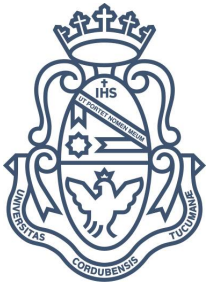
Facultad de Matemática, Astronomía,
Física y Computación

Universidad Nacional de Córdoba



Detección de objetos en imágenes mediante aprendizaje sin ejemplos

Licenciado en Ciencias de la Computación

Agustin Horacio Urquiza Toledo
Director: Jorge Sanchez

Córdoba, Argentina 2021

Agradecimientos

A la Universidad Nacional de Córdoba por haberme dado la oportunidad de formarme. Quiero agradecer a mi tutor Dr. Jorge Sanchez, que me guió y me enseñó lo necesario para realizar este trabajo. A mi familia, y en especial a mis padres, Graciela Toledo y Horacio Urquiza, quienes sin entender del todo lo que hago están siempre apoyándome. A mis compañeros de la Licenciatura, que crecí junto a ellos no sólo en lo profesional sino también en lo personal. Y por último a mis amigos de la vida, que siempre me escucharon y me ayudaron a despejarme de tanto estudio.

Resumen

En el año 2010 surgió la llamada "revolución" del aprendizaje profundo, y con esto, los métodos capaces de detectar objetos en una imagen progresaron considerablemente. Estos algoritmos o modelos fueron mejorando en cada año, hasta hoy en día, que alcanzaron un excelente rendimiento e innumerables aplicaciones. Pero estos poseen una limitación, necesitan tener una gran cantidad de imágenes anotadas, que en algunos casos resulta inviable. Para resolver este problema surgieron técnicas como la detección de objetos sin ejemplos (ZSD) por sus siglas en ingles *Zero-shot Object Detection*. Este es un problema de investigación recientemente propuesto, que tiene como objetivo localizar y reconocer simultáneamente objetos de clases nunca antes vistas. Para suplir la falta de ejemplos de entrenamiento de algunas clases, se combinan distintos atributos de las imágenes y de las clases para inferir los objetos novedosos. Generalmente, se proponen modelos de ZSD como una combinación de características visuales y descripciones semánticas, aprendiendo un mapeo visual-semántico.

En la actualidad existen modelos que obtienen información de las descripciones semánticas capaces de asociar clases similares con una gran eficiencia, en este trabajo se utiliza esta cualidad para poder transferir el conocimiento de las clases presentes en el entrenamiento a las que no estuvieron presentes.

En esta tesis, analizamos distintos artículos científicos y llevamos a cabo experimentos en el conjunto de datos COCO, con la idea de aportar una noción del estado actual en esta área.

Palabras claves

Detección de objetos en imágenes mediante aprendizaje sin ejemplos, aprendizaje automático sin ejemplos, aprendizaje profundo, deep learning, zero-shot learning, zero-shot object detection.

Sistema de clasificación

Detección de objetos.

Summary

In 2010 the deep learning revolution started and made possible the development of methods that are able to detect objects contained in images. These algorithms were improved with the years, and nowadays, they have achieved excellent performance and a great number of applications. However, these methods still have some limitations, mostly related to the great number of tagged images that need to be considered, which in some cases are impossible to obtain. In order to solve this problem, several techniques have been developed, such as the zero shoot detection (ZSD). This method has been recently proposed and aims to simultaneously localize and recognize objects of unseen classes. In order to compensate for the lack of "examples of training" of some classes, different attributes of the images and the classes are combined to infer the new objects. In general, ZSD models are designed as a combination of visual characteristics and semantic descriptions, which make it possible to learn a visual-semantic mapping.

Today, there are models that obtain information from semantic descriptions, and are able to associate similar classes in a very efficient way. This property is used in present work to transfer the knowledge of the classes seen during the training to the ones that were not present in it.

This thesis analyzes different scientific articles and presents experiments that were carried out within the data set COCO, with the objective of contributing to the current knowledge in this field.

Keywords

Deep learning, zero-shot learning, zero-shot object detection.

Computing Classification System

Object detection.

Índice general

Capítulo 1

Introducción y Motivación

1.1. Introducción

La detección de objetos es un campo de la visión por computadora que estudia cómo detectar la presencia de objetos en una imagen. Esta es una de las áreas de visión por computadora de mayor crecimiento en los últimos años. Si bien es un problema conocido y estudiado desde hace tiempo, recién en la última década sus modelos se volvieron más complejos y eficaces, debido a la aparición de técnicas de Aprendizaje profundo (*Deep Learning*). Sin embargo, estos modelos tienen un gran inconveniente, necesitan de un conjunto excesivamente grande de datos anotados para su entrenamiento. Conseguir un gran numero de anotaciones, puede resultar un gran desafío, ya sea por la naturaleza del problema o por los grandes costos que esto conlleva. Esto motivó la aparición de métodos como la detección de objetos sin ejemplos ZSD (por sus siglas en ingles), que intenten mitigar estas dificultades reduciendo el numero de anotaciones necesarias para entrenar un modelo.

Con el objetivo de entender mejor la detección de objetos sin ejemplos, comparemos esta tarea con otras más conocidas como la clasificación o la clasificación sin ejemplos. Antes de explayar los distintos

problemas de la visión por computadora, es necesario distinguir dos conjuntos. Por un lado, los datos de entrenamiento, que constan de las imágenes que se usan para entrenar el modelo con sus respectivas etiquetas, es decir, qué objetos se encuentran en la imagen, localización de los objetos, descripción de la imagen, o cualquier información extra que requiera la tarea. Por otro lado, las imágenes de prueba, que es el conjunto donde se observará o se medirá la eficiencia del modelo ya entrenado.

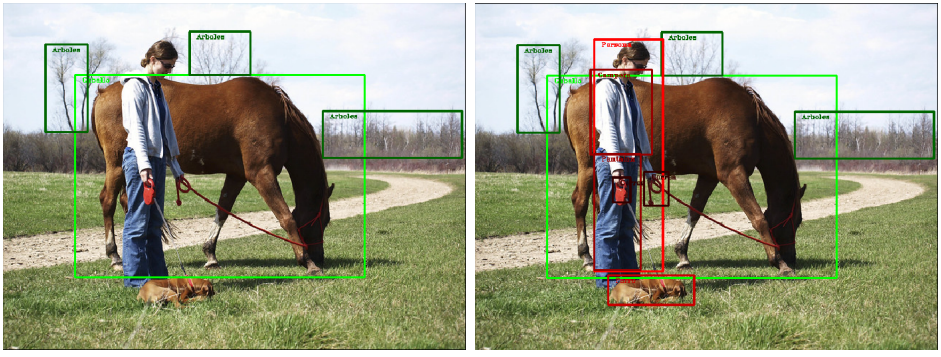
Supongamos que las etiquetas solo cuenta con dos tipos de información, qué **clase** de objeto es, es decir si es un "perro", "auto", "persona", etc. y su localización en la imagen. A todas las clases de objetos que aparecen en los datos de entrenamiento las llamaremos clases visibles o vistas, y todas aquellas clases que no sea una clase vista las llamearemos invisible o no vista. Dicho esto, los distintos problemas son:

- **Clasificación**: consta de un modelo capaz de predecir si una o varias clases específica está presente en una imagen. Para su entrenamiento solo es necesario anotar que objeto está en la imagen.

- **La detección de objetos**: además de reconocer objetos visibles, tiene que ser capaz de localizar dichos objetos. Para su entrenamiento se tiene que agregar la localización de todos los objetos. La Figura 1.1 (a) muestra un ejemplo de esta tarea.

- **Clasificación sin ejemplos**: ZSL por sus siglas en inglés *Zero-shot learning*, tiene que poder reconocer clases vistas y no vistas. Esta tarea utiliza las mismas etiquetas que el reconocimiento de imagen tradicional.

- **Detección de objetos sin ejemplos**: debe localizar y clasificar todas las instancias de objetos en la imagen, sin depender si es una clase vista o no. Necesita los mismo datos que la detección de objetos, pero no es necesario que todas las clases estén presentes



(a) **Detección de objetos** (b) **ZSD**

Figura 1.1: Ejemplo de tareas de (a) detección de objetos y (b) ZSD. En la escala de los verdes se encuentran las clases vistas {Caballo, Árbol}, y en rojo las clases invisibles {Perro, Persona, Campera, Pantalón, Correa}.

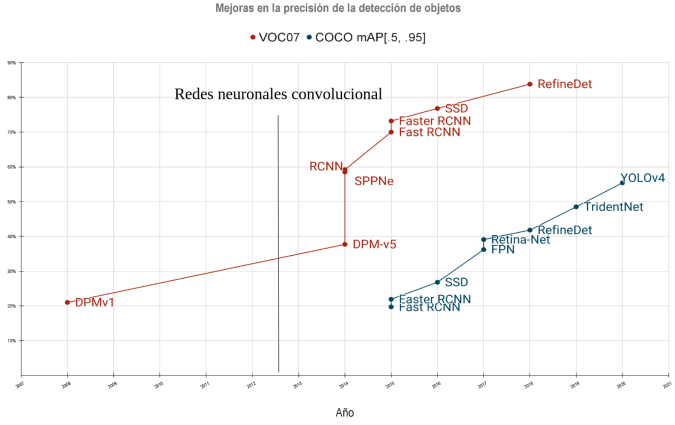
en el entrenamiento. En la Figura 1.1 (b) se muestra el resultado esperado por esta tarea.

1.2. Historia

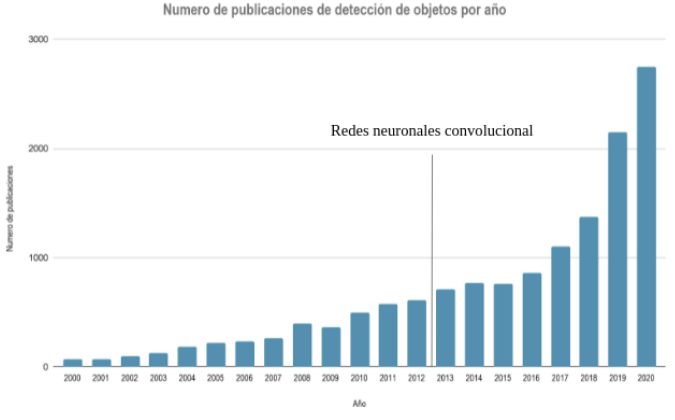
Para entender de donde surge la detección de objetos sin ejemplos, es necesario repasar los aportes más influyentes en la detección de objetos.

En 2001, Paul Viola y Michael Jones [1] presentaron el primer detector de rostros que funcionó en tiempo real. Aunque no se basaba en el aprendizaje profundo, el algoritmo tenía una relación con éste, ya que, al procesar imágenes aprendió qué características podrían ayudar a localizar rostros, inspirándose en un experimento de dos neurofisiólogos David Hubel y Torsten Wiesel de 1995 [2].

En 2006, comenzó la competencia de Pascal VOC que permitió evaluar el desempeño de diferentes métodos para el reconocimiento



(a) Mejoras de precisión en la detección de objetos en los conjuntos de datos VOC07 y COCO. Los detectores en esta figura son: DPM-v1 [3], DPM-v5 [4], RCNN [5], SPPNet [5], Fast RCNN [6], Faster RCNN [7], SSD [8], FPN [9], Retina-Net [10], RefineDet [11], TridentNet [12] y YOLO V4 [13].



(b) Crecimiento en el número de publicaciones sobre detección de objetos entre 2000 y 2020 (datos de la búsqueda avanzada de Google académico: allintitle: “object detection” OR “detecting objects”).

Figura 1.2: Estadísticas sobre la tarea de detección objetos. Estas imágenes están basadas en gráficos presentados en el trabajo [14].

de objetos. Más tarde en 2010, siguiendo los pasos de Pascal VOC, se inició el concurso de reconocimiento visual a gran escala ImageNet (ILSVRC) cuya tasa de error durante 2010 y 2011, en el desafío de clasificación de imágenes, rondaba el 26 %. En 2012, un equipo de la Universidad de Toronto ingresó a la competencia con un modelo de red neuronal convolucional (AlexNet) [15] que cambió todo, dado que logró una tasa de error del 16,4 %. Este suceso hizo que la tasa de error en clasificación disminuyera, pero también impulso a que problemas como la detección de objetos empezaran a utilizar este tipo de redes y mejoraran considerablemente. En la Figura 1.2 se muestran algunas estadísticas sobre detección de objeto y como este suceso influyo sobre éstas.

Todos estos modelos y competencias ayudaron a mejorar el rendimiento de la detección de objetos, pero a su vez empezaron a surgir limitaciones con la cantidad de datos que se necesitan para entrenar los modelos. Por este motivo, la comunidad científica empezó a analizar formas de mitigar estos problemas, dando nacimiento a la detección de objetos sin ejemplos.

1.3. Motivación

Por unos minutos dejémonos llevar por la imaginación y supongamos que se quiere crear un programa capaz de reconocer todos los objeto en una imagen, como: animales, plantas, artículos de limpieza, o cualquier cosa que se nos venga a la mente. Sería casi imposible, si es que no lo es, generar un conjunto de datos que contenga una cantidad considerable de imágenes de todos los objetos posibles. Esta idea de detectar todos los tipos de objetos puede sonar muy descabellada, pero no se puede negar su potencial y su gran cantidad de usos como en interpretaciones de escenas, seguridad, robótica, etc. En la actualidad se está experimentando con modelos de ZSD en robots para interiores [16] y comenzando a investigar su aplicación en áreas como medicina y conducción autónoma [17]. Pero a medida que ZSD

continúa desarrollándose, se espera ver más aplicaciones, como por ejemplo, mejores recomendaciones y soluciones más avanzadas que marcan automáticamente el contenido inadecuado dentro de las redes sociales, como así también un fuerte desarrollo en el campo de la robótica y conducción autónoma.

