucionales son la interacción que tiene el kernel con la imagen, donde este visita cada rincón de la imagen de entrada y la procesa con sus valores, pero debido a que el tamaño del kernel es menor que el de la imagen, es necesario que este se mueva dentro de ella tanto de manera vertical como horizontal, por lo que se deslizara cierta cantidad de celdas, conocidas como *stride*, para visitar cada celda de su matriz. Esto puede ser observado en la figura 2.18, específicamente en la columna izquierda, donde se puede apreciar que teniendo un *stride*=1, el kernel, correspondiente a la matriz verde, interactúa con cierta región de la imagen, siendo esta zona afectada de color azul, la cual se va deslizando cuanto mayor sea el *step* del proceso. La interacción que ocurre entre el kernel y las zonas de la imagen en cada *step* es el producto punto ente matrices, donde cada elemento de una matriz es multiplicado por el elemento que pertenece a la misma posición dentro de la otra matriz, y luego todos estos resultados son sumados, como se puede ver en el ejemplo, donde cada valor de la matriz de salida es una interacción del kernel con diferentes zonas de la imagen. Es importante mencionar que, debido a la naturaleza de la operación, se pierde información en la matriz de salida, dependiendo del tamaño que pueda llegar a tener el kernel, por lo que hay dos tipos de tratamiento a este suceso, llenando con ceros las columnas y filas que fueron eliminadas, y de esta forma la entrada y salida posean las mismas dimensiones, lo que se conocerá como *padding*, o simplemente considerar esta perdida dentro de la red.
- **Capa de *pooling***: El principal objetivo de esta capa es el sub-muestreo de los *feeature maps*, o mapas de características. obtenidas de la capa convolucional, manteniendo la mayor cantidad de información posible y disminuyendo el tamaño de estos mapas. Para

lograr este sub-muestreo se usa un concepto similar a los kernel visto anteriormente, pero con una operación diferente entre las matrices, siendo estas observadas en la figura 2.19, donde se puede observar que se presenta una matriz de menor tamaño de la imagen que se desliza en todos sus valores, pero con la diferencia que en este ejemplo, el *stride* de este movimiento es mayor al del kernel, evitando que se consideren las mismas celdas en dos diferentes operaciones. Los diferentes tipos de *pooling* observados en la figura son el *average pooling*, que obtiene el promedio de la zona procesada, el *max pooling* que obtiene el mayor valor de esta zona, y el *global average pooling* obtiene el promedio de toda la imagen, no de las zonas procesadas, por lo que a diferencia de sus otros dos métodos, que disminuyen el tamaño de la salida a la mitad que el de la entrada, este último lo reduce a solo un valor, perdiendo demasiada información en el proceso, pero para cierto tipo de tareas se pude acomodar bien.

- **Capas *fully connected***: O también conocidas como redes neuronales en este trabajo, son las capas donde los mapas de características anteriormente obtenidos en la capa de *pooling*, siendo re-dimensionadas a un vector de una dimensión, son procesados por una red con todas sus neuronas conectadas, y de esta forma, obtener una clasificación de la imagen ingresada.

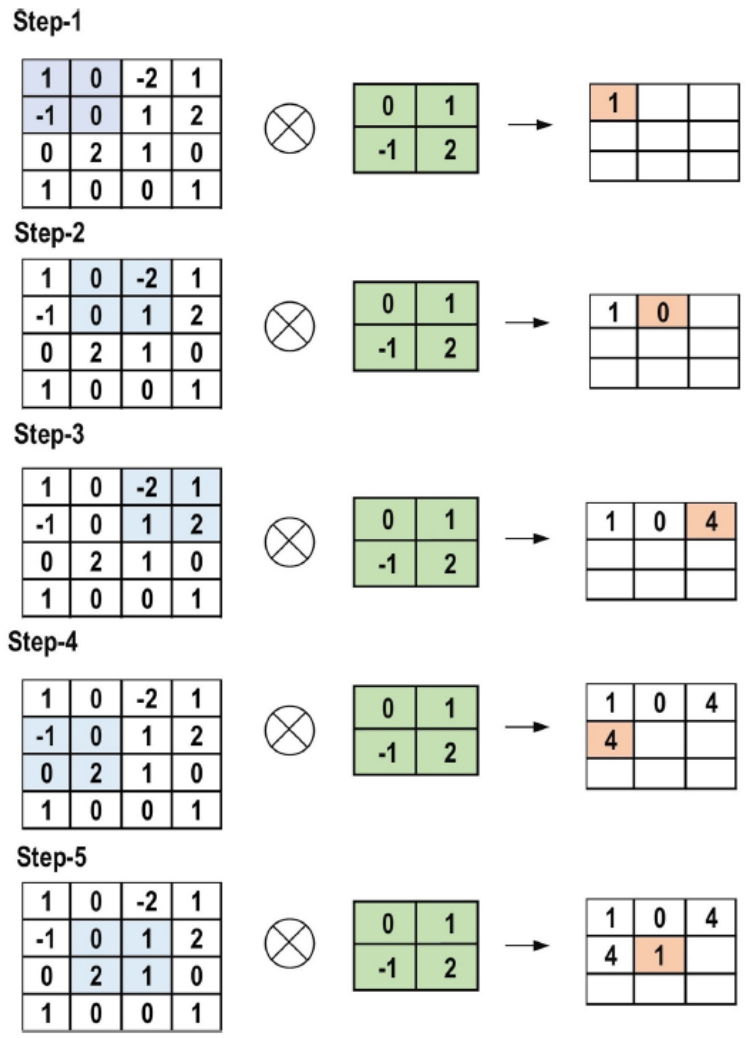


Figura 2.18: Operaciones convolucionales dentro de una imagen [17]

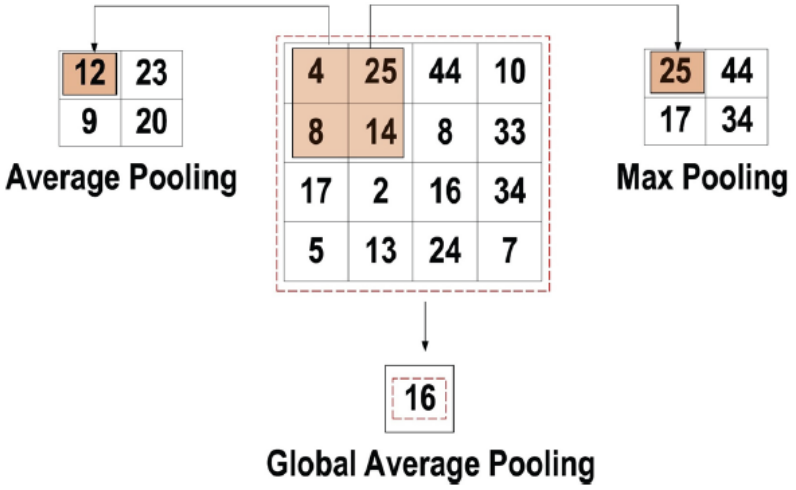


Figura 2.19: Diferentes tipos de *pooling* [17]

Como se vio en todo el proceso anterior, las capas convolucionales siempre vendrán acom-

pañadas con una capa de *pooling*, las que disminuirán el tamaño de estas, siendo este recorte dependiente del tamaño de la matriz de *pooling* y su *stride*. A esta combinación se le llamará bloque convolucional, no siendo necesario tener una proporción 1:1, ya que puede haber más de una capa convolucional antes del *pooling*, pero siempre terminará con este. Una CNN puede tener la cantidad de bloques convolucionales que requiera, siendo estos secuenciales entre ellos.

Este tipo de red es usado principalmente para la obtención de mapas de características, permitiendo de esta forma que las redes neuronales, con estas entradas, puedan clasificar imágenes de mejor manera que recibiendo estos datos crudos. Pero este no es único uso que se les puede dar, como se verá más adelante.

2.3.7. *ResNet*

Las redes convolucionales se volvieron la norma para el tratamiento de imágenes, viendo que cuantos más bloque posea, mejor será su resultado, pero esto se cumple hasta cierto punto, ya que si la red llega a ser demasiado profunda, la precisión se satura y empieza a degradarse rápidamente. Sorprendentemente la degradación no es causada por el *overfitting*, y agregar más capas a un modelo funcional, lo conduce a mayores errores de entrenamiento.

En el *paper Deep Residual Learning for Image Recognition* [10], se presenta una solución a este problema, donde en lugar de esperar que cada una de las capas apiladas se ajuste directamente a un mapeo subyacente deseado, dejan explícitamente que estas capas se ajusten a un mapeo residual. En la figura 2.20 se puede observar cómo se logra lo anteriormente mencionado, donde es casi igual a la estructura de una red convolucional normal, pero con la diferencia que cada entrada del bloque no necesariamente solo recibe lo entregado del bloque anterior, sino que también puede contener información del bloque anterior a ese, siendo agregado para la salida del bloque actual. Dependiendo la cantidad de total capas que contenga esta estructura, considerando también las capa *fully connected*, será modificado el numero dentro su nombre, como ejemplos están la *ResNet-34*, *ResNet-50*, *ResNet-101* y *ResNet-152*, siendo cada una más profunda que la otra.

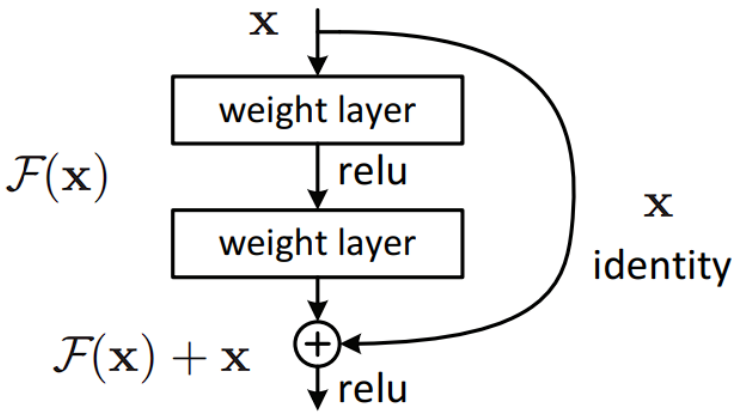


Figura 2.20: Bloques de *ResNet* [17]

2.3.8. *Backbone*

Como se había mencionado anteriormente, dentro del *deep learning* también se encuentra el paso de *feature extraction*, pero a diferencia de los métodos tradicionales, la extracción de características de las imágenes puede ser aprendido. Esto se logra mediante la *Backbone*, la cual en tareas de detección es usada como una herramienta para obtener datos más relevantes dentro de la red, y debido a su adaptabilidad se puede conseguir una mejorar en sus resultados.

Pero debido a que los modelo pueden llegar a presentar diferentes tipos de estructura, es necesaria la correcta elección de las *Backbone*, ya que dependiendo de cual se use podría llegar a mejorar los resultados. Existen varias redes populares que son usadas como *Bacbone*, destacando entre estas las redes *Resnet*, *VGG*, *Inception*, *GoogleNet* y *AlexNet*, que en un principio fueron diseñadas para problemas de clasificación, pero debido a sus buenos resultados en el pre-procesamiento de los datos de entrada empezaron a brillar también dentro de estas áreas.

Se ha demostrado que algunas redes entregan mejores resultados en ciertas tareas que en otras, pero la *Resnet* brilla por su variedad de usos, como pueden ser el resumen de vídeo, reconocimiento facial, reconocimiento de accione y detección de objetos [16]. Por lo que dentro de este trabajo será la única red que se usará para los *Backbone* de los diferentes modelos a presentar.

2.3.9. Optimizador

Como se ha mencionado anteriormente, cada red que tiene algún aprendizaje tiene parámetros modificables dentro ella, los cuales se ajustan dependiendo de la tarea a cumplir. Pero dentro de cada red es necesario una manera de que estos parámetros sean cambiados, proceso del cual se encarga los optimizadores, que, mediante el *backpropagation*, modifica el valor de los parámetros de cada capa. En el *backpropagation*, se compara el resultado predicho con el resultado real, y se calcula un error para cada una de las salidas. Los errores se propagan hacia atrás, partiendo de la capa de salida, hacia todas las neuronas de la capa oculta que contribuyen directamente a la salida. Sin embargo, las neuronas de la capa oculta solo reciben una fracción de la del error, basándose aproximadamente en la contribución relativa que haya aportado cada neurona a la salida original. Este proceso se repite, capa por capa, hasta que todas las neuronas de la red hayan recibido un error que describa su contribución relativa al error total.

Hay dos optimizadores que sobresalen por ser los más usados, el *Stochastic gradient descent* (SGD) que usa el gradiente para modificar los parámetros mediante la formula $w_{t+1} = w_t - \eta \cdot \nabla Q(w)$, con $Q(w)$ como la función de costo. El otro optimizador importante es el *Adaptive moment estimation* (ADAM), que comparte una formula semejante a la anterior con $w_{t+1} = w - \Delta w_t$, pero lo importante aquí es como se obtiene este Δw, siendo observado este proceso en la fórmula 2.4

$$\begin{aligned} v_t &= B * v_{t-1} - (1 - B_1) * g_t \\ s_t &= B * s_{t-1} - (1 - B_2) * g_t^2 \\ \Delta w &= -\epsilon \frac{v_t}{\sqrt{s_t + \eta}} * g_t \\ w_{t+1} &= w_t + \Delta w_t \end{aligned} \tag{2.4}$$

2.3.10. *Attention*

Al igual que el pensamiento y la visión, ha habido otros casos donde se ha imitado características humanas dentro de máquinas, siendo en este caso la atención, capacidad de seleccionar y concentrarse en los estímulos relevantes. Este proceso es importante para lograr que un modelo logre enfocarse en puntos de interés, pero sus orígenes no fueron principalmente usados para trabajar con imágenes, si no que fueron desarrollados para problemas de *Natural Language Processing* (NLP), siendo uno de sus principales usos dentro del área de traducción.

Naturalmente la atención es usada en redes del tipo *sequence to sequence (seq2seq)*, donde una red codifica la entrada en un vector, y otra red descodifica lo obtenido en una salida diferente. Los resultados de usar estas técnicas pueden ser observado en la figura 2.21, donde se puede observar que se pasa una frase del inglés al francés, pero no de forma directa, si no que revisando de palabra a palabra, y de esta forma, mediante la atención, ver cual palabra del otro idioma toma más importancia para la palabra revisada.

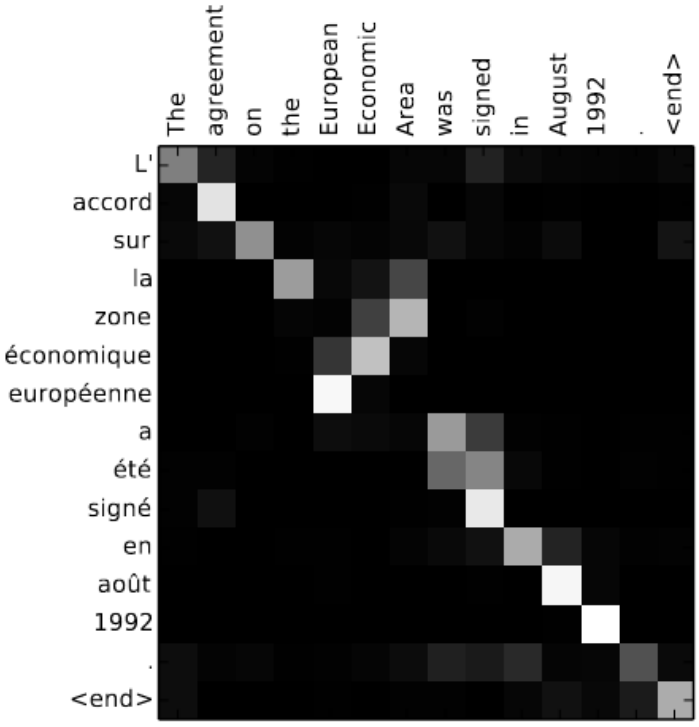


Figura 2.21: Ejemplo de importancia de cada palabra en ingles con su traducción al francés [15]

Hay varios métodos de atención dentro del *machine learning*, pero hay uno que se ha destacado mucho más que los otros en los últimos tiempos debido a sus muy buenos resultados, siendo este el presentado dentro del *paper* de *Attention is all you need* [8], donde generalizan la definición de atención a una función que mapea un *query* y un par de *key-value*, donde todos estos elementos, junto a la salida, son vectores. La salida se calcula como una suma ponderada de los *values*, donde el peso asignado a cada valor se calcula mediante una función de compatibilidad de la *query* con la *key* correspondiente. Por lo que el paper presenta el *"Scaled Dot-Product Attention"*, pudiendo ser observada en la función de la 2.5, donde recibe como entrada *queries* y *keys* de dimensión d_k, y *values* de dimensión d_v. Es importante mencionar que dentro del *paper* se menciona que su función es similar a otra usada anteriormente, pero que el factor $\frac{1}{\sqrt{d_k}}$ es una adición hecha por ellos, que mejora los resultados cuanto mayor sea el tamaño de d_k en comparación al no usarlo. En la figura 2.22 se puede observar la estructura de esta función.

$$Attention(Q, K, V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.5}$$

Otra cosa importante de recalcar que se presenta dentro del *paper* es el *Multi-head attention*, observada en la figura 2.22, que usa la misma estructura que la función de atención presentada anteriormente, pero con la diferencia de tener múltiples entradas, donde se usan los mismos *queries*, *keys* y *values*, pero con la diferencia que poseen h diferentes proyecciones lineales aprendidas. De esta manera el modelo atiende conjuntamente la información de distintos sub-espacios de representación en diferentes posiciones. En la función 2.6 se puede observar cómo los resultados de cada *head* son concatenados y multiplicados por una matriz,

y como cada *head* procesa las entradas, siendo los tres W_i las matrices que proyectan a estas.

$$\begin{gathered} MultiHead(Q,K,V) = Concat(head_1,...,head_h)W^O \\ \text{donde } head_i = Attention(QW_i^Q, KW_i^K, VW_i^V) \end{gathered} \tag{2.6}$$

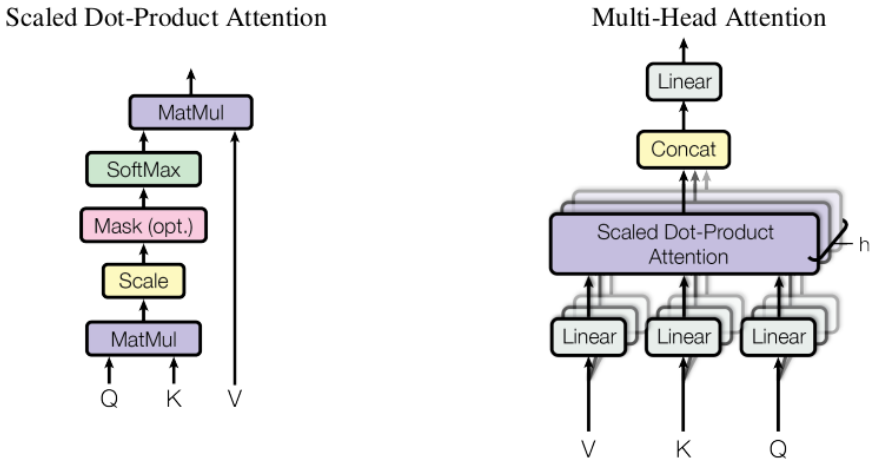


Figura 2.22: Estructuras del *Scaled Dot-Product Attention* y *Multi-Head Attention* [8]

2.3.11. *Transformers*

Naturalmente la atención era principalmente usada en las tareas de traducción, por lo que era necesario relacionar dos vectores diferentes para la obtención de un resultado, pero en *paper* de *"Attention is all you need"* se introduce el *self-attention*, que en vez de usar dos vectores diferentes como entrada para la relación de sus elementos, solo se usa uno, centrándose en la relación de los elementos internos del vector. Este método por si solo no aporta demasiado al tema, pero al unir los métodos de *self-attention* y *multi-head attention* se obtiene un nuevo método propuesto dentro del *paper*, los *transformers*, que aparte de lograr relacionar dos vectores diferentes, también lo consigue con el vector interés, por lo que en tareas de traducción se logra un mejor resultado que los otros métodos que usaban redes recursivas.

