

FACULTAD DE MATEMÁTICA, ASTRONOMÍA,
FÍSICA Y COMPUTACIÓN

UNIVERSIDAD NACIONAL DE CÓRDOBA



Zero-Shot Object Detection

TESIS

PARA OBTENER EL TÍTULO DE

LICENCIADO EN CIENCIAS DE LA COMPUTACIÓN

AGUSTIN HORACIO URQUIZA TOLEDO

DIRECTOR: JORGE SANCHEZ

CÓRDOBA, ARGENTINA

2021

DEDICATORIA

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

Pasado el año 2000, ocurrieron dos hechos que produjeron que las imágenes en Internet dieran un gran salto. Por un lado se empezaron a popularizar las cámaras digitales, y por otro, las conexiones de Internet subieron su velocidad. Esto generó la necesidad de crear métodos veloces y eficaces que faciliten la extracción de información en este tipo de datos. Luego, a partir del año 2010 con la “Revolución” del Aprendizaje profundo, surgieron una gran cantidad de métodos para realizar esta tarea, entre ellos los Detectores. Estos algoritmos necesitan tener una gran cantidad de imágenes anotadas que en algunos casos resulta inviable. Para resolver este problema surgieron técnicas como **Zero-shot Object Detection**.

Este trabajo final abordó este desafiante problema utilizando características visuales y descripciones semánticas, con el objetivo de detectar y reconocer simultáneamente instancias de objetos novedosos. Para esto, analizamos distintos trabajos y llevamos a cabo experimentos que aportan una noción del estado actual en esta área.

Summary

In the 2000s, the popularization of digital cameras and the increase of the Internet speed caused the rapid development of Internet images. This fact created the need of developing faster and more efficient methods to facilitate the extraction of information from these data. In 2010, with the Deep Learning “revolution”, many different methods emerged to carry out this task, among them Detectors. These algorithms use a large number of image annotations that, in some cases, results impossible to obtain. As an alternative to solve this problem, techniques such as **Zero-shot Object Detection** emerged.

This thesis tackles this problem using deep features and semantic embedding, with the aim of detecting and recognizing novel object instances. Hence, we analyzed different works from literature and we performed experiments that provide a notion of the current state in this area.

Índice general

1. Introducción y Motivación	1
1.1. Historia	1
1.2. Detectores y ZSD	4
1.3. Motivación	6
1.4. Estructura de la tesis	8
2. Preliminares	9
2.1. Redes neuronales convolucionales	9
2.2. Vectores densos de palabras	10
2.3. Propuesta de objetos	12
2.4. Multimodales	13
2.5. Aprendizaje por disparo cero (ZSL)	14
2.6. Detección de objeto por disparo cero (ZSD)	16
3. Metodología	19
3.1. Formalización de ZSD	19
3.2. Arquitectura y Diseño	21
3.2.1. Arquitectura	21
3.2.2. Conjuntos de datos	23
3.2.3. Detalles de la implementación	26
3.3. Experimentos	27
3.3.1. Experimentación con propuesta de objetos	27
3.3.2. Experimentación con CNN	28
3.3.3. Definición de métricas	30

3.3.4. Detalles de metodología de evaluación	33
4. Análisis de resultados	35
4.1. Resultados cuantitativos	35
4.1.1. Resultados ZSD	35
4.1.2. Resultados GZSD	38
4.2. Resultados cualitativos	40
5. Conclusiones y trabajo futuro	43
5.1. Conclusiones y aportes	43
5.2. Trabajo futuro	44
Bibliografía	47

Capítulo 1

Introducción y Motivación

1.1. Historia

La detección de objetos es una de las áreas de la visión por computadora que está creciendo más rápidamente. Gracias al aprendizaje profundo, cada año, los nuevos algoritmos/modelos siguen superando a los anteriores. Aunque la visión por computadora recientemente tomó gran importancia (el momento decisivo ocurrió en 2012 cuando AlexNet ganó ImageNet), ciertamente no es un nuevo campo científico.

Uno de los artículos más influyentes en Visión Informática fue publicado por dos neurofisiólogos, David Hubel y Torsten Wiesel [1], en 1959. Su publicación, titulada “*Receptive fields of single neurons in the cat’s striate cortex*”, en español “Campos receptivos de neuronas individuales en la corteza estriada del gato”, describió las propiedades de respuesta central de las neuronas corticales visuales y como la experiencia visual de un gato moldea su arquitectura cortical. Los investigadores establecieron a través de su experimentación (Figura 1.1) que existen neuronas simples y complejas en la corteza visual primaria, y que el procesamiento visual siempre comienza con estructuras simples como los bordes orientados y gradualmente identifica estructuras más complejas. En la actualidad, este es el principio básico detrás del

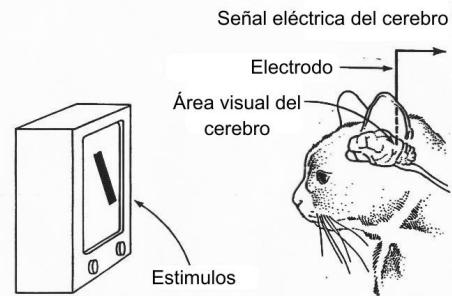


Figura 1.1: Simple explicación del experimento realizado por David Hubel y Torsten Wiesel

aprendizaje profundo.

Otro hecho importante en la historia de la visión por computadora fue en 1957, cuando Russell Kirsch y sus colegas desarrollaron un aparato que permitía transformar imágenes en cuadrículas de números que las máquinas de lenguaje binario podían entender.

Poco tiempo después, en la década de 1960 fue cuando la inteligencia artificial (IA) se convirtió en una disciplina académica y algunos de los investigadores eran extremadamente optimistas sobre el futuro del campo. En este período, Seymour Papert, profesor del laboratorio de IA del MIT, decidió lanzar el Proyecto de Verano y resolver, en pocos meses, el problema de la visión artificial. Los estudiantes debían diseñar una plataforma que pudiera realizar automáticamente segmentación de fondo y extraer objetos no superpuestos de imágenes del mundo real. Claro está que el proyecto no fue un éxito. Hoy en día, cincuenta años después, todavía no se ha podido resolver la visión por computadora. Sin embargo, ese proyecto fue el nacimiento oficial de esta disciplina como campo científico.

Los aportes más influyentes en este campo empezaron a surgir a partir de los años 2000. En 2001, Paul Viola y Michael Jones [2] 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 caras, inspirándose en el experimento de David Hubel y Torsten Wiesel.

En 2006, comenzó la competencia de Pascal VOC que permitió evaluar el desempeño de diferentes métodos para el reconocimiento 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) [3] que cambió todo, dado que logró una tasa de error del 16,4 %. En los años siguientes, las tasas de error en la clasificación de imágenes en ILSVRC cayeron a un pequeño porcentaje, como se observa en la Figura 1.2 y los ganadores, desde 2012, siempre han sido redes neuronales convolucionales.