1.4. Estructura de la tesis

Esta tesis se estructura de la siguiente manera. En el Capítulo 2 se detallan los conceptos fundamentales utilizado a lo largo del trabajo y se formaliza el problema de ZSD. En el Capítulo 3 se define la arquitectura empleada para resolver este problema. Además, se describe los conjuntos de datos utilizados para entrenar y medir el rendimiento del modelo, como así también las métricas utilizadas. En el Capítulo 4 se describen los distintos experimentos realizados y se analizan los resultados obtenidos, los cuales se comparan con distintos trabajos científicos. Por último, el Capítulo 5 expone las conclusiones que se obtuvieron, los aportes realizados por esta tesis, y los trabajos futuros o mejoras.

Capítulo 2

Marco teórico

En este capítulo se proporciona los antecedentes y se detallan los conceptos teóricos en lo que se basa esta tesis. Empezando con la detección de objetos tradicional y sus componentes. Luego, se desarrolla el problema de aprendizaje sin ejemplos y cómo este se usa en la detección de objetos sin ejemplos.

2.1. El problema de detección de objetos

Con los avances recientes en los modelos de visión por computadora basados en el aprendizaje profundo, las aplicaciones de detección de objetos son más fáciles de desarrollar que nunca. Además de importantes mejoras de rendimiento, estas técnicas también han aprovechado conjuntos de datos de imágenes masivos para reducir la necesidad de generar conjuntos de datos grandes. Con los enfoques actuales que se centran en arquitecturas con un pipeline de extremo a extremo, el rendimiento también ha mejorado significativamente, lo que permite casos de usos en tiempo real.

La detección de objetos es una importante tarea de visión por computadora, que consiste en localizar la presencia de objetos de de-

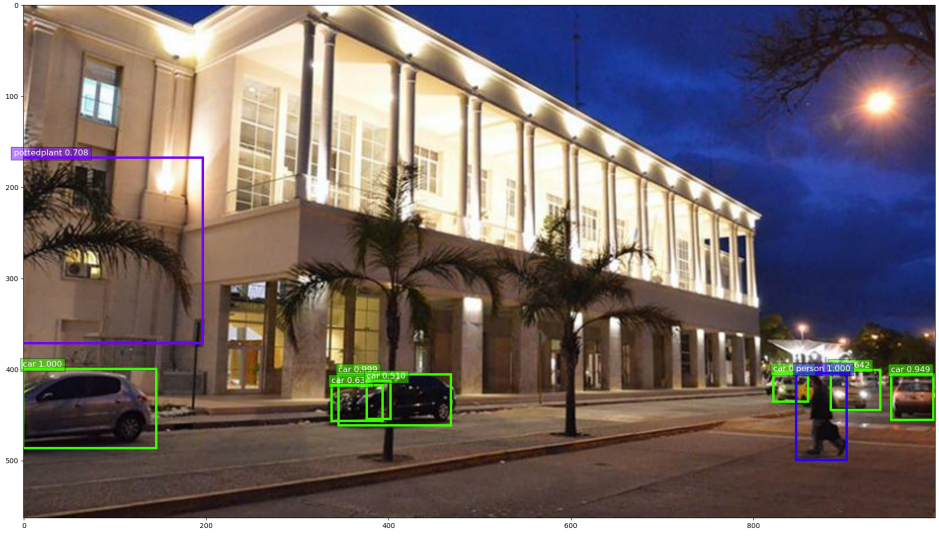


Figura 2.1: Ejemplo de detección de objetos, el modelo utilizado es Faster R-CNN [7]. Las clases que puede detectar esta implementación son: "Auto (*car*)", "Persona (*person*)", "Planta en maceta (*potted plant*)", entre otras.

terminadas clases, como humanos, automóviles, frutas, o animales, en imágenes digitales. El objetivo de la detección de objetos es desarrollar modelos y técnicas computacionales que proporcionen una de las piezas de información más básicas que necesitan las aplicaciones de visión por computadora: ¿Qué objetos están y dónde? En la Figura 2.1 se muestra un ejemplo del resultado generado por estos algoritmos.

Se puede considerar que la detección de objetos tiene asociado dos sub-problemas:

- **Localización:** localiza la presencia de un objeto en la imagen y lo representa con un cuadro delimitador (*Bounding box*). Toma una imagen como entrada y muestra la ubicación del cuadro delimitador en algún formato, por ejemplo: (posición, alto y an-

cho).

- **Reconocimiento:** tiene como entrada una sub-figura de una imagen (generalmente indicada por un cuadro delimitador) y genera la etiqueta de clase de esa figura, junto a una medida de la confianza que indica que tan "buena" es la predicción. Este problema esta asociado con la clasificación de imágenes, el cual genera una etiqueta y una medida de confianza para una imagen completa.

Teniendo en cuenta estos dos componentes, existen principalmente dos tipos de detectores de objetos. Por un lado, tenemos detectores de dos etapas (*two-stage*), como Faster R-CNN [7], que utiliza una red RPN (por su denominación en inglés *region proposal network*) para generar regiones candidatas a contener un objeto de cualquier clase, y luego enviar estas propuestas a una red de reconocimiento. Estos modelos alcanzan las tasas de precisión más altas pero suelen ser más lentos. Por otro lado, tenemos detectores de una etapa (*one-stage*), como YOLO [18], que tratan la detección de objetos como un simple problema de regresión, tomando una imagen de entrada y aprendiendo la predicción de clase y coordenadas del cuadro delimitadores en una misma red. Estos últimos tienen un diseño más eficiente y elegante.

Los detectores de dos etapas filtran la mayoría de las propuestas que no contienen ningún objeto, pasando a la etapa de clasificación un numero menor de cuadros. Mientras que los detectores de una etapa se enfrentan directamente a todas las regiones de la imagen reduciendo su desempeño.

En este trabajo nos centraremos en detectores de objetos de dos etapas, la Figura 2.2 muestra un ejemplo de como funcionan estos tipos de detectores.

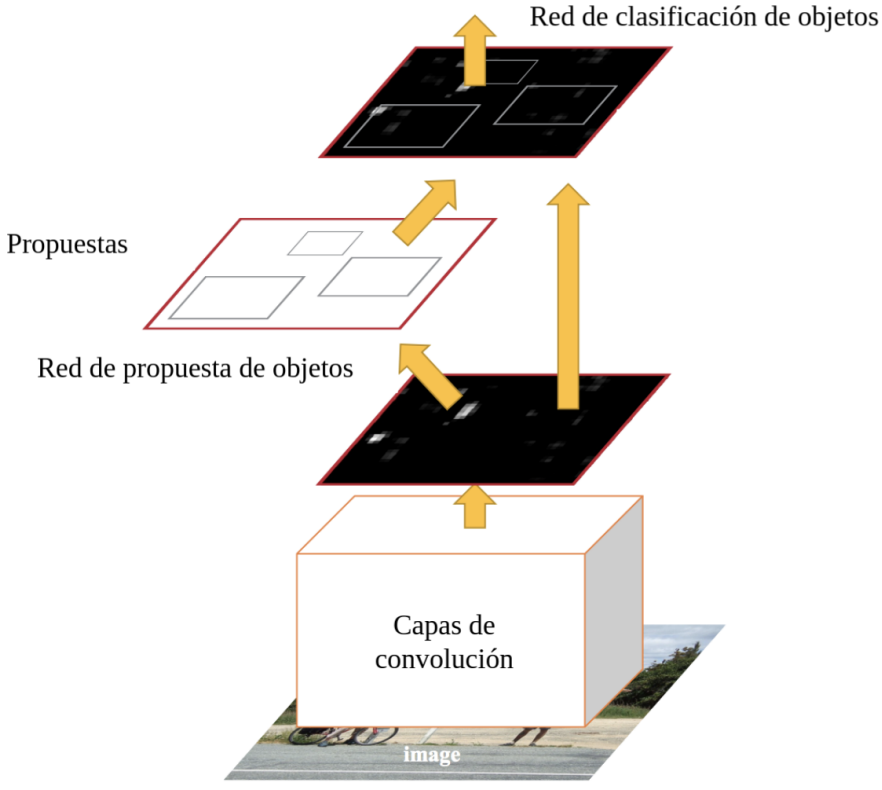


Figura 2.2: Las distintas etapas de un detector de objetos *two-stage*. Imagen adaptada del trabajo Faster R-CNN [7].

2.2. Datos de entrenamiento

Los datos, en general, son un elemento vital de los proyectos de aprendizaje automático supervisado. Cuantos más datos tenga, mejor será el producto final. Sin embargo, no basta con tener datos brutos, sino que se deben tener estos datos anotados. Dichas anotaciones surgen del proceso de etiquetado que consiste en indicar la "cosa" de interés. Dependiendo de los datos y el objetivo final, esta tarea varía. Las etiquetas junto a los datos brutos, "alimentan" al algoritmo de aprendizaje automático que luego permite identificar correctamente los objetos en una imagen, comprender el habla humana y muchas otras funcionalidades. La tarea de etiquetado implica un gran porcentaje del tiempo de desarrollo del proyecto de aprendizaje automático, y además, su costo es muy elevado. La razón por la que la anotación de datos es tan importante es que incluso el más mínimo error podría resultar en un modelo deficiente. Esta es una de las áreas en la que los humanos tenemos una ventaja respecto a las computadoras, ya que podemos lidiar mejor con la ambigüedad, descifrar la intención y muchos otros factores que intervienen en la anotación de datos.

Por otro lado, por lo general, las anotaciones sufren de un fenómeno denominado "cola larga" (o mas conocido como *long tail*). Este fenómeno surge cuando el número de ejemplos de entrenamiento por clase varía de miles, para las clases principales, a tan solo unos pocos, para las clases restantes. El problema de "cola larga" se puede observar en los datos anotados de COCO Figura 2.3, donde la clase persona tiene aproximadamente 260.000 instancias, y otras clases, como tijeras, oso o parquímetro, tienen solo 1000 anotaciones.

En el contexto de la detección de objetos los datos anotados son cuadros delimitadores y sus etiquetas. Una pregunta que siempre se hace es la siguiente: para un modelo de detección de objetos en un problema X, ¿cuántas imágenes se necesitan? La respuesta mas común que se encuentra en la red es 1000 imágenes por clase que se quiere detectar. El origen de este número mágico, proviene del desafío de clasificación de ImageNet original, donde el conjunto de datos tenía, 1000

categorías cada una con 1000 imágenes aproximadamente y por lo general un objeto por imagen. Pero si lo comparamos con un conjunto de datos actual (COCO) donde se evalúa detección de objetos, podemos observar que se requiere muchas más anotaciones (aproximadamente 10.000 anotaciones en promedio, ver Figura 2.3).

Esto ejemplos nos ayudan a orientarnos, pero dar un número exacto es difícil, ya que el número de imágenes necesarias varía mucho dependiendo del problema. Lo que se necesita es un conjunto de imágenes que representen todos los escenarios posibles que puede aparecer el objeto que se quiere detectar. Conseguir este conjunto puede que resulte muy difícil, ya sea por el costo que conlleva generar esas anotaciones, por que los escenarios donde aparece el objeto son demasiados, o por que simplemente es complicado conseguir una imagen del objeto.

Por estos motivos es necesario encontrar algún método que reduzca el numero de anotaciones necesarias para la detección de objetos. Es así como nacen los métodos denominados "aprendizaje sin ejemplos", los cuales analizaremos a continuación.

2.3. Aprendizaje sin ejemplos

El Aprendizaje sin ejemplos (ZSL, por su denominación en inglés *Zero-shot Learning*), es un conjunto de problemas de aprendizaje automático, donde en el momento de la prueba se observan muestras de clases que no se observaron durante el entrenamiento, y se necesita predecir la categoría a la que pertenecen. Esta se diferencia de las configuraciones estándares de aprendizaje automático, donde se espera que se clasifiquen correctamente las nuevas muestras en las clases que ya se han observado durante el entrenamiento.

Esta técnica se puede aplicar en problemas de visión por computadora, y algunos de sus escenarios son:

- **Cuando el número de clases objetivo es grande.** Generalmente, los seres humanos pueden reconocer una gran cantidad de

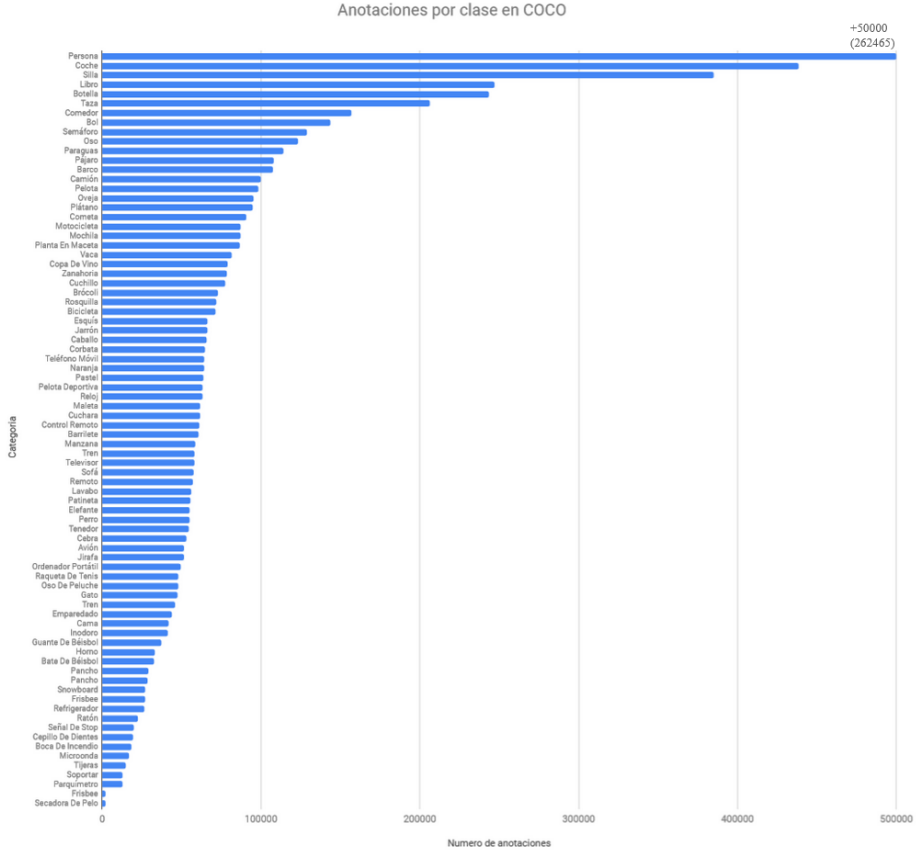


Figura 2.3: Numero de anotaciones por clases en el conjunto de datos COCO.

clases. Sin embargo, recopilar suficientes instancias etiquetadas para un número tan grande de clases es un desafío.

