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

DETECCIÓN DE OBJETOS USANDO REDES NEURONALES CONVOLUCIONALES JUNTO CON RANDOM FOREST Y SUPPORT VECTOR MACHINES

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO

DIEGO ALEJANDRO CAMPANINI GARCÍA

PROFESOR GUÍA:
JAVIER RUIZ DEL SOLAR SAN MARTÍN

MIEMBROS DE LA COMISIÓN:
PABLO ESTÉVEZ VALENCIA
FELIPE TOBAR HENRÍQUEZ

SANTIAGO DE CHILE
2018

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO
POR: DIEGO ALEJANDRO CAMPANINI GARCÍA
FECHA: 2018
PROF. GUÍA: SR. JAVIER RUIZ DEL SOLAR SAN MARTÍN

DETECCIÓN DE OBJETOS USANDO REDES NEURONALES CONVOLUCIONALES JUNTO CON RANDOM FOREST Y SUPPORT VECTOR MACHINES

En el presente trabajo de título se desarrolla un sistema de detección de objetos (localización y clasificación), basado en redes neuronales convolucionales (CNN por su sigla en inglés) y dos métodos clásicos de *machine learning* como Random Forest (RF) y Support Vector Machines (SVMs). La idea es mejorar, con los mencionados clasificadores, el rendimiento del sistema de detección conocido como Faster R-CNN (su significado en inglés es: ***R****egions with* ***CNN*** *features*).

El sistema Faster R-CNN, se fundamenta en el concepto de *region proposal* para generar muestras candidatas a ser objetos y posteriormente producir dos salidas: una con la regresión que caracteriza la localización de los objetos y otra con los puntajes de confianza asociados a los *bounding boxes* predichos. Ambas salidas son generadas por capas completamente conectadas. En este trabajo se interviene la salida que genera los puntajes de confianza, tal que, en este punto se conecta un clasificador (RF o SVM), para generar con estos los puntajes de salida del sistema. De esta forma se busca mejorar el rendimiento del sistema Faster R-CNN.

El entrenamiento de los clasificadores se realiza con los vectores de características extraídos, desde una de las capas completamente conectadas del sistema Faster R-CNN, específicamente se prueban las tres que contempla la arquitectura, para evaluar cuál de estas permite obtener los mejores resultados. Para definir, entre otras cosas, el número de capas convolucionales a utilizar y el tamaño de los filtros presentes en las primeras capas del sistema Faster R-CNN, se emplean los modelos de redes convolucionales ZF y VGG16, estas redes son solamente de clasificación, y son las mismas ocupados originalmente.

Para desarrollar los sistemas propuestos se utilizan distintas implementaciones o librerías para las cuales se dispone de su código de forma abierta. Para el detector Faster R-CNN se utiliza una implementación desarrollado en Python, para RF se comparan dos librerías: randomForest escrita en R y *scikit-learn* en Python. Por su parte para SVM se utiliza la librería conocida como LIBSVM escrita en C. Las principales tareas de programación consisten en desarrollar los algoritmos de etiquetado de los vectores de características extraídos desde las capas completamente conectadas; unir los clasificadores con el sistema base, para el análisis *online* de las imágenes en la etapa de prueba; programar un algoritmo para el entrenamiento eficiente en tiempo y en memoria para SVM (algoritmo conocido como *hard negative mining*)

Al evaluar los sistemas desarrollados se concluye que los mejores resultados se obtienen con la red VGG16, específicamente para el caso en que se implementa el sistema Faster R-CNN+SVM con *kernel* RBF (*radial basis function*), logrando un *mean Average Precision* (mAP) de 68.9 %. El segundo mejor resultado se alcanza con Faster R-CNN+RF con 180 árboles y es de 67.8 %. Con el sistema original Faster R-CNN se consigue un mAP de 69.3 %.

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

Capítulo 1

Introducción

1.1. Antecedentes sobre Sistemas de Detección de Objetos

En los últimos años algunos de los principales desafíos relacionados con visión computacional han sido las tareas de clasificación, detección y segmentación de objetos [9], [45]. Este trabajo se centra en la tarea de detección de objetos.

La detección de objetos se entiende como la tarea de predecir la ubicación de uno o más objetos en una imagen y ser capaz de clasificarlos, asignándoles una de las clases con las cuales se entrena el respectivo sistema, habitualmente se define detección como localización más clasificación.

En las competencias como PASCAL[1] Visual Object Classes (VOC) [9] e ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [45], para evaluar las detecciones se requiere que la localización sea caracterizada por un *bounding box*, que representa un rectángulo que encierra a un objeto, además a este se debe asociar un puntaje, que indica el nivel de confianza para la detección de una clase en específico.

A lo largo del tiempo la tarea de detección de objetos se ha abordado de distintas maneras, por ejemplo, en un principio para PASCAL VOC07 los sistemas utilizaban predominantemente el método de ventanas deslizantes para la localización de los objetos, y luego extraían características a cada ventana para su posterior clasificación. Para la extracción de características existe una variedad de métodos como por ejemplo *Histogram of Oriented Gradients* (HOG) [6] y *Scale Invariant Feature Transform* (SIFT) [34].

Los métodos clásicos para las tareas de clasificación y detección comienzan a ser largamente superados a partir del año 2012, específicamente con el trabajo que dió origen a la red AlexNet [28] que ganó la ILSVRC de aquel año, para la tarea de clasificación. Para esto se usó una red neuronal convolucional (CNN por su sigla en inglés) [30] con cinco capas convolucionales

[1]PASCAL es la sigla para *Pattern Analysis, Statistical Modelling and Computational Learning.*

y tres capas completamente conectadas. Luego se sucedieron numerosas investigaciones que profundizaron en el uso de CNN para las tareas de clasificación entre las cuales destacan las redes conocidas como ZF [59], VGG [48] y ResNet [22], [23].

Conjuntamente con el surgimiento de los sistemas de clasificación que emplean CNN, se desarrollaron sistemas de detección, algunos de los cuales se basaron en métodos del tipo *region proposal*, que se fundamentan en la idea que todos los objetos de interés en una imagen comparten características que los diferencian del fondo. De esta forma no es necesario hacer un recorrido exhaustivo por toda la imagen como sucede bajo el paradigma de ventana deslizante.

A continuación se destacan tres sistemas de detección de gran éxito, los cuales usan de alguna forma la idea de *region proposal*.

1. **R-CNN**: Este sistema usa un método de *region proposal* como *selective search* [54] para generar muestras que se utilizan en el entrenamiento de las capas completamente conectadas de una red convolucional [15]. Generó un aumento de 30 % en el mAP con respecto a los mejores métodos de detección de VOC2012. Demora 13 s/imagen en GPU y 52 s/imagen en CPU en la etapa de prueba.

2. **Fast R-CNN**: Al igual que el sistema anterior usa *region proposal* y CNN, sin embargo, solamente la imagen completa pasa por las capas convolucionales y no cada *proposal* como se hace en R-CNN, producto de esto se logra una considerable disminución en la detección de objetos en una imagen. Los *proposals* se utilizan para identificar regiones de interés sobre los *feature maps* de salida desde la última capa convolucional, estas regiones son en las que se concentran las capas completamente conectadas. Demora 0.3 s/imagen en GPU [14].

3. **Faster R-CNN**: Este sistema de detección se diferencia de los dos anteriores, ya que, utiliza la misma red convolucional para generar los *proposals*, localizar y clasificar, por lo que, se puede entrenar de extremo a extremo sin depender de ningún método externo. Toma en total 198 ms/imagen en GPU [41].

1.2. Objetivos

1.2.1. Objetivo General

El propósito del presente trabajo es mejorar el sistema de detección de objetos conocido como Faster R-CNN [41], el cual fue utilizado como base por los ganadores de varias tareas en la competencia ILSVRC 2015 [44]. Para esto se propone utilizar dos métodos clásicos de *machine learning* que son Random Forest (RF) y Support Vector Machines (SVMs), los cuales pasan a ser los encargados de la etapa de clasificación del detector. La fase de localización del sistema es mantenida como se propone originalmente [41]. El sistema completo debe ser capaz de analizar las imágenes de entrada de forma *online*, realizando la clasificación y la localización de objetos. Además cada clasificador debe ser entrenado con características extraidas

desde las capas completamente conectadas del sistema base, sin agregar características extras diseñadas manualmente.

1.2.2. Objetivos Específicos

Los objetivos específicos asociados al objetivo general son los siguientes:

- Analizar el efecto de utilizar las mejores implementaciones (librerías) de RF y SVM. Lo anterior para la tarea de clasificación.

- Evaluar la capacidad del sistema Faster R-CNN de generar vectores de características que permitan a un usuario entrenar de forma *offline* los clasificadores RF y SVM.