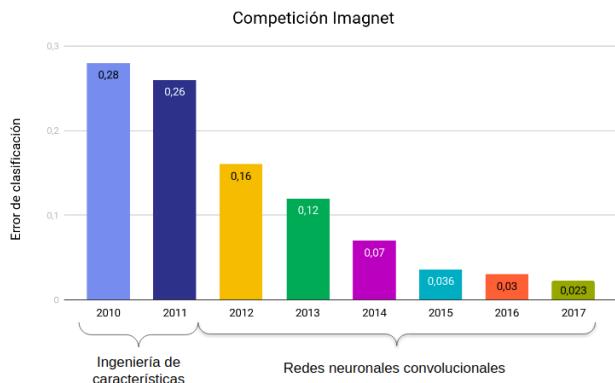


Figura 1.2: Evolución de los modelos propuestos en la competencia ILSVRC

1.2. Detectores y ZSD

La detección de objetos es un subproblema de la visión artificial, que estudia cómo detectar la presencia de objetos en una imagen. Debido a la complejidad de poder detectar todas las instancias de todos los posibles objetos en una imagen, se dividió en distintas tareas para disminuir la dificultad.

Par explicar los distintos problemas, es necesario distinguir dos conjuntos. Por un lado, los datos de entrenamiento, que consta de las imágenes que se usan para entrenar el modelo con sus respectivas etiquetas, es decir, que 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 medirá la eficiencia del modelo ya entrenado.

Supongamos que las etiquetas solo cuenta con dos tipos de información, que 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 clase que no sea una clase vista las llamearemos invisible o no vista. Dicho esto, los distintos problemas son:

- **Clasificación:** consta de un modelo capás de predecir si una clase específica esta presente en una imagen.
- **Clasificación más localización:** además de poder clasificar tiene que ser capas de ubicar el objecto en la imagen.
- **Reconocimiento de imagen:** predice que objetos perteneciente a las clases visibles están presente en la imagen.
- **La detección de objetos:** además de reconocer objetos visibles, tiene que ser capás de localizar dichos objetos.
- **Reconocimiento por disparo cero:** tiene que poder reconocer clases vistas y no vistas.

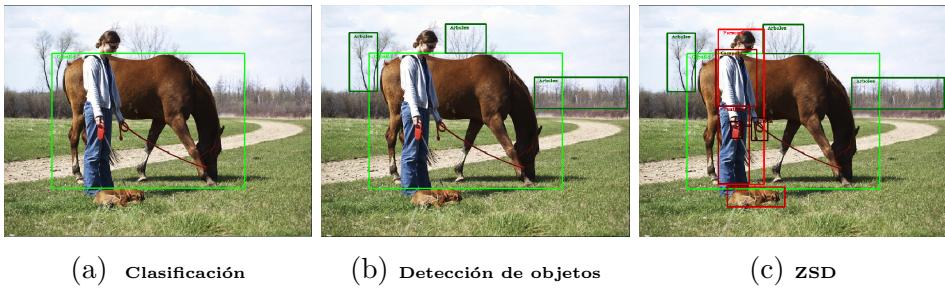


Figura 1.3: Ejemplo de tareas de clasificación más localización, detección de objetos y ZSD. En la escala de los verdes se encuentran las clases vistas {Caballo, Árbol}, y en rojo las clases invisibles {Perro, Persona, Campera, Pantalón, Correa}.

- **Detección de objetos por disparo cero (ZSD** por sus siglas en inglés): debe localizar y clasificar todas las instancias de objetos en la imagen, sin depender si es una clase vista o no.

La Figura 1.3 muestra un ejemplo de los resultados esperados por las distintas tareas.

Además de los problemas mencionados anteriormente, existen otros como la segmentación, que no desarrollaremos en este trabajo. Aquí, solo nos enfocaremos en ZSD y sus problemas asociados.

Existen muchas técnicas propuestas para resolver ZSD. Cuando se empezó a leer sobre este tema a fines del 2018, la más utilizada consistía en emplear multimodales. Puntualmente existían tres trabajos en paralelos [4][5][6] con una metodología similar. La idea de esta técnica es utilizar un espacio compartido entre las representaciones de visión y del lenguaje. Para lograr esto, se utiliza **incrustaciones de palabras y vectores con representaciones visuales**. Las primeras asignan a palabras una representación vectorial continua. Estos vectores se utilizan para medir similitudes semánticas y sintácticas entre palabras. Entre los modelos más famosos se encuentran Glove [7] y Word2vec [8]. Por otro lado, para obtener los vectores visuales de una

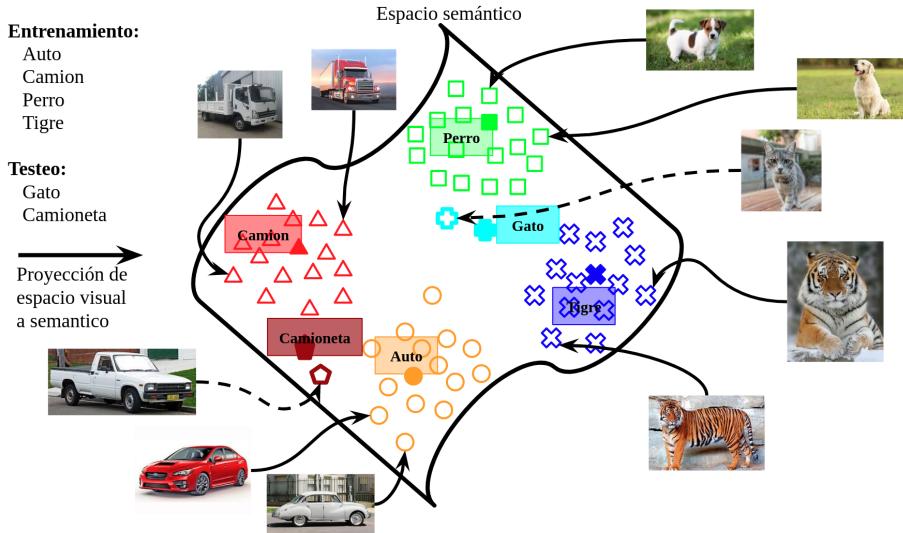


Figura 1.4: Descripción de la tarea de detección de objetos por disparo cero utilizando multimodales, donde los objetos “Auto”, “Camión”, “Perro” y “Tigre” se observan durante el entrenamiento, “Gato” y “Camioneta” son clases invisibles. El enfoque localiza estas clases no vistas aprovechando las relaciones del espacio semantico.

imagen se utilizan redes profundas. Entre los mejores modelos se encuentran VGG [9], ResNet [10] e Inception [11]. La Figura 1.4 describe como se utiliza la combinación de vectores de palabras y visuales para inferir un objetos nunca antes vistos por el modelo.

1.3. Motivación

Hoy en día, hay una gran cantidad de modelos, capaces de detectar objetos en una imagen, como son las redes YOLO o Faster R-CNN. Estos, como otros no mencionados, poseen una excelente rendimiento, pero tienen una gran limitación, necesitan una gran cantidad de

imágenes anotadas, para cada clase que se quiere detectar. Conseguir un gran numero de anotaciones, pude resultar un gran desafío, ya sea por la naturaliza del problema o por los grandes costo que esto conlleva. Esta dificultad se intenta mitigar con ZSD dado que puede inferir objetos no anotados.