- **Cuando las clases objetivo son raras.** Supongamos que queremos reconocer flores de diferentes especies. Es difícil recopilar suficientes instancias de imágenes para todas las especies. Además, para muchas flores raras, no podemos encontrar las instancias etiquetadas correspondientes.

- **Cuando las clases de interés cambian con el tiempo.** Un ejemplo es reconocer imágenes de productos pertenecientes a cierto estilo o marca. Dado que los estilos y marcas cambian con frecuencia, para algunos productos nuevos, es difícil encontrar los casos etiquetados correspondientes.

- **Cuando el costo de obtener las instancias etiquetadas es alto.** El proceso de etiquetado de instancias es caro y requiere mucho tiempo. Por ejemplo, supongamos que queremos detectar un animal en extinción para poder rastrearlos y tener una forma de contabilizarlos y proteger las áreas donde se encuentran, por ejemplo el yaguareté que habita el Chaco Argentino. Poder conseguir un conjunto considerable de fotos del mismo en distintos entornos, situaciones y ubicaciones, como de día, noche, corriendo, descansando, etc. puede resultar un gran desafió.

Todos estos problemas se pueden intentar solucionar con ZSL, entrenando los modelos con clases similares y luego evaluando las clases que nos interesan. Por ejemplo, para el problema del yaguareté, se puede proponer un modelo de ZSL, entrenado con imágenes de animales similares como leopardo, puma, guepardo, etc., y luego utilizar este modelo para detectar el yaguareté.

Es necesario entender que en este tipo de configuración, existen dos tipos de clases. Las vistas, que son todas aquellas que tienen al menos una instancia en los datos de entretenimiento y las invisibles que no

tienen ninguna instancia en los datos de entrenamiento. Dicho esto, el aprendizaje sin ejemplos se puede dividir en categorías según los datos presentes durante la fase de entrenamiento y la fase de prueba:

- En base a los datos disponibles en el momento de entrenar un modelo.
  - **Zero-shot learning inductivo:** se tiene acceso a las anotaciones de solo las clases vistas.
  - **Zero-shot learning transductivo:** además de las anotaciones de las clases vistas, se tiene acceso a algún dato de las clases no vistas, de esta manera soportar la predicción para las clases invisibles.
- Basado en los datos disponibles en el momento de la inferencia.
  - **Zero-shot learning convencional (ZSL):** en las pruebas solo se evalúan las clases no vistas.
  - **Zero-shot learning generalizado (GZSL):** en las pruebas se evalúan tanto las clases vista como las no vistas.

En el aprendizaje sin ejemplo, no se tiene información sobre las clases no vistas en el entrenamiento. La forma de suplir la falta de esa información sobre las clases que se quiere predecir, es utilizar otro espacio que pueda representar tanto las clases vista como las invisibles. Para poder generar ese espacio, se necesita algún tipo de información auxiliar, que pueda representar todas las clases. Algunos ejemplos de información auxiliar son: una estructura de clases, una descripción textual en lenguaje natural, etc. La Figura 2.4 muestra algunos tipos de información subsidiaria para un mismo dato de entrada.

En obras existentes, el enfoque de información auxiliar se inspira en la forma en que los seres humanos reconocen el mundo. Los seres humanos pueden realizar un aprendizaje sin ejemplo con la ayuda de algunos conocimientos semánticos. Por ejemplo, con el conocimiento

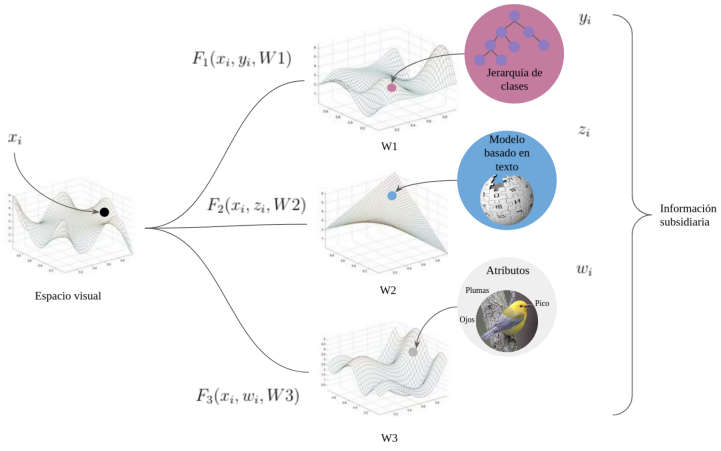


Figura 2.4: Idealización de distintos tipos de información subsidiaria con sus respectivos espacios. Las funciones F_i mapea elementos del espacio visual al espacio de la información subsidiaria(W_i).

de que "una zebra se parece a un caballo y tiene rayas", podemos reconocer una zebra incluso sin haberla visto antes, siempre que sepamos cómo es un caballo y cómo se ve el patrón rayas. De esta manera, la información auxiliar involucrada por los métodos de aprendizaje sin ejemplos existentes suele ser información semántica que forman un espacio que contiene tanto las clases visibles como las invisibles.

Esta técnica de utilizar dos tipos de información, semántica y visual se denomina multimodales. La modalidad se refiere a la forma en que algo sucede o se experimenta. Un problema de investigación se caracteriza como multimodal cuando incluye datos de distinta naturaleza. La Figura 2.5 muestra un ejemplo de como funciona un clasificador sin ejemplos usando multimodales, asociando las imágenes a un vector perteneciente a un espacio que representa a todas las clases.

A continuación se detalla como se puede procesar y representar los dos tipos de información utilizado en este trabajo, visuales y semánti-

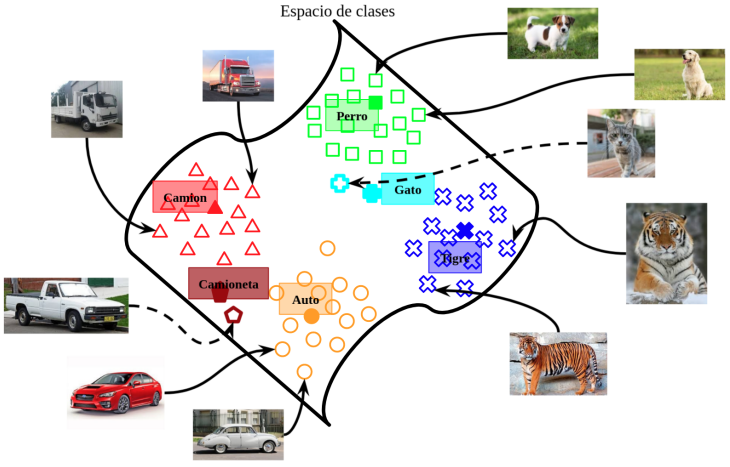


Figura 2.5: Descripción de como funciona un clasificador de imágenes usando aprendizaje sin ejemplo. “Auto”, “Camión”, “Perro” y “Tigre” se observan durante el entrenamiento, “Gato” y “Camioneta” son clases invisibles. El formato de las imágenes sufren una transformación y se mapean a un espacio donde se contiene la información de todas las clases. Este mapeo agrupa distintas imágenes de una clase en espacios cercanos y mientras más distinta es la clase más alejadas están.

cos.

2.3.1. Redes neuronales convolucionales

Las redes neuronales convolucionales, o CNN por sus siglas en inglés, son un tipo de modelos de aprendizaje profundo (*Deep learning*) utilizadas para procesar distintos tipos de datos, pero empleadas generalmente en el dominio de las imágenes. Está inspirado en la organización de la corteza visual de los animales, diseñada para aprender de forma automática y adaptativa patrones en jerarquías, de bajo a alto nivel. Es decir, de formas simples como cuadrados y círculos a formas mas complejas como autos. Esto quiere decir que las primeras

capas pueden detectar lineas, curvas y se van especializando hasta llegar a capas más profundas que reconocen formas complejas como un rostro. Por lo general una red CNN se compone de tres tipos de capas: convolución, agrupación y capas completamente conectadas, como se puede ver en la Figura 2.6. El rol de cada capa es:

- Convolución: somete los datos de entrada a un conjunto de filtros convolucionales, cada uno de los cuales activa ciertas características de los datos. Generalmente, esta acompañada de una capa de activación, que permite un entrenamiento más rápido y eficaz al asignar los valores negativos a cero y mantener los valores positivos. De esta manera, permite que solo las características activadas pasen a la siguiente capa. La función de activación más utilizada para este tipo de redes neuronales es la llamada ReLu por *Rectifier Linear Unit*.

- Agrupación: se coloca generalmente después de la capa convolucional. Su utilidad principal radica en la reducción de su entrada para la siguiente capa convolucional, reduciendo así el número de parámetros que la red necesita aprender. La mas utilizada es *max-pooling* que encuentra el valor máximo entre una ventana de muestra y pasa este valor como resumen de características sobre esa ventana.

- Capas completamente conectadas: Es la encargada de relacionar los datos de las capas anteriores y generar una salida, que por lo general es utilizada para clasificar los datos de entrada.

Algunos ejemplos de redes CNN son: VGG16 [19] (que posee 13 capas de convolución, 5 de agrupación y una totalmente conectada) y AlexNet [15] (que contiene 5 capas convolucionales, 3 capas de agrupación y 3 capas completamente conectadas).

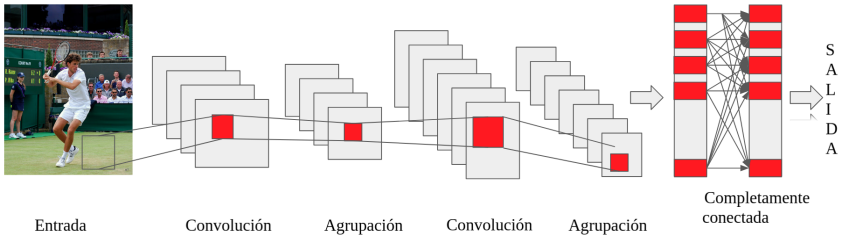


Figura 2.6: Una arquitectura simplificada de una red neuronal convolucional.

2.3.2. Vectores de palabras (Word embedding)

Así como en las imágenes utilizamos las redes CNN, para obtener un vector que represente a la misma, es necesario un procedimiento para representar clases de interés con algún objeto matemático. Hay muchas formas de representar palabras, los más usados son los *word embedding*. Esta es una técnica de aprendizaje en el campo de procesamiento del lenguaje natural (PLN), capaz de capturar el contexto de una palabra en un documento, esto es útil para calcular similitud semántica y sintáctica con otras palabras.

Para entender como funcionan, consideremos las oraciones con un significado similar: "Que tengas un buen día." y "Que tengas un gran día.". Si construimos un vocabulario exhaustivo:

$$V = \{que, tengas, un, buen, gran, dia\}.$$

A partir de esto, se puede crear un vector codificado para cada una de estas palabras, en donde cada vector tenga el tamaño de V, cuyos componentes sean todos 0 excepto por el elemento en el índice que representa la palabra correspondiente en el vocabulario, que contiene un 1. Esta representación no resulta conveniente ya que la distancia entre *gran* y *buen* es la misma que entre *tengas* y *buen*. El objetivo es que las palabras con un contexto similar ocupen posiciones espacia-

les cercanas. Para lograr esto, se introduce cierta dependencia de una palabra con las otras.

Word2Vec [20] desarrollado por Tomas Mikolov en 2013, es un modelo particularmente eficiente desde el punto de vista computacional. Este modelo se encuentra disponible de dos formas: *Continuous Bag-of-Words* (CBOW) o el modelo *Skip-Gram*. En CBOW, las representaciones distribuidas de contexto (o palabras circundantes) se combinan para predecir la palabra en el medio. En nuestro ejemplo *gran* y *buen* están rodeado de un contexto similar por lo cual resultan en vectores similares. Es varias veces más rápido de entrenar que el *Skip-gram*, y tiene una precisión ligeramente mejor para las palabras frecuentes. Mientras que en el modelo *Skip-gram*, la representación distribuida de la palabra de entrada se usa para predecir el contexto. Se entrena con una tarea que, dada una palabra, intenta predecir las palabras vecinas. En realidad, el objetivo es solo aprender los pesos de la capa oculta que corresponden a los vectores de palabras que estamos tratando de aprender. Por ejemplo, *Gran* se entrena para predecir el contexto *un* y *día*, al igual que *buen*. Funciona bien con una pequeña cantidad de datos de entrenamiento.