- Evaluar distintos métodos de etiquetado de los vectores de características.

- Analizar cuál capa completamente conectada entrega los vectores de características que permiten obtener un mayor rendimiento del clasificador SVM.

- Implementar técnicas que permitan el entrenamiento eficiente, en tiempo y en uso de memoria, para SVM.

- Comparar los resultados de los sistemas implementados para dos arquitecturas de redes convolucionales distintas: ZF y VGG16.

1.3. Estructura del Documento

En este documento se ordena la información en 5 capítulos, incluyendo el actual. Los otros 4 se describen a continuación.

El Capítulo 2 presenta una revisión bibliográfica, destacando aspectos generales relacionados con la redes neuronales convolucionales, además describe en detalle el sistema que se utiliza como base, es decir, Faster R-CNN [41]. Finalmente se dasarrolla una descripción de SVM y Random Forest, centrándose en técnicas que se utilizan en los apartados posteriores.

En el tercer capítulo se describe la metodología seguida para implementar los sistemas propuestos, se detallan los algoritmos utilizados y las dificultades computacionales que se abordaron. Adicionalmente se describe la base de datos empleada en el desarrollo del sistema.

El cuarto capítulo trata sobre los resultados obtenidos con las distintas pruebas y un análisis de estos, realizando comparaciones con el sistema base para las dos redes empleadas, y para los métodos de etiquetado. Además se muestran ejemplos de detecciones del mejor sistema implementado. Por último el Capítulo 5 expone las conclusiones de este trabajo.

Capítulo 2

Antecedentes Generales

2.1. Deep Learning

Deep learning es un área de *machine learning*, la cual usa algoritmos de aprendizaje que efectúan procesamientos no lineales en múltiples capas, logrando la extracción de características en cada uno de los niveles, un aspecto clave es que estas características no son diseñadas manualmente, si no que son determinadas desde los datos en bruto por medio de algoritmos de aprendizaje. Los modelos que utilizan *deep learning* han mejorado el estado del arte en tareas tales como detección de objetos, clasificación de acciones de personas, procesamiento de lenguaje natural, y en muchas otras áreas donde se requiere aprender alguna representación desde los datos de entrada [1], [29].

Entre las distintas redes profundas, destaca particularmente en tareas de visión las redes neuronales convolucionales (CNNs) [30], las cuales fueron inspiradas por la estructura del sistema visual animal [1]. Estas redes han sido aplicadas con gran éxito en tareas de detección, segmentación y reconocimiento de objetos en imágenes desde principio de los 2000s, tal como se señala en [29].

El año 2012 ocurre un hecho importante en favor del uso de las CNNs. Se aplican estas resdes a la base de datos ILSVRC para la tarea de clasificación de objetos, logrando un rendimiento del 15.3 % para el error *top-5* en el conjunto de *test* [28], superando ampliamnete al segundo mejor resultado que fue de 26.2 %, conseguido con los métodos clásicos de clasificación de objetos usados hasta esa fecha (HOG [6], SIFT [34], entre otros). Luego de este éxito las CNNs se transformaron en el enfoque predomiante en las tareas de clasificación y detección de objetos [17], [29]. Lo antes señalado, corresponde a algunas de las razones por las cuales, las CNNs son discutidas con especial énfasis en el presente trabajo.

2.1.1. Redes Neuronales Convolucionales

Las redes neuronales convolucionales (CNNs) son un tipo de arquitectura profunda, que realizan procesamiento de la señal de entrada en diversas capas, siendo la principal, la capa convolucional [30]. Algunas de las características de estas redes neuronales es que son más fáciles de entrenar y poseen menos parámetros que estimar, en comparación con otras arquitecturas profundas. El entrenamiento de estas se realiza utilizando esencialmente dos algoritmos: gradiente descendente y *backpropagation* [17], [26], [28], [29], [41], [43].

El gradiente descendente es un algoritmo de optimización de primer orden, empleado para encontrar el mínimo o máximo de una función. Consiste básicamente en actualizar los parámetros de una función objetivo en la dirección opuesta al gradiente de esta con respecto a los parámetros que se actualizan [17], [43].

En [43] se mencionan tres variantes del gradiente descendente: gradiente descendente por *batch*, gradiente descendente estocástico (SGD por su sigla en inglés), y gradiente descendente por *mini-batch*, este último es el algoritmo seleccionado típicamente para entrenar redes neuronales y se suele llamar SGD, aunque en estricto rigor SGD corresponde al gradiente descendente estocástico [26], [43].

Las tres variantes del gradiente descendente señaladas en el párrafo anterior se diferencian en la cantidad de datos que utilizan para calcular el gradiente de la función objetivo [26], [43]. A continuación se describen brevemente los tres métodos.

- **Gradiente descendente por *batch*:** Calcula el gradiente usando todo el conjunto de entrenamiento y realiza con este una actualización de los parámetros.

$$w = w - \eta \bigtriangledown_w L(w) \tag{2.1}$$

- **Gradiente descendente estocástico:** A diferencia del método anterior que usa todos los datos de entrenamiento, este algoritmo utiliza 1 sola muestra del conjunto de entrenmiento ($x^{\mathrm{i}} \rightarrow \mathrm{e}jemplo, y^{\mathrm{i}} \rightarrow \mathrm{e}tiqueta$), para realizar una actualización de los parámetros.

$$w = w - \eta \bigtriangledown_w L(w; x^{\mathrm{i}}; y^{\mathrm{i}}) \tag{2.2}$$

- **Gradiente descendente por *mini-batch*:** Este método toma los mejores aspectos de los dos anteriores y realiza una actualización por cada *mini-batch* de n ejemplos de entrenamiento.

$$w = w - \eta \bigtriangledown_w L(w; x^{\mathrm{i:i+}n}; y^{\mathrm{i:i+}n}) \tag{2.3}$$

En las ecuaciones (2.1), (2.2) y (2.3), η representa el paso o la tasa de aprendizaje del algoritmo. Este hiperparámetro determina que tan grande es la actualización que se realiza. La elección de la tasa de aprendizaje es de suma importancia para el éxito del entrenamiento de las redes neuronales, un valor pequeño implicará una convergencia muy lenta, por otro

lado, un valor demasiado grande puede causar que la función objetivo o de pérdidas, fluctúe entorno al mínimo o que el algoritmo diverja.

En conjunto con el gradiente descendente se utiliza el algoritmo de *backpropagation* para entrenar las redes neuronales, tal como se mencionó al inicio de esta sección. El algoritmo de *backpropagation* se refiere al método para calcular el gradiente mediante la aplicación recursiva de la regla de la cadena [17], [26].

Una de las principales deficiencias de las redes neuronales en tareas relacionadas con reconocimiento de imágenes y patrones de voz, es que no son invariantes ante traslaciones, escalamientos y distorsiones de la señal de entrada. En las CNNs se logra cierto grado de invariancia con respecto a las distorsiones antes mencionadas combinando tres ideas en su arquitectura: campos receptivos locales, pesos compartidos y sub-muestreo espacial o temporal.

Los campos receptivos son las entradas a las unidades de una capa. En CNN estos son forzados a ser locales, ya que, en imágenes o en representaciones tiempo-frecuencia de voz, las variables cercanas están altamente correlacionadas. Esta correlación local es la razón para combinar características locales antes del reconocimiento de objetos, porque, de esta forma, por ejemplo, los pixeles cercanos se pueden clasificar en bordes, esquinas, entre otros.

Estas redes comparten pesos, lo que se refiere a que un conjunto de unidades que tienen como entradas los campos receptivos locales aplicados en distintas partes de una imagen, poseen el mismo vector de pesos. Las unidades se organizan en planos llamados *feature maps*, que realizan la misma operación de convolución en distintos sectores de una imagen.

El sub-muestreo es una forma de reducir la exactitud con la que la ubicación de las características son codificadas en un *feature map*. Esto se realiza porque la posición exacta de una característica no es tan importante, incluso puede llegar a ser dañina, ya que, aumenta la sensibilidad de la red ante desplazamientos y distorsiones de los ejemplos.

2.1.1.1. Arquitectura de las CNNs

La arquitectura de las CNNs contempla un gran número de capas, las más usuales son las capas convolucionales, las de sub-muestreo, las completamente conectadas y las de normalización, aunque el uso de esta última es menos común. Cabe mencionar que en algunos trabajos la función de activación también es señalada como una capa. En la Figura 2.1 se muestra un diagrama de una CNN con sus respectivas capas. A continuación se proceden a describir.

- **Capa Convolucional:** Esta capa es la principal de la red y está compuesta por varios *feature maps* con distintos pesos. Las unidades dentro de un mismo *feature maps* comparten los pesos y bias.