Debido a que los *transformers* fueron hechos principalmente para tareas de NLP, se explicará su estructura mediante un ejemplo de traducción, donde se busca pasar una frase del inglés, *"The Black cat"*, al español, "El gato negro". En la figura 2.23 se puede observar todos los bloques importantes dentro de su estructura, destacando las presencias de dos tipos de *multi-head attention*, siendo el primero el que recibe sus tres vectores de entrada (Q, K y V) de un solo vector, como se observa de los primeros pasos del bloque izquierdo y derecho, por lo que se considerarán como *self-attention*, y el segundo que recibe más de un vector de entrada. El bloque izquierdo, correspondiente al *encoder*, es el que recibe la entrada que se quiere traducir, que en este caso sería la frase en inglés, y es procesada por una *self-attention* para luego ser usada por el otro bloque, mientras que el bloque derecho, correspondiente al *decoder*, puede recibir dos tipos de entrada, dependiendo si está en el proceso de entrenamiento o de evaluación, siendo su traducción al español en el primer caso, y las posibles traducciones en caso de evaluación. En el *decoder*, aparte de ser procesada por un *self-attention*, también

lo es por una *multi-head attention*, que, a diferencia de su predecesora, recibe como entrada el resultado de la *self-attention* y la salida del *encoder*, para finalmente obtener el resultado.

En el ejemplo de la frase en inglés, *"the black cat"*, no es una traducción directa de palabra a palabra, ya que como se sabe, en el español a diferencia del inglés, los adjetivos van después de lo sustantivos, por lo que la traducción directa "el negro gato" no es la correcta, siendo necesaria una mejor comprensión de estas características del lenguaje. La frase en inglés sería considerada como la entrada del *encoder*, ya que se desea pasar del inglés al español. Lo importante de aquí es la entrada del *decoder*, ya que la salida del *encoder* entra a este bloque en la mitad del proceso, no al comienzo, siendo primeramente ingresado las posibles correctas traducciones secuenciales de la frase. Explicado con el ejemplo, en el primer paso el *decoder* no recibe ninguna palabra, ya que aún no se procesa la frase en ingles procesada por el *encoder*, pero tiene como salida la posible traducción de la primera palabra de la frase en español, siendo esta "el", para el segundo paso, el *decoder* recibe como entrada su salida del paso anterior ("el"), siendo procesada dentro de su estructura junto a la salida del *encoder*, por lo que el *decoder* supone que la segunda palabra de la frase en español debe ser "gato", por lo que la nueva salida sería "el gato", para el último paso solo se recibe como entrada la última palabra de la salida del paso anterior ("gato"), siendo procesada junto a la salida del *encoder*, y dando como resultado la frase completa "el gato negro". El ejemplo mencionado anteriormente puede ser observado en la figura A.1, donde se explica de mejor manera los pasos tomados para la traducción completa.

Es importante destacar que este método es mejor para este tipo de trabajo que su competencia directa, que serían las redes recursivas, ya que ha diferencias de estas, no sufren degradación de sus resultados al recibir frases más largas, y tardan mucho menor al no trabajar de forma totalmente secuencial, teniendo partes que pueden trabajar en paralelo.

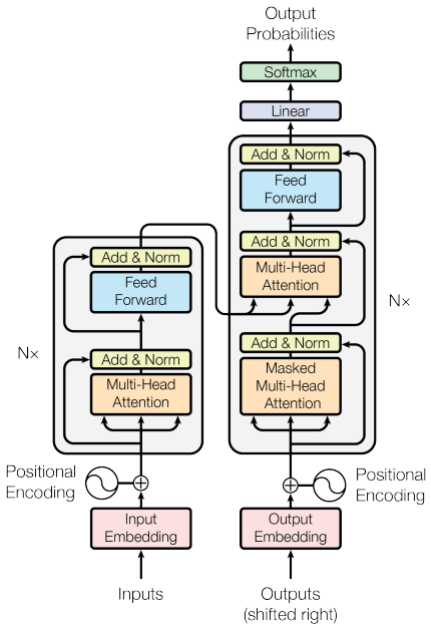


Figura 2.23: Estructura del *transformer* [8]

2.4. Métricas

Debido a la naturaleza de este trabajo, donde se busca un modelo que entregue buenos resultados en los objetivos propuestos, es necesario tener alguna forma de poder calificar cual de todos los modelos a probar será el mejor, por lo que es necesario tener algunas métricas de comparación, y de esta forma poder identificar cual modelo resuelve de mejor forma el problema propuesto.

Es importante la métrica a usar, ya que al ser un problema de detección dentro de imágenes, no le servirán totalmente las métricas usadas para tareas de clasificación, por lo que es necesario repasar algunos conocimientos para saber cuál va a ser usada al final.

Primero que nada, es importante reconocer que ningún modelo va a ser 100 % efectivo, por lo que van a haber casos donde se realizará predicciones incorrectas, como son en los casos de los falsos negativos y falsos positivos, siendo el primero todas las predicciones que dicen que la salida es negativa cuando el resultado real es positivo, y el segundo caso siendo todo lo contrario, con resultados que predicen que la salida es positiva cuando esta realmente es negativa. El caso contrario a estos serían el verdadero negativo y verdadero positivo, donde tanto la predicción como el resultado real son del mismo tipo. Con el contenido de la tabla 2.1 debería quedar claro lo anteriormente mencionado.

Tabla 2.1: Tabla de posibles resultados de un modelo.

		Real	
		Positivo	Negativo
Predicho	Positivo	Verdadero Positivo (VP)	Falso Positivo (FP)
	Negativo	Falso Negativo (FN)	Verdadero Negativo (VN)

Ya teniendo esto claro, se puede hacer un repaso en las métricas necesarias para la obtención de la que se usará.

2.4.1. Precision

Con la precisión se consigue el porcentaje de la cantidad de resultados positivos que fueron correctamente predichos (verdaderos positivos) entre la cantidad de todos los positivos predichos que habían (conjunto de verdadero positivos y falsos positivos), en la fórmula 2.7 se puede observar cómo se calcula esta métrica.

$$Precisión = \frac{VP}{VP + FP} \tag{2.7}$$

2.4.2. *Recall*

Con el *recall* se consigue el porcentaje de la cantidad de resultados positivos que fueron correctamente predichos (verdaderos positivos) entre la cantidad de todos los positivos que realmente habían (conjunto de verdadero positivos y falsos negativos), en la fórmula 2.8 se puede observar cómo se calcula esta métrica.

$$Recall = \frac{VP}{VP + FN} \tag{2.8}$$

2.4.3. *Intersection Over Union (IoU)*

El *Intersection Over Union* (*IoU*) es una de las métricas usadas principalmente para calificar el desempeño de los modelos de detección de objetos, comparando los *bounding boxes* predichos con los reales.

No se había mencionado anteriormente, pero el *bounding box* es usado para delimitar visualmente un área de un problema de dos dimensiones o más, como puede ser una imagen, vídeo, datos en el tiempo, etc. Debido a que todos estos datos están representados en más de 1 dimensión, pueden ser señalados mediante figuras rectangulares, por lo que principalmente se guardarán las posiciones de las esquinas de esta figura dentro de los datos.

Volviendo al *IoU*, al tener que comparar dos *bounding boxes* es necesario saber cómo hacerlo, y la mejor forma de lograrlo es mediante sus áreas, tanto de su intersección como de su unión. Y al dividir estas dos áreas, de la intersección sobre la de la unión, se consigue la métrica *IoU*, donde si el resultado es más cercano a 1, significa que las dos *bounding boxes* están casi en las mismas posiciones, mientras que cuanto más cercano a 0 sea el valor, significa que no hay ninguna similitud en sus posiciones.

En la figura 2.24 se puede observar un ejemplo de cómo se calcula esta métrica, donde la imagen de la izquierda contiene la *bounding box* del gato completo en verde, mientras que la azul corresponde a la *bounding box* predicha, el *IoU* se calcularía con el área de intersección de las *bounding box* azul y verde, dividido por la unión de ambas. Se puede observar que el área superior corresponde a menos de la mitad de la inferior, por lo que su puntuación del *IoU* sería menor que 0,5, haciéndola una predicción no muy buena.

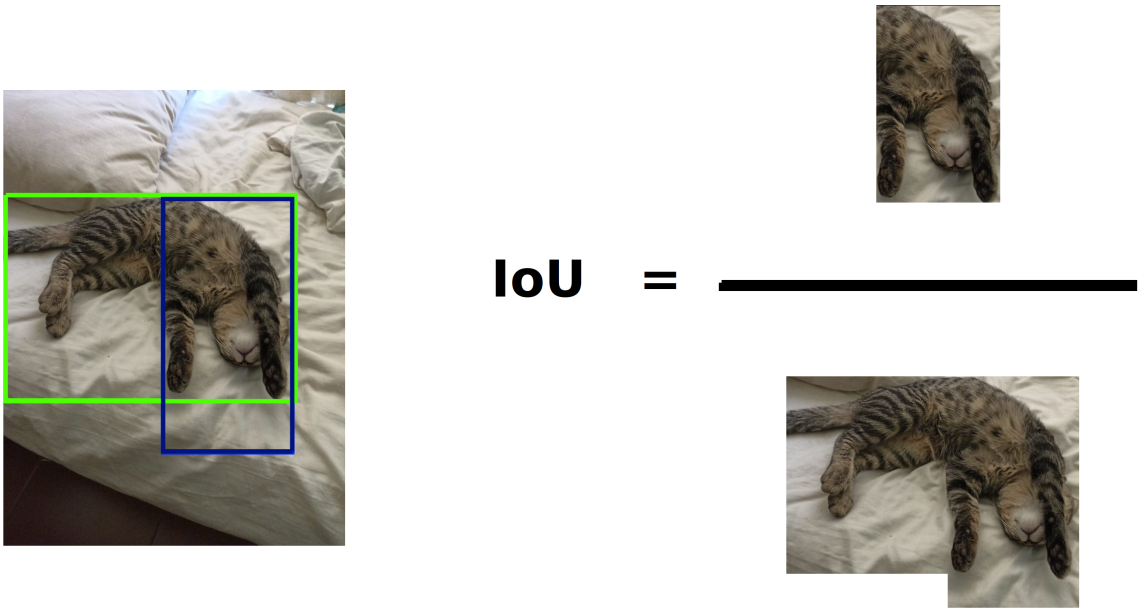


Figura 2.24: Ejemplo del calculo del *IoU*

2.4.4. *Average Precisión (AP)*

Dentro de los problemas de detección, al tener *bounding boxes* predichas y reales, se necesita una nueva definición para los casos de verdaderos (y falsos) positivos, dado que van haber situaciones en las *bounding boxes* sean similares, pero no exactamente iguales, por lo que debe haber un *threshold* que defina cuando se considere un verdadero o falso positivo. Esto se consigue mediante la métrica de *IoU*, ya que de esta manera se puede calificar que tan cercana fue la predicción con la ubicación real. Naturalmente en los *papers* se considera como un verdadero positivo si el *IoU* es mayor que 0.5, por lo que todo otro caso que sea menor será considerado como un falso positivo.

Otro *threshold* importante a considerar es el de *score* de la detección, que, dependiendo de su valor, puede considerarse si el modelo presenta una propuesta de región, ya que toda detección con un valor menor que el *theshold* del score será considerado un falso positivo.

Para todo resultado siempre se buscará un valor alto para las métricas de *recall* y precisión, pero dependiendo del *threshold* del *score* de detección, estos valores se inclinan hacia un lado o el otro, ya que si este *threshold* es alto, no habrá varia detecciones, pero todas las que hayan tiene más posibilidades de realmente contener un objeto buscado, y por ende, aumentar la precisión, mientras que el *threshold* el tener un valor pequeño, se harán más detecciones y por ende, es menos probable no haber considerado un objeto buscado, y por ende, aumento el *recall*, por lo que es necesario tener un *trade-off* entre ambas. Una curva de precisión-*recall* muestra el valor de la precisión en función del *recall* para diferentes valores del *thresholds*, siendo esta una buena herramienta para fijar el valor de este, al elegir el que tenga los valores más altos de *recall* y precisión.

El *Average precision* (AP) es el área bajo la curva de precisión-*recall*, la que nos indicaría si la precisión y el *recall* son altos sin depender de la selección de un *threshold*. El número que naturalmente acompaña al AP es el *threshold* del IoU, el cual solo se consideran verdaderos positivos si su valor es mayor que este. Dentro de esta definición también se encontraría el *mean Average precisión* (mAP), que dependiendo del conjunto de datos con el que se trabaje, puede variar. Para PASCAL-VOC es simplemente el promedio del AP de todas las clases, mientras que para COCO se calcula el AP para los valores del IoU entre 0.5 y 0.95, a pasos de 0.05, y todos estos valores son promediados, obteniéndose el mAP. Solamente la primera definición de mAP es de interés dentro de este trabajo, ya que se trabajará solamente con PASCAL-VOC, y por ende, solo se considera su definicón.

Capítulo 3

Estado del arte

Actualmente hay una gran variedad de *papers* dedicados a la localización de objetos mediante fotos, en los cuales se usan diferentes tipos de técnicas, pero siendo los más actuales y los que entregan mejores resultados los que usan estructuras de *Deep learning*. Y de igual forma hay una limitada cantidad de *papers* que abarcan el tema de la localización de objetos usando bosquejos, pero hay dos *papers* que sobresalen sobre estos temas, siendo el *paper Sketch-Guided Object Localization in Natural Images* [7], el que entrega una metodología de red que puede relacionar bosquejos con objetos de búsqueda y poder localizarlo, y el *paper Adaptive Image Transformer for One-Shot Object Detection*, que entrega los mejores resultados en detección de objetos dentro de una imagen, siendo estos objetos obtenidos igualmente de una imagen. La idea principal sería obtener una mezcla de ambas estructuras, por lo que van a haber varios modelos intermedios, que usarán las estructuras de otros *papers* importantes dentro de este trabajo.

3.1. Sketch-Guided Object Localization in Natural Images

Uno de los pocos *papers* sobre la detección de objetos mediante *sketch*, que se caracteriza por sus dos partes principales, siendo este el *proposal generation* y el *proposal scoring*, en su primera parte se destaca el uso del método propuesto dentro de este *paper*, el *Cross-modal attention*, que prioriza incrustar información del *query (sketch)* en las representación de características de la imagen, antes de generar una propuesta de región, por lo que, es intrínsecamente capaz de generar propuestas relevantes de objetos, incluso para categorías de objetos que no se ven durante el tiempo de entrenamiento (clases no vistas). Mientras que el *proposal scoring* compara el resultado de la parte anterior con el *query* para localizar precisamente el objeto de interés.