Los modelos de ZSL, aprovechan la capacidad de capturar similitudes semántica que tiene *word embedding*, para relacionar las clases vistas con las clases invisibles.

2.4. Detección de objetos sin ejemplos

2.4.1. Introducción

El aprendizaje sin ejemplos identifica objetos invisibles para los que no hay imágenes de entrenamiento disponibles. Los enfoques de ZSL convencionales están restringidos a una configuración de reconocimiento donde cada imagen de prueba se clasifica en una de varias

clases de objetos invisibles. Esta configuración no es adecuada para aplicaciones del mundo real donde los objetos invisibles aparecen como parte de una escena completa. Para abordar esta limitación, aparece una nueva configuración, la detección de objetos sin ejemplos ZSD (por sus siglas en ingles *Zero-shot Object Detection*), que tiene como objetivo reconocer y localizar simultáneamente instancias de objetos, incluso en ausencia de ejemplos visuales de esas clases durante la fase de entrenamiento.

Se podría considerar que un modelo de detección de objetos sin ejemplos, es un modelo de ZSL pero con un paso extra, ubicar todas las instancias de objetos que aparecen en una imagen. Este paso se denomina propuesta de objetos y tiene que ser capaz de diferenciar fondos y generar una lista con cuadros delimitadores con posibilidad de contener algún objeto. En visión artificial, la forma más popular de representar la ubicación de los objetos es con la ayuda de cuadros delimitadores (*Bounding Boxes*). Existen muchos algoritmos y redes que intenta resolver este problema, algunos ejemplos son ventana deslizante (*slide window*), Edge-Boxes [21] y búsqueda selectiva (*selective search*) [22]. En ZSD la propuesta de objetos cumple un papel importante, ya que se necesita extraer todas las instancias de los objetos, pero también tiene que discriminar fondos como cielo, ciudades, veredas, etc.

Como veremos en el Capítulo 4, en este trabajo se experimentó con Edge-Boxes [21] y *selective search*) [22], ya que estos generan una cantidad de propuestas significativamente menor a algoritmos del estilo de ventana deslizante. Esto se debe a que no recorren toda la imagen generando distintos cuadros para distinto tamaños, sino que extraen algún tipo de información de la imagen y generan los cuadros basándose en dicha información, refinando y reduciendo el numero de cuadros. En el caso de Edge-Boxes [21] se utiliza un método para generar propuestas de cuadro delimitador utilizando bordes, que proporcionan una representación escasa pero informativa de una imagen.

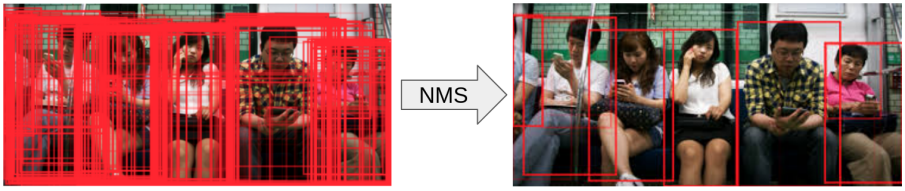


Figura 2.7: Salida de un generador de propuesta de objetos y el resultado después de usar NMS.

La principal observación es que el número de contornos que están totalmente contenidos en un cuadro delimitador es indicativo de la probabilidad de que el cuadro contenga un objeto. De esta maneara con 1.000 cuadros pueden detectar hasta el 90 % de los objetos. Por otro lado, *selective search* [22] combina los beneficios de la búsqueda exhaustiva y de la segmentación, que agrupa los pixeles en segmentos, ya sea por color, texturas, etc. Este algoritmo genera aproximadamente 10.000 propuestas que detecta hasta el 99 % de los objetos.

Aún así, Edge-Boxes [21] y *selective search* [22], siguen generando una cantidad considerable de cuadros (del orden de los miles) y además, muchos cuadros con una gran superposición. Esto da lugar a una técnica de refinamiento denominada supresión de no máximos (NMS), ejemplificada en la Figura 2.7. La salida de NMS es un conjunto más reducido de propuestas, en la cual se filtraron todas las que se consideran repetidas y retorna solo las más representativas.

El requisito estricto de no utilizar ninguna imagen de clase invisible durante el entrenamiento es una condición difícil. Además, existen otras dificultades en la tarea de ZSD relacionadas al conjunto de datos de entrenamiento y prueba, es decir entre las clases vistas e invisibles. Estas dificultades son:

- **Rareza**: los conjuntos de datos, por lo general, contiene un problema de distribución, es decir, muchas clases raras tienen me-

nos cantidad de instancias. Este problema hace que las clases con mayor cantidad de instancias sesguen el modelo y las clases más raras sean marcadas incorrectamente en la etapa de prueba. Esto es un problema al momento de comparar dos modelos que fueron entrenados con distintas clases, ya que algunas separaciones de las clases resultan mejores que otras.

- **Tamaño del objeto**: algunas clases de objetos raros (tijeras, lápices, celulares, etc.), suelen tener un tamaño pequeño. Los objetos más pequeños son difíciles de detectar y reconocer. También, tienen el problema de que por lo general están junto a objetos más grandes como una mesa o una persona y se ven opacadas por estas clases.

- **Diversidad**: cuando una clase invisible no tiene otras clases visualmente similares, resulta muy difícil aprender el aspecto visual de esta. Por ejemplo, "auto" tiene muchas clases similares en comparación con "cartel". Esto permite una descripción visual inadecuada de la clase invisible "cartel" que eventualmente afectará el rendimiento de ZSD, a diferencia de lo que sucede con la clase "auto".

- **Ruido en el espacio semántico**: cuando se utiliza los vectores semánticos como Word2Vec [20] o GloVe [23], las embeddings resultante en general son ruidosas, ya que se generan automáticamente a partir de la minería de texto no anotado. Esto también afecta significativamente el rendimiento de ZSD.

2.4.2. Trabajos recientes en ZSD

Existen muchas técnicas propuestas para resolver ZSD. Cuando se empezó a leer sobre este tema a fines del 2018, la más utilizada consistía en crear una combinación de aspectos visuales y semánticos de cada objeto. Puntualmente existían tres trabajos en paralelo con una metodología similar. Bansal *et al.* [24] propuso un enfoque basado en

características donde las propuestas de objetos se generan mediante edge-box. Zhu *et al.* [25] propone un método basado en el detector YOLO [18]. Rahman *et al.* [26] propuso una extensión de Faster R-CNN [7] junto a un nuevo enfoque transductivo para asociar objetos novedosos en el espacio semántico.

En los últimos dos años se publicaron nuevos trabajos utilizando esta técnica. Rahman *et al.* [27] que mejora los modelos y resultados de su trabajo previo [26]. Gupta *et al.* [28] donde combina predicciones obtenidas en dos espacios de búsqueda diferentes, es decir, del espacio semántico al visual, y viceversa. Rahman *et al.* [29] proponen una función de pérdida novedosa que maneja el desequilibrio de clases y busca alinear adecuadamente los vectores visuales y semánticos.

Este trabajo se basa en el artículo científico de Bansal *et al.* [24], además utiliza muchos conceptos sobre zero-shot learning generalizado [30]. Se propone un modelo zero-shot inductivo, es decir, solo se observan imágenes de clases vistas y etiquetas que indican a que clase pertenece. Estas etiquetas son palabras del lenguaje natural sin ninguna estructura. Luego, se puede inferir todas las clases o solo las invisibles, dependiendo de si se quiere el evaluar aprendizaje por zero-shot generalizado o convencional, respectivamente.

2.4.3. Formalización de ZSD

Para formalizar ZSD denotamos el conjunto de las clases como $\mathcal{C} = \mathcal{S} \cup \mathcal{U}$, donde $\mathcal{S}$ son las clases vistas para entrenamiento y $\mathcal{U}$ las clases no vistas, utilizadas en la etapa de pruebas. Además se tiene que $\mathcal{S} \cap \mathcal{U} = \emptyset$. Aunque no es necesario definir el conjunto de clases de pruebas, ya que el modelo tiene que ser capaz de detectar tanto clases vista como las no vista, esto se hace para poder tener una evaluación cuantitativa.

Denotamos a una imagen como $\mathcal{I} \in \mathbb{D}^{\mathcal{H} \times \mathcal{W} \times 3}$. Donde $\mathbb{D} = \{0, ..., 255\}$, $\mathcal{H}$ es el largo de la imagen, $\mathcal{W}$ el ancho. Esta es la forma en la que se

representa cada pixel de la imagen en el formato **RGB**, donde se tiene 3 canales que caracterizan la intensidad de los colores rojo, verde y azul.

Por cada imagen se provee un conjunto de cuadros delimitadores $\mathbb{B} = \{b_0, ..., b_k \mid b_i \in N^4\}$ (cada b_i representa las coordenadas de un cuadro) y sus etiquetas asociadas como $\mathbb{Y} = \{y_0, ..., y_k \mid y_i \in \mathcal{C}\}$. Para cada cuadro delimitador b_i extraemos una característica profunda utilizando una red neuronal convolucional denotada como $\phi(b_i) \in \mathbb{R}^{D_1}$.

Denotamos los vectores semánticos $w_j \in \mathbb{R}^{D_2}$ obtenido por algún modelo como Word2Vec [20]. El conjunto de todas las imágenes de entrenamiento se indica con $\mathcal{X}^s$, que contiene ejemplos de todas las clases de objetos visibles. El conjunto de todas las imágenes de prueba que contienen muestras de clases de objetos invisibles se indica con $\mathcal{X}^u$. En particular, no hay ningún objeto de clase invisible en $\mathcal{X}^s$, pero $\mathcal{X}^u$ puede contener objetos vistos.

El objetivo es encontrar una matriz de proyección W_p, tal que

$$\psi_i = W_p \phi(b_i) \quad , \quad W_P \in \mathbb{R}^{D_2 \times D_1}, \quad \psi_i \in \mathbb{R}^{D_2}$$

Note que ψ_i y los vectores semánticos se encuentran en el mismo dominio. Como mencionamos en secciones anteriores, el espacio vectorial semántico, tiene una gran capacidad de capturar similitudes semánticas. Por lo cual, resulta clave encontrar una matriz que para cada cuadro delimitador se proyecte lo más cerca posible al vector semántico de su clase.

El resultado es una función

$$f : \mathcal{X}, W_p \rightarrow \{y_0, ..., y_k \mid y_i \in \mathcal{C}\} \quad \text{con} \quad \mathcal{X} = \mathcal{X}^s \cup \mathcal{X}^u$$

con una minimización empírica del riesgo regularizado $\mathcal{R}$ definido de la siguiente manera:

$$\arg\min_{f \in F} \mathcal{R}(f(x, W_p)) \quad ,$$

donde $x \in \mathcal{X}^s$ durante el entrenamiento. La función de mapeo utilizada en la etapa de inferencia, tiene la siguiente forma

$$f(x, W_p) = \arg \max_{y \in \mathcal{C}} \max_{b \in \mathbb{B}(x)} (F(x, y, b, W_p)) \quad ,$$

donde $\mathbb{B}(x)$ es el conjunto de propuestas de cuadros delimitadores para la imagen x. Por ultimo, F calcula la similitud entre la proyeccion ψ_i y el vector semantico de la clase y. Intuitivamente se buscan los cuadros delimitadores de mejor puntuación y se les asigna la categoría de objeto de puntuación máxima.

Ahora que tenemos una idea general del problema de ZSD y como se define formalmente, podemos proponer una arquitectura para resolver este problema, como así también, dónde y cómo se puede evaluar dicha arquitectura. Estos temas serán tratados en el siguiente capítulo.

Capítulo 3

Modelado y conjuntos de datos

En este capítulo se detalla la metodología que se utilizó en la presente tesis para resolver ZSD. Específicamente, se explayan las distintas etapas, se presentan los conjuntos de datos utilizados y se desarrolla como puede evaluarse el rendimiento de los modelos de ZSD Y GZSD en estos conjuntos de datos.

En la actualidad la arquitectura más utilizada por la comunidad científica para resolver el problema de ZSD, es utilizar el espacio que forman modelos como Word2Vec [20] o GloVe [23] para transferir el conocimiento de las clases vistas a las invisibles. Luego de analizar los distintos artículos (ver Subsección 2.4.2), y basándonos en la complejidad del modelo y sus resultados, se decidió apoyarse en el trabajo de Bansal *et al.* [24] para abordar el problema de ZSD. En las siguientes secciones se abordaran las distintas etapas y los detalles de la arquitectura.

3.1. Modelado

Antes de detallar el entrenamiento de la arquitectura, es necesario explayar como se modifican los datos para poder ser utilizados. Cada dato de entrenamiento consiste de una imagen y un conjunto de cuadros delimitadores con el nombre de la clase del objeto que se encuentra dentro del cuadro. El objetivo de esta etapa es generar por cada imagen un conjunto de puntos en el espacio visual y semántico que corresponde a cada cuadro. Por un lado, se recortan los cuadros de la imagen x_i y se reescalan a un tamaño fijo, por ejemplo 224×224. Luego se utiliza una CNN pre-entrenada como VGG16 [19] para generar los vectores visuales de cada cuadro. La salida de este paso es:

$$B_i = [\phi(b_0), ..., \phi(b_k) \mid \phi(b_i) \in \mathbb{R}^{D_1}]$$

Donde B_i son todos los vectores de características visuales de la imagen x_i.