  La capa consiste en filtros convolucionales entrenables, los pesos de estos son los parámetros que se entrenan. Algunos aspectos que se tienen que elegir en esta capa son el número de filtros, el tamaño de estos y el paso con el que irán recorriendo la imagen.

- **Capa de Submuestreo:** Esta capa tiene como función reducir la cantidad de parámetros y tamaño de la red. Para esto se condensa la información de unidades vecinas en un *feature map*, por medio de una operación matemática, como puede ser, obtener el máximo o el promedio.

  El submuestreo se puede realizar sin o con solapamiento, aunque con este último se han obtenido mejores resultados como se expone en [28]. Algunos de los parámetros que se deben fijar es el tamaño del filtro, el paso y el método a utilizar (máximo, promedio, entre otros).

- **Capa de Normalización** En [28] se propone utilizar en la red un esquema de normalización local para ayudar a la generalización, logrando una disminución del error al probarlo en el conjunto de datos CIFAR-10. Sin embargo, en [48] no logran mejorar sus resultados al emplear normalización en el conjunto de datos de ILSVRC. Además obtienen como consecuencia de utilizar esta técnica, un aumento en el tiempo de cálculo y en el consumo de memoria.

- **Capa Completamente Conectada:** Es la capa final de una arquitectura típica de CNN que se caracteriza por estar completamente conectada a las unidades de la capa anterior, como en las redes neuronales clásicas. Un parámetro que se debe fijar es el número de neuronas.

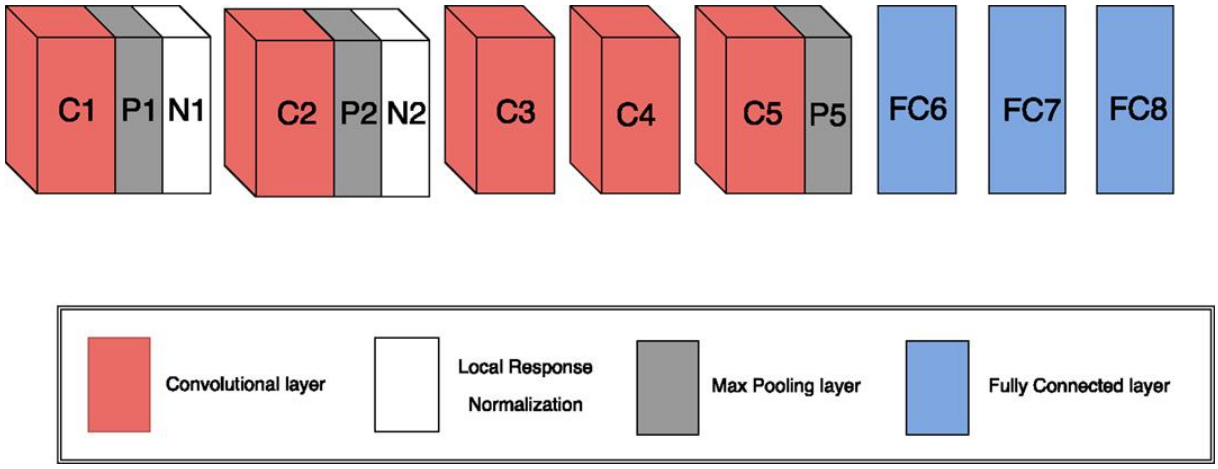


Figura 2.1: Esquema que ejemplifica los tipos de capas y el orden en que se disponen en la red convolucional conocida como AlexNet [28]. Imagen obtenida desde [49].

Cabe mencionar que después de cada capa de convolución y completamente conectada, se aplica de forma habitual una función de activación no lineal. Las más comúnmente utilizadas son la función tangente hiperbólica ($f(x) = tanh(x)$), función sigmoidea ($f(x) = (1+\mathrm{e}^{-x})^{-1}$) y función *Rectified Linear Units* o ReLU ($f(x) = max(0, x)$). En [28] se señala que la función ReLU es la más rápida en términos del tiempo de entrenamiento. En la Figura 2.2 se muestran las funciones de activación antes mencionadas.

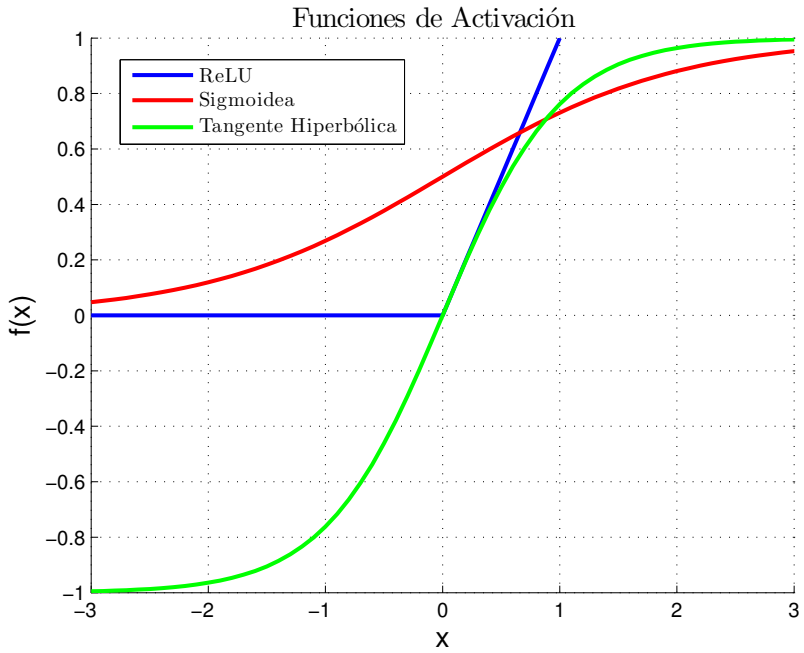
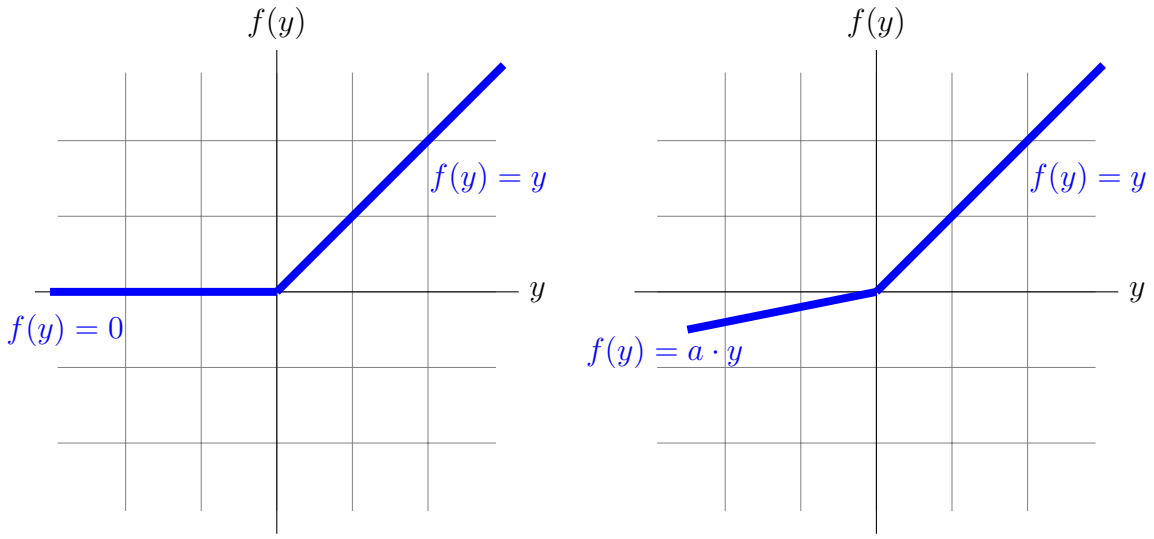


Figura 2.2: Funciones de activación usadas comúnmente en CNN.

En [21] se propone una nueva generalización para la función ReLU denominada *Parametric ReLU* (PReLU). Esta función logra mejorar los resultados en un 26 % comparado con la red ganadora de la competencia de clasificación más localización de objetos ILSVRC 2014, se obtiene un 4.94 % de error top-5, con casi cero costo computacional. Además con la utilización de PReLU se ha conseguido superar el rendimiento humano (5.1 % top-5 error [45]) en la tarea señalada para la misma base de datos. La función PReLU se define como sigue:

$$f(y_{\mathrm{i}}) = \begin{cases} y_{\mathrm{i}} & \text{si } y_{\mathrm{i}} > 0 \\ a_{\mathrm{i}} y_{\mathrm{i}} & \text{si } y_{\mathrm{i}} \leq 0 \end{cases} \tag{2.4}$$

En la ecuación 2.4, y_{i} es la entrada a la función de activación por cada canal, por su lado, a_{i} es un parámetro entrenable que puede variar en los diferentes canales. Cuando $a_{\mathrm{i}} = 0{,}01$ corresponde a la función propuesta en [35], la cual es denominada *leaky rectified linear* (LReL) y fue utilizada en redes neuronales profundas para modelos acústicos, obteniendo casi idénticos resultados que con la función ReLU estándar. En la Figura 2.3 se compara la función ReLU con la PReLU.