En la figura 3.1 se puede observar la estructura de este modelo, donde se tiene una imagen real como imagen objetivo, y el *sketch* como entrada *query*. La primera etapa *proposal generation*, los vectores de características correspondientes a diferentes regiones del mapa de características de la imagen (que se muestra en rosa) se califican con la representación global del *sketch* (que se muestra en azul) para identificar la compatibilidad (Bloque-1). Luego, estos puntajes de compatibilidad (mostrados en violeta) se multiplican con el mapa de características de la imagen (rosa) para obtener las características de atención (Bloque-2). Además, estos mapas de características de atención están concatenados con los mapas de caracterísi-

cas originales, y proyectadas a un espacio de menor dimensión, que son procesadas a través de una *Region Proposal Network* para generar propuestas de objetos relevantes (Bloque-3). En la segunda etapa *proposal scoring*, las propuestas de objetos resumidas (que se muestran usando índigo) se puntúan con el vector de características del bosquejo (que se muestra en azul) para localizar el objeto de interés (Bloque-4). [7]

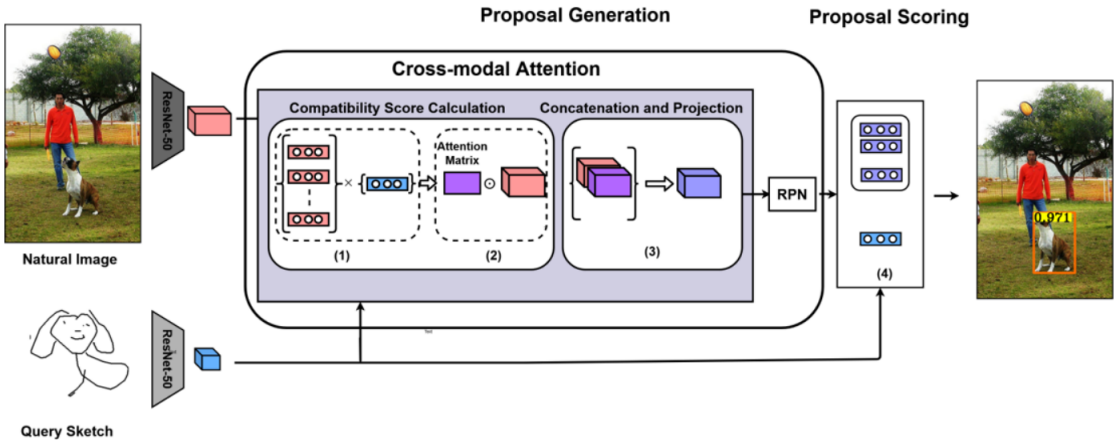


Figura 3.1: Estructura de la red del paper *Sketch-Guided Object Localization in Natural Images (2020) [13].*

Esta red entrega buenos resultados tanto para objetos pertenecientes a las clases vista dentro del entrenamiento, como a las no vistas. Y será usado principalmente para comparar resultados con el modelo obtenido, ya que es el modelo con mejores resultados usando bosquejos como imágenes de consulta, y se puede considerar como la predominante en su área en el día de hoy.

3.2. Adaptive Image Transformer for One-Shot Object Detection

El *paper Adaptive Image Transformer for One-Shot Object Detection* implementa una nueva técnica de localización de objeto, ya que sigue una idea parecida a la del paper anterior, con la diferencia que las áreas propuestas, que anteriormente eran puntuadas de forma inmediata, ahora son procesadas mediante *transformers*, que mejoran la relación entre el bosquejo y las áreas propuestas, por lo que su puntuación final será más precisa y, finalmente, de las áreas con mejores puntuaciones se revisan diferentes canales que tienen diferentes resultados y se eligen los mejores puntuados. Este modelo es el que entrega mejores resultados hasta la fecha en la tarea de localización de objetos, y está construida para usar imágenes reales tanto para la imagen objetivo como para la entrada de consulta.

Este modelo tiene tres partes claves dentro de su estructura, siendo la primera el *Multi-head Co-attention* (a), que correlaciona la imagen objetivo y la de consulta a través de varios *embeddings*. El mecanismo de atención considera conjuntamente la imagen de objetivo y la de consulta explorando diferentes aspectos de las características visuales y genera un mapa de características correspondiente que codifica tal relación, sobre la cual el RPN podría generar

más propuestas de regiones relevantes para el *query*. La segunda parte clave es el módulo *Adaptive Image Transformer* (b), está diseñado para explorar cómo cada par propuesta-*query* comparte atributos semánticos comunes sobre los rasgos visuales profundos, específicamente, se emplea un esquema de traducción de características. En su formulación, AIT transformaría adaptativamente el mapa de características de cada propuesta para que coincida con las características del *query* mediante el empleo de mecanismos de atención aprendidos. Es decir, dado una entrada de consulta, los aspectos de las características visuales ya sean, la forma, textura y/o color, al ser destacas podrían variar entre las propuestas de regiones. Y finalmente, la última parte clave corresponde al uso selectivo de canales (*Selective Channel Attention*), usados para mejorar la efectividad al optimizar con *ranking loss*. A pesar de que el módulo AIT puede transformar las características de las propuestas para que coincida con las características del *query*, la similitud podría diferir significativamente sobre la dimensión de canal respectivo. Así, sería beneficioso para realzar la importancia de esos canales de alta similitud antes de evaluar un par propuesta-*query*. Esta estructura se puede observar en la figura 3.2

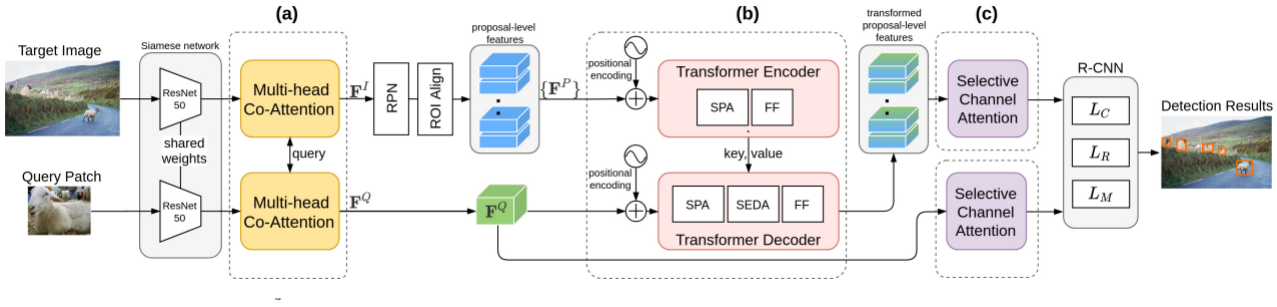


Figura 3.2: Estructura de la red del paper *Adaptive Image Transformer for One-Shot Object Detection* (2021) [2].

Dentro de este trabajo se intentará imitar este modelo, al funcionar de buena manera tanto para las clases vistas, como las no vistas. Pero debido a los limites de tiempo solo se mencionará como el modelo teóricamente perfecto para el objetivo de este trabajo.

3.3. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks

En el paper de *Faster R-CNN* [9] se presenta por primera el *Region Proposal Network* (RPN) que comparte características convolucionales de las imágenes completas con la red de detección, lo que permite propuestas de regiones casi gratuitas. Una RPN es una *fully-convolutional network* que predice simultáneamente los límites de los objetos y las puntuaciones de la probabilidad de encontrarlos en cada posición. La RPN esta entrenada *end-to-end* para generar propuestas regionales de alta calidad, que el *Fast R-CNN* utiliza para la detección.

Su sistema de detección de objetos, llamado *Faster R-CNN*, esta compuesto por dos módulos. Siendo el primero un *fully-convolutional network* que propone regiones, y el segundo módulo es el detector *Fast R-CNN*, como se puede ver en la imagen 3.3, que utiliza las regiones propuestas. El sistema entero en una única red unificada para la detección de objetos.

El módulo de la RPN le dice al módulo *Fast R-CNN* dónde buscar, mediante mecanismos de atención.

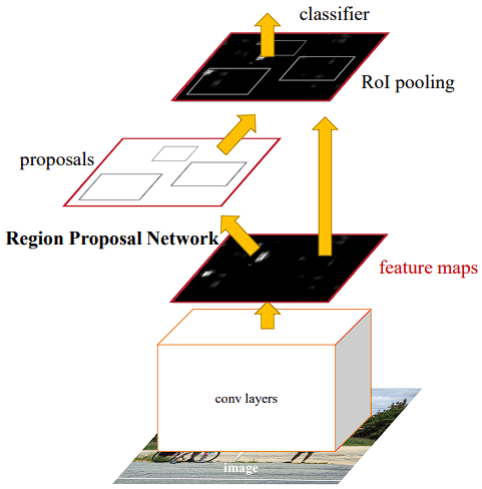


Figura 3.3: Estructura de la red del *Faster R-CNN* (2021) [9] (2016).

Esta red en si no fue utilizada, pero fue usada como base para fabricar otros modelos de detección, que se verán más adelante.

3.4. Few-Shot Object Detection with Attention-RPN and Multi-Relation Detector

En el *paper Few-Shot Object Detection with Attention-RPN and Multi-Relation Detector* [3] se presenta una modificación de la estructura del modelo del *Faster R-CNN* [9], donde cambia el *Region Proposal Network* por un mecanismo de atención y el detector por módulos *multi-relation* para producir un preciso análisis entre el *query* (renombrado *support* dentro de este paper) y los potenciales *boxes* de la imagen objetivo (llamados *query* dentro de este trabajo).

Dentro de este *paper* se recalca que el RPN no sólo debe distinguir entre objetos y no objetos, pero también filtrar objetos negativos que no pertenezcan a la misma clase que el *query*. Sin embargo, si no se tiene ninguna información de la imagen de consulta, el RPN será activado en cada objeto potencial con alta probabilidad de presencia de objetos, aunque estos no pertenezcan a la misma clase que el *query*, sobrecargando así las posteriores tareas de clasificación del detector con un gran número de objetos irrelevantes. Debido a este problema, proponen el *Attention-RPN*, observado en la figura 3.4, que utiliza información de la imagen de consulta para permitir el filtrado de la mayoría de los cuadros de *background* y las que no coincidan con la misma clase. Por lo tanto, un conjunto más pequeño y preciso de candidatas son generadas, que contienen una alta probabilidad de contener objetos de interés.

Al usar un *framework* R-CNN, el módulo RPN será seguido por un detector que tiene como rol de volver a puntuar las propuestas y reconocer las clases. Por lo tanto, desarrolla-

ron un detector con una fuerte capacidad discriminatoria para distinguir diferentes clases, que mide de forma eficaz la similitud entre las regiones propuestos dentro **target** y los objetos de la imagen de consulta, aprendiendo sus relaciones a diferentes niveles de profundidad.

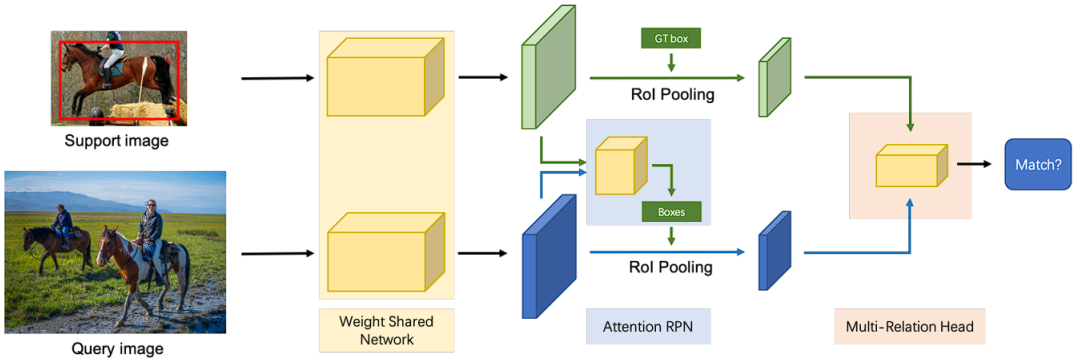


Figura 3.4: Estructura de la red del paper *Few-Shot Object Detection with Attention-RPN and Multi-Relation Detector* (2020) [3].

Este modelo será utilizado como *Baseline* dentro este trabajo, aunque su principal área de trabajo se centra en el *few-shot detection*, pero al también considerar el *one-shot* dentro de sus resultados, se considero como un buen punto para partir.

3.5. CAT: Cross-Attention Transformer for One-Shot Object Detection

La arquitectura propuesta dentro del *paper CAT: Cross-Attention Transformer for One-Shot Object Detection* [4], está compuesta de tres partes como se puede observar en la figura 3.5, incluyendo la extracción de características (*Backbone*), el módulo *cross-attention* y el cabezal de detección basado en similitud. Para el primer paso, optan por la *ResNet-50* para extraer características, tanto para la imagen *query* como el *target*, con los parámetros de la *backbone* compartidas entre ambas entradas, solo usando los tres primeros bloques de la *Resnet-50* para obtener mapas de caracteritícas de mayor resolución.

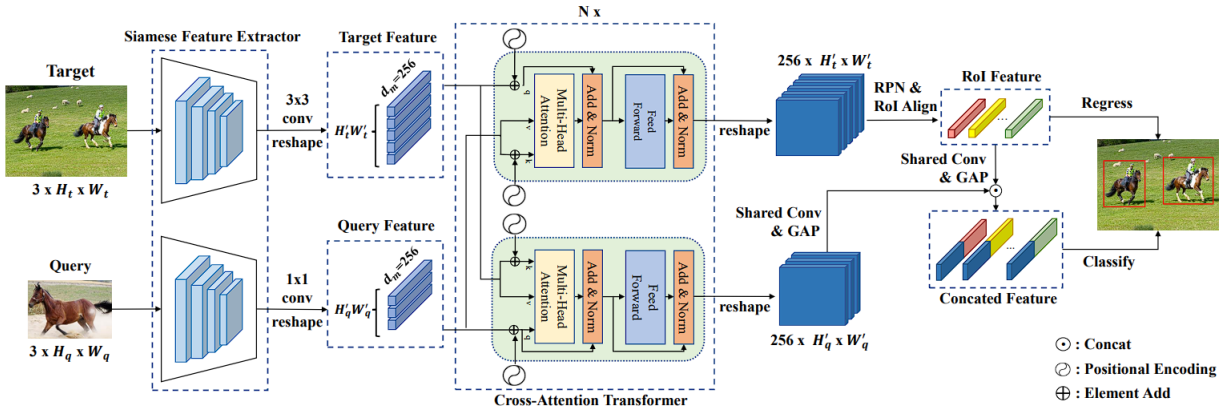


Figura 3.5: Estructura de la red del *paper CAT: Cross-Attention Transformer for One-Shot Object Detection* (2021) [4].

Es red será considerada como el punto intermedio entre la estructura del *Attention-RPN* y la del *Adaptive Image Transformer*, siendo elegida por la similitud de su estructura con el primer modelo mencionado y el uso de *transformer* al igual que ek segundo, por lo que se puede imitar de manera más fácil que la del *Adaptive Image Transformer*.

3.6. Evaluación de métodos auto-supervisados y semi-supervisados para la extracción de características visuales en el contexto de recuperación de imágenes basada en Dibujos

La estructura observada en la figura 3.6 se obtuvo del trabajo de título de Javier Morales [5], siendo esta una de las soluciones propuesta.

La arquitectura de BYOL es propuesta para aprendizaje auto-supervisado, con la idea de tener una red "profesor" de aprendizaje lento, conocida como *target-encoder*, que enseña a otra red "alumno", o conocida como *student-encoder*, siendo este un concepto muy interesante que puede ser extendido a otros problemas. Por eso propone utilizar una arquitectura BYOL bimodal, es decir, que sea capaz de extraer características de dos modalidades o dominios, en este caso fotos y bosquejos, para enfrentar el problema de recuperación de imágenes basada en bimodal. Para eso se plantea asignar un dominio específico a cada red, de forma de tener una red especializada en bosquejos y otra en fotos. El modelo es entrenado con pares de bosquejos y fotos que compartan la misma clase, por lo que no es realmente auto-supervisado pues se necesita generar estos pares, lo que busca es mejorar la generalización evitando utilizar etiquetas y perdidas de clasificación durante el entrenamiento.

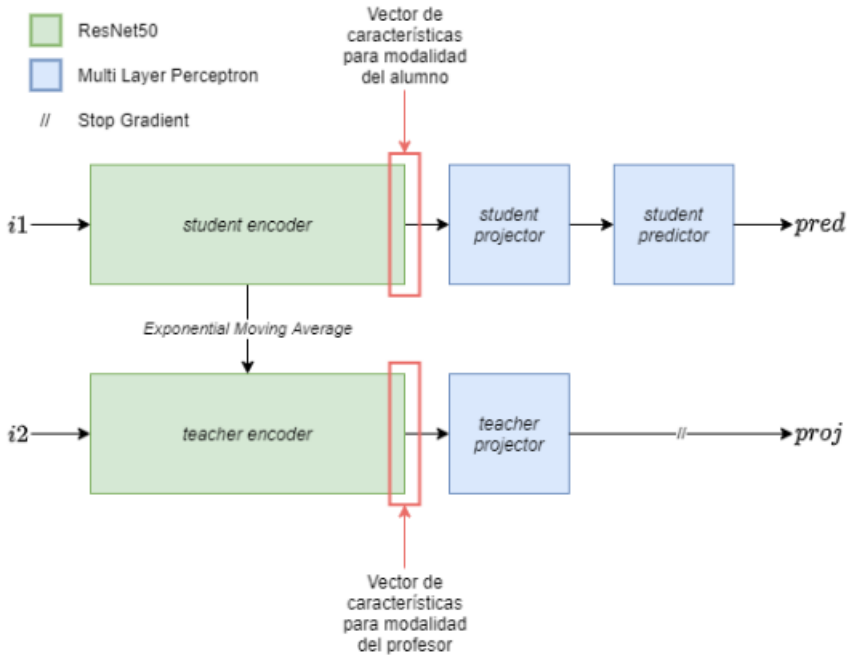


Figura 3.6: Estructura de la red modificada (2021) [5].

Este trabajo de título en si tiene como objetivo la extracción de características, pero su importancia dentro de este trabajo es el uso de imágenes reales y bosquejos dentro de sus entradas, y al tener un modelo que utiliza una *Backbone ResNet-50* para la extracción de características, se considera un buen punto de inicio para considerar parámetros diferentes dentro de las redes originales, incluso llegando a usar su modelo para entrenar nuevos parámetros.

Capítulo 4

Desarrollo

Siendo uno de los principales objetivos de este trabajo la detección de objetos mediante *sketch*, y la entrega de mejores resultados en comparación con el estado del arte, se inició con dos ideas de cómo conseguir el modelo que cumpliera esta tarea. La primera idea que se tuvo fue la recreación del modelo que tenía mejor resultados en detección en esos momentos (*Adaptive Image Transformer for One-Shot Object Detection*) [2], y solo modificarle los datos de entrada para volverlo a entrenar, pero desgraciadamente esto se había intentado antes del comienzo de este trabajo, con el problema de que se obtenían peores resultados al *paper* del modelo, usando los mismos datos, lo que indicaba que este camino no era el indicado.