Por otro lado, cada cuadro se asocia con el vector semántico de la clase que tiene asignado, que se puede obtener con modelos de vectores de palabras previamente entrenados, como Word2Vec [20] o GloVe [23]. Dando como resultado:

$$W_i = [w_0, ..., w_k \mid w_i \in \mathbb{R}^{D_2}]$$

Donde W_i son los vectores semánticos asociado a cada cuadro. En la Figura 3.1 se ejemplifica los pasos mencionados anteriormente.

En la inferencia se quiere predecir la clase a la que pertenece cada cuadro delimitador en una imagen, para esto, se computa el vector visual $\phi(b_i)$ obtenido por una CNN como VGG16 [19], y luego utilizando la matriz de proyección W_p (que nos permite pasar del dominio visual al semántico, ver Subsección 2.4.3), calculamos el vector semántico asociado al vector visual. Por último se calcula la similitud coseno entre el vector semántico obtenido y el de todas las clases que se quiere

evaluar, asignándole a esa propuesta la que obtenga un mejor puntaje. La Figura 3.2 muestra la arquitectura empleada en la etapa de inferencia.

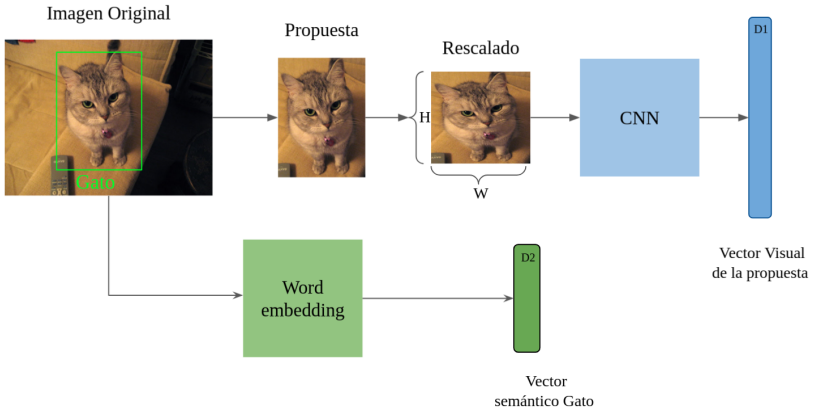


Figura 3.1: Esquema del pre-procesamiento de las imágenes y de como se obtienen los vectores semánticos y visuales.

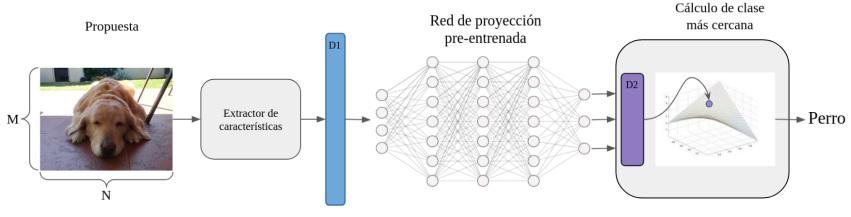


Figura 3.2: Arquitectura propuesta para inferir los vectores semánticos (por ende a la clase que pertenece un cuadro delimitador) a partir de un vector visual.

3.2. Entrenamiento del modelo

3.2.1. Función de costo

Utilizamos el espacio semántico (R^{D_2}) para calcular una medida de similitud entre las proyecciones de $\phi(b_j) \in B_i$ y los vectores semánticos $w_j \in W_i$. Luego, para entrenar la proyección W_p, que nos permite transformar un vector del espacio visual a uno del espacio semántico, definimos una función de costo, que imponga la restricción que el puntaje de la similitud de un cuadro delimitador, con su clase verdadera, debe ser más alto que el de otras clases. Por ejemplo, la proyección de un cuadro que tiene un "perro", tiene que estar lo más cerca posible del vector semántico "perro", y a su vez lejos de cualquier otro vector semántico como "gato" o "auto".

Utilizaremos la función de costo definida por [24] Bansal *et al.*:

$$\mathcal{L}(\psi_i, w_i) = \sum_{j \in \mathcal{S}, j \neq i} max(0, m - S_{ii} + S_{ij})$$

donde m es el margen máximo, y S_{ij} es la similitud entre la proyección visual *i-esima* y el vector semántico *j-esimo*.

Para comprender mejor esta función supongamos un margen máximo de 1, y que para todo S_{ij} se cumple $0 < S_{ij} < 1$. Si la similitud entre la proyección y su vector semántico (S_{ii}) es cercano a 1, la función sólo dependen de los S_{ij}, con $j \neq i$, cuando estos valores se acercan a 0 la función de costo se minimiza, y cuando aumentan la función de costo también lo hace. Por otro lado, si la similitud S_{ii} es aproximadamente 0, estaríamos penalizando la función sin importar los S_{ij}, pero si estos aumentan la función de costo crecerá a la par de ellos.

También se agrega una función de costo de reconstrucción ($\mathcal{L}_r$) como sugiere Kodirov *et al.* [31]. En esta función se utilizan las características del cuadro delimitador proyectadas para reconstruir los vectores visuales originales, y calcular la pérdida de reconstrucción

como la distancia $L2$ entre el vector reconstruido y el original:

$$\mathcal{L}_r = \|\phi(b_i) - \psi_i W_p^T\|^2$$

Luego, definimos λ como un coeficiente de ponderación que controla la importancia del primer y segundo término, que corresponden a las pérdidas de proyección y reconstrucción, respectivamente. Por lo cual, la función de perdida total es:

$$\mathcal{L}_t = \lambda\mathcal{L} + (1 - \lambda)\mathcal{L}_r$$

Es común que en la detección de objetos se incluya una clase de fondo, para obtener un detector robusto que pueda discriminar eficazmente entre objetos de primer plano y objetos de fondo. En ZSD, esto no es un problema trivial, ya que no se sabe si un cuadro de fondo incluye elementos como cielo, tierra, bosque, etc. o una instancia de una clase de objeto invisible. En muchos trabajos [24, 26] se proponen distintas técnicas para abordar este problema, pero no presentan mejoras en evaluaciones cuantitativas. Es por esto que no se incluye una arquitectura que discrimine cuadros de fondos.

3.3. Conjuntos de datos

En la actualidad no existe un conjunto de datos pensado para evaluar ZSD, es por esto que se tiene que adaptar otros conjuntos de datos para poder medir el rendimiento de los modelos. Otra posibilidad es crear un conjunto de dato sintético que emule imágenes de la vida real para ser utilizados en ZSD.

3.3.1. Common Objects in Context (COCO)

COCO es una base de datos que tiene como objetivo ayudar en la investigación de detección de objetos, posee varias características como segmentación de instancias, subtítulos de imágenes y localización

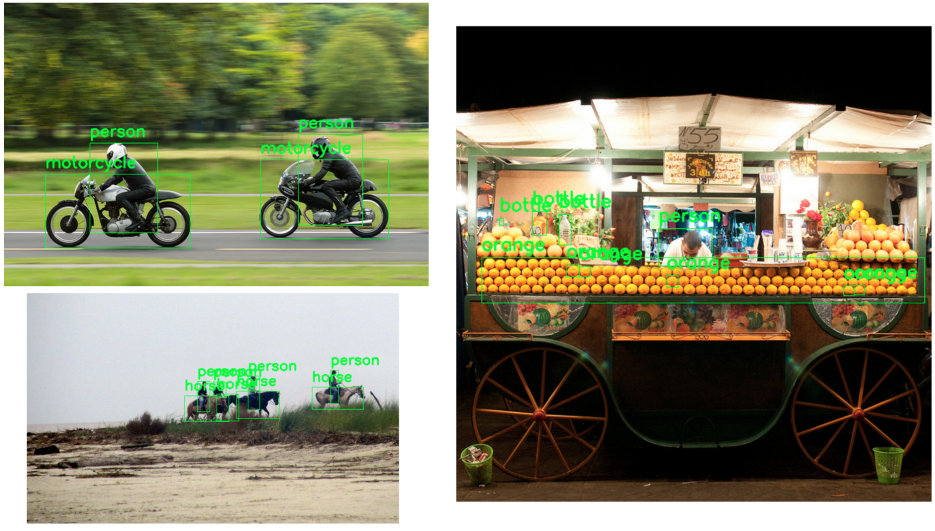


Figura 3.3: Ejemplos de imágenes del conjunto de datos COCO.

de puntos clave de personas. Este conjunto de datos contiene 80 tipos de objetos o clases (65 para COCO 2014), con un total de 2.5 millones de instancias etiquetadas en 328.000 imágenes. La Figura 3.3 muestra algunas imágenes que forman parte del conjunto COCO.

La gran cantidad de instancias de objetos y de categorías, resulta en un conjunto ideal para entrenar y evaluar modelos de ZSD. Además, la mayoría de la imágenes constan de una gran cantidad de objetos, a diferencia de conjuntos como Visual Genome [32]. Estas tipo de imágenes generan un contexto en el que varios objetos se relacionan y se superponen, emulando de una mejor manera situaciones de la vida real.

En este trabajo se utilizan las imágenes de entrenamiento del conjunto COCO 2014 e imágenes del conjunto de validación para realizar las pruebas de ZSD.

Como COCO no provee una separación de los datos para evaluar modelos de ZSD, es necesario crear una forma de dividir las clases en

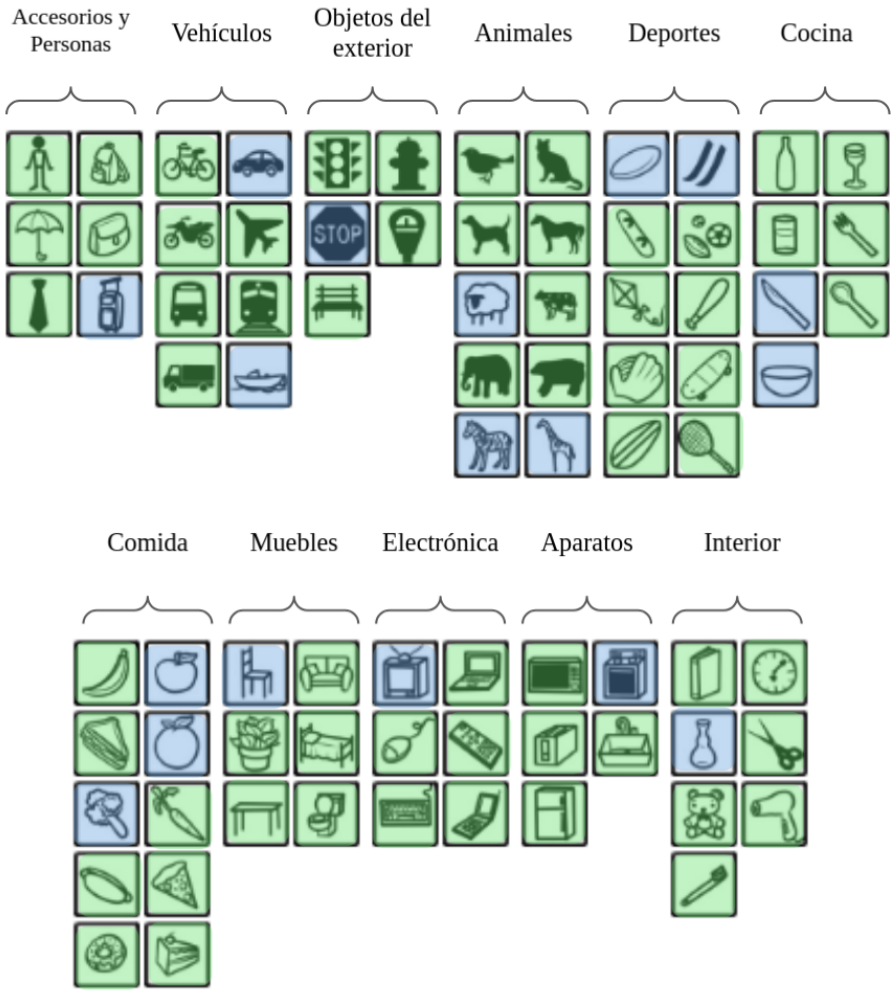


Figura 3.4: División de las clases para entrenamiento (verde) y pruebas (azul).

vistas e invisibles. Esta separación resulta de suma importancia, ya que se debe cumplir que para todo objeto del conjunto prueba, exista otro de aspecto similar que este presente durante el entrenamiento. Además, no se puede encontrar ningún objeto de prueba en los datos de entrenamiento. Para esto, se aprovecha que COCO tiene agrupadas las clases por "Clases superiores" donde se agrupan objetos que tienen alguna relación. Por ejemplo, la clase superior animales contiene las clases zebra, perro, gato, etc. Por cada uno de estos grupos se elige de forma aleatoria un 70 % de clases para entrenamiento y un 30 % para pruebas. Es decir, 47 y 18 clases, respectivamente, de un total de 65 clases de COCO 2014. En la Figura 3.4 se puede observar el resultado de esta división. Por último, se eliminaron todas las imágenes de entrenamiento que contengan al menos una instancia de las clases de prueba. Esto resulta en 42.564 imágenes, con 261.258 instancias de entrenamiento, y 3.008 imágenes con 10.878 instancias de prueba.

Bansal *et al.* [24], divide el conjunto de datos de manera similar, utilizando la misma cantidad de clases para las etapas de prueba y entrenamiento. Pero la diferencia radica en que utiliza los vectores densos de palabras para agrupar las clases, utilizando la similitud coseno entre los vectores como métrica. Por último, elige de forma aleatoria las clases visibles e invisibles de cada grupo. En este trabajo también se utiliza esta separación para logra una comparación de modelos más justa.