(a) *Rectified Linear Units* (ReLU). **(b)** *Parametric* ReLU (PReLU).

Figura 2.3: Funciones de activación ReLU y PReLU, estas son usadas habitualmente en CNN luego de una capa convolucional o una completamente conectada. Imagen basada en [21].

2.1.1.2. Reducción del Sobreajuste

Como fue mencionado previamente las redes neuronales profundas poseen múltiples capas ocultas no lineales, las cuales posibilitan aprender complejas relaciones entre entradas y salidas. Sin embargo, debido al gran número de parámetros, si se cuenta con pocos datos en el conjunto de entrenamiento, la configuración trabajará de manera óptima solamente sobre este conjunto y no sobre uno de prueba, produciendo un sobreajuste de la red.

Es necesario recurrir a técnicas para combatir el sobreajuste y de esta forma lograr una disminución del error en las bases de datos de prueba. Estas técnicas por lo general apuntan a aumentar artificialmente el número de datos o incluir un mecanismo de regularización en el proceso de entrenamiento.

Aumento de los Datos Existen varias formas de aumentar artificialmente los datos de una misma base. Por ejemplo en [28] se proponen dos métodos, el primero consiste en extraer aleatoriamente una porción de 224x224 pixeles desde las imágenes de entrada que son de 256x256 y entrenar con estas porciones. Además se utilizan las reflexiones horizontales de las porciones consideradas.

La segunda forma consiste en modificar las intensidades de los canales RGB en las imágenes de entrenamiento, para esto se buscan sus componentes principales en la colección de pixeles RGB del conjunto. Luego la modificación de las intensidades de los pixeles se realiza adicionando la siguiente cantidad:

$$[p_1.p_2, p_3][\alpha_1\lambda_1, \alpha_2\lambda_2, \alpha_3\lambda_3]^T \tag{2.5}$$

En la ecuación anterior p_i y λ_i son los vectores y valores propios de la matriz de covarianza de 3x3 de los valores RGB de los pixeles, y α_i es una variable aleatoria obtenida desde una distribución gaussiana con media cero y desviación estándar 0.1.

En la misma línea del primer método en [16] aumentan los datos para la detección de personas, utilizando porciones de la imagen, que coincidan en más de un 70 % del área con el *ground truth*, logrando un incremento de 100 veces el conjunto de entrenamiento.

Dropout El dropout es una técnica que consiste en retirar de forma aleatoria las unidades de una red neuronal durante la etapa de entrenamiento, tal como se ejemplifica en la Figura 2.4. Se ha mostrado que la utilización de esta técnica mejora el rendimiento en múltiples tareas que requieren la utilización de redes neuronales [24],[50].

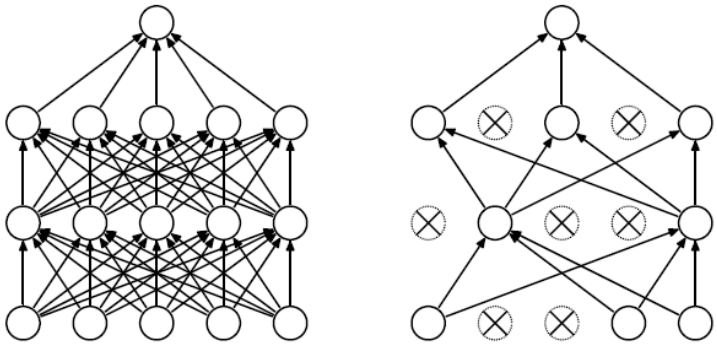


Figura 2.4: A la izquierda se muestra una red sin dropout y a la derecha una red luego de aplicar esta técnica en la etapa de entrenamiento. Imagen obtenida desde [50].

Un procedimiento que la mayoría de las veces es efectivo en la reducción del sobreajuste, consiste en promediar las predicciones hechas por un gran número de redes diferentes, esto se puede realizar con redes que posean diversas arquitecturas o que son entrenadas en conjuntos de datos distintos. Sin embargo, lo anterior resulta poco práctico por varias razones, como la elección de parámetros de las redes, el tiempo de entrenamiento, la cantidad de datos necesarios y el tiempo de respuesta del sistema completo.

El dropout permite de cierto modo realizar la tarea de promediar muchas redes, considerando como una red a las unidades que sobreviven a este. Una forma de implementar esta técnica es fijando la probabilidad de retención de cada unidad en p, el valor de este puede ser elegido usando un conjunto de validación o ser fijado en 0.5, sin embargo, para las unidades de entrada se encontró que, en la mayoría de los casos, 0.8 es el valor óptimo.

En la etapa de prueba lo que se realiza es multiplicar por la probabilidad de retención p cada peso que sale desde una respectiva unidad. Lo anterior constituye una forma aproximada de promediar los modelos de redes entrenadas utilizando dropout. Un inconveniente de esta técnica es que toma 2-3 veces más tiempo de entrenamiento que una red estándar con la misma arquitectura.

Una extensión del dropout es el DropConnect [55], el cual consiste en retirar de forma

aleatoria los pesos dentro de una red al momento de entrenar, en vez de retirar las unidades completas como lo realiza el dropout. Este método logra mejores resultados en varios casos estudiados. Otra variación al dropout se presenta en [56], en la cual se logra un aumento de la rapidez de entrenamiento en un orden de magnitud.

2.1.1.3. Transferencia de Aprendizaje

En términos generales la transferencia de aprendizaje se refiere a la capacidad de un sistema de aplicar a una tarea nueva el conocimiento aprendido en una tarea previa [38], [58]. Este método de transferencia puede resultar particularmente útil cuando el problema abordado inicialmente cuenta con significativamente más datos que un segundo problema para el cual se utiliza la transferencia [17].

La transferencia de aprendizaje en CNN se puede realizar por ejemplo utilizando como extractor de características a una red pre-entrenada. Para efectuar lo anterior se emplea la red completa como un extractor de características fijo y se usan las salidas desde las capas completamente conectadas para entrenar un nuevo clasificador (por ejemplo SVM), sobre una base de datos distinta a la que se usó para entrenar la red previamente. Si los nuevos datos están lo suficientemente relacionados con los anteriores se obtendrán buenos resultados, dado que las primeras capas de las CNNs extraen características genéricas (por ejemplo bordes o manchas de colores) [26].

Se han desarrollado numerosos trabajos explorando la transferencia de aprendizaje de las CNNs [7], [37], [40], [58]. Por ejemplo en [7] usan la red AlexNet [28] (entrenada en la base de datos de ILSVRC 2012 para clasificación como tarea original) para extraer características desde las capas completamente conectadas y con estas entrenar, por ejemplo, un clasificador SVM. Lo anterior lo realizan para distintas bases de datos de reconocimiento de objetos, tales como: Caltech-101 (reconomiento de objetos), *Office dateset* (adaptación de dominio), Caltech-UCSD (reconocimiento de subcategorías) y SUN-397 (reconocimiento de escenas) [7].

Las bases de datos mencionadas en el párrafo anterior incluyen clases distintas que las usadas originalmente para entrenar la red AlexNet. Cabe destacar que mediante la transferencia de aprendizaje aplicada en [7], se logró superar los resultados obtenidos con los métodos clásicos para las tareas de reconocimiento, en las bases de datos antes citadas.

2.1.2. Tipos de Arquitecturas Comunes para CNNs

Existen varios tipos de arquitecturas de CNNs que han sido de importancia para mejorar el rendimiento de estas redes en trabajos posteriores [26]. La primera red que obtuvo exitosos resultados es la de Yann LeCun en los 90 [31], en su trabajo propuso la red conocida como LeNet, la cual fue utilizada para reconocer dígitos escritos a mano (Figura 2.5). Posterior a este desarrollo se destaca la red propuesta en [28], conocida como AlexNet (Figura 2.1), que obtuvo una notable disminución del error en la tarea de clasificación de la base de datos

ImageNet 2012. A continuación se describen dos redes conocidas como ZFNet y VGGNet, las cuales son posteriores a LeNet y AlexNet, y que ayudaron a aumentar la precisión de las CNNs y fueron las redes empleadas en el sistema Faster R-CNN.

Figura 2.5: Ejemplos de la base de datos MNIST. Imagen obtenida desde [31].