La segunda idea fue el uso de un algoritmo de código abierto, donde se podía descargar el modelo y usar/entrenar al instante, pero el principal problema de esta solución es que no todos los códigos están disponibles para todo el público, y por desgracia tampoco lo estaba el que se deseaba ocupar.

Debido a que escribir el código desde cero no aseguraba los resultados esperados, y la indisponibilidad de un código ya listo que nos asegure estos resultados, se llegó a la conclusión de que la dirección que se debía tomar era la obtención de un modelo lo bastante parecido al deseado, y modificarlo hasta hacerlo lo más semejante a este, pero el principal obstáculo que se podía presentar era la similitud del modelo, ya que su estructura debería parecerse lo suficiente como para poder llegar al mismo resultado.

El nuevo modelo por modificar debía tener estos rasgos como mínimo: Estar hecho para tareas de detección dentro de imágenes, por lo que debería recibir como mínimo dos entradas, la imagen objetivo donde se realizara la búsqueda, y la imagen de consulta que contiene a la clase que se desea buscar, contener un mecanismo de atención y tener como principal objetivo el *One-Shot Object Detection*. Mediante una búsqueda se encontró un modelo que cumplía casi todos estos puntos, siendo este perteneciente al *paper Few-Shot Object Detection with Attention-RPN and Multi-Relation Detector* [3], pero como se puede deducir de su nombre, esta principalmente diseñado para problemas del tipo *Few-Shot*, pero al ser *One-Shot* un subconjunto de este grupo, se decidió el uso de este modelo como la estructura base (*Baseline*) de este trabajo.

Es importante mencionar y dar los créditos pertinentes al repositorio *MMFewShot*, del cual se obtuvo la *Baseline* y se utilizó de su estructura para el desarrollo de este trabajo, me-

diante las modificaciones necesarias. Este repositorio pertenece al proyecto *OpenMMlab*, del *Multimedia Laboratory* de la Universidad de Hong Kong, siendo este uno de los institutos pioneros en *deep learning*. Este proyecto entrega varias herramientas de trabajo para diferentes problemas de visión computacional, como puede ser detección, clasificación, segmentación, procesamiento, etc.

Lo importante de este repositorio es que está diseñado para trabajar solamente con fotos como entrada, por lo que es necesario modificarlo para que reciba fotos como imágenes objetivo, y bosquejos como entradas de consulta.

4.1. Obtención de pares

Debido a que este trabajo se centra en la detección de objetos, es necesario tener información de lo que se desea buscar y el dónde, siendo estos datos correspondientes a la entrada de consulta y a la imagen objetivo respectivamente. Al querer lograr la localización de objetos mediante *sketch*, es importante señalar que el lugar donde se busca localizar los objetos es dentro de una imagen real, lo que lo haría la imagen objetivo, mientras que, al querer usar bosquejos como guía para encontrar estos objetos, estos pasarían a ser la entrada de consulta.

El modelo por modificar esta escrito en *python*, usando principalmente la librería de *pytorch* para todas las tareas de *deep learning*. Una cosa para recalcar de este código es la buena separación de las partes de su estructura, como puede ser el pre-procesamiento de los *datasets*, *Backbone*, *ROI-Head*, técnica de RPN, etc, por lo que cualquiera de estas partes se podía modificar sin la necesidad de tocar todo el código, con la única necesidad de que información mantenga el formato correspondiente para el siguiente paso.

En la parte de pre-procesamiento del *dataset* fue donde se realizaron mayormente las modificaciones, debido a que era el encargado de unir el par de imágenes objetivo y *query*, haciéndola una parte clave para lograr introducir bosquejos dentro del modelo. Debido a la importancia de este apartado dentro de este trabajo, es necesario entender cómo es su funcionamiento, estructura y todos los pasos que son tomados hasta conseguir el par deseado.

En el repositorio a trabajar hay dos grupos de archivos claves, siendo el más importante los archivos que contiene el código en sí, los cuales son del tipo *"py"*, donde se definen todas las estructuras posibles de cada sección del modelo. Pero el segundo grupo de archivos es el que define al modelo que se usa, siendo estos archivos conocidos como *"config"*, los cuales son del tipo *"txt"*, y contienen toda la información correspondiente de la estructura final del modelo, indicando el valor de todas las variables importantes, cuáles serán las estructuras usadas para cada tarea dentro del modelo, y señalando de donde se obtienen todos los datos con los que deben trabajar. Se pueden considerar los *configs* como la estructura final que se quiere lograr, solo entregando información de lo que se desea, mientras que los archivos *"py"* cumplen tanto el rol de piezas de esta estructura, herramientas para armarla, y las encargadas de obtener todos los datos necesarios, en esta última parte se va a centrar la discusión de esta sección.

Una de las informaciones que posee el *config* es como se debe organizar la información de entrada del modelo, ya que este repositorio está diseñado con la idea de que todas sus partes sean intercambiables, por lo que el tipo de datos con los que se trabajara puede va-

riar, pero para el caso de este trabajo solo es importante uno, siendo este llamado dentro del código como *"QueryAwareDataset"*, la cual se encarga de emparejar imágenes *query* y *target* para tareas de detección, aunque dentro del código se le llama *support* y *query* respectivamente, pero se seguirá usando la misma nomenclatura que se ha usado hasta este momento.

Este repositorio fue hecho principalmente para trabajar con dos conjuntos de datos, PASCAL-VOC y COCO, de los cuales solo se ha explicado la estructura del primero, pero lo necesario saber del conjunto de datos de COCO es que comparándolo con PASCAL-VOC, es más extenso, con más clases y con una mayor cantidad de objetos por imagen, y guarda de manera diferente la información de las imágenes, pero dentro de los límites de este trabajo solo se usará PASCAL-VOC para comparar los modelos, ya que al ser más sencilla que su compañera, será un buen punto de partida para hacer comparaciones. Estos dos conjuntos de datos se pueden usar de forma directa, siguiendo los pasos que da el mismo código, pero en el caso de datos externos, como sería el caso de los bosquejos, se debe ajustar sus salidas dependiendo de la base de datos a la que esta complementando, por lo que si el *dataset* de las imágenes objetivos son obtenidas de PASCAL-VOC, se deberá entregar los mismos resultados que este, y lo mismo para COCO.

Al solo usar PASCAL-VOC dentro de este trabajo, es necesario saber cómo se obtienen todos los datos importantes que se usarán dento del modelo, por lo que se repasará el código encargado de esta tarea. Desde el *"config"* se establece la ruta de donde se van a obtener los datos, y al trabajar con PASCAL-VOC basta con revisar un archivo *txt* de una de sus carpetas, donde posee el nombre de todas las imágenes del conjunto revisado (entrenamiento, validación o testo), por lo que se puede acceder a la información de estas directamente, ya sea mediante su archivo *xml* o solo obteniendo la imagen en si. Esto se puede observar en la entrada de la figura 4.1, donde para el caso de la imagen solamente se usa una dirección, mientras que para los bosquejos se tiene una ruta para cada clase, ya que cada una de sus carpetas posee un archivo que enlista todos los elementos que posee para cada conjunto, y se realizo de esta forma debido al tipo de estructura que poseía este conjunto de datos.

La Clases de *python "FewShotVOCDataset"* y *"DrawShotVOCDataset"* se encargan de obtener una lista de todas las imágenes, y *sketch*, que pertenecen al conjunto revisado (entrenamiento, validación o testo), teniendo la información de la id de la imagen, que solamente se usa para ordenarlo dentro de la lista, el nombre de la imagen, sus dimensiones, y un archivo *annotations* (ann), que posee información sobre los objetos dentro de las imágenes y bosquejos, obteniéndose los *bounding boxes* y sus clases mediante los archivos *xml* en el primer caso, y para los bosquejos se obtiene mediante la dimensión de estos y a la carpeta que pertenecesn respectivamente. En la figura 4.1 se puede observar este proceso en la mitad de la imagen, donde se observa que para ambos conjuntos se tiene los mismos datos, con la diferencia que los *filename* cambian para cada caso, al tener diferentes tipos de rutas para cada caso. Otra cosa importante para mencionar es las anotaciones de cada una, ya que PASCAL-VOC puede poseer más de una *bounding box* de diferentes tamaños en la figura, incluso con diferentes clases, pero para el caso de los bosquejos no ocurre esto, ya que la única *bounding box* que posee rodea a la totalidad del *sketch*, debido a que todo su contenido solo debe poseer un objeto de una única clase que ocupa todo el espacio.

Finalmente, mediante el *QueryAwareDataset* se obtiene el par *query-target*, al obtener dos

elementos de las listas anteriores que comparten una clase. Lo particular del modelo con el que se trabaja es que usa el método *"N-Way"*, donde se puede tener más de una imagen para el *query*, una como un ejemplo positivo que comparte clase con el *target*, y otro como un ejemplo negativo, que puede poseer cualquier clase diferente a la del *target*, y de esta forma se obtienen mejores resultados que el método tradicional de 1-1 [3]. La primera parte del modelo que trabaja con este resultado son las *Backbone*, las cuales reciben información sobre la imagen, como son sus rutas, tamaños y pre-procesamiento recibido, también recibe la imagen pre-procesada dentro de un tensor, las *bounding boxes* que poseen, y las clases que contienen. En la figura 4.1 se puede observar todos los pasos que se mencionaron anteriormente para la obtención del *dataset* con el que trabajara el modelo.

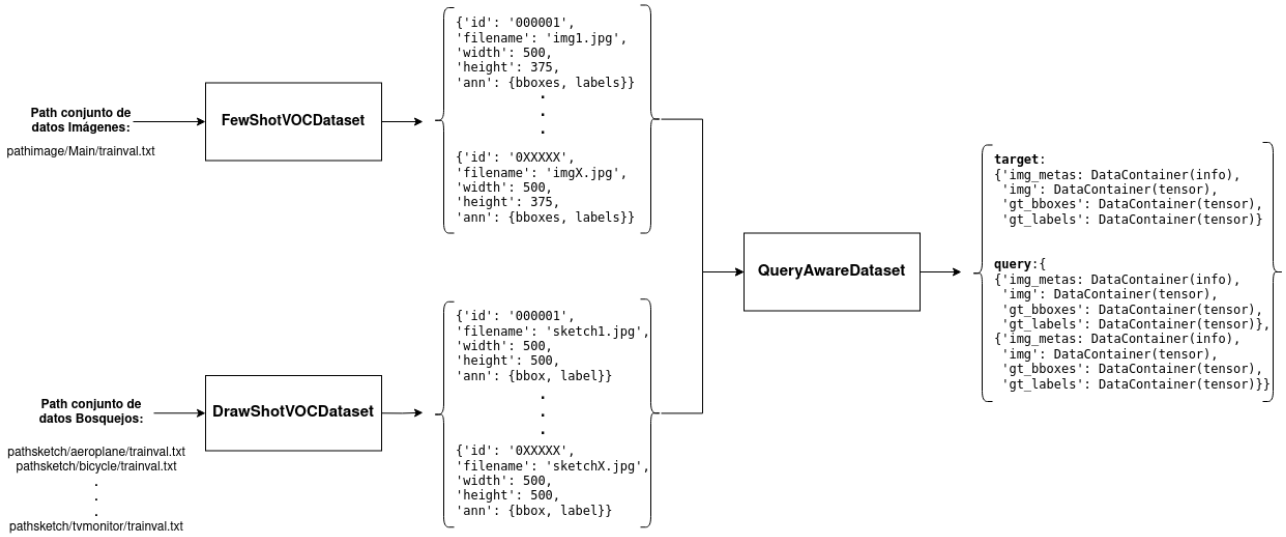


Figura 4.1: Flujo de datos para obtención de *dataset* de entrada del modelo.

Dentro del proceso antes mencionado, las modificaciones que se debieron hacer fue dentro del código de *"DrawShotVOCDataset"*, siendo creada desde cero usando como molde al *"'FewShotVOCDataset'*, ya que debía tener el mismo formato de salida. Este trabajo no fue tan sencillo como parece, ya que si bien, el repositorio usado recibía datos *query* y *target*, estaba realmente fabricado para que estas dos se obtuvieran del mismo conjunto de datos, por lo que fue necesario un buen entendimiento de todo el código encargado de esta parte, siendo importante la modificación de partes puntuales para usar dos conjuntos diferentes.

Si bien, dentro del *config* se especificaba de donde se obtenían los datos para la fabricación del *dataset*, se tenía la particularidad de tener que repetir este proceso para cada fase del modelo, de entrenamiento, validación, testeo y *modelinit*. Solamente dentro del conjunto de entrenamiento se podía formar el par *query* y *target* mediante el *QueryAwareDataset*, ya que todos los otros solo poseían un elemento del paso anterior al emparejamiento, siendo el *target* para el caso de validación y testeo, con el *FewShotVOCDataset*, mientras que *modelinit* era el encargado de guardar la información del *query*, con el *DrawShotVOCDataset*. Por lo que para los pasos de testeo y validación se unían estos respectivos *datasets* con el del *modelinit*.

Habían códigos y *datasets* diferentes para el entrenamiento/validación y testeo, ya que

dentro del primer grupo solo se debían ingresar datos de las imágenes que pertenecían a las clases vistas, mientras que el segundo se incluían estas mismas imágenes, pero agregándole las clases no vistas, por lo que este último utilizaba la totalidad de los conjunto de datos disponibles. En la figura 4.2 se puede observar los *datasets* que se generan, siendo cuatro necesarios para el entrenamiento, dos con ruta a las imágenes de entrenamiento y dos con ruta a las imágenes de validación, con solo imágenes con una cantidad limitada de clases, mientras que para el testeo se necesitan solo dos *datasets*, teniendo ruta a las imágenes de prueba, conteniendo todas las clases disponibles.

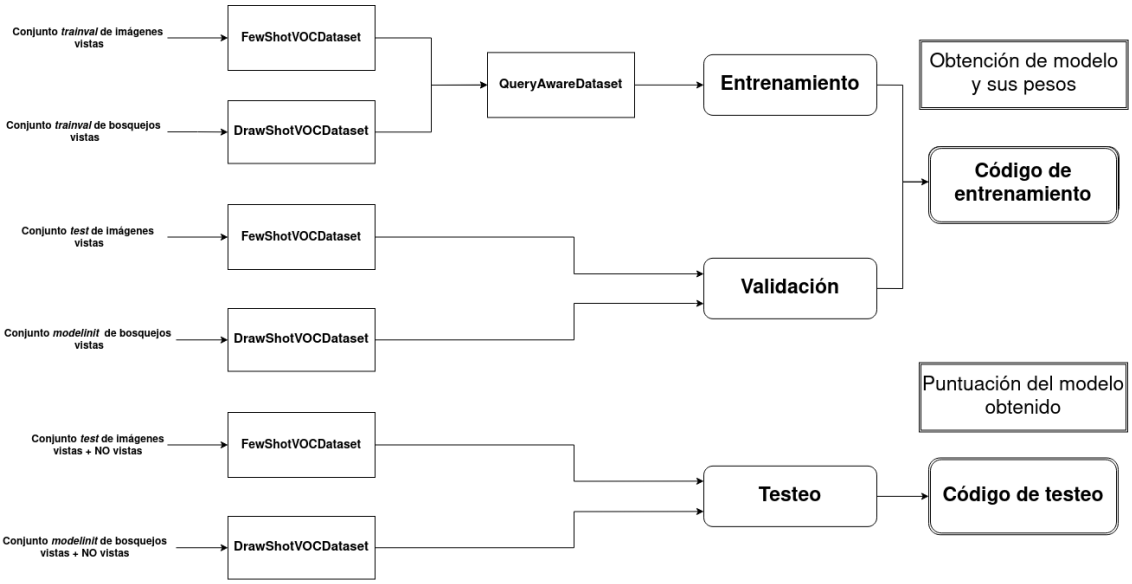


Figura 4.2: Todos los datasets que se usan *Datasets*.

4.2. Modificación al modelo base

Ya obtenidos los conjuntos de datos de cada etapa, se ingresan al modelo para obtener los resultados preliminares del *baseline*, para tener un punto de comparación con los otros modelos modificados, pudiendo ser todos estos resultados observados en la sección 5. Van a haber tres grandes modificaciones dentro del modelo base, siendo uno relacionado a su pre-procesamiento de imágenes, otro con modificaciones en los *Backbones* y finalmente un modelo que modificará por completo una parte de la estructura de modelo.

4.2.1. Modificaciones al Pre-procesamiento

Las primeras modificaciones realizadas al modelo base fue dentro del tratamiento de las entradas, pero solamente a las relacionadas con los bosquejos, ya que las que afectan a las imágenes se consideran las óptimas al ser las mismas con las que se realizó el código original. El *Baseline* contiene los siguientes tratamientos para las entradas: *"CropResizeInstance"*, *"RandomFlip"* y *"Normalize"*, la primera recorta solo una parte de la imagen siendo esta la que es utilizada al final, *"RandomFlip"* voltea el *sketch* de manera aleatoria y *"Normalize"* normaliza a los bosquejos con un valor predeterminado. Por lo que los primeros experimen-

tos serán relacionado a esto, eliminando uno de estos procesos para ver cómo afecta a los resultados, como se puede ver en la figura 4.3, donde las imágenes solo poseen un tipo de procesamiento, mientras que los bosquejos pueden tener un tratamiento eliminado.

Otra modificación que se le realizará al *Baseline* es el cambio al tamaño de los *batch*, donde el original, al tener 3 GPU's en el servidor utilizado, se divide el trabajo de procesamiento del *batch* en cada una de estas, siendo en este caso 2 *samples* por GPU, y por ende, un *batch* de tamaño 6. Se usaron otros dos valores para el tamaño del *batch*, siendo 18 y 27 los elegidos, siendo este último el máximo posible por los límites de recursos usados por el servidor.