ZSD es una habilidad que los humanos ya tienen. De hecho, podemos aprender muchas cosas con solo un “conjunto de datos mínimo”. Por ejemplo, tendemos a diferenciar variedades de la misma fruta o frutas de aspecto similar, aun si hemos visto muy pocas veces cada tipo de fruta. La situación es diferente para las máquinas. Necesitan muchas imágenes para aprender a adaptarse a la variación que se produce de forma natural en lo humanos. Esta habilidad proviene de nuestra base de conocimientos lingüísticos existente, que proporciona una descripción de alto nivel de una clase nueva o no vista y establece una conexión entre ella y las clases que ya conocemos. Como la visión por computadora intenta simular en una maquina el comportamiento de la visión humana, resulta razonable intentar resolverlo este problema de manera similar a como nosotros interpretamos el contexto visual.

Por unos minutos dejemos llevarnos por la imaginación y supongamos que se quiere crear un programa capas de reconocer todos los objeto en una imagen, pero objetos de cualquier índole, animales, plantas, artículos de limpieza, o cualquier cosa que se te venga a la mente. Seria casi imposible, si es que no lo es, generar un conjunto de datos que contenga una cantidad considerable de imágenes de todos los objetos posible. Esta idea puede sonar muy descabellada, o no, pero no se puede negar su potencial y su gran cantidad de usos como en interpretaciones de escenas, seguridad, etc. A medida que ZSD continúa desarrollándose, se espera ver más aplicaciones, como 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.

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. En el Capítulo 3 se comienza formalizando el problema de ZSD y se define la arquitectura empleada para resolverlo. Ademas se describe los conjuntos de datos utilizados para entrenar y medir el rendimiento del modelo, como así también los detalles de nuestra implementación. Por último se define las distintas métricas utilizadas y describe los distintos experimentos realizados. Luego en el Capítulo 4 se analizan los resultados obtenidos y se compara con distintos trabajos. El Capítulo 5 expone las conclusiones que se obtuvieron, los aportes realizados por esta tesis, y los trabajos futuros o mejoras.

Capítulo 2

Preliminares

2.1. Redes neuronales convolucionales

Las redes neuronales convolucionales, o CNN por sus siglas en inglés, es un tipo de modelo de aprendizaje profundo para procesar datos que tiene un formato de cuadrícula, como 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. 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 2.1. Las dos primeras realizan extracción de características, mientras que la tercera, las relaciona y genera una salida. La capa de convolución desempeña un papel clave en CNN y consiste de una pila de operaciones matemáticas, como la convolución. En las imágenes en 2D estas redes son muy utilizadas, por su alta eficiencia para tareas de visión artificial, como en la clasificación y segmentación de imágenes, entre otras aplicaciones.

Algunos ejemplos de redes CNN son: VGG16 [9] (que posee 13 capas de convolución, 5 de agrupación y una totalmente conectada) y AlexNet [3] (que contiene 5 capas convolucionales, 3 capas de agru-

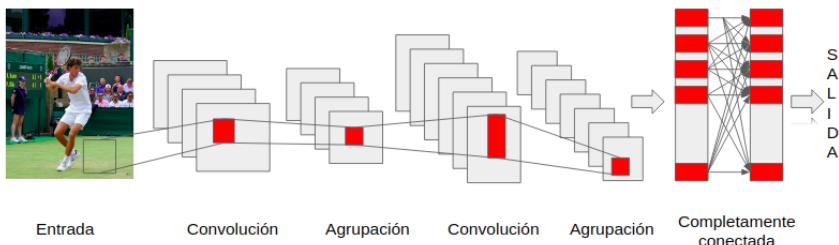


Figura 2.1: Una arquitectura simplificada de una rede neuronal convolucional.

pación y 3 capas completamente conectadas).

Las redes CNN se a utilizado para resolver distintos problemas como, la detección de objetos, Fast R-CNN [12], la comprensión visual de escenas de calles urbanas [13], entre otros. En este trabajo utilizamos la salida de las redes CNN (la capa completamente conectada), como un vector de características visual de la imagen. Debido a que las CNN son muy eficaces reconociendo patrones, si dos imágenes tienen un aspecto similar, los vectores también tendrán una semejanza.

2.2. Vectores densos de palabras

Así como en las imágenes utilizamos las redes CNN, para obtener un vector que represente a la misma, es necesario un procedimiento para representar palabras con algún objeto matemático. Hay muchas formas de representar palabras, la más usada son los vectores densos de palabras, conocidos en inglés como *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, 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 espaciales cercanas. Para lograr esto, se introduce cierta dependencia de una palabra con las otras.

Word2Vec [14] 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 falsa 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.

En este trabajo, aprovechamos la capacidad de capturar similitudes semántica que tiene vectores densos de palabras, para relacionar las clases vistas con las clases invisibles.

2.3. Propuesta de objetos

En problemas de detección de objetos, generalmente se tiene que encontrar todos los objetos posibles en la imagen. La localización de objetos se refiere a identificar la ubicación de uno o varios objetos en la imagen. Un algoritmo de localización de objetos generará las coordenadas de la ubicación de los objetos con respecto a la imagen. 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 [15] y búsqueda selectiva (*selective search*) [16]. 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.

En este proyecto, como veremos en el Sección 3.3 se experimentó con Edge-Boxes y selective search, ya que estas generan una cantidad de propuestas significativamente menor a algoritmos del estilo de ventana deslizante. Aun así, procesar todas estas propuestas es engorroso. Ademas, estos modelos por lo general dan como resultados muchos cuadros con una gran superposición. Esto da lugar a una técnica denominada supresión no máxima (NMS), ejemplificada en la Figura 2.2b. Este algoritmo necesita de un puntaje que indica la confianza del cuadro delimitador y un criterio para comparar entre distintos cuadros. El criterio más común es Intersección sobre Unión (IoU), en la Figura 2.2a se muestra como se calcula sobre dos Bounding Boxes. La salida de NMS es un conjunto más reducido de propuestas, en la

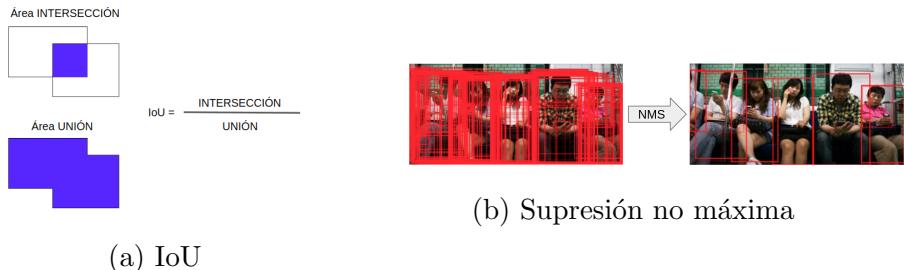


Figura 2.2: (a) Calculo de Intersección sobre Unión. (b) Salida de la propuesta de objetos y el resultado después de NMS.