2.1.2.1. ZFNet

La red convolucional conocida como ZFNet fue planteada en [59]. En este trabajo toman como arquitectura base la red propuesta por [28] (AlexNet), planteando un perfeccionamiento de esta a partir de la introducción de una técnica de visualización de los estímulos que excitan *feature maps* individuales en cualquier capa.

La arquitectura utilizada en varios de los experimentos en [59] consiste de una imagen de entrada reescalada al tamaño de 224x224, la primera capa está compuesta de 96 filtros convolucionales, cada uno de estos de tamaño 7x7 (en AlexNet son de 11x11 para la primera capa), además el paso es de 2 pixeles en las dos dimensiones (en AlexNet el paso es de 4 pixeles). Luego el resultado de esta convolución pasa por una función ReLU, después se aplica un submuestreo mediante la extracción del máximo en regiones de 3x3 con un paso de 2. Finalmente se aplica una operación de normalización, a través de cada *feature maps*, lo cual resulta en 96 salidas de 55x55.

En las capas 2, 3, 4 y 5 se realizan operaciones similares a las antes descritas, por su parte, las capas 6 y 7 corresponden a capas completamente conectadas, la capa 8 es la salida de la red, la que se calcula como una función softmax para las C clases que tiene como respuesta el sistema completo. La Figura 2.6 muestra un esquema de la arquitectura de la red.

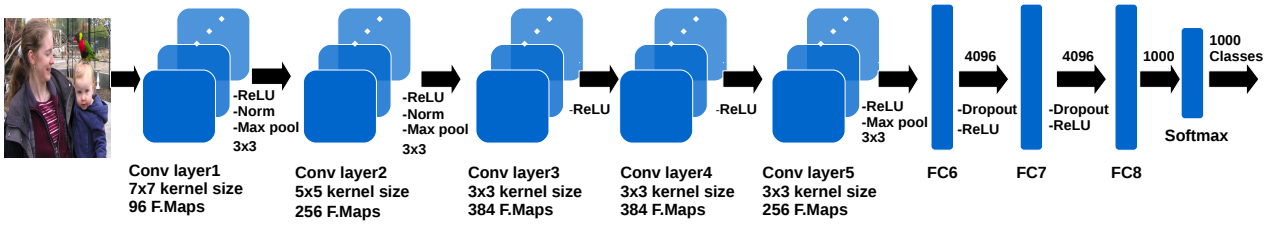


Figura 2.6: Modelo de la CNN denominada ZFNet, en esta se especifica el número de *feature maps* de salida en cada capa, el tamaño de los *kernels* y las operaciones que se aplican entre una capa y otra (ReLU, normalización, *max pooling* o *dropout*).

2.1.2.2. VGGNet

El modelo VGGNet es planteado en [48], trabajo en el cual se muestra la importancia de la profundidad de las redes convolucionales para mejorar la precisión de estas. En particular se consideran arquitecturas con 11, 13, 16 y 19 capas (distribuidas en n capas convolucionales y 3 capas completamente conectadas).

En la Figura 2.7 se detalla la configuración con 16 capas. Cabe destacar que la red con 19 capas es fundamentalmente lo mismo, salvo que antes de las tres últimas operaciones de *pooling* se agrega una capa convolucional, respetando el número de *feature maps* de la capa que las antecede, es decir, se agruegan las siguientes capas: Conv layer3-4 (256 *feature maps*), Conv layer4-4 (512 *feature maps*) y Conv layer5-4 (512 *feature maps*).

La red VGGNet, en todas sus configuraciones, comienza con un grupo de capas convolucionales, con campo receptivo de 3x3, con paso de 1 pixel, algunas de estas son seguidas de una operación de submuestreo, que se realiza calculando el máximo en ventanas de 2x2 pixeles, con paso de 2. El grupo de capas convolucionales es seguido por 3 capas completamente conectadas. Las dos primeras poseen 4096 canales, mientras que la tercera posee 1000, finalmente la salida es una función softmax. Otro aspecto a destacar es que todas las unidades ocultas utilizan como no linealidad la función ReLU.

La configuración con 19 capas (16 capas convolucionales y 3 capas completamente conectadas) es la que obtiene los mejores resultados [48], esto al evaluar las distintas profundidades de redes en la tarea de clasificación de ILSVRC-2014. Por otro lado, al mezclar los modelos con 16 y 19 capas, logran superar los rendimientos obtenidos por las redes de forma individual, disminuyendo el error en el conjunto de prueba en 0.2 puntos porcentuales en comparación con lo alcanzado por la red de 19 capas [48].

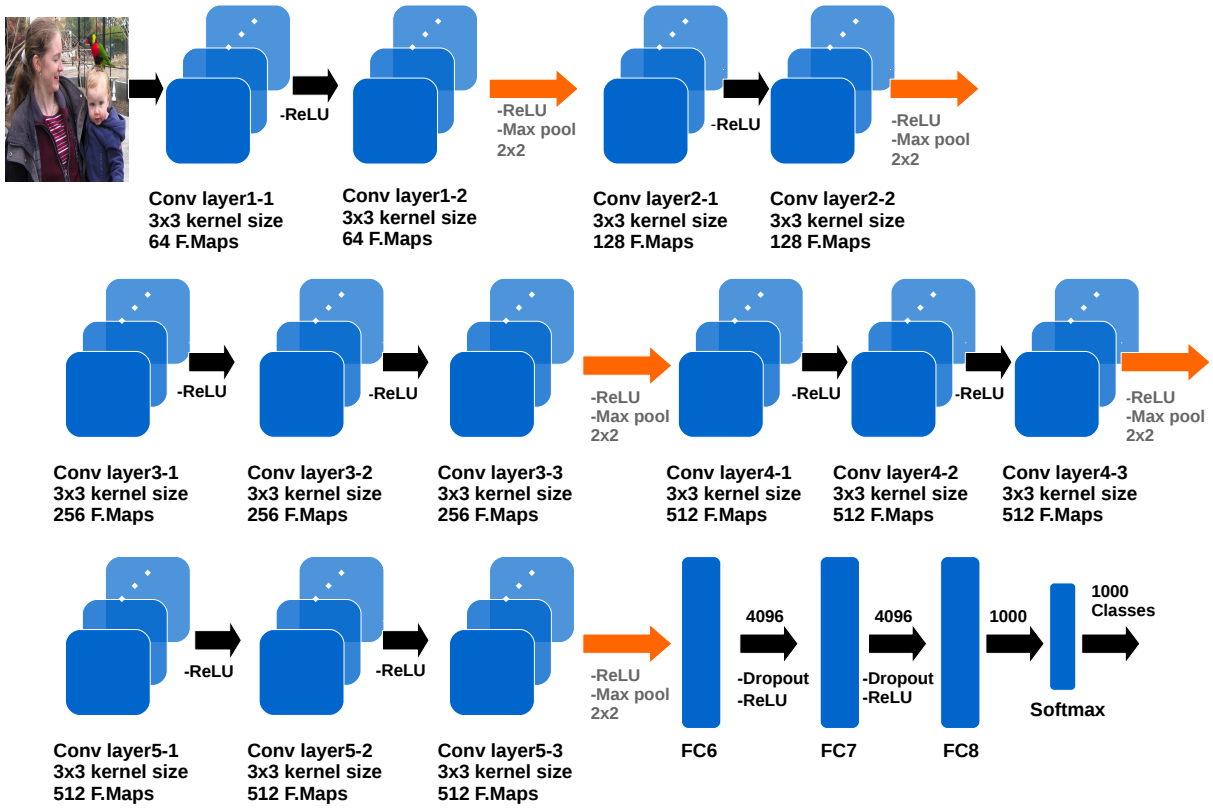


Figura 2.7: Esquema para la red VGG16, compuesta por 13 capas convolucionales y 3 capas completamente conectadas, además la salida es una capa *softmax*. Notar que las capas convolucionales se agrupan en 5 conjuntos, los cuales están separados por una operación de *max pooling* (flecha naranja).

2.1.3. Sistemas de Detección de Objetos Basados en Region Proposals

2.1.3.1. R-CNN

El sistema de detección de objetos conocido como R-CNN (su significado en inglés es: ***R****egions with* ***CNN*** *features*) se presenta en [15]. Este trabajo propone utilizar un método externo de *region proposals* para extraer candidatos a ser objeto en una imagen de entrada, luego cada propuesta es dilatada en $p = 16$ pixeles y escalada a una dimension de 227x227, puesto que, la red CNN utilizada requiere entradas de esta dimensión. A continuación estas regiones entran a la red CNN y se extrae desde una capa completamente conectada un vector de dimensión 4096 (uno por cada región propuesta). Con los vectores extraídos para cada región, se procede a entrenar un conjunto de SVMs lineales para cada clase del problema, estos clasificadores son los que predicen la clase de salida de la red. La Figura 2.8 presenta un esquema del sistema de detección antes descrito.