3.3.2. CIFAR-ZSD

COCO puede resultar pesado en término computacional. Para solucionar esto se creó un conjunto de datos sintético basado en CIFAR-100 dataset, el cual denominamos CIFAR-ZSD. Este consta de imágenes localizadas, rotadas y re-escalada aleatoriamente con un fondo de otra imagen (algunos ejemplos se pueden ver en la Figura 3.5). Con esto se intenta simular imágenes reales en la cual un objeto puede aparecer con distintos aspectos y escalas. Este conjunto esta dividido de tal forma que ninguna instancia de prueba aparezca en el conjunto

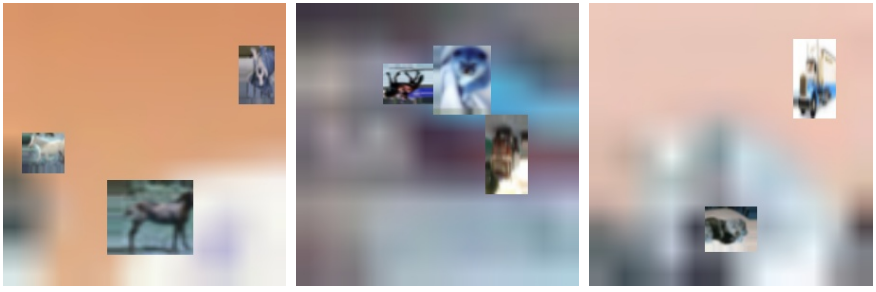


Figura 3.5: Ejemplos de imágenes del conjunto de datos CIFAR-ZSD.

de entrenamiento.

Aunque este conjunto resulta muy útil para hacer pruebas de modelos, no es bueno para reportar métricas reales, pero en combinación con COCO, que si lo es, facilita los experimentos a realizar.

3.3.3. Definición de métricas

Entre los diferentes conjuntos de datos anotados utilizados por los desafíos de detección de objetos y la comunidad científica, la métrica más común utilizada para medir la precisión de las detecciones es el *Mean Average Precision (mAP)*, seguida por *Recall*. Un dificultad que tienen los modelos de detección a diferencia de los de clasificación, es el calculo de estas métricas ya que no es trivial definirlas. Además, no existe una implementación estándar y pública para calcularlas, y aquellas implementaciones públicas están muy encapsuladas en el código, y resulta muy difícil adaptarlo para medir rendimientos de modelos propios. Como ya se mencionó anteriormente, el código de Bansal *et al.* [24] no esta disponible, por este motivo fue necesario encontrar alguna implementación de estas métricas. A partir de estas búsqueda se encontraron varias opciones, sin embargo los resultados variaban mucho de un código a otro. Esto se debe a la falta de consenso en

diferentes trabajos e implementaciones de AP, que es un problema al que se enfrentan las comunidades académicas, tal como se plantea en artículo de Padilla *et al.* [33]. Además, [33] propone una definición y un código para estandarizar las métricas, de manera que se puedan comprar distintos modelos de una forma "justa". Por estos motivos decidimos utilizar este trabajo y su implementación para calcular nuestras métricas, aunque los resultados no den exactos a los reportados por Bansal *et al.* [24].

Ahora definamos las métricas, basándonos en el trabajo [33]. Primero es necesario definir algunos conceptos:

- Intersección sobre unión (**IoU**): es un término utilizado para describir el grado de superposición de dos cuadros. Cuanto mayor sea la región de superposición, mayor será el IoU, la Figura 3.6 ilustra este concepto.

- Falso negativo (**FN**): Para un cuadro delimitador verdadero no se obtuvo ninguna detección en absoluto, o una propuesta tiene IoU $>$ *umbral* con algún cuadro verdadero y no se predijo correctamente la clase.

- Falso positivo (**FP**): Una propuesta predijo correctamente la clase de un cuadro delimitador verdadero pero el IoU $<$ *umbral*, o es un predicción duplicada, es decir, ya se marco otra con mayor IoU como **TP**, o se detecto un objeto inexistente con IoU $<$ *umbral* para todo cuadro verdadero.

- Verdadero positivo (**TP**): Una propuesta predijo correctamente la clase y obtuvo un IoU $>$ *umbral* con algún cuadro verdadero.

- Verdadero negativo (**TN**): Esto sólo tiene sentido si se quisiera medir propuestas que no tienen un IoU $>$ *umbral* con todos los cuadros verdaderos, y además se predijo como clase de fondo. Pero en este trabajo no es utilizada.

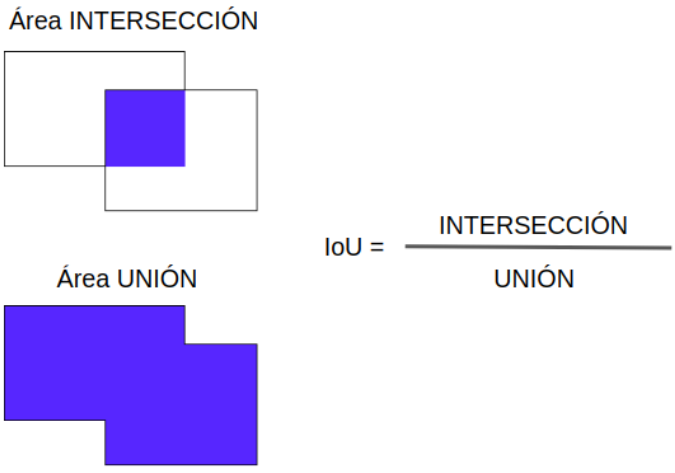


Figura 3.6: Intersección sobre unión de dos cuadros.

El umbral por lo general es 0.5, pero se puede cambiar para exigir que las propuestas tenga una mayor superposición con los objetos en la imagen.

La *Recall*, también conocida como exhaustividad, mide la probabilidad de que los objetos anotados en la imagen se detecten correctamente, y viene dado por:

$$Recall = \frac{TP}{FN + TP} \tag{3.1}$$

En otras palabras la *recall* contabiliza cuantos objetos se detectaron correctamente de todos los anotados en una imagen.

El trabajo de Bansal *et al.* [24], define *Recall* de la siguiente manera:

"Un cuadro delimitador predicho se marca como verdadero positivo

solo si tiene una superposición de IoU mayor que un cierto umbral t con un cuadro delimitador existente en la imagen y no se ha asignado ningún otro cuadro delimitador de mayor confianza al mismo cuadro. De lo contrario, se marca como falso positivo."

Según esta definición solo se tienen en cuenta los objetos que tuvieron al menos una propuesta con un IoU $> 0{,}5$, y el resto quedan fuera del cálculo de esta métrica. Esto genera una diferencia enorme entre los resultados calculados con esta definición y con los obtenidos usando la Ecuación 3.1. Con el objetivo de poder comparar los resultados con otros modelos, en este trabajo se calcula la *Recall* de ambas formas.

Bansal *et al.* [24] además calcula una variación denominada *K@Recall*, donde sólo se tienen en cuentan las K mejores propuestas basándose en la confianza de la predicción y el resto son descartadas.

AP, es una métrica popular para evaluar la precisión de los detectores de objetos mediante la estimación del área bajo la curva (AUC), que viene dada por la relación de la *precisión* y la *recall*. Donde la precisión consiste en medir el proporción de predicciones positivas correctas entre todas las predicciones realizadas y se define como:

$$Precision = \frac{TP}{FP + TP} \tag{3.2}$$

Para dibujar la curva AUC necesitamos obtener múltiples pares de valores de *precisión* y *recall*, esto se logra cambiando un límite de puntuación. Este limite trata como un falso positivo o todas aquellas propuesta que tengan un puntaje de confianza menor.

Para entender mejor supongamos un limite tal que genera un numero de FP bajo, la *precisión* será alta. Sin embargo, en este caso, se pueden pasar por alto muchos aspectos interesantes de analizar, como por ejemplo un numero de FN alto y por lo tanto una *recall* baja. Pero si uno baja el limite se aceptaran más positivos y la *recall* aumentará, pero el numero FP también puede aumentar, disminuyendo la *precisión*. De esta manera a media que aumentamos la *recall* (bajamos el

limite) la *presision* se tiene que mantener alta. Por esto una área alta bajo la curva (AUC) tiende a indicar tanto una alta *recall* como una alta *precisión*.

Se define *mAP* para la detección de objetos como el promedio del AP calculado para todas las clases. Por lo general, se indica sobre que IoU se calcula, puede ser un único valor, como por ejemplo mAP@0.5, o un conjunto de umbrales, como *mAP@[x, y]* promediando el valor de *mAP* para cada IoU. El trabajo de Bansal *et al.* [24] reporta *mAP*, pero no indica sobre que IoU se calcula, por lo cual se asume que se utilizo un valor de 0,5. Muchos trabajos que utilizan COCO, reportan *mAP@[.5, .95]*. Esta métrica resulta muy útil si se quiere comparar rendimientos entre distintos trabajos.

Capítulo 4

Experimentos

En este capitulo explican los distintos experimentos desarrollados, se informar sobre los valores obtenidos en este proyecto y se compara con otros trabajos.

4.1. Experimentos previos

4.1.1. Experimentación con propuesta de objetos

Como se mencionó anteriormente, el número de propuestas es un parámetro clave. Algunas métricas son muy sensibles a la cantidad de propuestas, afectando así los resultados finales. Esto se observó cuando se obtuvieron las primeras métricas, donde los valores estaban muy lejos de los esperados, y a medida que se aumentaba la cantidad de propuestas, los resultados empeoraban. Por este motivo, se probaron dos algoritmos, *Edge Boxes* [21] y *Selective Search* [22], con algunas combinaciones de sus parámetros, con el objetivo de obtener una cantidad de propuestas que se superponga con el mayor número de objetos sin afectar las métricas.

Para no sesgar el experimento con los datos de prueba, se definió la metodología de la siguiente manera: por cada imagen de entrena-

miento se corrió el generador de propuestas, y se calculó el tiempo y la cantidad de cuadros verdaderos que tenían un IoU $> 0{,}5$, con algún cuadro verdadero. En nuestros experimentos, el tiempo es un parámetro importante ya que algunos algoritmos soy muy lentos y resultan difíciles de evaluar con los recursos disponibles.

Como se puede observar en la Tabla 4.1, *Selective Search* obtiene una mayor cantidad de superposición, pero con un número exageradamente grande de propuestas, por lo que resulta mas ventajoso usar *Edge-boxes* [21]. En cuanto número de propuestas totales, resulta más conveniente entre 100 y 500 propuestas como máximo, ya que al aumentar este numero no se generan mejoras en superposición. Si tenemos en cuenta el tiempo, resulta mas practico *Edge-boxes* [21], ya que demora una fracción de lo que tarda *Selective Search* [22]. En la Figura 4.1 se pueden observar las propuestas de cuadros delimitadores para ambos algoritmos.

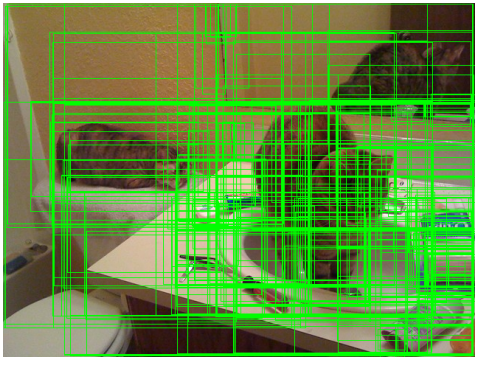
Algoritmo		Numero de propuestas	Tiempo promedio(S)	proporción(*)
Edge Boxes		100	0.11	0.01953
		500	0.11	0.00607
		1000	0.12	0.00355
		5000	0.12	0.00120
Selective Search	Single	$\approx$ 5000	5.40	0.00063
	Fast	$\approx$ 1000	1.41	0.00212

Tabla 4.1: Resultados de correr los distintos algoritmos de propuestas de regiones en los datos de entrenamiento. La última columna muestra el promedio para todas las imágenes de la relación: $(*)(Propuestas\ con\ IOU > 0{,}5)/numero\ de\ Propuetas$.

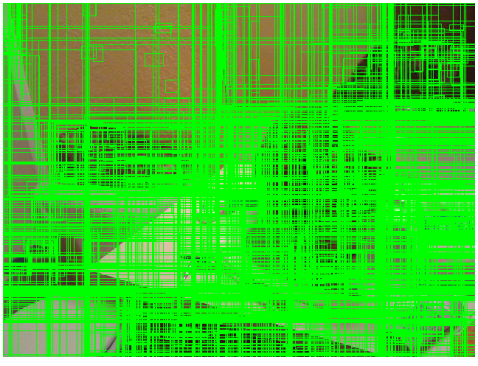
4.1.2. Experimentación con CNN

Se decidió analizar la CNN ya que el modelo final es muy dependiente de esta red y de su capacidad de extraer características visuales útiles. Lo que se quiere aquí es que la red sea capaz de asociar las

(a) Imagen Original



(b) Edge Boxes [21]



(c) Selective Search [22]

Figura 4.1: Se muestran los resultados obtenidos para los algoritmos Edge Boxes [21] con 500 propuestas, y Selective Search [22] (Single $\approx$ 5000 propuestas). Ademas, se muestra la imagen original para referencia.