cual se filtraron todas las que se consideran repetidas y retorna solo las más representativa. A continuación se muestra el pseudocódigo de NMS.

```

1: procedure NMS(B, S, t)
2:   D =  $\emptyset$ 
3:   for  $B \neq \emptyset$  do
4:      $b_i = \text{SelPropuestaMaxPuntaje}(B, S)$ 
5:     Eliminar( $b_i$ , B)
6:     for  $b_j \in B$  do
7:       if  $\text{IoU}(b_i, b_j) > t$  then
8:         Eliminar( $b_j$ , B)
9:       end if
10:      end for
11:    end for
12:    return D
13: end procedure

```

2.4. Multimodales

Nuestra experiencia del mundo es multimodal, vemos objetos y sus entornos, escuchamos música y ruidos, sentimos la textura de los

distintos materiales, etc. Inconscientemente asociamos una situación a los distintos estímulos que recibimos en ese momento y los relacionamos entre si, para genera una idea de lo que esta sucediendo. De esta manera, sabemos que si algo huele mal, lo más probable es que sepa igual, o podemos relacionar una imagen del campo con el sonido de los pájaros.

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. Para que la inteligencia artificial avance en la comprensión del mundo que nos rodea, necesita poder interpretar y relacionar estas distintas señales. Si bien la combinación de diferentes modalidades para mejorar el rendimiento parece una tarea intuitivamente atractiva, en la práctica, es un desafío, ya que una ineducada combinación de distintos tipos de información, puede generar confusión y conflictos.

La idea general es, a partir de dos objetos matemáticos distintos, uno de cada modal, poder transformarlos para lograr que pertenezcan a un tercer objeto que corresponde a la representación multimodal. Las imágenes suelen estar asociadas con etiquetas y explicaciones de texto. En este trabajo nos aprovechamos de esto y tratamos de encontrar un espacio común entre el vector que representan a la imagen del objeto y el que representa la sintaxis de la clase.

2.5. Aprendizaje por disparo cero (ZSL)

ZSL 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 en el aprendizaje automático, donde se espera que se clasifiquen correctamente las nuevas muestras en las clases que ya se han observado durante el entrenamiento. Podemos diferenciar dos tipos de

clases, las vidas que están presente en el entrenamiento y las invisibles o novedosas que no estuvieron en el mismo. Se debe proporcionar algún tipo de información complementaria sobre estas clases invisibles, este tipo de dato puede ser variado. Como por ejemplo, una descripción estructurada predefinida, que al tenerla en cuenta mejora el aprendizaje, o una descripción textual, donde las clases van acompañadas de un comentario en lenguaje natural. Por último, tanto las clases visibles como las invisibles están relacionadas en un espacio vectorial, donde el conocimiento de las clases vistas se puede transferir a clases invisibles.

El problema de aprendizaje por disparo cero 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.
 - **Aprendizaje por disparo cero inductivo:** Se tiene acceso a datos de imágenes etiquetadas de las clases vistas.
 - **Aprendizaje por disparo cero transductivo:** Además de los datos de imagen etiquetados de las clases vistas, también se tiene acceso a las imágenes no etiquetadas de las clases no vistas.
- Basado en los datos disponibles en el momento de la inferencia.
 - **Aprendizaje por disparo cero convencional (ZSL):** En las pruebas solo se evalúan las clases no vistas.
 - **Aprendizaje por disparo cero generalizado (GZSL):** En las pruebas se evalúan tanto las clases vista como las no vistas.

2.6. Detección de objeto por disparo cero (ZSD)

La Detección de objeto por disparo cero (ZSD), tiene como objetivo reconocer y localizar simultáneamente instancias de objetos que pertenecen a categorías novedosas sin ningún ejemplo de entrenamiento. Como estas categorías no están presente en el entrenamiento, no se tiene ninguna información sobre su aspecto visual, lo cual no nos permite detectarlas ni reconocerlas. Por lo tanto, es necesario encontrar algún dominio que tenga la capacidad de guardar la información de todas las clases, para luego relacionarlas con el aspecto visual de las categorías novedosas.

Este trabajo se basa en el artículo científico de Bansal *et al.* [6], además utiliza muchos conceptos sobre disparo cero generalizado [17]. Se propone un modelo de disparo cero 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 evaluar aprendizaje por disparo cero generalizado o convencional, respectivamente.

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 detección de disparo cero 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 menos 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 detección de disparo cero, a diferencia de lo que sucede con la clase “auto”.
- **Ruido en el espacio semántico:** cuando se utiliza los vectores de incrustación semántica no supervisados como word2vec o GloVe, las incrustaciones 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 la detección de disparo cero.

Existen algunas variaciones de ZSD que intentan atenuar estos problemas. Como **Detección de metaclase de disparo cero (ZSMD)**, que dada una imagen de prueba, el objetivo es localizar cada instancia de una clase de objeto invisible y categorizarla en una de las superclases. **Etiquetado de disparo cero (ZST)**, consiste en reconocer una o más clases invisibles en una imagen de prueba, sin identificar su ubicación. **Etiquetado de metaclase de disparo cero (ZSMT)**, reconoce una o más metaclases en una imagen de prueba, sin identificar su ubicación. Estas tareas son atractivas a la hora de calcular métricas, pero no para ser aplicadas a problemas reales.

Capítulo 3

Metodología

3.1. 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 $\mathcal{I} \in \mathbb{D}^{\mathcal{M} \times \mathcal{N} \times 3}$. Donde $\mathbb{D} = \{0, \dots, 255\}$, \mathcal{M} es el largo de la imagen, \mathcal{N} 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 $\mathbb{B} = \{b_0, \dots, b_k \mid b_i \in N^4\}$ y sus etiquetas asociadas como $\mathbb{Y} = \{y_0, \dots, y_k \mid y_i \in \mathcal{C}\}$. Para cada cuadro delimitador b_i extraemos una característica profunda utilizando una red neuronal convolucional denominada como $\phi(b_i) \in \mathbb{R}^{D_1}$.

Denotamos las incrustaciones semánticas $w_j \in \mathbb{R}^{D_2}$ obtenido por algún modelo como Wor2Vec. 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 las incrustaciones semánticas 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 a la incrustación semántica de su clase.

El resultado es una función

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

con un riesgo empírico regularizado mínimo \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 los $\mathbb{B}(x)$ es el conjunto de propuestas de la imagen x . Intuitivamente se buscan los cuadros delimitadores de mejor puntuación y se les asigna la categoría de objeto de puntuación máxima.