El sistema R-CNN utiliza para generar los *region proposals* el método conocido como

selective search [54]. En una primera implementación la localización la generaba el método de *region proposal*, pero luego de analizar las fuentes de error del sistema se agrega una regresión para refinar la localización generada en primera instancia [15]. Esta regresión es la encargada de producir los *bounding boxes* de salida para cada objeto localizado en una imagen.

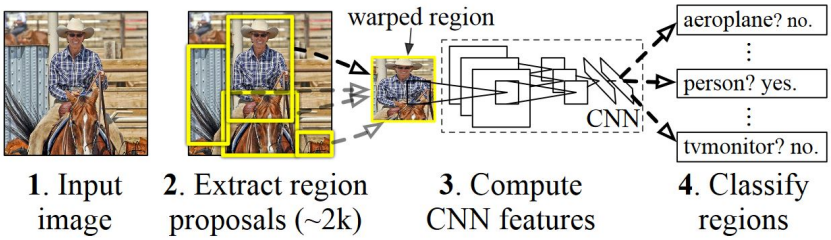


Figura 2.8: Esquema del sistema R-CNN. Se distinguen 4 pasos: (1) Entra una imagen al sistema. (2) Se extraen *region proposals*, alrededor de 2000. Antes que cada *proposal* entre a la red, se desforman para cumplir con las dimensiones requeridas por la primera capa convolucional. (3) Los *region proposals* desformados entran a la red y se genera un vector de dimensión 4096 desde la segunda capa completamente conectada. (4) Finalmente se clasifica cada vector por un conjunto de SVMs lineales específicos para cada clase. Para obtener la localización se proponen dos formas en [15], una directa desde los *region proposals* y otra desde un regresor que refina la localización de los *proposals*. Imagen obtenida desde [15].

2.1.3.1.1. Entrenamiento del Sistema de Detección en Múltiples Etapas

El entrenamiento del sistema de detección R-CNN se realiza en múltiples etapas. Primero se utiliza una red pre-entrenada en la base de datos ILSVRC 2012, para la tarea auxiliar de clasificación. La red pre-entrenada con la mencionada base de datos es la que se empleó en [15] para obtener los resultados en la tarea de detección en la base de datos PASCAL VOC 2010-12. La arquitectura de red empleada en R-CNN es la conocida como AlexNet [28].

Luego de pre-entrenar la red AlexNet, se realiza la adaptación de esta a la nueva tarea (detección) y al nuevo dominio, es decir, a los *proposals* escalados obtenidos desde las imágenes de la base de datos VOC. Por otro lado, dado que la última capa completamente conectada es de dimensión 1000 (número de clases de la base de datos ILSVRC 2012), se reemplaza por una capa del mismo tipo pero de dimensión 21 inicializada aleatoriamente, este valor representa las 20 clases de la base de datos VOC más el fondo. Luego se continúa con el *stochastic gradient descent* (SGD) usando una tasa de aprendizaje de 0.001 (10 veces más pequeña que la tasa inicial del pre-entrenamiento).

Para realizar la adaptación antes especificada, es decir, para entrenar la red con la nueva capa, se procede a etiquetar los *region proposals*. Si alguno posee IoU[1] con un *ground truth* mayor o igual a 0,5 son considerados como ejemplos positivos y el resto como negativos. En cada iteración se muestrean uniformemente 32 ventanas positivas (sobre todas las clases) y 96 ventanas del fondo [15].

[1]IoU es la sigla para *Intersection over Union.*

Una vez entrenada la red para la nueva tarea en el nuevo dominio, se extrae desde la penúltima capa completamente conectada un vector de dimensión 4096 para cada *region proposal*. Estos se etiquetan considerando que si el IoU de una región con un *ground truth* es menor a 0.3 entonces se le asigna la clase fondo. Por otro lado, la clase positiva serán solamente los *ground truth*. Posterior al etiquetado se procede a entrenar un SVM lineal por clase.

Destacar que los umbrales de etiquetado son distintos para la etapa del *fine-tuning* (entrenamiento en un dominio específico) y para el entrenamiento de los clasificadores SVM. La diferencia se justifica, porque, para el primer caso se necesitan muchos ejemplos. Además al permitir ejemplos positivos ruidosos se controla el *overfitting*. En cambio para SVM los ejemplos ruidosos resultan nocivos para su rendimiento.

2.1.3.1.2. Refinamiento de la Localización

Los primeros resultados expuestos en [15] para el sistema R-CNN arrojaron que una importante fuente de error surgía de las imprecisiones en la localización, la cual era proveniente del método externo de *proposal*. Para solucionar esto proponen agregar al sistema una regresión capaz de predecir un nuevo *bounding box* a partir del inicial, obtenido desde el método externo, por ejemplo *selective search* [54].

Se entrena un regresor para predecir un nuevo *bounding box*, para esto se aprende una transformación que tiene por objetivo mapear un *box* propuesto P a un *ground truth* G. Lo anterior se implementa utilizando como entrada al regresor las características provenientes desde la capa $pool_5$ para un *proposal* P. La transformación se parametriza en término de 4 funciones $\mathrm{d}_x(P)$, $\mathrm{d}_y(P)$, $\mathrm{d}_w(P)$ y $\mathrm{d}_h(P)$ [15]. Los índices x, y representan el centro del *bounding box* P y w, h el ancho y alto. El *bounding box* predicho a partir del *proposal* P se expresa de la siguiente forma:

$$\hat{G}_x = P_w \mathrm{d}_x(P) + P_x \tag{2.6}$$
$$\hat{G}_y = P_h \mathrm{d}_y(P) + P_y \tag{2.7}$$
$$\hat{G}_w = P_w \cdot exp(\mathrm{d}_w(P)) \tag{2.8}$$
$$\hat{G}_h = P_h \cdot exp(\mathrm{d}_h(P)) \tag{2.9}$$

Las funciones $\mathrm{d}_*(P)$ ($*$ es x, y, h, w según corresponda) son modeladas como una función lineal de las características del *proposal* P provenientes desde $pool_5$ [15]. Se tiene que $\mathrm{d}_*(P) = w_*^T \phi_5(P)$, donde el primer término corresponde al vector que se aprende y el segundo a las características provenientes de las salidas de $pool_5$ para el *proposal* P. El vector w_* se aprende minimizando la siguiente expresión:

$$w_* = \underset{\hat{w}_*}{\operatorname{argmin}} \sum_{\mathrm{i}}^{N} (t_*^{\mathrm{i}} - \hat{w}_*^T \phi_5(P^{\mathrm{i}}))^2 + \lambda ||\hat{w}_*||^2 \tag{2.10}$$

Finalmente los objetivos de la regresión t_* para el par de entrenamiento (P,G) son definidos

como sigue:

$$t_x = (G_x - P_x)/P_w \quad (2.11)$$
$$t_y = (G_y - P_y)/P_h \quad (2.12)$$
$$t_w = log(G_w/P_w) \quad (2.13)$$
$$t_h = log(G_h/P_h) \quad (2.14)$$

2.1.3.2. Fast R-CNN

El sistema de detección R-CNN logró excelentes resultados, mejorando el mAP en más de 30 % relativo al mejor resultado previo en VOC 2012 [15]. Sin embargo, posee algunos inconvenientes, uno de ellos es su entrenamiento en múltiples etapas lo cual hace al sistema difícil de entrenar, además es caro en tiempo y en almacenamiento en disco. Finalmente en la etapa de *test* también es lento producto que para cada imagen se extraen *proposals* que son procesados por todas las capas de la red convolucional.

Buscando resolver los inconvenientes mencionados en el párrafo anterior surge el sistema conocido como Fast R-CNN [14]. Algunos de los aportes del citado trabajo con respecto a R-CNN es que logran un mAP más alto, desarrollan un algoritmo de entrenamiento en un único paso, usando una función de pérdidas multi-tarea (clasificación y localización) y además no requieren almacenar vectores de características en el disco, dado que no usan clasificadores SVMs, solamente emplean *softmax*.

2.1.3.2.1. Arquitectura y Entrenamiento del Sistema Faster R-CNN

El sistema R-CNN es lento producto que procesa todos los *proposals* generados en una imagen ($\sim$ 2000 en la etapa de prueba). En cambio, en Fast R-CNN solamente la imágen completa es la que pasa por la red, posteriormente los *region proposals* generados con algún método externo son usados para proyectar estas regiones de interés en los *feature maps* de salida desde la última capa convolucional.