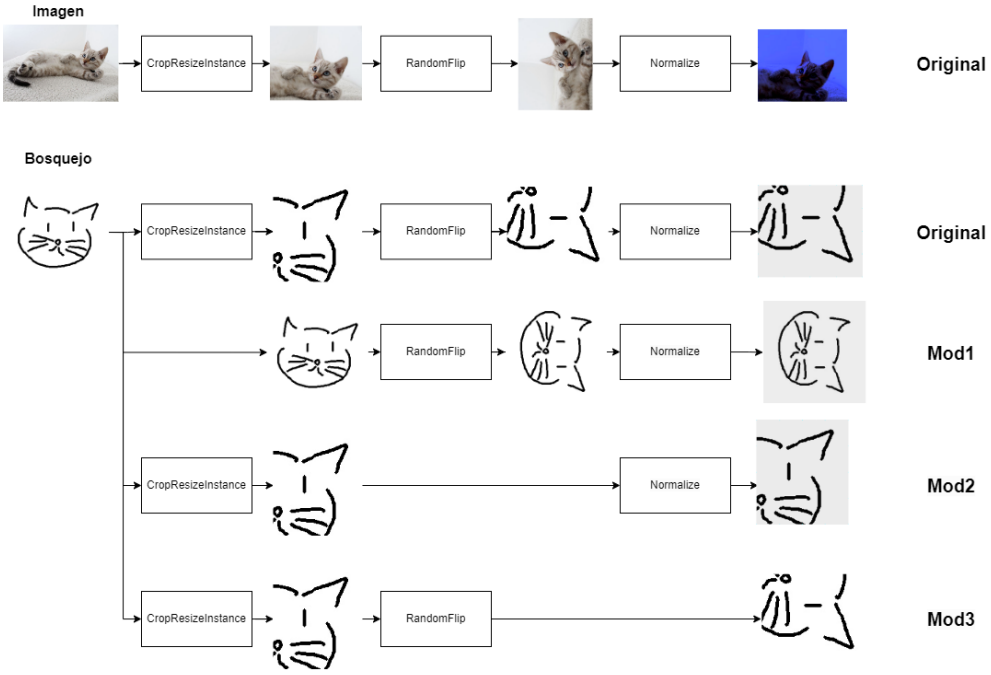


Figura 4.3: Tipos de modificaciones posibles a la imágenes/bosquejos

4.2.2. Modificaciones al *Backbone*

El modelo al usar *Backbones* para la extracción de características necesita cargar parámetros a su red para una mejora de resultados, siendo obtenidas directamente desde el mismo repositorio, pero al ser solamente consideradas imágenes en la construcción de estas redes, se tuvo la idea de modificarlos por otros pesos que pertenecieran al mismo tipo de *Backbone* que se está usando en el código original, siendo este una *ResNet-50*, la cual es usada en una gran variedad de modelos por sus buenos resultados. Lo óptimo sería usar parámetros entrenados tanto con imágenes reales como con bosquejos, por lo que todo problema que use estos dos tipos de entrada será de interés. El trabajo de Javier Morales Rodríguez, *´´Evaluación de métodos auto-supervisados y semi-supervisados para la extracción de características visuales en el contexto de recuperación de imágenes basadas en dibujos"* [5], tiene como una de sus soluciones una red *BYOL*, la cual también utiliza una *ResNet-50* para la extracción de características, y las entrena usando tanto imágenes reales como bosquejos. Otra característica importante de esta elección es el objetivo principal de la arquitectura de *BYOL*, ya que busca obtener características que permitan determinar si dos elementos pertenecen a la misma

clase, lo que la hace la candidata perfecta para la obtención de los parámetros que se usarán.

BYOL utiliza dos *ResNet-50* en su estructura, las cuales contienen diferentes valores en sus parámetros, siendo conocidas como *target-encoder* y *online-encoder*, por lo que usando los pesos de estas redes entrenadas se harán varias combinaciones junto a los parámetros obtenidos del repositorio original, viendo diferentes casos dentro del *Baseline*. Se modificará toda parte del modelo que utilice alguna *ResNet-50* dentro de su estructura, por lo que tanto los *Backbones* como los *ROI-Head* verán modificados sus parámetros para ver si la introducción de esta nueva red puede aportar en la solución de este problema.

Dentro de estas modificaciones habían 4 diferentes opciones de donde obtener los parámetros de la red *BYOL*, siendo la primera opción el uso directo de los pesos usados dentro de esta red, la cual se entrenó usando la base de datos de *Imagenet*, para las imágenes objetivos, junto a la de *sketchy*, para las entradas *query*. La segunda opción que se tenía era el uso de los pesos obtenidos en el trabajo de Javier [5], donde usando los mismos pesos que los de la primera opción, se realizó un *fine-tunning* con ejemplos de zapatos. La tercera opción era entrenar la red *BYOL* usando la totalidad de conjunto de datos usados dentro de este trabajo (PASCAL-VOC y *Quick, Draw*). Y finalmente, la última opción era similar a la anterior, pero en vez de usar todo el conjunto de datos, solamente se usarían los datos que contuvieran a las clases vistas.

4.3. Modelo con *transformers*

Otra modificación importante dentro de la red será el cambio del mecanismo usado para el *Region Proposal Network* (RPN), cambiando el de tipo *attention* por uno que utiliza *transformers*. La principal idea de este intercambio era dejar el modelo final parecido al presentado dentro del paper *´´CAT: Cross-Attention Transformer for One-Shot Object Detection"* [4], pero solamente logrando la estructura mostrada en la figura 4.4, donde se puede observar que solamente la imagen objetivo es procesada dentro del *transformer* en su totalidad, mientras que el *query* es usado en en su forma original en la última sección del modelo. Se usarán tantos los parámetros originales como los obtenidos de *BYOL*, viendo si hay alguna diferencia y/o mejora con respecto al modelo original.

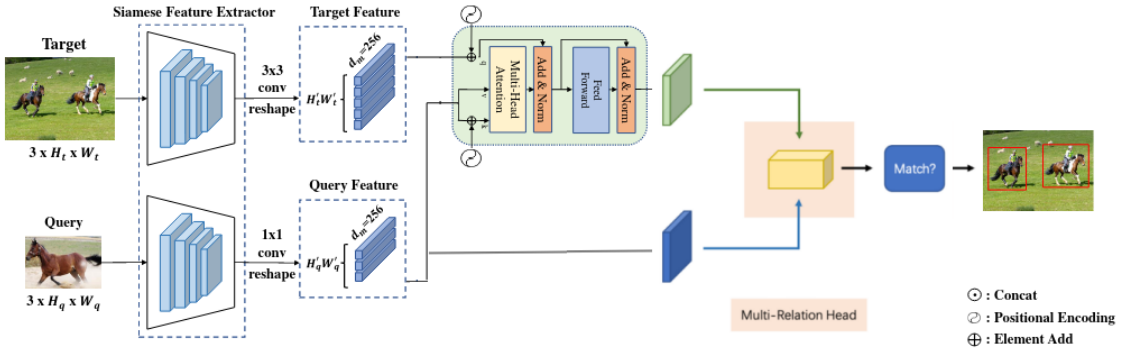


Figura 4.4: Estructura final del modelo utilizado.

Los experimentos realizados con todas estas redes obtenidas se pueden observar en la sección 5, de Resultados y Discusiones, donde se detallará lo realizado dentro de cada experimento, y lo rescatable dentro de cada una de las modificaciones realizadas. No se mezclaron modificaciones generales dentro de experimento, por lo que solamente se verán resultados donde se haya modificado únicamente el pre-procesamiento, los *Backbones* o el método RPN, aunque en este último caso se hizo la excepción de también agregarle los nuevos pesos obtenidos de BYOL, para observar cómo afectaba en este nuevo modelo

Capítulo 5

Resultados y Discusiones

Con el modelo base obtenido desde el paper *Few-Shot Object Detection with Attention-RPN and Multi-Relation Detector* [3], se hicieron las modificaciones necesarias para tener bosquejos como entradas *query*. Pero este no fue el único cambio realizado dentro de su estructura, ya que al trabajar con diferentes tipos de datos que los originales, pueden haber pequeñas modificaciones dentro de los parámetros o hyper-parámetros que mejoren los resultados finales, por lo que la totalidad de estos se pueden dividir de tres maneras, siendo los primeros resultados correspondientes al modelo base (Baseline), donde la única modificación que se realiza dentro del modelo es la entrada que recibe, por lo que es la versión más parecida al obtenido del repositorio de *openmmlab*. El segundo tipo de resultados son los pertenecientes a las variaciones de este modelo base, donde la estructura se mantiene semejante al original, pero realizando pequeñas modificaciones en ciertos aspectos, como puede ser pre-procesamiento de las imágenes, tamaño de *batch*, o cambio de los pesos originales de algunas redes internas por otros. Finalmente, el ultimo tipo de resultado es la modificación del modelo base en sí, donde ya no se puede considerar como tal al modificar un aspecto importante dentro de este, siendo este el cambio del método usado para el *Region Proposal Network* (RPN), pasando desde un mecanismo de atención (*Attention-RPN*) a uno que usa *transformers (Transformers-RPN)*.

Debido a la naturaleza de este trabajo, hay dos resultados que toman critica importancia, siendo la métrica de las clases vistas y las no vistas. Donde se recalcará los modelos que obtengan mejores valores en cada caso.

5.1. Configuraciones iniciales

Dentro de los *configs* hubo parámetros que se prefirió no modificarlos, debido a que no se consideró importante dentro del problema a resolver, y otros simplemente no se cambiaron porque emperoraban los resultados, como es en el caso del *learning rate* del optimizador *SGD*, donde dentro de los archivos del código se recomendaba usar la formula: $\#GPUs \cdot \frac{lr}{8}$, donde el valor original del *learning rate* (lr) era igual a $0,004$, por lo que el servidor al tener 3 GPU's, se debía usar un lr igual a $0,0015$, lo que empeoraba los resultados en comparación a usar el lr original. Finalmente, el optimizador que se eligió usar fue el que venía por defecto, un *SGD* con lr=0,004 y un *momentum=0,9*.

Para todos los experimentos se usó un ejemplo positivo y otro negativo para los *query*, debido a la mejora de resultados que se mostraban dentro del repositorio al usar esta técnica. Dentro de la estructura no se cambió el tipo de *Backbone* que se usaba, una *ResNet-50*, pero si el valor de sus parámetros.

Dentro del RPN se usaba la técnica de *Non Maximum Suppression*, la que evita un superposición de entidades al solo considerar una, lo que disminuye la cantidad de propuestas hechas para un mismo objeto, y en este caso un *threshold* de 0.7.

Se eligió una cantidad de 36000 iteraciones para el entrenamiento, lo que en tiempo real equivaldría a 8 horas aproximadamente, siendo este el elegido debido a que el aumento de este valor no mostraba una mejora en sus resultados, solamente aumentado el tiempo de entrenamiento necesario. Cada 6000 iteraciones se guardaban los *Checkpoints* (pesos) del modelo, por lo que para cada experimento se obtenían 6 diferentes *checkpoints*, del cual solamente era usado el ultimo para la comparación.

5.2. Resultados del estado del arte

Como se había mencionado en 3.1, los resultados del *paper Sketch-Guided Object Localization in Natural Images* [7] son considerados como los mejores hasta este momento, debido a ser uno de los únicos trabajos que se han hecho con respecto a este mismo tema, por lo que sus resultados serán importantes para saber que tan buenos son los obtenidos dentro de este trabajo. Dentro de este mismo *paper* también contenía resultados de otros modelos usados para la misma tarea, por lo que también se agregarán.

Tabla 5.1: Resultados del estado del arte.

Resultados usando PASCAL-VOC

Modelo	mAP
Faster-R-CNN modificado	**0.65**
Matchnet	0.61
Sketch-guided [7]	**0.65**

Resultados usando COCO

	mAP	
Modelo	Clases Vistas	Clases No Vistas
Faster-R-CNN modificado	0.074	0.345
Matchnet	0.124	**0.491**
Sketch-guided [7]	**0.150**	0.488

Se usan dos diferentes conjuntos de datos para la obtención de los resultados, siendo estos PASCAL-VOC y COCO, pero con la desgracia que solamente para este último se hace las pruebas tanto para las clases vistas como las no vistas, por lo que dentro de este trabajo se va a centrar más en los de PASCAL-VOC. En la tabla 5.1 se pueden observar estos resultados, viéndose que solamente con el *dataset* de COCO se separan en clases vistas y no vistas, mientras que el PASCAL-VOC solo se posee información en las clases vistas.

5.3. Resultados en modelo base (*Baseline*)

Se observará los resultados dados en tres diferentes divisiones del mismo *dataset*, conteniendo la misma imágenes para entrenamiento y testeo, pero cambiando las clases que son consideradas dentro del entrenamiento, siendo las clases vistas la que participarán en este, y las no vistas solo consideradas dentro del testeo.

Solo se mostrará la puntuación de cada clase en estos resultados, para hacer visible como se consideran los puntajes del mAP y como se divide cada *split*.

Tabla 5.2: Resultados del *Split* 1.

Clases Vistas	**Recall**	**AP50**
Aeroplane	0,846	0,763
Bicycle	0,958	0,781
Boat	0,856	0,677
Car	0,923	0,846
Cat	0,978	0,853
Chair	0,836	0,579
Diningtable	0,932	0,589
Dog	0,984	0,792
Horse	0,960	0,797
Sheep	0,876	0,661
Train	0,957	0,754
Tvmonitor	0,890	0,727
	mAP	**0,735**

Clases No Vistas	**Recall**	**AP50**
Bird	0,723	0,083
Bus	0,901	0,050
Cow	0,951	0,153
Motorbike	0,880	0,210
Sofa	0,782	0,009
	mAP	**0,101**

	Recall	**AP50**
All Classes	**0,896**	**0,548**

Tabla 5.3: Resultados del *Split* 2.

Clases Vistas	**Recall**	**AP50**
Bicycle	0,947	0,813
Bird	0,852	0,739
Boat	0,852	0,657
Bus	0,962	0,827
Car	0,932	0,858
Cat	0,969	0,865
Cair	0,861	0,588
Dinningtable	0,922	0,545
Dog	0,984	0,700
Motorbike	0,917	0,780
Sheep	0,884	0,608
Train	0,940	0,816
	mAP	**0,733**

Clases No Vistas	**Recall**	**AP50**
Tvmonitor	0,185	0,001
Aeroplane	0,467	0,025
Cow	0,959	0,111
Horse	0,891	0,162
Sofa	0,707	0,009
	mAP	**0,062**

	Recall	**AP50**
All classes	**0,837**	**0,536**

Tabla 5.4: Resultados del *Split* 3.

Clases Vistas	**Recall**	**AP50**
Aeroplane	0,835	0,771
Bicycle	0,947	0,808
Bird	0,832	0,736
Bus	0,958	0,804
Car	0,923	0,847
Chair	0,841	0,583
Cow	0,955	0,693
Diningtable	0,898	0,568
Dog	0,980	0,714
Horse	0,954	0,855
Train	0,933	0,831
Tvmonitor	0,890	0,734
	mAP	**0,745**

Clases No Vistas	**Recall**	**AP50**
Boat	0,490	0,021
Cat	0,916	0,195
Motorbike	0,834	0,196
Sheep	0,831	0,096
Sofa	0,561	0,006
	mAP	**0,103**

	Recall	**AP50**
All classes	**0,858**	**0,556**

5.4. Resultados en modelo modificado

En este caso solo se mostrará los puntajes mAP tanto de las clases vistas como no vistas solo del *split* 1, y de esta forma conseguir que modificaciones entrega mejores resultados.

Los resultados van a pertenecer a las modificaciones del modelo base y al modelo de *transformers*, siendo este primer grupo dividido en diferentes tipos de modificaciones, ya sea los diferentes pre-procesamientos a las imágenes, diferente tamaño del *batch* en el entrenamiento, y el cambio de los pesos en las Resnets usadas. Es necesario mencionar que estas modificaciones son excluyentes entre ellas, por lo que a cada modelo solo se le ha modificado una parte de la estructura, y todo aspecto que se no sea mencionado dentro modificaciones, exceptuando los pesos, será semejante al del modelo base, que no contiene ninguna modificación y solo posee pesos obtenidos desde el repositorio original para las *Resnet-50*. El modelo con *transformers* solo se le modificarán los pesos de sus Resnets, ya que como se ha mencionado anteriormente, se desea ver como se consiguen mejores resultados, y al cambiar el método para la parte de RPN quizás funciones mejor con pesos diferentes.

Los *Backbones* de la entrada de consulta y de la imagen objetivo no necesariamente tendrán los mismos pesos, pero siempre compartirán la misma estructura, siendo esta una *Resnet-50*. Las primeras capas de los dos *Backbones* serán congeladas en el entrenamiento, por lo que no se modificarán de inicio a fin. Hay dos tipos de pesos posibles de usar, siendo uno el obtenido del repositorio, y usado en el modelo original, conocido como *Resnet50*, mientras que el obtenido mediante el uso del modelo *BYOL* será conocido como *Resnet-byol*. Dentro de los pesos *Resnet-byol* hay 4 diferentes versiones, siendo la primera la original obtenida directamente del trabajo de titulación de Javier Morales [5], la segunda también perteneciente a este trabajo, pero siendo el obtenido como resultado final después de un *fine-tunning* con sus datos de trabajo (*Resnet-byol-V2*), el tercero es el *fine-tunning* con todos los objetos obtenidos de los conjunto de datos usados dentro de este trabajo, siendo estos *Quick, Draw!* y *Pascal-VOC* (*Resnet-byol-Q*), y el ultimo es semejante al anterior, pero con la diferencia que solo se usó los objetos pertenecientes a las clases vistas, y excluyendo a las no vistas (*Resnet-byol-QN*).