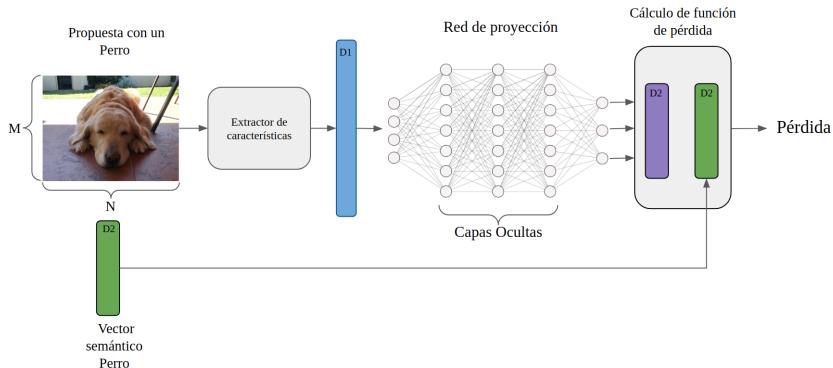


Figura 3.1: Arquitectura propuesta para ZSD, utilizando incrustaciones semánticas y visuales, ademas se muestra la dimensión en cada paso.

3.2. Arquitectura y Diseño

3.2.1. Arquitectura

Como se menciono anteriormente, hemos decidido basarnos en el modelo propuesto por Bansal *et al.* [6] para abordar el problema de ZSD. La arquitectura propuesta en este trabajo se puede dividir en las siguientes etapas:

- **Pre-procesamiento:** Se recorren todas las imágenes de entrenamiento y se extrae una característica profunda por cada cuadro delimitador, utilizando una red neuronal convolucional. Éste se asocia con el vector semántico de la clase que tiene asignada el cuadro, que se puede obtener con modelos de incrustación de palabras previamente entrenados, como Glove o Word2vec. Esta etapa nos genera como salida dos listas

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

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

- **Entrenamiento:** Utilizamos el espacio de incrustación común (R^{D_2}) para calcular una medida de similitud entre las proyecciones $\phi(b_i)$ y los vectores densos de palabras w_i . Luego, se entrena la proyección usando una función de pérdida, que impone 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.

La función de pérdida definida como:

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

donde m es el margen máximo, y S_{ij} es la similitud entre la proyección i -esima y la incrustación j -esima.

También se agrega una función de pérdida de reconstrucción como sugieren Kodirov *et al.* [18]. Se utilizan las características del cuadro delimitador proyectadas para reconstruir las características profundas originales, y calcular la pérdida de reconstrucción como la distancia $L2$ entre la característica reconstruida y la característica profunda 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:

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

En la Figura 3.1 se puede apreciar la arquitectura completa propuesta en este trabajo.

- **Evaluación:** Por cada imagen de entrenamiento se genera un conjunto de propuestas de cuadros delimitadores. Luego, se eliminan todos los que no tienen un puntaje de confianza mayor

a un umbral t . Para cada cuadro obtenido se computa la característica profunda $\phi(b_i)$ y utilizando la matriz de proyección W_p , se predicen las características semánticas. Por último, se calcula la similitud con las características semánticas de todas clases invisibles, asignando al cuadro delimitador la que tenga mayor puntaje.

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 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.

3.2.2. Conjuntos de datos

Common Objects in Context (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 de puntos clave de personas. Este conjunto de datos contiene 91 tipos de objetos o clases, con un total de 2.5 millones de instancias etiquetadas en 328.000 imágenes.

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 las imágenes consta de una gran cantidad de objetos, a diferencia de conjuntos como Visual Genome. Esto 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.

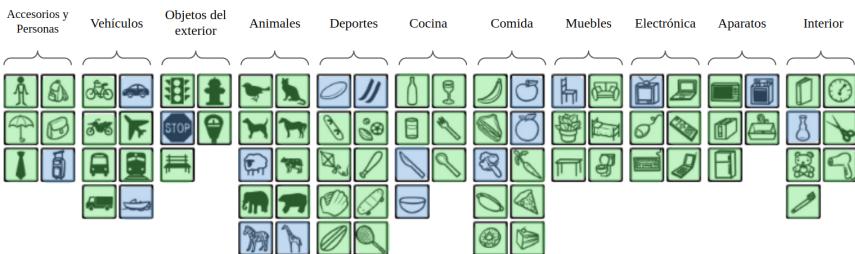


Figura 3.2: División de las clases para entrenamiento (verde) y pruebas (azul).

Como COCO no provee una separación de los datos para evaluar modelos de ZSD, es necesario crear una forma de dividir las clases en 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 de que COCO tiene agrupadas las clases por “Clases superiores”, y de 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 3.2 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 42564 imágenes con 261258 instancias de entrenamiento, y 3008 imágenes con 10878 instancias de prueba.

Bansal *et al.* [6], 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 lograr una comparación de modelos más justa.

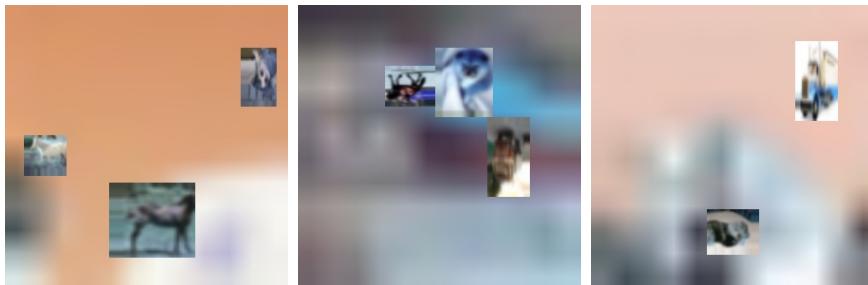


Figura 3.3: Ejemplos de imágenes del conjunto de datos CIFAR-ZSD.

COCO puede resultar pesado en término computacional. Para solucionar esto se creó un conjunto de datos sintético basado en CIFAR-100 datasets, el cual denominamos CIFAR-ZSD. Éste consta de imágenes localizadas, rotadas y re-escalada aleatoriamente con un fondo de otra imagen (algunos ejemplos se pueden ver en la Figura 3.3). Con esto se intenta simular imágenes reales en la cual un objeto puede aparecer con distintos aspectos y escalas. Este conjunto está dividido para que ninguna instancia de prueba aparezca en el conjunto 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, ayudan a enfrentar el problema de ZSD de una forma más práctica.

3.2.3. Detalles de la implementación

El paper de Bansal *et al.* [6], carece de una implementación de acceso público, por lo cual, se realizó una implementación propia, basándose en los detalles que se pueden extraer del documento publicado. Fue necesario realizar algunas suposiciones, pero también, nos otorgó flexibilidades a la hora de codificar. Se buscó obtener resultados lo más cercanos posibles a los reportados, siendo fiel a la información disponible. Para esto se decidió utilizar Python 3 con el framework Keras ejecutándose sobre TensorFlow.