Una vez proyectados los *object proposals* en los *feature maps*, las proyecciones entran a una capa de *pooling* denominada *region of interest (RoI) pooling layer*, la cual extrae un vector de características de largo fijo desde cada region de interés en un *feature map*. Este vector es la entrada a una secuencia de capas completamente conectadas, las cuales finalmente se ramifican en dos capas de salida, una que predice K+1 puntajes (*softmax*), para las K clases del problema más el fondo, y la otra capa genera 4 valores que codifican los *bounding boxes* refinados para cada una de las clases [14]. Un esquema general de la arquitectura del sistema se muestra en la Figura 2.9

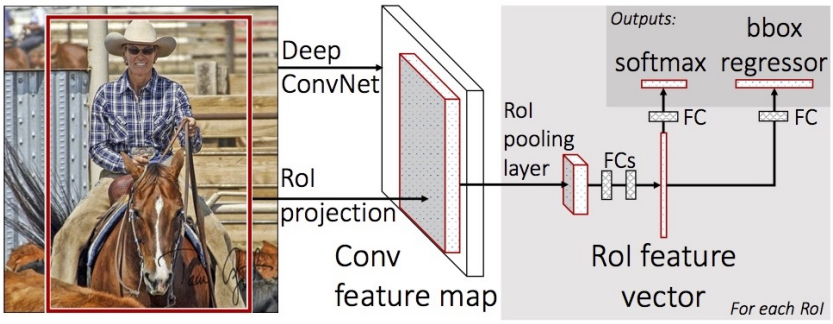


Figura 2.9: Esquema del sistema Fast R-CNN. Primero entra a la red una imagen completa, luego se proyectan en los *feature maps* de salida de la última capa convolucional los *region proposals* formando las RoIs, estas entran a la capa RoI *pooling* que genera un vector de salida de largo fijo por cada RoI, este vector pasa por dos capas completamente conectadas y finalmente se generan las dos salidas del sistema (vector de probabilidades y vector con los *bounding boxes* predichos). Imagen obtenida desde [14].

En [14] se define una RoI como una ventana rectangular en un *feature map*. Cada RoI esta determinada por un vector de dimensión 4 (r, c, h, w), donde (r, c) especifica la esquina superior izquierda y (h, w) son el alto y ancho de la RoI. Una capa de RoI *pooling* divide una RoI de dimensión $h \times w$ en una grilla de $H \times W$ con sub-ventanas de dimensión $h/H \times w/W$. A continuación a cada sub-ventana se le extrae el máximo y se forma un vector de largo fijo igual a $H \times W$. Este procedimiento se ejecuta para cada RoI y para todos los *feature maps* de salida desde la última capa convolucional. La capa de RoI *pooling* se basa en el trabajo sobre *spatial pyramid pooling networks* [20], el cual introduce una red conocida como SPPnet.

Para desarrollar el sistema se experimenta con tres redes pre-entrenadas en ImageNet (AlexNet [28], VGG_CNN_M_1024 y VGG16 [48]). Estas redes para convertirlas en el detector Fast R-CNN sufren tres cambios:

1. La última capa de *max pooling* es reemplazada por una capa de RoI *pooling* y se configuran sus valores H,W para ser compatibles con la primera capa completamente conectada de la red.

2. La última capa completamente conectada y la *softmax* son reemplazadas por dos capas hermanas, una compuesta por una capa completamente conectada más una función *softmax*, la cual se encarga de predecir los puntajes para cada clase para cada RoI y la segunda es una capa completamente conectada encargada de producir los *bounding boxes* de salida del sistema.

3. La red es modificada para tomar dos datos de entrada: una lista de imágenes y una lista de RoIs en estas imágenes.

2.1.3.2.2. Función de Pérdidas Multi-tarea

Otro de las mejoras que incorporó el sistema Fast R-CNN con respecto a trabajos previos

como R-CNN [15] y SPPNet [20], es que el entrenamiento se puede realizar en una sola etapa, este aspecto se desarrolló gracias a la introducción de una función de pérdidas multi-tarea, que permite entrenar dos capas de salida en paralelo, una para generar la clasificación y otra encargada de producir la regresión de los *bounding boxes* que determinan la localización de los objetos en las imágenes.

La primera salida del sistema es un vector de probabilidaddes por cada RoI que posee la siguiente forma $p = (p_0, ..., p_K)$ para las $K + 1$ categorías, incluido el fondo, p es obtenido desde una *softmax* alimentada por las $K + 1$ salidas de una capa completamente conectada. La segunda salida es la que produce para cada una de las K clases un *bounding box* definido por $t^k = (t^k_x, t^k_y, t^k_w, t^k_h)$, estos son generados por una capa completamente conectada.

Para efectuar el entrenamiento del sistema cada RoI es etiquetado de la siguiente forma: si IoU con algún *ground truth bounding box* es de al menos 0.5 se le asigna una etiqueta $u \geq 1$, es decir, son considerados como los ejemplos positivos. Si IoU(RoI, *ground truth*) $\in$ [0,1; 0,5) entonces el RoI se etiqueta como clase fondo $u = 0$, este etiquetado sigue lo planteado en [20]. Agregar que el 25 % de los RoIs usados en el entrenamiento se toman desde las clases positivas y el resto desde la negativa.

La función de pérdidas multi-tarea propuesta en [14], la cual se usa sobre los RoIs etiquetados para entrenar conjuntamente para clasificación y regresión del *bounding box* está definida por:

$$L(p, u, t^u, v) = L_{cls}(p, u) + \lambda[u \geq 1]L_{loc}(t^u, v) \tag{2.15}$$

El primer término de la ecuación (2.15) es una función de pérdida logarítmica para la clase verdadera u, esta se define como $L_{cls}(p, u) = -log(p_u)$. El hiperparámetro λ se agrega para controlar el balance entre las dos funciones de pérdidas, se le asigna el valor de 1 [14]. El término que sigue a λ es conocido como corchete de Iverson y es 1 cuando $u \geq 1$ y 0 en otros casos. Finalmente L_{loc} tienen como argumentos dos tuplas, una que es el *bounding box* verdadero para la clase u dada por $v = (v_x, v_y, v_w, v_h)$ y una tupla para la predicción $t^u = (t^u_x, t^u_y, t^u_w, t^u_h)$. Matemáticamente L_{loc} se define de la siguiente forma:

$$L_{loc}(t^u, v) = \sum_{\mathrm{i} \in \{x,y,w,h\}} smooth_{L_1}(t^u_{\mathrm{i}} - v_{\mathrm{i}}) \tag{2.16}$$

$$smooth_{L_1}(x) = \begin{cases} 0{,}5x^2 & \text{si } |x| < 1 \\ |x| - 0{,}5 & \text{otro caso} \end{cases} \tag{2.17}$$

2.1.3.3. Faster R-CNN

La utilización de redes convolucionales profundas más un algoritmo de *region proposal* se ha estudiado ampliamente para abordar la tarea de localización más clasificación de objetos [14], [15]. No obstante lo anterior, la determinación de los *proposals* significan un cuello de botella para disminuir los tiempos de respuesta de los sistema de detección (localización más clasificación). Un cambio en el algoritmo para realizar esta tarea es propuesto en [41], trabajo

que introduce una red convolucional para calcular los *proposals*. Esta red es llamada *Region Proposal Network* (RPN), la cual comparte los pesos de las capas convolucionales con la red denominada Fast R-CNN [14], que se encarga de la detección.

El mencionado sistema de detección de objetos, contempla dos módulos (RPN y Fast R-CNN) y recibe el nombre de Faster R-CNN. El sistema completo termina siendo una sola red para la tarea de detección de objetos (Figura 2.10), ya que, ambos módulos comparten los pesos de las capas convolucionales. Para la implementación en [41] utilizan las arquitecturas de redes ZFNet [59] y VGGNet [48].

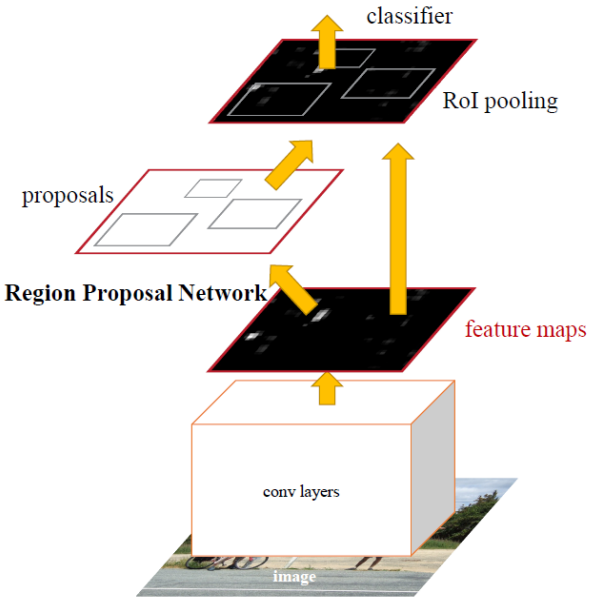


Figura 2.10: Sistema Faster R-CNN para la detección de objetos. Se destaca el módulo de *region proposal network* (RPN) que permite generar *proposals* sobre los *feature maps* de salida desde la última capa convolucional del sistema, de esta forma se logra prescindir de un método externo de *region proposal*. Imagen obtenida desde [41].