Es necesario mencionar que usando el método de *Attention-RPN*, la estructura del ROI head usará las ultimas capas de una *Resnet*, por lo que también se procede a modificar estos pesos. Al usar la estructura del *Resnet-50* también se pueden utilizar los pesos que usa el *Backbone*, pero con la diferencia que todos sus pesos pueden ser modificados, al no tener ninguna capa congelada.

Las primeras modificaciones serán solo al pre-procesamiento de las imágenes, como se pude ver en la primera sub-tabla la tabla 5.5, observando que al cambiar el valor de la rotación aleatoria de 0 a 0,5 aumento sus resultados con respecto a al modelo original. En la segunda sub-tabla se puede observar que al aumentar el tamaño de los *batch* solo se mejora los resultados de las clases vistas, y disminuyendo para las no vistas, esto se le atribuye al *overfitting* en el entrenamiento con las clases vistas. En la tercera sub-tabla, y donde se hicieron la mayoría de los experimentos, se probó diferentes combinaciones para los parámetros de la *Backbone* y la *ROI-Head*, siendo destacables dos resultados importantes, siendo el primero un experimento para ver qué sucederá si la *Backbone* no es entrenada previamente para los bosquejos, donde los resultados obtenidos demuestran que es imposible

la detección de los objetos buscados, y el segundo tipo de resultado importante es la obtención de la mejor combinación, la cual entrega los mejores resultados de todos los modelos, siendo logrados al usar casi toda la estructura original, exceptuando por los pesos de la *Backbone* de las entradas *query*, la cual fue cargada con los parámetros entrenados dentro de la *BYOL* usando lo *datasets* de *Imagenet* y *Sketchy*. Y finalmente, al cambiar el método de la RPN, se obtuvieron peores resultados para todos los casos probados.

Tabla 5.5: Resultados de diferentes modificaciones del modelo base.

				Clases Vistas	**Clases no vistas**
Método	**Modificaciones**	**Backbone del target**	**Backbone del query**	**mAP**	**mAP**
Attention RPN	Ninguna	Resnet50	Resnet50	0,735	0,101
Attention RPN	Flip 0,5	Resnet50	Resnet50	0,708	0,123
Attention RPN	No Crop	Resnet50	Resnet50	0,733	0,104
Attention RPN	No Mean	Resnet50	Resnet50	0,244	0,011
	Tamaño de Batch				
Attention RPN	16 por Batch	Resnet50	Resnet50	**0,752**	0,083
Attention RPN	27 por Batch	Resnet50	Resnet50	0,741	0,087
	Cambio de pesos de Backbones y Roi-Head (RH)				
Attention RPN	No carga pesos en RH	Resnet-Byol online	Resnet-Byol online	0,140	0,045
Attention RPN	Resnet-byol online en RH	Resnet-Byol online	Resnet-Byol online	0,256	0,069
Attention RPN	Resnet-byol online en RH	Resnet50	Resnet50	0,742	0,102
Attention RPN	Resnet50 en RH	Resnet-Byol online	Resnet-Byol online	0,252	0,070
Attention RPN	Resnet50 en RH	Resnet50	Resnet-Byol online	0,728	**0,131**
Attention RPN	Resnet50 en RH	Resnet50	Resnet-Byol-V2 online	0,729	0,096
Attention RPN	Resnet50 en RH	Resnet50	Ninguna	0,000	0,000
Attention RPN	Resnet50 en RH	Resnet50	Resnet-Byol target	0,715	0,113
Attention RPN	Resnet50 en RH	Resnet-Byol online	Resnet-Byol target	0,244	0,078
Attention RPN	Resnet-byol online en RH	Resnet-50	Resnet-Byol target	0,726	0,124
Attention RPN	Resnet50 en RH	Resnet-Byol-Q online	Resnet-Byol-Q target	0,238	0,068
Attention RPN	Resnet-byol-Q online en RH	Resnet50	Resnet-Byol-Q target	0,725	0,117
Attention RPN	Resnet50 en RH	Resnet-Byol-QN online	Resnet-Byol-QN target	0,244	0,050
Attention RPN	Resnet-byol-QN online en RH	Resnet50	Resnet-Byol-QN target	0,726	0,118
	Cambio de metodo				
Transformer RPN	-	Resnet50	Resnet50	0,493	0,059
Transformer RPN	-	Resnet-Byol online	Resnet-Byol target	0,193	0,048
Transformer RPN	-	Resnet50	Resnet-Byol target	0,499	0,046

5.5. Discusiones

Primero que nada, es importante recordar que los dos objetivos más importantes dentro de este trabajo son la mejora de resultados en comparación al estado del arte, siendo tanto los resultados de clases vistas como las no vistas fundamentales dentro de este trabajo.

5.5.1. Comparación con el estado del arte

Lo primero que se puede observar al comparar la primera sub-tabla de la tabla 5.1, perteneciente al *State of the Art* (Sota), es que es superada por todo modelo que use los parámetros originales del *Baseline* para el conjunto de las clases vistas, desde el modelo base con sus modificaciones, hasta los que combinan los pesos originales con los entrenados en *BYOL*.

Pero debido a que el modelo Sota no contiene información de las clases no vistas dentro del *dataset* de PASCAL-VOC, no se puede hacer una comparación directa, pero si una indirecta con los resultados del conjunto de datos de *COCO*, ya que estos al contener una mayor cantidad de imágenes, clases, y objetos por imagen, se espera una disminución en el puntaje de los resultados, lo cual se puede ver dentro de las clases vistas al comparar las dos sub-tablas de la tabla 5.1, por lo que de igual forma se esperaría una baja en los resultados de las clases no vistas en comparación a PASCAL-VOC. Esto indica que todos los modelos que se obtuvieron son peor en la detección de las clases no vistas, siendo el mejor resultado obtenido dentro de PASCAL-VOC es igual a 0.131, mientras que en COCO el modelo *SOTA* obtuvo 0.15, que al ser calculado dentro de este *dataset* aumenta mucho más la diferencia.

5.5.2. Modificación del modelo

Un detalle importante a mencionar dentro de los resultados de los modelos modificados en el pre-procesamiento, es la importancia del uso del proceso del *normalize* dentro de todos los conjuntos de datos, ya que el no uso de este degrada los resultados. Otro detalle importante de observar son los resultados al aumentar el tamaño del *batch*, que si bien, aumenta el mAP de las clases vistas con 16 elementos por *batch*, se empeora el de las clases no vistas, siendo esto atribuido a un *overfitting* con las clases vistas.

5.5.3. Usar pesos obtenidos de *BYOL*

Otro detalle importante para discutir es el uso exclusivo de parámetros entrenados dentro de *BYOL*, que empeoran de forma notable los resultados entregados, teniendo como máximo puntaje un 0.256 en las clases vistas, y esto se le atribuye a cambiar los parámetros de la *Backbone* perteneciente a la imagen objetivo, ya que al mezclar los pesos originales con los de *BYOL* no se ve realmente perjudicado los resultados. Debido a esto, es necesario mantener los pesos originales en la *backbone* del *target*, ya que a pesar de modificar los pesos de la *backbone* del *query*, pero no los del *target*, se obtienen de los mejores resultados, teniendo incluso el mejor de estos con la *Resnet-Byol online*.

Al entrenar nuevos parámetros con *BYOL*, teniendo los mismos *datasets* que son usados dentro de este trabajo (PASCAL-VOC y *Quick, Draw!*), no se obtiene mejora de ningún tipo, y ni siquiera se nota la diferencia al considerar, o no, las clases no vistas dentro de estos entrenamientos. Por lo que se deduce que *BYOL* no aporta mucho dentro de estas soluciones, pero de igual manera aporta lo suficiente como para no estorbar, ya que al tener el *backbone* de la *query* sin ningún parámetro entrenado, se pudo observar que le es imposible obtener algún match incluso no teniendo ninguna de sus capas congeladas, por lo que entrenando con *BYOL* dentro de bosquejos, se obtienen iguales resultados que la *ResNet* del repositorio, mejorándolos un poco incluso.

5.5.4. Modelo con *transformers*

Otra cosa importante de observar es el bajo desempeño del modelo que usa *transformers*, que todos sus resultados son aproximadamente la mitad de los mejores, por lo que se deduce que el modelo obtenido, semi-completo, no es el adecuado, por lo que solo se puede tener

resultados concluyentes al tener el modelo con *transformers* que procese cada entrada.

5.5.5. Similitud entre clases

Lo último importante que se observó, fue la importancia de contener objetos parecidos entre las clases vistas y no vistas, ya que en todos los resultados se observó lo mismo con ciertas clases, pero más notable en las tablas 5.2, 5.3 y 5.4, que las clases que obtuvieron mejor puntuación en las clases no vistas fueron las que compartían algún rasgo con los de la clase vista, como es el caso de *Cow* y *Motorbike* en el split 1, *Cow* y *Horse* en el split 2, y *Cat* y *Motorbike* en el split 3. Todos estos tienen un semejante en las clases vistas, como son las clases *Horse* (o *Sheep*) y *Bicycle* respectivamente para el split 1, *Dog* y *Sheep* respectivamente para el split 2, y *Dog* y *Bicycle* respectivamente para el split 3. Por lo que se destaca la importancia de tener un conjunto de datos bien variados, y modelos que puedan generalizar estos rasgos para mejorar la detección de clases que las contengan.

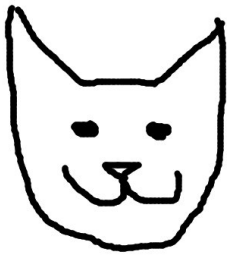
Como curiosidad llama la atención el alto *recall* pero bajo mAP para las clases no vistas, por lo que podría haber un exceso de falsos positivos al tener un *score* de detección muy bajo, lo que disminuiría la precisión.

5.6. Ejemplos de Resultados

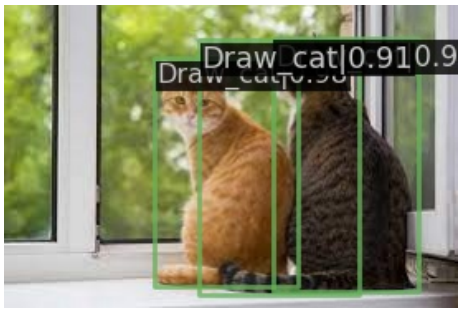
Se mostrará diferentes ejemplos de los resultados obtenidos al usar diferentes entradas en dos modelos diferentes, siendo el primero el modelo base, y el segundo un modelo que entrega los mejores resultados.

Se usarán primero cuatro clases para estos ejemplos, uno perteneciente a las clases vistas (Gato), observado en la figura 5.1, dos a las clases no vistas (Moto y Bus), observados en las figuras 5.2 y 5.3 respectivamente, y una totalmente externa al conjunto de datos usado (Bolso), observado en la figura 5.4.

También se mostrará ejemplos para todos los objetos vistos (Figuras 5.5 y 5.6) o no vistos (figura 5.7) del *dataset* de PASCAL-VOC, usando un *theshold* de 0.8, por lo que todo elemento que tengo un menor valor no será detectado dentro de las imágenes.

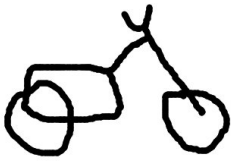


(a) *Sketch* de consulta



(b) Imágenes objetivo (c) Resultados entregados

Figura 5.1: Ejemplos de la clase Gato en modelo base.

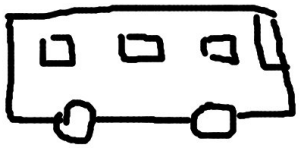


(a) *Sketch* de consulta

(b) Imágenes objetivo

(c) Resultados entregados

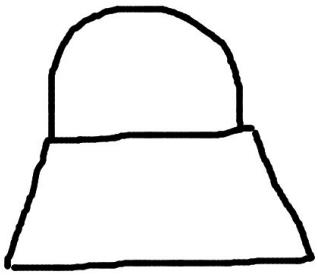
Figura 5.2: Ejemplos de la clase Moto en modelo base.



(a) *Sketch* de consulta

(b) Imágenes objetivo (c) Resultados entregados

Figura 5.3: Ejemplos de la clase Bus en modelo base.



(a) *Sketch* de consulta

(b) Imágenes objetivo (c) Resultados entregados

Figura 5.4: Ejemplos de la clase Bolso en modelo base.

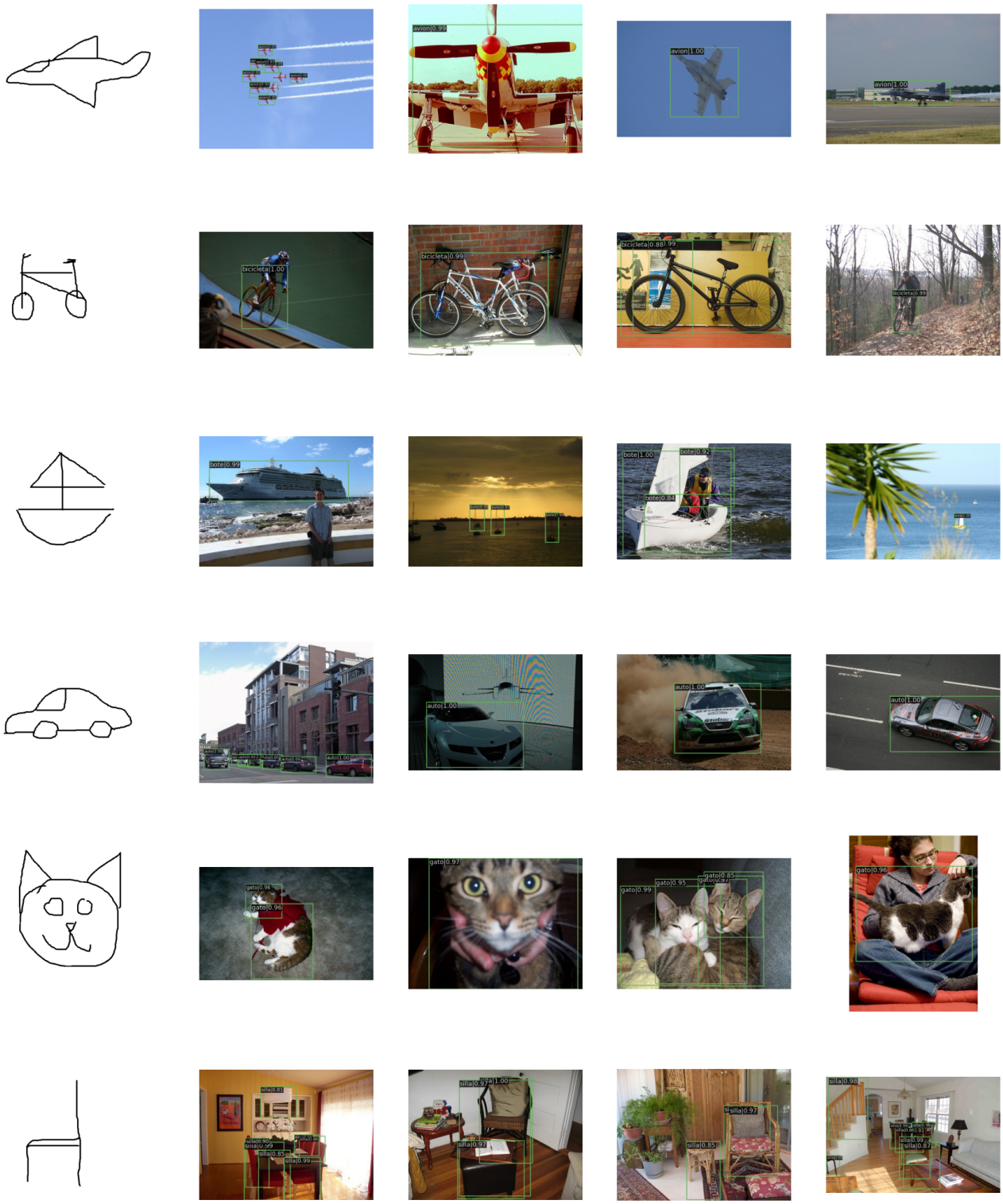


Figura 5.5: Ejemplos para primer grupo clases vistas.