Primero se realiza un preprocesamiento a todas las imágenes del conjunto de datos de entrenamiento, que consiste en generar propuestas de objetos utilizando *Edge Boxes* o *Selective Search*. En la Sección 3.3 se muestran los resultados de cada algoritmo. Luego por cada propuesta se calcula la intersección sobre unión con todos los cuadros delimitadores verdaderos. Si el IoU > 0,5 con algún cuadro verdadero, se guarda la propuesta y se la asocia con la clase del cuadro delimitador verdadero. Este preprocesamiento aumenta significativamente la cantidad de instancias para la etapa de entrenamiento, que resulta en un total de 1.436.835 instancias para COCO y 137.204 para CIFAR-ZSD. El siguiente paso consiste en generar el vector de características visuales y semánticas de cada cuadro delimitador. Como las CNN, utilizan un tamaño de entrada fijo, es necesario rescalar todos los cuadros. Para *VGG16* utilizamos 224×224 , y en *Inception ResNet V2* 299×299 , que son las dimensiones por defecto. El tamaño del vector de salida de cada red es de 512 en VGG y 1536 ResNet. Para ambas usamos pesos preentrenados en Imagenet. Este paso es diferente a como se hace en el artículo de Bansal *et al.* [6], ya que en este trabajo el modelo se entrena de extremo a extremo, es decir los pesos de la CNN se ajustan en la etapa de entrenamiento. Para obtener las características semánticas, utilizamos *Word2Vec* previamente entrenado por *Google News 300*. Este incluye vectores para un vocabulario de 3 millones de palabras y frases, entrenado en aproximadamente 100 mil

millones de palabras de un conjunto de datos de Google News. La longitud del vector resultante es de 300. Luego se guardan dos archivos por cada imagen, en uno se encuentran los vectores visuales de cada clase y en el otro los semánticos.

Por último, para el entrenamiento, se creó una red con una sola capa oculta, una capa de entrada del tamaño del vector de características visuales, y una capa de salida de la dimensión de las características semánticas. Lo cual resulta en 153.900 parámetros entrenables con *VGG16* y 461.100 con *Inception ResNet V2*. Para esta etapa, se utilizó el optimizador Adam, un taza de aprendizaje de 10e-3, un tamaño del lote de 64 muestras y no se uso ninguna activación. Para la función de perdida se uso un lambda de 10e-3 y un margen máximo de 1.

3.3. Experimentos

3.3.1. Experimentación con propuesta de objetos

Como se mencionó anteriormente, el número de propuestas es un parámetro clave. Algunas métricas son muy sensible 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* y *Selective Search*), 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 entrenamiento se corrió el generador de propuestas, y se calculó el tiempo

Algoritmo	Edge Boxes				Selective Search	
	-				Single	Fast
Numero de propuestas	100	500	1000	5000	≈ 5000	≈ 1000
Tiempo promedio (s)	0,11	0,11	0,12	0,12	5,48	1,41
Propuestas Totales	4.415.244	22.050.071	43.802.935	161.809.194	350.535.591	95.643.172
Propuestas con IOU > 0,5	86.233	133.942	155.584	194.891	221.551	203.563

Tabla 3.1: Resultados de correr los distintos algoritmos de propuestas de regiones en los datos de entrenamiento. El número de propuestas verdaderas es 261.258.

y la cantidad de cuadros verdaderos que tenían un $\text{IoU} > 0,5$, con algún cuadro verdadero. El tiempo es un parámetro importante ya que algunos algoritmos soy muy lentos y resultan imposible de usar.

Como se puede observar en la Tabla 3.1, *Selective Search* obtiene una mayor cantidad de superposición, pero con un número exageradamente grande de propuestas, por lo que la mejor opción es usar *Edge-boxes*. 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 pero si aumenta el número de propuestas. Si tenemos en cuenta el tiempo, resulta mejor *Edge-boxes*, ya que demora una fracción de lo que tarda *Selective Search*.

3.3.2. 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. Lo que se quiere aquí es que la red sea capaz de asociar las 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.* [6] propone utilizar *Inception ResNet*

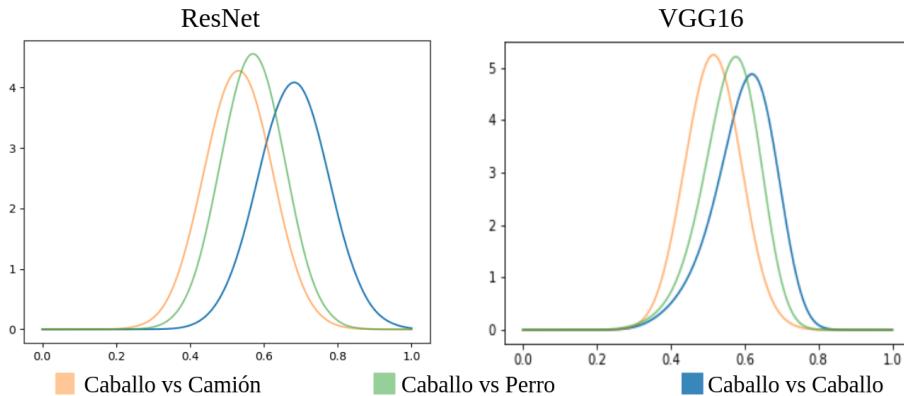


Figura 3.4: Frecuencia de la similitud coseno de los vectores de características visuales, entre la misma y distintas clases, para las CNN **Inception ResNet V2** y **VGG16**.

V2, 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*, que reduce el número de parámetros en las capas convolucionales y mejorar el tiempo de ejecución, además es una de la más utilizada.

El experimento consistió en comparar miles de cuadros de 3 clases de entrenamiento, caballo, perro y camión. Por cada cuadro se generó el vector de características 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 3.4 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 animales es más grande que con un vehículo. Se observó que para *Inception ResNet V2* existe una mayor separación entre clases, aunque sus similitudes están más dispersas. *VGG16* 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.

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 problema que tienen las métricas en detección de objetos, es la falta de una implementación estándar para calcularlas. Además, aquellas implementaciones públicas están muy encapsuladas al código, y resulta muy difícil adaptarlo para medir rendimientos de modelos propios. Como ya se mencionó anteriormente, el código de Bansal *et al.* [6] no está disponible, por este motivo fue necesario encontrar alguna implementación de estas métricas. A partir de estas búsquedas 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 y científicas, tal como se plantea en artículo de Padilla *et al.* [19]. Además, propone una definición y un código para estandarizar las métricas, de manera que se puedan comparar 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.* [6].

Ahora definamos las métricas, basándonos en el trabajo [19]. Primero es necesario estandarizar:

- Falso negativo (**FN**): Para un cuadro delimitador verdadero no se obtuvo ninguna detección en absoluto, o una propuesta tiene $\text{IoU} > \text{umbral}$ 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 una predicción duplicada, es decir, ya se marcó otra con mayor IoU como **TP**, o se detectó 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.

El umbral por lo general es 0.5, pero se puede cambiar para exigir que tenga una mayor superposición.

La *Recall*, también conocida como sensibilidad, mide la probabilidad de que los objetos verdaderos (los que se encuentran en la imagen) se detecten correctamente, viene dado por:

$$Recall = \frac{TP}{FN + TP} \quad (3.1)$$

El trabajo de Bansal *et al.* [6], 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 de verdadero y no se ha asignado ningún otro cuadro delimitador de mayor confianza al mismo cuadro de verdadero. 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. Esto dificulta la tarea de comparar con otros modelos, con lo cual es en este trabajo reportamos ambas formas.