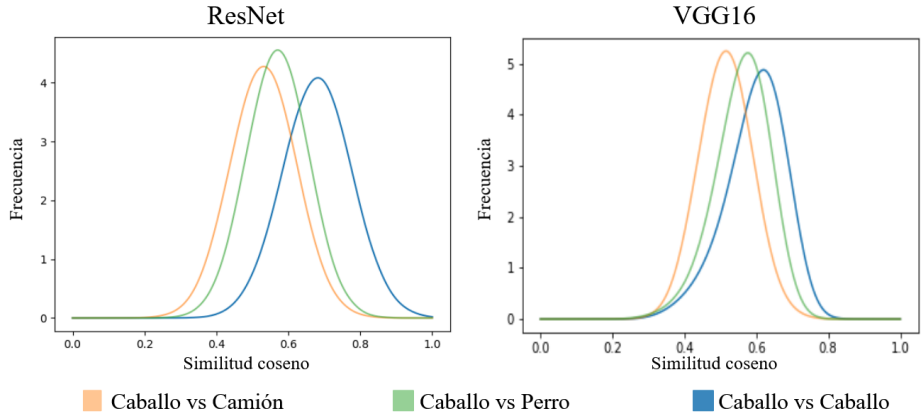


Figura 4.2: Frecuencia de la similitud coseno de los vectores de caracteristicas visuales ente caballo vs caballo, caballo vs camión y caballo vs perro, para las CNN Inception ResNet V2 [34] y VGG16 [19]. Los histogramas están normalizados para una mejor visualización de la superposición.

características visuales de objetos similares, y diferenciar los elementos de distinta naturaleza. En otras palabras, el espacio resultante tiene que distribuirse de tal manera que, por ejemplo, las imágenes de los animales estén muy cerca y a su vez alejadas de vehículos o electrodomésticos, pero también mantengan una separación entre los distintos animales como perro y gato. Bansal *et al.* [24] propone utilizar Inception ResNet V2 [34], el problema de esta red es que puede resultar pesada en cuanto a tiempo de ejecución y memoria. Por este motivo se decidió intentar con VGG16 [19], que reduce el número de parámetros en las capas convolucionales y mejorar el tiempo de ejecución, además, es una de la más utilizadas.

El experimento consistió en comparar miles de recuadros de 3 clases de entrenamiento, caballo, perro y camión. Por cada cuadro se generó el vector de caracteristicas visuales. Luego se comparó utilizando la similitud coseno, entre todas las características de caballo

vs caballo, caballo vs camión y caballo vs perro. En la Figura 4.2 se graficaron las frecuencias de los resultados para cada CNN. Con esto se intenta observar como se distribuyen en el espacio visual las distintas clases. Como se esperaba, la similitud entre entre animales es más grande que con un vehículo. Se observó que para Inception ResNet V2 [34] existe una mayor separación entre clases, aunque sus similitudes están más dispersas. VGG16 [19] parece tener una menor dispersión, pero la similitud coseno entre distintas clases tiene valores muy cercanos. Esto puede afectar de manera negativa ya que camión y caballo no poseen una gran diferencia y el modelo podría interpretarlo como clases similares.

4.2. Detalles de metodología de evaluación

Como ya se menciono, el trabajo de Bansal *et al.* [24] carece de una implementación publica. Por este motivo el principal experimento consistió en replicar los resultados de este trabajo, la falta de una implementación nos permitió experimentar con las distintas etapas y parámetros del modelo. Además, decidimos enfocarnos en los modelos que presentaban un mejor desempeño en el trabajo de Bansal *et al.* [24]. Por ejemplo, los experimentos con clases de fondo, no obtuvieron buenos resultados, en comparación con los que no la utilizan, por este motivo no lo replicamos.

Ahora definamos la metodología de evaluación. El primer paso consiste generar propuestas para cada imagen, luego, cada cuadro propuesto es reescalado al tamaño de la capa de entrada que tiene la CNN, y se le extrae el vector de características visuales. Después, se utiliza el modelo entrenado para inferir el vector de características semánticas, y se calcula la similitud coseno con los vectores semánticos de todas las clases o solo las invisibles, dependiendo si se quiere

evaluar GZSD o ZSD. Aquella clase que obtenga el mayor puntaje es asignada a la propuesta, y también se guarda la similitud coseno como la confianza de predicción, que indica que tan "correcta" es la predicción. Por último, se agrupan todas las propuestas que se tengan asignada la misma clase y se corre un algoritmo de supresión no máxima. Éste elimina las predicciones repetidas y retorna las mejores propuestas de cada grupo. Al final obtenemos como resultado un conjunto de propuestas, sus clases y su respectivo puntaje. Con estos datos alimentamos la implementación de Padilla *et al.* [33], para obtener los resultados de las métricas.

4.3. Resultados cuantitativos

Esta sección desarrolla de forma numérica los resultados obtenidos por los distintos modelos y en las distintas métricas. Se analizan las configuraciones ZSD Y GZSD.

4.3.1. Resultados ZSD

			Modelo propuesto		
Métrica	Baseline [24]	DSES [24]	VGG	ResNet	Mejor resultado de [27]
100@Recall (Bansal)	22.14	27.19	26.34	**28.91**	-
100@Recall	-	-	5.44	6.38	**12.27**
mAP@0.5	0.32	0.54	0.19	0.23	**5.05**
mAP@[.5, .95]	-	-	0.17	**0.21**	-

Tabla 4.2: Comparación de los resultados obtenidos en el presente trabajo, con los obtenidos en Bansal *et al.* [24] y Rahman *et al.* [27]. Se presentan las distintas métricas *recall* y *mAP*, evaluados en COCO.

La Tabla 4.2, muestra los valores de las métricas *100@Recall*, en la versión desarrollada por Bansal *et al.* [24], la de Padilla *et al.* [33], *mAP@0.5* y *mAP[.5, .95]*, para los modelos de Bansal *et al.* [24], y los dos propuestos por nosotros, uno utilizando *VGG16* y el otro Inception ResNet V2. Además, se agregan los mejores resultado presentado

por el trabajo de Rahman *et al.* [27]. Se eligió este documento ya que es un trabajo más actual y aborda de una manera similar a la nuestra el problema de ZSD, aunque presenta algunas mejoras y un modelo más complejo.

La *100@Recall* es un buen punto de partida para analizar el modelo propuesto ya que refleja el numero de propuestas predichas correctamente sobre el total de cuadros verdaderos. Obtuvimos 6.38 puntos en esta métrica, que resulta por debajo de lo esperado. Pero esto no significa necesariamente que el modelo no funciona correctamente, existen varios parámetros que influyen en este resultado. El punto que más afecta es el generador de propuestas, ya que menos del 50 % de los cuadros verdaderos obtienen una propuesta con $IoU > 0{,}5$ lo cual reduce mucho la esperanza de esta métrica. Otro punto, es la capacidad de la CNN de obtener un espacio de características visuales, que agrupe las clases visualmente similares y separe las diferentes. Como se vio en la Subsección 4.1.2, *ResNet* supera a *VGG16* en esta tarea, lo cual se refleja en la pequeña mejora del modelo que utiliza *ResNet*.

En cuanto a *mAP* obtuvimos 0.23 puntos, esto es un bajo desempeño comparado con los trabajos publicados recientemente, pero al igual que *100@Recall* se ve afectada por los puntos antes mencionados. Además, existe otro factor que la afecta muy fuertemente, debido a la naturaleza de la matriz que proyecta las características visuales al espacio semántico, que hace que dos objetos proyectados obtengan una similitud coseno poco distanciada, la cual ronda entre los valores 0.3 y 0.6. Es decir, si se proyectan dos imágenes con muy pocas diferencias se obtendrá un similitud como máximo de 0.6. Esta similitud es utilizada como el puntaje de confianza de una predicción, y como se explicó en la Subsección 3.3.3, para calcular la curva AUC, *mAP* varia el límite de confianza y lo compara con los puntajes obtenidos por cada predicción. Esto hace que cuando el límite sea superior a 0.6 se obtengan valores muy bajos o incluso nulos de *precisión*.

El mejor resultado de Bansal *et al.* [24], el cual se denomina *Den-*

sely Sampled Embedding Space (DSES), consiste en aumentar los datos de entrenamiento con datos adicionales de fuentes externas y obtiene 27.19 puntos en su definición de *recall*. Nuestro modelo base, que usa *ResNet*, supera ese valor obteniendo 28.91 puntos. Esto se debe a que en la etapa de depuración se modificaron algunos parámetros por defecto del entrenamiento, como el numero de lote, la tasa de aprendizaje, el optimizador, etc.

En cuanto mAP, Bansal*et al.* [24] no aporta mucha información y su implementación es desconocida, por lo cual asumimos que lo reportado es *mAP@0.5*. De inmediato se puede observar que los valores son muy bajos 0.54. Esto genera una discrepancia con su alto rendimiento de *100@Recall* y refleja lo poco representativa de esta última métrica.

Si comparamos con un trabajo más actual [27] que obtiene 12.27 en *100@Recall* y un excelente desempeño en *mAP@0.5* con 5.05 puntos, refleja una consistencia con nuestros valores y demuestra que la implementación utilizada para calcular las métricas está mejor encaminada.

Los resultados de la Tabla 4.2 se calcularon utilizando la división de clases propuesta por Bansal, ya que ambos trabajos con los que comparamos utilizan esta división. También se corrieron algunas pruebas con la partición propuesta en este trabajo (Subsección 3.3.1), y se observó que los resultados se vieron afectados entre un 4 % y 7 % menos al utilizar nuestra división. Esta reducción se debe a que el documento de Bansal *et al.* [24] utiliza como criterio de división los vectores semánticos de las clases, lo cual afecta positivamente ya que es el mismo espacio utilizado para asociar e inferir las clases invisibles.

También se calcularon las métricas para el conjunto de datos CIFAR-ZSD, sin embargo, fueron necesarias algunas mejoras para adaptar las diferencias con COCO. Se utilizó una tamaño de entrada para la CNN más pequeña de 32x32, ya que las imágenes no tenían una gran resolución. También, se redujo considerablemente el número máximo de propuestas, de 500 al orden de 50. La justificación de esto es que los objetos sobresalen del fondo de la imagen y es más fácil su detección.

Dicho esto, los resultados obtenidos fueron, 8.83 en *100@Recall* (implementación de Padilla *et al.* [33]) y 0.72 para *mAP@0.5*. Estos valores, al contrario de los reportados para COCO, se ven influenciado por la calidad de la imagen, lo que hace muy difícil de diferenciar el aspecto visual de las distintas clases.

En conclusión, sabiendo que nuestro modelo de base es muy sencillo y no utiliza ningún tipo de información extra, los valores obtenidos son aceptables para *100@Recall*, aunque resulta importante resaltar su bajo desempeño en *mAP*.

4.3.2. Resultados GZSD

Por último, se analizaron los resultados en el desafio de GZSD. La configuración generalizada de aprendizaje sin ejemplos es más realista que la configuración de aprendizaje sin ejemplos discutida anteriormente, porque tanto las clases visibles como las invisibles están presentes durante la evaluación.

La metodología de evaluación es la misma que ZSD estándar solo que ahora se agrega las clases visibles a las pruebas. Algunos trabajos modifican la metodología de evaluación para que las clases invisibles tengan más oportunidad sobre las vistas, pero esto agrega información extra que en situaciones reales no tenemos.

La Tabla 4.3 muestra los resultados para GZSD evaluados en COCO. Se puede observa que el despeño promedio de las clases vistas e invisible para las métricas *100@Recall* y *mAP@0.5*, fue 3.84 puntos y 0.13 respectivamente. Como es de esperarse se obtuvo un mejor rendimiento para las clases vistas en ambas métricas.

Si comparamos los resultados de ZSD vs GZSD, se observa una disminución en los valores de las métricas promedio, que también se ve reflejado en los trabajos [24] y [27]. El motivo de esto es que las clases vistas, al estar en entrenamiento, tienden a tener un mejor puntaje en la etapa de evaluación que las clases invisibles pertenecientes

Modelo	GZSD		
	Clases vistas	Clases Invisibles	Media
	mAP/Recall Bansal/Recall	mAP/Recall Bansal/Recall	mAP/Recall Bansal/Recall
Mejor resultado [24]	-/15.02/-	-/15.32/-	-/15.17/-
Nuestro modelo base	0.15/**20.98**/4.77	0.11/**18.53**/2.92	0.13/**19.75**/3.84
Mejor resultado de [27]	**13.93**/-/**20.42**	**2.55**/-/**12.42**	**4.31**/-/**15.45**

Tabla 4.3: Resultados obtenidos, en el desafió GZSD, para los modelos de Bansal *et al.* [24], nuestro (ResNet) y Rahman *et al.* [27]

a una misma clase superior. Por ejemplo, supongamos que la clase superior "animal" esta conformada por "gato" (clase vista) y "perro" (invisible), por la semejanza visual de estas dos clases, la proyección entrenada con "gato" va a tender a ubicar la proyección de una imagen de un "perro" cerca del vector semántico "gato", generando así un falso positivo, y causando una disminución en las métricas. Por este motivo, muchos objetos que en la configuración anterior se predecían correctamente, ahora se predicen incorrectamente por que una clase visible obtiene mejor puntaje.

4.4. Resultados cualitativos

Para tener una idea más realista del comportamiento del modelo propuesto, se realizaron pruebas sobre algunas imágenes de muestra. Con el objetivo de obtener ejemplos más claros fue necesario modificar la metodología que se utilizó para calcular los resultados cuantitativos. De esta manera, i) disminuimos el número de propuestas al orden de 10, y ii) se descartaron todas aquellas propuestas que obtuvieran un puntaje de confianza menor a 0.5. Esto hace que los ejemplos solo contengan los objetos más relevantes pero, en contra parte, se ignoran muchos objetos que se encuentran en segundo plano. Otra aclaración importante es que se evalúa ZSD y no GZSD, es por esto que se ignoran o confunden clases visibles. La Figura 4.3 muestra las detecciones del modelo propuesto en el conjunto de datos COCO. Los cuadros en azul muestran detecciones incorrectas y en verde las que acertaron a que clase pertenece el objeto.