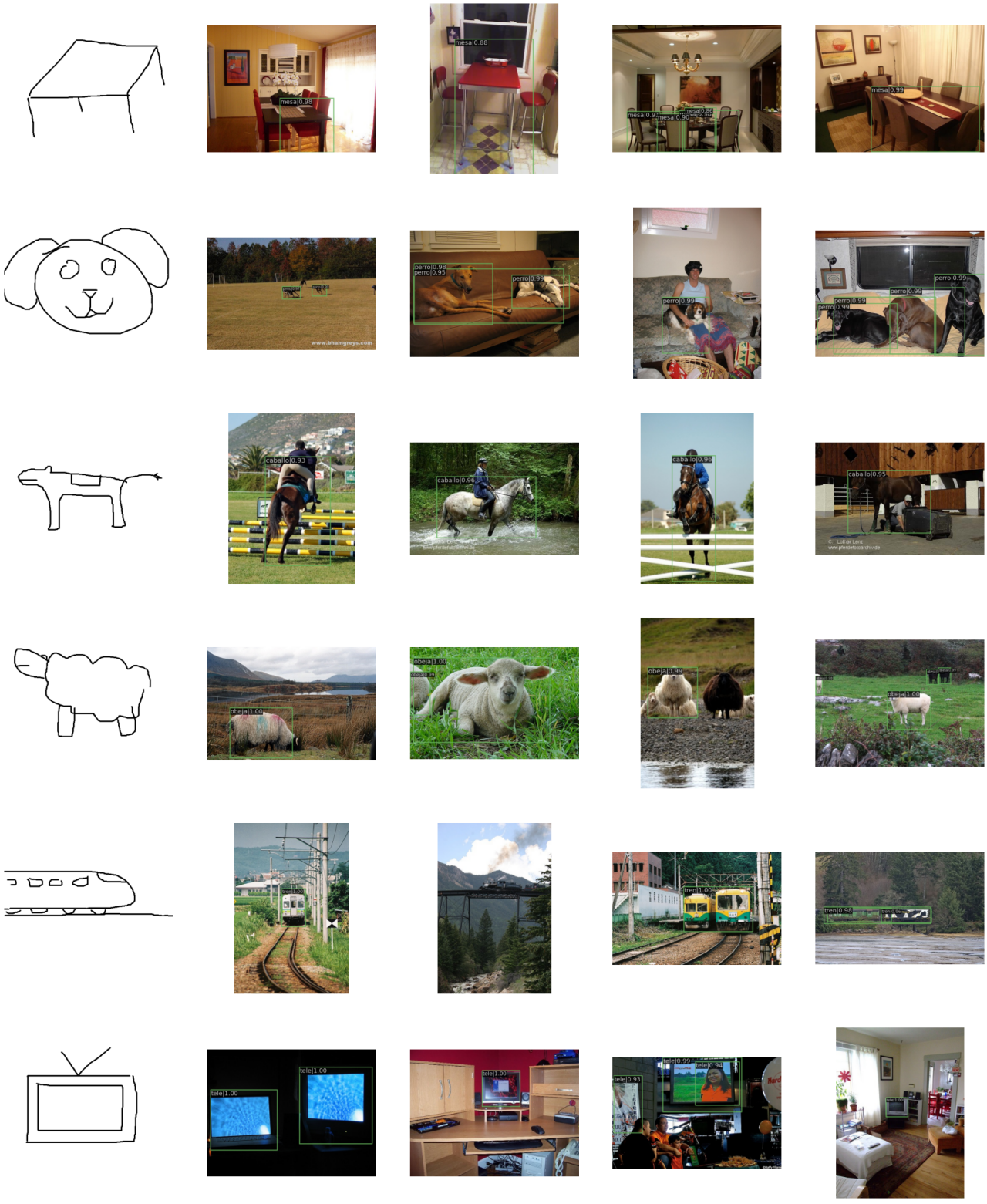


Figura 5.6: Ejemplos para segundo grupo clases vistas.

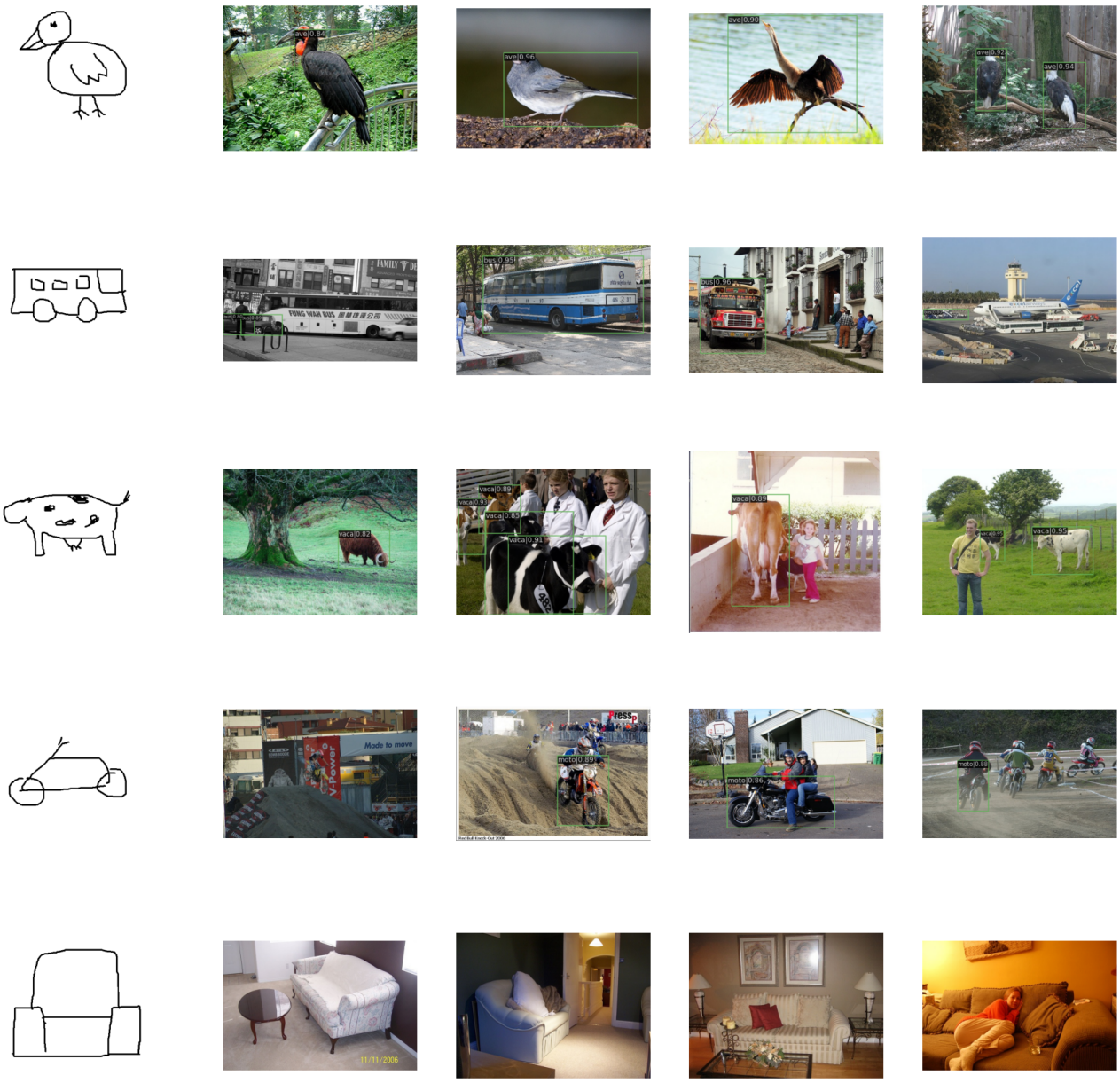


Figura 5.7: Ejemplos para clases no vistas.

Capítulo 6

Conclusión

Se pudo modificar un modelo fabricado para imágenes reales, para que este de igual manera pudiese trabajar con bosquejos como entrada, logrando de esta forma, localizar objetos mediante *sketch*. Se realizaron bastantes experimentos con la esperanza de obtener los mejores resultados posibles, incluso incluyendo modelos de clasificación para la mejora de resultados.

Los resultados mejoraron solamente en el conjunto de las clases vistas, comparándolas con el estado del arte, pero no se pudo obtener el mismo resultado con las clases no vistas, siendo en todos los modelos peores. Esto se pudo haber obtenido por una mala generalización dentro de los modelos, lo cual pudo haber sido resultado de ocupar el conjunto de datos de PASCAL-VOC, que solamente contenía 12 clases disponibles en el entrenamiento, mucho menor a las 80 que contiene COCO, que le permitiría aprender al modelo una mejor generalización, y mejores resultados en las clases no vistas. El único argumento en contra que podría haber de usar un *dataset* más complejo, es la mayor cantidad de objetos por imagen que posee, que obviamente disminuirá el puntaje en las tareas de detección.

La modificación del pre-procesamiento no afecta demasiado a los resultados del modelo, solo importando la normalización de todas las entradas. El aumento del tamaño de *batch* solo puede llegar a mejorar de forma no muy notable a los resultados de las clases vistas, e incluso disminuir el puntaje de las clases no vistas. El usar *BYOL* mejora muy poco el desempeño de los modelos, pero se pudo observar que es necesario el uso de parámetros entrenados exclusivamente con imágenes reales dentro de la *Backbone* del *target*, a diferencia de la *Backbone* del *query*, que obtiene resultados similares si se usan pesos entrenados exclusivamente con imágenes, o siendo entrenada con bosquejos, o ya sea con los mismos *datasets* usados o diferentes.

El usar un modelo con *transformers* a medio terminar pudo ser el responsable de los bajos resultados obtenidos, pudiendo haber mejora dentro de esta área, pero se puede concluir que el uso del modelo en su estado actual no es factible ya sea para la detección de objetos mediante *sketch* o mediante imágenes reales.

Finalmente, en los ejemplos de resultados se pudo observar que un *threshold* para el IoU muy bajo hace que un mismo objeto sea detectado más de una vez, empeorando cuando hay más de uno en una imagen, pero un valor alto puede impedir la detección de algunas instancias. Por lo que sería bueno definir un valor intermedio que evite detectar al mismo objeto, pero también a los más posibles.

Capítulo 7

Trabajo a futuro

Como se puedo observar, el modelo con *transformers* no se pudo usar con su estructura completa, por lo que sería un buen inicio usarla una vez esté acabada, y si es posible, probar este mismo ejercicio en el modelo *Adaptive Image Transformer*, y de esta forma conseguir mejores resultados.

En este trabajo solo se usó el conjunto de datos de PASCAL-VOC, pero también se vio necesario el uso del conjunto de datos de COCO, ya que los resultados de comparación para las clases no vistas se presentaban en este conjunto. También es importante para poder comparar con cualquier otro *paper* que trate del mismo tema, ya que ambos *datasets* son los más usados en toda tarea de detección.

Los resultados en las clases no vistas no fueron mejores que del estado del arte, pero podría solucionarse con un *fine-tunning* en un conjunto de datos que solo contenga un bosquejo y una imagen, para una buena cantidad de clases no vistas, y de esta forma generalizar aún más el modelo. El problema de esta solución es que la cantidad de clases a añadir deberían ser alrededor de las 1000 para obtener una mejora considerable, lo cual implicaría una gran cantidad de datos a obtener.

Finalmente, debido a lo alto que se medía el *recall* en todas las situaciones, incluyendo para las clases no vistas que entregaban un mAP muy bajo, se encuentra necesario hacer un repaso a las detecciones realizadas, quizás al disminuir un poco el *recall* se pueda mejorar el mAP para todas las clases.

Bibliografía

[1] Yang Liu, Yao Zhang, Yixin Wang, Feng Hou, Jin Yuan, Jiang Tian, Yang Zhang, Zhongchao Shi, Jianping Fan and Zhiqiang He *"A Survey of Visual Transformers"* arXiv:2111.06091v2, 2021.

[2] Ding-Jie Chen1, He-Yen Hsieh1, and Tyng-Luh Liu*"Adaptive Image Transformer for One-Shot Object Detection"* IEEE, 2021.

[3] Qi Fan, Wei Zhuo, Chi-Keung Tang and Yu-Wing Tai *"A Few-Shot Object Detection with Attention-RPN and Multi-Relation Detector"* arXiv:1908.01998v4, 2020.

[4] Weidong Lin, Yuyan Deng *"CAT: Cross-Attention Transformer for One-Shot Object Detection"* arXiv:2104.14984v1, 2021.

[5] Javier Morales Rodriguéz, *"Evaluación de métodos auto-supervisados y semi-supervisados para la extracción de características visuales en el contexto de recuperación de imágenes basada en Dibujos"* Universidad de Chile, 2021.

[6] Alexey Dosovitskiy, Lucas Beyer *"An Image Is Worth 16x16 Words: Transformers For Image Recognition At Scale"* arXiv:2010.11929v2, 2021.

[7] Peng Xu, Kun Liu, Tao Xiang, Timothy M. Hospedales, Zhanyu Ma, Jun Guo, and Yi-Zhe Song *"Fine-Grained Instance-Level Sketch-Based Video Retrieval"* arXiv:2002.09461v1, 2020.

[8] Ashish Vaswani, Noam Shazeer *"Attention Is All You Need"* arXiv:1706.03762v5 Dec 2017.

[9] Shaoqing Ren, Kaiming He *"Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks"* arXiv:1506.01497v3, 2016.

[10] Kaiming He, Xiangyu Zhang *"Deep Residual Learning for Image Recognition"* arXiv:1512.03385v1, 2015.

[11] Ting-I Hsieh1, Yi-Chen Lo, Hwann-Tzong Chen, Tyng-Luh Liu *"One-Shot Object Detection with Co-Attention and Co-Excitation"* arXiv:1911.12529v1, 2019.

[12] Yizhou Zhao, Xun Guo and Yan Lu *"Semantic-aligned Fusion Transformer for One-shot Object Detection"* arXiv:2203.09093, 2022.

[13] Aditay Tripathi, Rajath R Dani, Anand Mishra and Anirban Chakraborty *"Sketch-Guided Object Localization in Natural Images"* arXiv:2008.06551, 2020.

[14] Zhengsu Chen, Lingxi Xie and Jianwei Niu *"Visformer: The Vision-friendly Transformer"* arXiv:2104.12533, 2021.

[15] Dzmitry Bahdanau, KyungHyun Cho *"Neural machine translation by jointly learning to align and translate"* arXiv:1409.0473v7 2016.

[16] Omar Elharroussa, Younes Akbari *"Backbones-Review: Feature Extraction Networks for Deep Learning and Deep Reinforcement Learning Approaches"* arXiv:2206.08016v1, 2022.

[17] Laith Alzubaidi, Jinglan Zhang *"Review of deep learning: concepts, CNN architectures, challenges, applications, future directions"* Springer Open, 2021.

[18] Gregory Koch, Richard Zemel *"Siamese neural networks for One-Shot Image Recognition"* Department of Computer Science, University of Toronto, Canada, 2015.

[19] Seyyid Ahmed Medjahed, *"A Comparative Study of Feature Extraction Methods in Images Classification"* MECS, 2015.

[20] Warren S.Mcculloch and Walter Pitts *"A Logical Calculus of the Ideas Immanent in Nervous Activity"*, University of Chicago, Chicago, 1943

[21] J. Rosenblatt *"The perceptron: a probabilistic model for information storage and organization in the brain"*, Semantic scholar, 1958

[22] Mark Everingham, Luc Van Gool, Christopher K. I. Williams, John Winn and Andrew Zisserman, *"The PASCAL Visual Object Classes (VOC) Challenge"*, disponible en Intenet, 2008.

[23] KnowledgeForAll, *"PASCAL – Pattern Analysis, Statistical Modelling and Computational Learning"* , disponible en Intenet, 2020.

[24] Srishilesh P S, *"Understanding PASCAL VOC Dataset"* , disponible en Intenet, 2022.

[25] Google Creative Lab , *"Quick, Draw!"* , disponible en Intenet, 2017.

[26] Amazon Web Services, ¿Qué es la inteligencia artificial?, disponible en Intenet, 2022.

[27] Amazon Web Services, ¿Qué es el machine learning?, disponible en Intenet, 2022.

ANEXOS

Anexo A. Marco Teórico

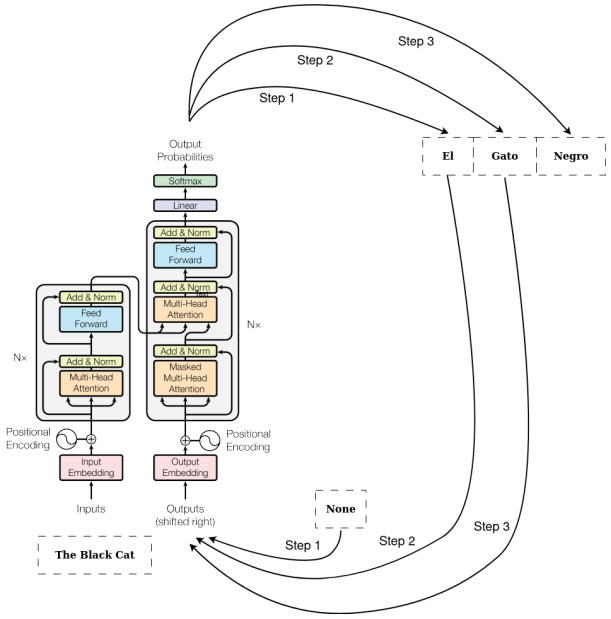


Figura A.1: Ejemplo del *transformer* [8]

Anexo B. Tablas de los resultados

Tabla B.1: Resultados del *Split* 1 al modificar el *Flip*.

Clases Vistas	**Recall**	**AP50**
Aeroplane	0,856	0,750
Bicycle	0,958	0,740
Boat	0,878	0,669
Car	0,923	0,830
Cat	0,975	0,853
Chair	0,844	0,564
Diningtable	0,913	0,576
Dog	0,986	0,672
Horse	0,960	0,744
Sheep	0,913	0,649
Train	0,957	0,722
Tvmonitor	0,867	0,728
mAP	**0,919**	**0,708**

Clases No Vistas	**Recall**	**AP50**
Bird	0,747	0,092
Bus	0,930	0,173
Cow	0,947	0,134
Motorbike	0,877	0,212
Sofa	0,536	0,005
mAP	**0,807**	**0,123**

	Recall	**AP50**
All classes	**0,886**	**0,536**

Tabla B.2: Resultados del *Split* 1 al eliminar el *Crop*.

Clases Vistas	**Recall**	**AP50**
Aeroplane	0,842	0,766
Bicycle	0,944	0,792
Boat	0,871	0,696
Car	0,921	0,835
Cat	0,980	0,853
Chair	0,833	0,572
Diningtable	0,922	0,585
Dog	0,982	0,782
Horse	0,948	0,745
Sheep	0,880	0,689
Train	0,933	0,745
Tvmonitor	0,877	0,736
mAP	**0,911**	**0,733**

Clases No Vistas	**Recall**	**AP50**
Bird	0,741	0,079
Bus	0,897	0,510
Cow	0,930	0,202
Motorbike	0,855	0,182
Sofa	0,686	0,007
mAP	**0,822**	**0,196**

	Recall	**AP50**
All classes	**0,885**	**0,575**

Tabla B.3: Resultados del *Split* 1 al eliminar el *Mean*.

Clases Vistas	**Recall**	**AP50**
Aeroplane	0,179	0,169
Bicycle	0,715	0,571
Boat	0,411	0,297
Car	0,495	0,401
Cat	0,316	0,231
Chair	0,245	0,211
Diningtable	0,291	0,200
Dog	0,057	0,068
Horse	0,078	0,091
Sheep	0,364	0,244
Train	0,213	0,221
Tvmonitor	0,277	0,225
mAP	**0,303**	**0,244**

Clases No Vistas	**Recall**	**AP50**
Bird	0,000	0,000
Bus	0,028	0,036
Cow	0,008	0,008
Motorbike	0,083	0,009
Sofa	0,000	0,000
mAP	**0,024**	**0,011**

	Recall	**AP50**
All classes	**0,221**	**0,175**

Tabla B.4: Resultados del *Split* 1 al aumentar el tamaño del *batch* a 16.