Bansal *et al.* [6] además calcula una variación denominada *K@Recall*, donde sólo se tienen en cuenta 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 porcentaje de predicciones positivas correctas entre todas las predicciones realizadas y se define como:

$$\text{Precision} = \frac{TP}{FP + TP} \quad (3.2)$$

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

Para entender mejor supongamos un límite tal que genera un número 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 límite 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 límite) la *precision* 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 qué 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.* [6] 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 distinto trabajos.

3.3.4. Detalles de metodología de evaluación

El principal experimento consistió en replicar los resultados de Bansal *et al.* [6], aunque no se realizaron exactamente los mismos experimentos, ya que esto no aportaría nada nuevo. Así que, se decidió analizar sus resultados y solo replicar los que consideramos indispensable y que sería un buen punto de partida. 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. También, se guarda la puntuación como la confianza de predicción. Por último, se agrupan todas las propuestas que se tengan asignada la misma clase y se corre un algoritmo de supresión no máxima. Este 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. Estos datos se guardan en un archivo y luego se corre la implementación de Padilla *et al.* [19], para obtener los resultados de las métricas.

Capítulo 4

Análisis de resultados

En este capítulo se analizan los distintos resultados obtenidos en nuestros experimentos, detallados en la Sección 3.3, se comparan con los de Bansal *et al.* [6] y se explican las diferencias obtenidas.

4.1. 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.

4.1.1. Resultados ZSD

Metrica	Baseline [6]	DSES [6]	Nuestros con VGG	Nuestros con ResNet	Mejor resultado de [20]
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 4.1: Resutados obtenidos por Bansal *et al.* [6], nosotros y Rahaman *et al.* [20]. Se presentan las distintas métricas *recall* y *mAP*, evaluados en COCO.

La idea inicial de esta tesis era replicar el modelo base de Bansal *et al.* [6], una vez logrado esto, modificar nuestra implementación con el objetivo de mejorar los resultados. Pero esta idea se frustró al no obtener valores similares a los reportados. Para encontrar la causa de esta diferencia se necesitó depurar cada etapa del código. Luego de confirmar que la implementación no tenía ningún error aparente y replicaba correctamente lo que se define en el trabajo de Bansal *et al.* [6], se comenzó a modificar los distintos parámetros de cada etapa, aun sin éxito, se analizaron las métricas, y como ya se explicó anteriormente se encontró una diferencia en la definición, esto explica la discrepancia en los resultados que analizaremos a continuación.

La Tabla 4.1, muestra los valores de las métricas *100@Recall*, en la versión desarrollada por Bansal *et al.* [6] y la de Padilla *et al.* [19], también se muestran los resultados para *mAP@0.5* y *mAP[.5, .95]*. Se observan 2 modelos propuestos por nosotros, uno utilizando *VGG16* y el otro *Inception ResNet V2*, ademas se agregan los mejores resultado presentado por el trabajo de Rahman *et al.* [20]. 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 número de propuestas predichas correctamente sobre el total de cuadros verdaderos. Obtuvo 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 3.3.2, *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 por la comunidad científica. Pero al igual que *100@Recall* se ve afectada por los puntos antes mencionados, ademas, existe otro que la afecta muy fuertemente. Por la naturaleza de la matriz que proyecta las características visuales al espacio semantico, hace que dos objetos proyectados obtengan una similitud coseno poco distanciada, la cual ronda entre los valores 0.3 y 0.6. Es decir que si proyecto dos imágenes con muy pocas diferencias obtendrán un similitud como máximo de 0.6. Esta similitud es utilizada como el puntaje de confianza de una predicción y como se explico en la Subsección 3.3.3, *mAP* varia un limite de confianza de predicciones para calcular la curva AUC. Esto hace que para limites superiores a 0.6 se obtengan valores muy bajos o incluso nulos de *precision*.

En comparacion con el mejor resultado de Bansal *et al.* [6] el cual denomina *Densely Sampled Embedding Space (DSES)* y consiste en aumentar el procedimiento de entrenamiento con datos adicionales de fuentes externas que contienen casillas que pertenecen a clases distintas a las clases invisibles. Obtiene 27.19 puntos en su definición de *recall*, el cual nuestro modelo base supera con 28.91 puntos usando *ResNet*. Esto se debe a que en la etapa de depuración se modificaron algunos parámetros por defecto del entrenamiento, como el numero de lote, la taza de aprendizaje, el optimizador, etc.

En cuanto *mAP*, Bansal 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 *recall* y refleja lo poco representativa de esta ultima métrica.

Si comparamos con un trabajo más actual [20] 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 que la implementación utilizada para calcular las métricas esta mejor encaminada.

Estos resultados fueron calculados sobre la división de clases propuesta por Bansal ya que ambos trabajos con cuales comparamos utilizan esta. Pero también se corrieron las evaluaciones con la partición propuesta en este trabajo. 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.* [6] utiliza como criterio de división los vectores semánticos de las clases, esto afecta positivamente ya que es el mismo espacio utilizado para inferir las clases invisibles.

También se calcularon las métricas para el conjunto de datos CIFAR-ZSD. Pero fueron necesario algunas mejoras para adaptarse a las diferencias con COCO. Primero se utilizó una tamaño de la entrada de la CNN más pequeña de 32x32. Esto se debe a que las imágenes no tienen una gran resolución. También, se reduzco considerablemente el numero máximo de de propuestas, del 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.* [19]) 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 es solo un modelo base muy sencillo sin utilizar ningún tipo de información extra, obtuvimos valores aceptables para *100@Recall*, aunque resulta importante resaltar su bajo desempeño en *mAP*.

4.1.2. Resultados GZSD

Por ultimo, se analizaron los resultados en el desafío de GZSD. La configuración generalizada de aprendizaje de disparo cero es más realista que la configuración de disparo cero discutida anteriormente, porque tanto las clases visibles como las invisibles están presentes

Modelo	GZSD		
	Clases vistas	Clases Invisibles	Media
	mAP/Recall Bansal/Recall	mAP/Recall Bansal/Recall	mAP/Recall Bansal/Recall
Mejor resultado [6]	-/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 [20]	13.93/-/20.42	2.55/-/12.42	4.31/-/15.45

Tabla 4.2: Resultados obtenidos, en el desafío GZSD, para los modelos de Bansal *et al.* [6], nuestro (ResNet) y Rahman *et al.* [20]

durante la evaluación.

La Tabla 4.2, se muestra los resultados para GZSD evaluados en COCO. En esta se puede observar el desempeño obtenido en $100@Recall$ y $mAP@0.5$, para las cuales este trabajo obtuvo 3.84 puntos y 0.13 respectivamente en promedio con las clases visibles e invisibles. Como es de esperarse se obtuvo un mejor rendimiento para las clases vistas en ambas métricas.

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

Si comparamos los resultados de ZSD vs GZSD, se observa un decrecimiento en los valores de las métricas promedio. Esta disminución también se ve reflejado en los trabajos [6] y [20]. El motivo de esto es que las clases vistas al estar en entrenamiento, tienden a tener un mejor puntaje en la etapa de evaluacion que las clases invisibles perteneciente a su clase superior. Por esto muchos objetos que en la configuración anterior, predecía correctamente ahora una clase visible obtiene mejor puntaje.