Si bien el modelo confunde algunas instancias de objetos, cabe destacar que por lo general se equivoca dentro de una misma clase superior, confirmado que el modelo propuesto es capaz de relacionar aspectos visuales y detectar clases invisibles sin observar ninguna muestra durante el entrenamiento.

Otro punto es que debido a que reducimos la cantidad de propuestas, éstas no están centradas en los objetos, y no detectan otros que se encuentran en un primer plano. Además, con esta configuración resulta difícil encontrar un ejemplo en el que se observen objetos pequeños, como un “cuchillo”; ya que las propuestas son a escalas más grandes y se ven opacadas por los objetos que lo rodean, como un “plato”.

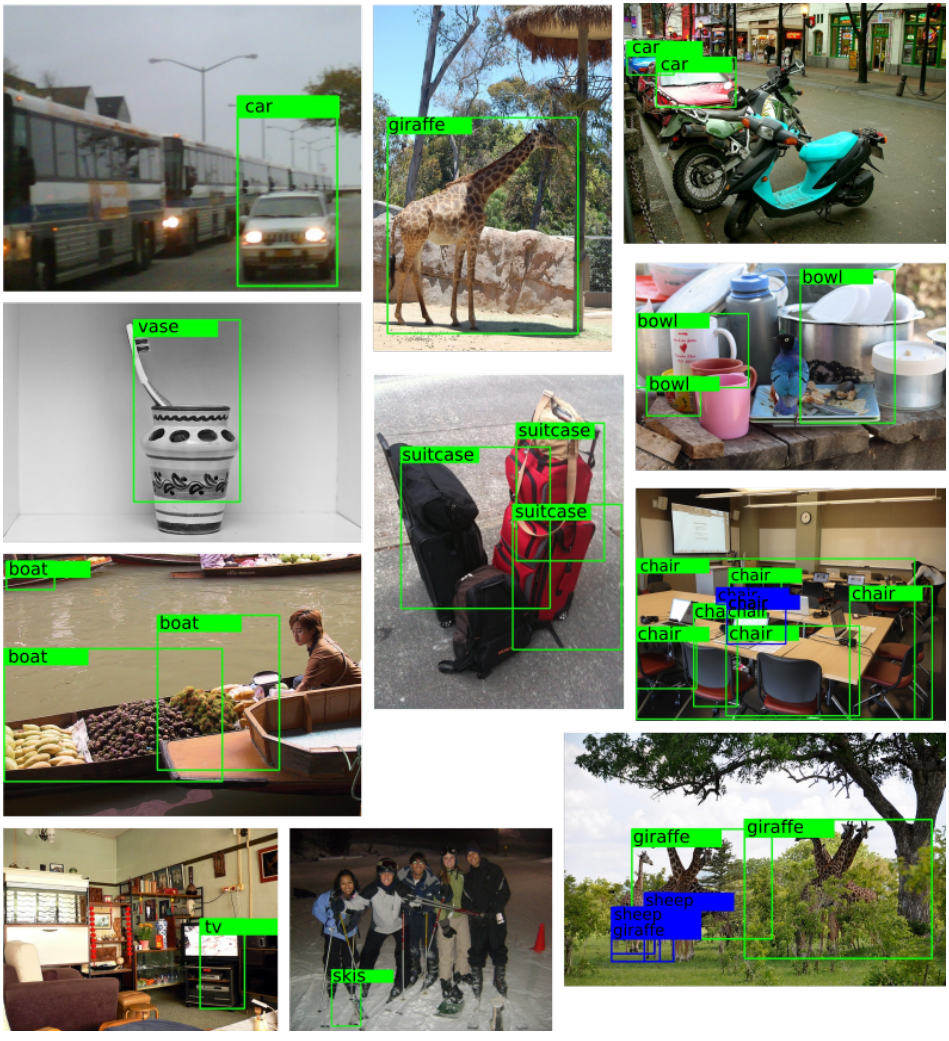


Figura 4.3: Ejemplo del comportamiento del modelo sobre clases invisibles. Los cuadros azul se muestran las predicciones incorrectas y en verde las correctas.

Capítulo 5

Conclusiones y trabajo futuro

5.1. Conclusiones y aportes

Durante este trabajo se analizó de forma detallada y objetiva el desafiante problema de ZSD. Desde un principio, sabíamos que era un campo de investigación nuevo y que esto dificultaría el desarrollo de esta tesis. Los objetivos se fueron modificando en transcurso del tiempo, pero aún así logramos genera un aporte en esta disciplina.

El primer paso de esta tesis fue la lectura y análisis de los distintos trabajos sobre ZSD. Un aspecto que tenían en común la mayoría es la utilización de vectores visuales y semánticos para abordar el problema. Por este motivo, decidimos utilizar esta metodología, para proponer un modelo base, basándonos en el trabajo de Bansal *et al.* [24]. La falta de una implementación por parte de este artículo nos obligó a profundizar en cada etapa del desarrollo, reduciendo las metas planteadas, pero aportando un conocimiento más detallado de la solución.

Si bien el modelo de base no fue propuesto por nosotros, aportamos detalles que surgieron de nuestra experimentación y comprendimos

como estos afectan a los resultados. También se analizaron aspectos como los generadores de propuestas y las CNN, que a nuestro entender, fueron ignorados por el trabajo original pero resultan de crucial importancia.

Otro aspecto importante analizado es el conjunto de datos. Aún no existe uno específico para el problema de ZSD, ni una adaptación consensuada de alguno ya existente. Por ello, propusimos una manera sencilla de dividir COCO y la comparamos con la división del trabajo original, que al parecer, beneficia al modelo considerablemente.

Lo que creemos que es el mayor aporte es el análisis de la métricas. Esto es una gran debilidad en los trabajos relacionados actuales, ya que impide una comparación justa y correcta. Debido a que los resultados obtenidos no eran los esperados, decidimos investigar sobre las métricas, y nos encontramos con una definición poco específica que generó una mala interpretación. Para resolver esto, se analizaron distintos artículos que señalaban el problema que tuvimos. El trabajo de Padilla *et al.* [33] sobresale a los demás, porque posee una clara definición de las métricas y una implementación fácil de utilizar, por lo que se recomida su uso.

Si bien los resultados no representan una mejora respecto al estado del arte, aportan una idea de lo que los modelos de ZSD son capaces de alcanzar. Por otro lado, creemos que este trabajo aporta resultados más transparentes y detallados, haciendo posible agregar mejoras y ver su progreso de una manera cuantitativa.

5.2. Trabajo futuro

Teniendo en cuenta los resultados obtenidos en esta tesis, existen distintas alternativas para seguir profundizando. Las cual podemos dividir en cuatro grupos:

- Mejorar el algoritmo que genera propuestas, que afecta sobre todo la etapa de evaluación. Trabajos actuales utilizan varios generadores simultáneamente, obteniendo algunas mejoras. Otros

plantean aumentar el número de propuestas considerablemente y utilizan un criterio más complejo para eliminar casillas repetidas y de fondo.

- Mejorar del modelo propuesto. Existe muchas formas, algunas ideas pueden ser: considerar la fusión de diferentes vectores de palabras (*Word2vec* y *GloVe*); utilizar otro espacio que no sea el semántico y mapear ambas características a este; o utilizar una única red unificada entrenada de extremo a extremo, capaz de predecir la ubicación de diferentes objetos y clasificarlos como lo hace *Faster R-CNN*.

- Resulta interesante suavizar el problema de ZSD, y en vez de clasificar por clase, se pueden utilizar sub-clases. Si bien esto no es una mejora, puede ayudar a entender si el modelo realmente esta relacionando objetos, ya que no es lo mismo confundir un perro con un auto, que un perro con lobo.

- Una debilidad del modelo actual es la forma de calcular el valor de confianza asociado a una predicción. Se puede analizar una manera más compleja para realizar esto, como por ejemplo agregar al cálculo la similitud coseno las demás clases y no solo la que obtenga mayor puntaje.

Bibliografía

[1] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," in *Proceedings of the 2001 IEEE computer society conference on computer vision and pattern recognition. CVPR 2001*, vol. 1, pp. I–I, IEEE, 2001.

[2] D. H. Hubel and T. N. Wiesel, "Receptive fields of single neurones in the cat's striate cortex," *The Journal of physiology*, vol. 148, no. 3, pp. 574–591, 1959.

[3] P. Felzenszwalb, D. McAllester, and D. Ramanan, "A discriminatively trained, multiscale, deformable part model," in *2008 IEEE conference on computer vision and pattern recognition*, pp. 1–8, IEEE, 2008.

[4] M. A. Sadeghi and D. Forsyth, "30hz object detection with dpm v5," in *European Conference on Computer Vision*, pp. 65–79, Springer, 2014.

[5] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 580–587, 2014.

[6] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial pyramid pooling in deep convolutional networks for visual recognition," *IEEE transactions on pattern analysis and machine intelligence*, vol. 37, no. 9, pp. 1904–1916, 2015.

[7] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *arXiv preprint arXiv:1506.01497*, 2015.

[8] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "Ssd: Single shot multibox detector," in *European conference on computer vision*, pp. 21–37, Springer, 2016.

[9] T.-Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, "Feature pyramid networks for object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2117–2125, 2017.

[10] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *Proceedings of the IEEE international conference on computer vision*, pp. 2980–2988, 2017.

[11] S. Zhang, L. Wen, X. Bian, Z. Lei, and S. Z. Li, "Single-shot refinement neural network for object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4203–4212, 2018.

[12] Y. Li, Y. Chen, N. Wang, and Z. Zhang, "Scale-aware trident networks for object detection," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 6054–6063, 2019.

[13] C.-Y. Wang, A. Bochkovskiy, and H.-Y. M. Liao, "Scaled-yolov4: Scaling cross stage partial network," *arXiv preprint arXiv:2011.08036*, 2020.

[14] Z. Zou, Z. Shi, Y. Guo, and J. Ye, "Object detection in 20 years: A survey," *arXiv preprint arXiv:1905.05055*, 2019.

[15] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in*

neural information processing systems, vol. 25, pp. 1097–1105, 2012.

[16] A. Abdalwhab and H. Liu, "Zero-shot object detection for indoor robots," in *2019 International Joint Conference on Neural Networks (IJCNN)*, pp. 1–8, IEEE, 2019.

[17] M. Rezaei and M. Shahidi, "Zero-shot learning and its applications from autonomous vehicles to covid-19 diagnosis: A review," *Intelligence-based medicine*, p. 100005, 2020.

[18] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 779–788, 2016.

[19] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[20] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, pp. 3111–3119, 2013.

[21] C. L. Zitnick and P. Dollár, "Edge boxes: Locating object proposals from edges," in *European conference on computer vision*, pp. 391–405, Springer, 2014.

[22] J. R. Uijlings, K. E. Van De Sande, T. Gevers, and A. W. Smeulders, "Selective search for object recognition," *International journal of computer vision*, vol. 104, no. 2, pp. 154–171, 2013.

[23] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1532–1543, 2014.

[24] A. Bansal, K. Sikka, G. Sharma, R. Chellappa, and A. Divakaran, "Zero-shot object detection," in *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 384–400, 2018.

[25] P. Zhu, H. Wang, and V. Saligrama, "Zero shot detection," *IEEE Transactions on Circuits and Systems for Video Technology*, 2018.

[26] S. Rahman, S. Khan, and F. Porikli, "Zero-shot object detection: Learning to simultaneously recognize and localize novel concepts," in *Asian Conference on Computer Vision*, pp. 547–563, Springer, 2018.

[27] S. Rahman, S. H. Khan, and F. Porikli, "Zero-shot object detection: Joint recognition and localization of novel concepts," *International Journal of Computer Vision*, vol. 128, no. 12, pp. 2979–2999, 2020.

[28] D. Gupta, A. Anantharaman, N. Mamgain, V. N. Balasubramanian, C. Jawahar, *et al.*, "A multi-space approach to zero-shot object detection," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, pp. 1209–1217, 2020.

[29] S. Rahman, S. Khan, and N. Barnes, "Improved visual-semantic alignment for zero-shot object detection," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, pp. 11932–11939, 2020.

[30] Y. Xian, C. Lampert, B. Schiele, and Z. Akata, "Zero-shot learning - a comprehensive evaluation of the good, the bad and the ugly," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PP, 07 2017.

[31] E. Kodirov, T. Xiang, and S. Gong, "Semantic autoencoder for zero-shot learning," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 3174–3183, 2017.

[32] R. Krishna, Y. Zhu, O. Groth, J. Johnson, K. Hata, J. Kravitz, S. Chen, Y. Kalantidis, L.-J. Li, D. A. Shamma, M. Bernstein, and L. Fei-Fei, "Visual genome: Connecting language and vision using crowdsourced dense image annotations," 2016.

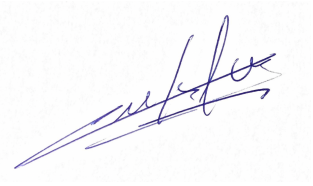
[33] R. Padilla, S. L. Netto, and E. A. da Silva, "A survey on performance metrics for object-detection algorithms," in *2020 International Conference on Systems, Signals and Image Processing (IWSSIP)*, pp. 237–242, IEEE, 2020.

[34] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 770–778, 2016.

Los abajo firmantes, miembros del Tribunal de evaluación de tesis, damos fe que el presente ejemplar impreso se corresponde con el aprobado por este Tribunal.

Cristian Cardellino

Jorge Sánchez



Matias Molina