Clases Vistas	**Recall**	**AP50**
Aeroplane	0,818	0,752
Bicycle	0,923	0,800
Boat	0,802	0,657
Car	0,908	0,844
Cat	0,964	0,869
Chair	0,782	0,547
Diningtable	0,937	0,652
Dog	0,973	0,839
Horse	0,951	0,809
Sheep	0,872	0,699
Train	0,915	0,799
Tvmonitor	0,886	0,761
mAP	**0,894**	**0,752**

Clases No Vistas	**Recall**	**AP50**
Bird	0,664	0,072
Bus	0,836	0,062
Cow	0,930	0,136
Motorbike	0,655	0,139
Sofa	0,502	0,008
mAP	**0,717**	**0,083**

	Recall	**AP50**
All classes	**0,842**	**0,556**

Tabla B.5: Resultados del *Split* 1 al aumentar el tamaño del *batch* a 27.

Clases Vistas	**Recall**	**AP50**
Aeroplane	0,814	0,761
Bicycle	0,914	0,797
Boat	0,791	0,635
Car	0,902	0,840
Cat	0,955	0,869
Chair	0,755	0,544
Diningtable	0,903	0,628
Dog	0,965	0,831
Horse	0,951	0,791
Sheep	0,864	0,686
Train	0,911	0,764
Tvmonitor	0,857	0,748
mAP	**0,882**	**0,741**

Clases No Vistas	**Recall**	**AP50**
Bird	0,612	0,086
Bus	0,798	0,076
Cow	0,947	0,132
Motorbike	0,615	0,134
Sofa	0,406	0,006
mAP	**0,676**	**0,087**

	Recall	**AP50**
All classes	**0,821**	**0,549**

Tabla B.6: *Backbones* con pesos *ResNet-Byol online* y *ROI-Head* sin pesos.

Clases Vistas	**Recall**	**AP50**
Aeroplane	0,674	0,216
Bicycle	0,700	0,217
Boat	0,563	0,050
Car	0,518	0,304
Cat	0,872	0,114
Chair	0,459	0,121
Diningtable	0,835	0,138
Dog	0,869	0,139
Horse	0,813	0,059
Sheep	0,459	0,128
Train	0,837	0,055
Tvmonitor	0,406	0,139
mAP	**0,667**	**0,140**

Clases No Vistas	**Recall**	**AP50**
Bird	0,525	0,015
Bus	0,737	0,029
Cow	0,537	0,096
Motorbike	0,711	0,073
Sofa	0,870	0,011
mAP	**0,676**	0,045

	Recall	**AP50**
All classes	**0,670**	**0,112**

Tabla B.7: *Backbones* y *ROI-Head* con pesos *ResNet-Byol online*.

Clases Vistas	**Recall**	**AP50**
Aeroplane	0,705	0,311
Bicycle	0,724	0,350
Boat	0,608	0,197
Car	0,553	0,381
Cat	0,869	0,242
Chair	0,505	0,152
Diningtable	0,859	0,295
Dog	0,879	0,158
Horse	0,851	0,232
Sheep	0,579	0,247
Train	0,876	0,264
Tvmonitor	0,523	0,244
mAP	**0,711**	**0,256**

Clases No Vistas	**Recall**	**AP50**
Bird	0,562	0,025
Bus	0,765	0,127
Cow	0,639	0,101
Motorbike	0,732	0,082
Sofa	0,879	0,010
mAP	**0,715**	0,069

	Recall	**AP50**
All classes	**0,712**	**0,201**

Tabla B.8: *Backbones* con pesos *ResNet50* y *ROI-Head* con pesos *ResNet-Byol online*.

Clases Vistas	**Recall**	**AP50**
Aeroplane	0,870	0,767
Bicycle	0,950	0,794
Boat	0,863	0,675
Car	0,921	0,841
Cat	0,969	0,859
Chair	0,845	0,584
Diningtable	0,922	0,629
Dog	0,988	0,785
Horse	0,960	0,785
Sheep	0,888	0,677
Train	0,950	0,743
Tvmonitor	0,912	0,768
mAP	**0,920**	**0,742**

Clases No Vistas	**Recall**	**AP50**
Bird	0,715	0,101
Bus	0,897	0,049
Cow	0,947	0,145
Motorbike	0,886	0,205
Sofa	0,682	0,007
mAP	**0,825**	0,101

	Recall	**AP50**
All classes	**0,892**	**0,554**

Tabla B.9: *Backbones* con pesos *ResNet-Byol online* y *ROI-Head* con pesos *ResNet50*.

Clases Vistas	**Recall**	**AP50**
Aeroplane	0,695	0,308
Bicycle	0,721	0,342
Boat	0,620	0,199
Car	0,552	0,378
Cat	0,863	0,240
Chair	0,499	0,151
Diningtable	0,845	0,286
Dog	0,881	0,146
Horse	0,862	0,232
Sheep	0,566	0,240
Train	0,879	0,249
Tvmonitor	0,536	0,258
mAP	**0,710**	**0,252**

Clases No Vistas	**Recall**	**AP50**
Bird	0,569	0,020
Bus	0,770	0,126
Cow	0,635	0,110
Motorbike	0,735	0,085
Sofa	0,870	0,010
mAP	**0,716**	0,070

	Recall	**AP50**
All classes	**0,712**	**0,199**

Tabla B.10: *Backbone* del *target* y *ROI-Head* con pesos *ResNet50*, *Backbone* del *query* con pesos *ResNet-Byol online*.

Clases Vistas	**Recall**	**AP50**
Aeroplane	0,860	0,778
Bicycle	0,947	0,803
Boat	0,856	0,684
Car	0,908	0,832
Cat	0,978	0,850
Chair	0,825	0,577
Diningtable	0,913	0,601
Dog	0,984	0,636
Horse	0,966	0,773
Sheep	0,847	0,682
Train	0,943	0,783
Tvmonitor	0,870	0,741
mAP	**0,908**	**0,728**

Clases No Vistas	**Recall**	**AP50**
Bird	0,721	0,174
Bus	0,887	0,054
Cow	0,939	0,257
Motorbike	0,825	0,157
Sofa	0,711	0,010
mAP	**0,817**	0,130

	Recall	**AP50**
All classes	**0,881**	**0,552**

Tabla B.11: *Backbone* del *target* y *ROI-Head* con pesos *ResNet50*, *Backbone* del *query* con pesos *ResNet-Byol-V2 online*.

Clases Vistas	**Recall**	**AP50**
Aeroplane	0,835	0,781
Bicycle	0,935	0,793
Boat	0,852	0,687
Car	0,916	0,841
Cat	0,975	0,848
Chair	0,823	0,584
Diningtable	0,913	0,623
Dog	0,975	0,615
Horse	0,957	0,775
Sheep	0,860	0,679
Train	0,936	0,778
Tvmonitor	0,880	0,740
mAP	**0,905**	**0,729**

Clases No Vistas	**Recall**	**AP50**
Bird	0,723	0,091
Bus	0,864	0,045
Cow	0,943	0,171
Motorbike	0,843	0,162
Sofa	0,770	0,010
mAP	**0,829**	0,096

	Recall	**AP50**
All classes	**0,882**	**0,543**

Tabla B.12: *Backbone* del *target* y *ROI-Head* con pesos *ResNet50*, *Backbone* del *query* con ningún peso.

Clases Vistas	**Recall**	**AP50**
Aeroplane	0,000	0,000
Bicycle	0,000	0,000
Boat	0,000	0,000
Car	0,000	0,000
Cat	0,000	0,000
Chair	0,000	0,000
Diningtable	0,000	0,000
Dog	0,000	0,000
Horse	0,000	0,000
Sheep	0,000	0,000
Train	0,000	0,000
Tvmonitor	0,000	0,000
mAP	**0,000**	**0,000**

Clases No Vistas	**Recall**	**AP50**
Bird	0,000	0,000
Bus	0,000	0,000
Cow	0,000	0,000
Motorbike	0,000	0,000
Sofa	0,000	0,000
mAP	**0,000**	0,000

	Recall	**AP50**
All classes	**0,000**	**0,000**

Tabla B.13: *Backbone* del *target* y *ROI-Head* con pesos *ResNet50*, *Backbone* del *query* con pesos *ResNet-Byol target*.

Clases Vistas	**Recall**	**AP50**
Aeroplane	0,846	0,767
Bicycle	0,938	0,785
Boat	0,848	0,669
Car	0,909	0,830
Cat	0,964	0,854
Chair	0,823	0,566
Diningtable	0,913	0,604
Dog	0,978	0,565
Horse	0,954	0,744
Sheep	0,864	0,648
Train	0,940	0,796
Tvmonitor	0,883	0,747
mAP	**0,905**	**0,715**

Clases No Vistas	**Recall**	**AP50**
Bird	0,706	0,088
Bus	0,869	0,048
Cow	0,947	0,227
Motorbike	0,809	0,192
Sofa	0,753	0,009
mAP	**0,817**	0,113

	Recall	**AP50**
All classes	**0,879**	**0,538**

Tabla B.14: *Backbone* del *target* con pesos *ResNet-Byol online*, *Backbone* del *query* con pesos *ResNet-Byol target* y *ROI-Head* con pesos *ResNet50*.

Clases Vistas	**Recall**	**AP50**
Aeroplane	0,674	0,293
Bicycle	0,715	0,336
Boat	0,578	0,172
Car	0,548	0,368
Cat	0,877	0,232
Chair	0,483	0,138
Diningtable	0,850	0,342
Dog	0,875	0,169
Horse	0,865	0,204
Sheep	0,512	0,216
Train	0,872	0,212
Tvmonitor	0,494	0,241
mAP	**0,695**	**0,244**

Clases No Vistas	**Recall**	**AP50**
Bird	0,551	0,021
Bus	0,756	0,130
Cow	0,623	0,156
Motorbike	0,732	0,074
Sofa	0,887	0,009
mAP	**0,710**	0,078

	Recall	**AP50**
All classes	**0,700**	**0,195**

Tabla B.15: *Backbone* del *target* con pesos *ResNet50*, *Backbone* del *query* con pesos *ResNet-Byol target* y *ROI-Head* con pesos *ResNet-Byol online*.

Clases Vistas	**Recall**	**AP50**
Aeroplane	0,846	0,775
Bicycle	0,947	0,820
Boat	0,848	0,683
Car	0,908	0,842
Cat	0,972	0,854
Chair	0,817	0,577
Diningtable	0,917	0,591
Dog	0,982	0,617
Horse	0,957	0,771
Sheep	0,888	0,671
Train	0,940	0,775
Tvmonitor	0,867	0,734
mAP	**0,907**	**0,726**

Clases No Vistas	**Recall**	**AP50**
Bird	0,704	0,178
Bus	0,883	0,051
Cow	0,951	0,201
Motorbike	0,828	0,183
Sofa	0,607	0,007
mAP	**0,795**	0,124

	Recall	**AP50**
All classes	**0,874**	**0,549**

Tabla B.16: *Backbone* del *target* con pesos *ResNet-Byol-Q online*, *Backbone* del *query* con pesos *ResNet-Byol-Q target* y *ROI-Head* con pesos *ResNet50*.

Clases Vistas	**Recall**	**AP50**
Aeroplane	0,688	0,289
Bicycle	0,718	0,338
Boat	0,582	0,182
Car	0,544	0,362
Cat	0,880	0,209
Chair	0,447	0,138
Diningtable	0,820	0,297
Dog	0,869	0,128
Horse	0,554	0,213
Sheep	0,862	0,244
Train	0,419	0,253
Tvmonitor	0,540	0,201
mAP	**0,660**	**0,238**

Clases No Vistas	**Recall**	**AP50**
Bird	0,540	0,029
Bus	0,770	0,056
Cow	0,652	0,163
Motorbike	0,708	0,083
Sofa	0,874	0,009
mAP	**0,709**	0,068

	Recall	**AP50**
All classes	**0,675**	**0,188**

Tabla B.17: *Backbone* del *target* con pesos *ResNet50*, *Backbone* del *query* con pesos *ResNet-Byol-Q target* y *ROI-Head* con pesos *ResNet-Byol-Q online*.

Clases Vistas	**Recall**	**AP50**
Aeroplane	0,839	0,776
Bicycle	0,929	0,796
Boat	0,848	0,683
Car	0,908	0,831
Cat	0,972	0,848
Chair	0,828	0,590
Diningtable	0,908	0,620
Dog	0,980	0,633
Horse	0,963	0,756
Sheep	0,868	0,674
Train	0,929	0,759
Tvmonitor	0,867	0,740
mAP	**0,903**	**0,726**

Clases No Vistas	**Recall**	**AP50**
Bird	0,704	0,126
Bus	0,836	0,042
Cow	0,947	0,237
Motorbike	0,831	0,174
Sofa	0,669	0,008
mAP	**0,797**	0,117

	Recall	**AP50**
All classes	**0,872**	**0,547**

Tabla B.18: *Backbone* del *target* con pesos *ResNet-Byol-QN online*, *Backbone* del *query* con pesos *ResNet-Byol-QN target* y *ROI-Head* con pesos *ResNet50*.

Clases Vistas	**Recall**	**AP50**
Aeroplane	0,670	0,281
Bicycle	0,706	0,344
Boat	0,593	0,187
Car	0,536	0,357
Cat	0,860	0,199
Chair	0,462	0,139
Diningtable	0,840	0,301
Dog	0,865	0,149
Horse	0,848	0,252
Sheep	0,537	0,220
Train	0,863	0,261
Tvmonitor	0,451	0,234
mAP	**0,686**	**0,244**

Clases No Vistas	**Recall**	**AP50**
Bird	0,545	0,018
Bus	0,746	0,041
Cow	0,602	0,111
Motorbike	0,695	0,071
Sofa	0,891	0,008
mAP	**0,696**	0,050

	Recall	**AP50**
All classes	**0,689**	**0,187**

Tabla B.19: *Backbone* del *target* con pesos *ResNet50*, *Backbone* del *query* con pesos *ResNet-Byol-QN target* y *ROI-Head* con pesos *ResNet-Byol-QN online*.

Clases Vistas	**Recall**	**AP50**
Aeroplane	0,824	0,759
Bicycle	0,950	0,801
Boat	0,848	0,688
Car	0,912	0,835
Cat	0,983	0,852
Chair	0,821	0,595
Diningtable	0,898	0,578
Dog	0,984	0,631
Horse	0,966	0,789
Sheep	0,868	0,671
Train	0,926	0,771
Tvmonitor	0,870	0,739
mAP	**0,904**	**0,726**

Clases No Vistas	**Recall**	**AP50**
Bird	0,715	0,095
Bus	0,869	0,049
Cow	0,930	0,287
Motorbike	0,834	0,152
Sofa	0,695	0,008
mAP	**0,809**	0,118

	Recall	**AP50**
All classes	**0,876**	**0,547**

Tabla B.20: RPN mediante *transformers* y *Backbones* con pesos *ResNet50*.

Clases Vistas	**Recall**	**AP50**
Aeroplane	0,554	0,495
Bicycle	0,635	0,495
Boat	0,456	0,361
Car	0,497	0,443
Cat	0,874	0,772
Chair	0,417	0,227
Diningtable	0,835	0,617
Dog	0,830	0,642
Horse	0,770	0,594
Sheep	0,401	0,306
Train	0,773	0,608
Tvmonitor	0,455	0,361
mAP	**0,625**	**0,493**

Clases No Vistas	**Recall**	**AP50**
Bird	0,427	0,046
Bus	0,582	0,033
Cow	0,447	0,044
Motorbike	0,603	0,162
Sofa	0,494	0,008
mAP	**0,511**	0,059

	Recall	**AP50**
All classes	**0,591**	**0,366**

Tabla B.21: RPN mediante *transformers*, *Backbone* de *target* con pesos *ResNet-Byol online* y *Backbone* de *query* con pesos *ResNet-Byol target*.

Clases Vistas	**Recall**	**AP50**
Aeroplane	0,519	0,204
Bicycle	0,576	0,263
Boat	0,430	0,110
Car	0,471	0,314
Cat	0,818	0,253
Chair	0,333	0,103
Diningtable	0,786	0,293
Dog	0,763	0,162
Horse	0,655	0,127
Sheep	0,314	0,124
Train	0,699	0,202
Tvmonitor	0,331	0,159
mAP	**0,558**	**0,193**

Clases No Vistas	**Recall**	**AP50**
Bird	0,436	0,018
Bus	0,662	0,076
Cow	0,418	0,068
Motorbike	0,578	0,069
Sofa	0,849	0,012
mAP	**0,589**	0,049

	Recall	**AP50**
All classes	**0,567**	**0,150**

Tabla B.22: RPN mediante *transformers*, *Backbone* de *target* con pesos *ResNet50* y *Backbone* de *query* con pesos *ResNet-Byol target*.

Clases Vistas	**Recall**	**AP50**
Aeroplane	0,579	0,509
Bicycle	0,638	0,553
Boat	0,468	0,376
Car	0,505	0,481
Cat	0,888	0,763
Chair	0,402	0,232
Diningtable	0,845	0,617
Dog	0,851	0,547
Horse	0,793	0,574
Sheep	0,421	0,299
Train	0,791	0,661
Tvmonitor	0,432	0,374
mAP	**0,634**	**0,499**

Clases No Vistas	**Recall**	**AP50**
Bird	0,438	0,050
Bus	0,474	0,027
Cow	0,418	0,074
Motorbike	0,551	0,070
Sofa	0,460	0,007
mAP	**0,468**	0,046

	Recall	**AP50**
All classes	**0,586**	**0,366**