4.2. Resultados cualitativos

Para poder tener una idea más realista del comportamiento del modelo propuesto, se probó sobre algunas imágenes de muestra. Con el objetivo de obtener ejemplos más claro fue necesario modificar la metodología que se utilizó para calcular los resultados cuantitativos. Para esto, disminuimos el número de propuestas en el orden de 10 y se descartó todas aquellas que obtuvieron un puntaje de confianza menor a 0.5. Esto hace que los ejemplos solo contengan los objetos más relevantes, pero una contra de esto es que descarta muchos que están en un segundo plano. Otra aclaración importante es que se evalúa ZSD y no GZSD, es por esto que se ignoran o confunde clases visibles.

La Figura 4.1 muestra las detecciones del modelo propuesto en el conjunto de datos COCO. Los cuadros azules muestran detecciones incorrectas y en verdes los que acertaron a la clase que 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 a pesar de que reducimos la cantidad de propuestas, 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 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”.

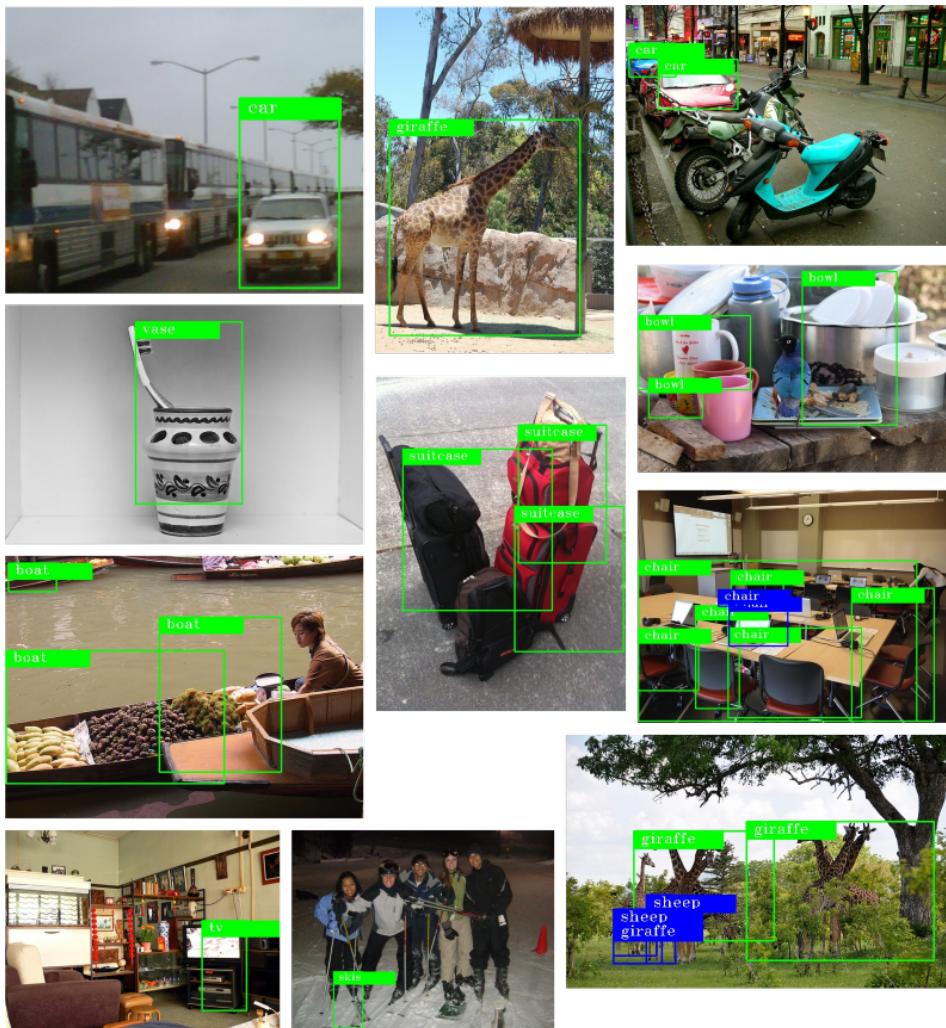


Figura 4.1: Ejemplo del comportamiento del modelo sobre clases invisibles. Los cuadros azul son las predicciones incorrectas y en verde las correctas.

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 generar 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 incrustaciones visuales y semánticas para abordar el problema. Por este motivo decidimos utilizar esta metodología, para proponer un modelo base, basándonos en el trabajo de Bansal *et al.* [6]. 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 al nuestro entender, fueron ignorados por el trabajo original pero resulta cruciales.

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

Pero 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 genera una mala interpretación. Para resolver esto, se analizaron distintos artículos que señalaban el problema que tuvimos. Pero el trabajo de Padilla *et al.* [19] sobresale a los demás, porque pose una clara definición de las métricas y una implementación fácil de utilizar, por lo que se recomienda 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 transparente 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 cuales podemos dividir en tres 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 cosenos con las demás clases y no solo la que obtenga mayor puntaje.

Bibliografía

- [1] 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.
- [2] 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.
- [3] 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.
- [4] 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.
- [5] P. Zhu, H. Wang, and V. Saligrama, “Zero shot detection,” *IEEE Transactions on Circuits and Systems for Video Technology*, 2018.
- [6] 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.

- [7] J. Pennington, R. Socher, and C. Manning, “Glove: Global vectors for word representation,” in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, (Doha, Qatar), pp. 1532–1543, Association for Computational Linguistics, Oct. 2014.
- [8] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient estimation of word representations in vector space,” *arXiv preprint arXiv:1301.3781*, 2013.
- [9] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” *arXiv preprint arXiv:1409.1556*, 2014.
- [10] 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.
- [11] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, “Going deeper with convolutions,” in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.
- [12] R. Girshick, “Fast r-cnn,” in *Proceedings of the IEEE international conference on computer vision*, pp. 1440–1448, 2015.
- [13] M. Cordts, M. Omran, S. Ramos, T. Rehfeld, M. Enzweiler, R. Benenson, U. Franke, S. Roth, and B. Schiele, “The cityscapes dataset for semantic urban scene understanding,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 3213–3223, 2016.
- [14] 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.

- [15] C. L. Zitnick and P. Dollár, “Edge boxes: Locating object proposals from edges,” in *European conference on computer vision*, pp. 391–405, Springer, 2014.
- [16] 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.
- [17] 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.
- [18] 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.
- [19] 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.
- [20] 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.
- [21] D. Marr, “Vision: A computational investigation into the human representation and processing of visual information,” New York, NY: W.H. Freeman and Company, 1982.
- [22] L. G. Roberts, *Machine perception of three-dimensional solids*. PhD thesis, Massachusetts Institute of Technology, 1963.
- [23] Y. Aytar, C. Vondrick, and A. Torralba, “See, hear, and read: Deep aligned representations,” *arXiv preprint arXiv:1706.00932*, 2017.

- [24] Y. Zhang, L. Yuan, Y. Guo, Z. He, I.-A. Huang, and H. Lee, “Discriminative bimodal networks for visual localization and detection with natural language queries,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 557–566, 2017.