2.1.3.3.1. Region Proposal Network

El módulo de Region Proposal Network (RPN) es un tipo de red completamente convolucional, que recibe como entrada una imagen de cualquier tamaño y produce como salida un conjunto de *proposals* y un puntaje asociado a ellos, el cual es generado desde una capa softmax de dos clases (objeto y no objeto).

Para generar los *region proposals* se desliza una pequeña red por el *feature map* de salida desde la última capa convolucional compartida entre los módulos [41]. Esta red tiene como entrada ventanas de 3x3 y sus salidas provienen desde una capa de regresión para obtener el *bounding box* y otra desde una capa de clasificación de los *bounding boxes* (Figura 2.11).

En cada ventana deslizante se predicen múltiples *region proposals*, el número máximo se denota como k. Estos son parametrizados relativos a k referencias denominadas *anchors*, los cuales están centrados en la respectiva ventana deslizante y es asociado con 3 escalas y 3

relaciones de aspecto (ancho de una imagen dividido en su altura). Por lo tanto en cada posición se tendrán k=9 *anchors*.

Previo a las capas de salida de la red RPN se ubica una capa que redimensiona las características, para el caso de la red ZFNet la dimensión es de 256 y es seguida por una función ReLU. El número de salidas desde la capa de clasificación son $2k$ (por las 2 clases) y desde la capa de regresión son $4k$ (por las 4 coordenadas del *bounding box*).

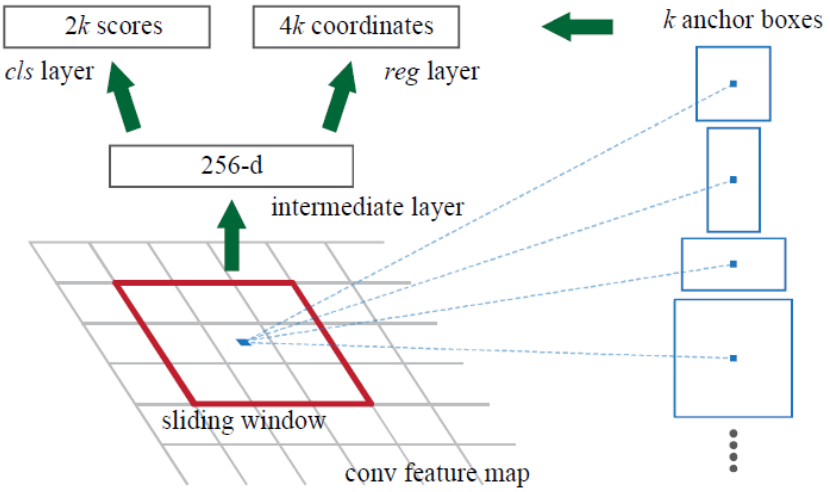


Figura 2.11: Esquema de la red RPN actuando en un *feature map* de la última capa convolucional del sistema. Las salidas desde RPN son dos: una indicando si un *proposal* generado es o no objeto (*cls layer*) y otra que produce las 4 coordenadas que caracterizan al *bounding box* (*reg layer*). Imagen obtenida desde [41].

En la etapa de entrenamiento para la red RPN se minimiza una función de pérdidas que está definida para una imagen, de la siguiente forma:

$$L(\{p_i\},\{t_i\}) = \frac{1}{N_{cls}} \sum_i L_{cls}(p_i, p_i^*) + \lambda \frac{1}{N_{reg}} \sum_i p_i^* L_{reg}(t_i, t_i^*) \tag{2.18}$$

A continuación se especifican los términos de la ecuación (2.18). Primero mencionar que un *anchor* en un *mini-batch* se identifica por el índice i, luego se tiene lo siguiente:

- $\mathbf{p_i}$ es la probabilidad que un *anchor* sea objeto. El valor de esta se obtiene desde una función softmax.

- $\mathbf{p_i^*}$ representa la etiqueta del *ground-truth*, la que será 1 si el anchor pertenece a la clase positiva o cero si es negativo, la pertenencia a una de las dos clases se determina mediante el cálculo de la intersección sobre la unión (IoU) entre los anchors y el *ground-truth* de la siguiente forma:

    - **Etiquetas positivas:** Se asignan a dos tipos de *anchors*, primero el o los anchors con el más alto IoU o un *anchor* que tiene un IoU mayor a 0.7 con cualquier *ground-truth boxes*

- **Etiquetas negativas:** Serán los *anchors* con IoU menor a 0.3 para todos los *ground-truth boxes*

- $\mathbf{t_i}$ es un vector que representa las 4 coordenadas parametrizadas del *bounding box* predicho.

- $\mathbf{t_i^*}$ es el vector de coordenadas del *ground truth* asociado con un *anchor* positivo.

- $\mathbf{L_{cls}}(\mathbf{p_i}, \mathbf{p_i^*})$ es una función de pérdida logarítmica para dos clases [14].

- $\mathbf{L_{reg}}(\mathbf{t_i}, \mathbf{t_i^*}) = \mathbf{R}(\mathbf{t_i} - \mathbf{t_i^*})$ donde R es la función *smooth* L_1 propuesta en [14].

- El primer término de la función de pérdidas es normalizado por $\mathbf{N_{cls}}$ que corresponde al tamaño del *mini-batch*.

- El segundo término es normalizado por $\mathbf{N_{reg}}$ que es el número de ubicaciones de los *anchors*.

- λ es un parámetro que se incluye para que ambos términos de la ecuación (2.18) tengan prácticamente la misma ponderación. Se fija en 10, aunque el rendimiento de la red es casi insensible a este en un rango de 1 a 100.

La parametrización para la regresión del *bounding box*, está dada por las ecuaciones (2.19)-(2.22), las que se presentan a continuación:

$$t_x = \frac{x - x_a}{w_a}, \quad t_y = \frac{y - y_a}{h_a} \tag{2.19}$$

$$t_w = log\left(\frac{w}{w_a}\right), \quad t_h = log\left(\frac{h}{h_a}\right) \tag{2.20}$$

$$t_x^* = \frac{x^* - x_a}{w_a}, \quad t_y^* = \frac{y^* - y_a}{h_a} \tag{2.21}$$

$$t_w^* = log\left(\frac{w^*}{w_a}\right), \quad t_h^* = log\left(\frac{h^*}{h_a}\right) \tag{2.22}$$

En las ecuaciones anteriores x, y denotan el centro del *bounding box* y w, h corresponden al ancho y alto de este. Además las coordenadas con el subíndice a pertenecen al *anchor*, aquellas que solamente poseen el superíndice $*$ corresponden a las cajas del *ground-truth* y las que no tienen ningún tipo de índice se usan para identificar las cajas predichas.

2.1.3.3.2. Integración entre RPN y Fast R-CNN

Un paso importante para completar el sistema Faster R-CNN es el desarrollo de las técnicas que permiten que el módulo de *proposal* RPN y la parte correspondiente al sistema Fast R-CNN compartan las capas convolucionales, destacar que ambos módulos pueden ser entrenados de forma independiente.

En [41] se mencionan tres métodos para entrenar los dos módulos (RPN y Fast R-CNN), en este trabajo se discutirán dos de estos, que son los que se utilizaron finalmente en las implementaciones compartidas del sistema Faster R-CNN [2].

El primer método de entrenamiento consiste en 4 pasos, primero se entrena la red RPN (Figura 2.12 izquierda), con *backpropagation* y gradiente descendente estocástico *stochastic gradiente descent* (SGD), las capas que posteriormente serán compartidas se inicializan desde un modelo pre-entrenado, por su lado, las capas nuevas se inician aleatoriamente desde una gaussiana de media cero y desviación estándar 0.01. En el segundo paso se entrena de forma separada la red Fast R-CNN usando los *proposals* que provienen de la red RPN (Figura 2.12 derecha), al igual que en el paso 1 la inicialización se realiza desde un modelo pre-entrenado. En el paso tres se usa la red Fast R-CNN para inicializar el entrenamiento del módulo RPN fijando las capas convolucionales compartidas y solamente ajustando las capas de RPN (desde ahora los dos módulos comparten las capas convolucionales). El cuarto paso y final mantiene las capas compartidas y ajusta las capas de Fast R-CNN.

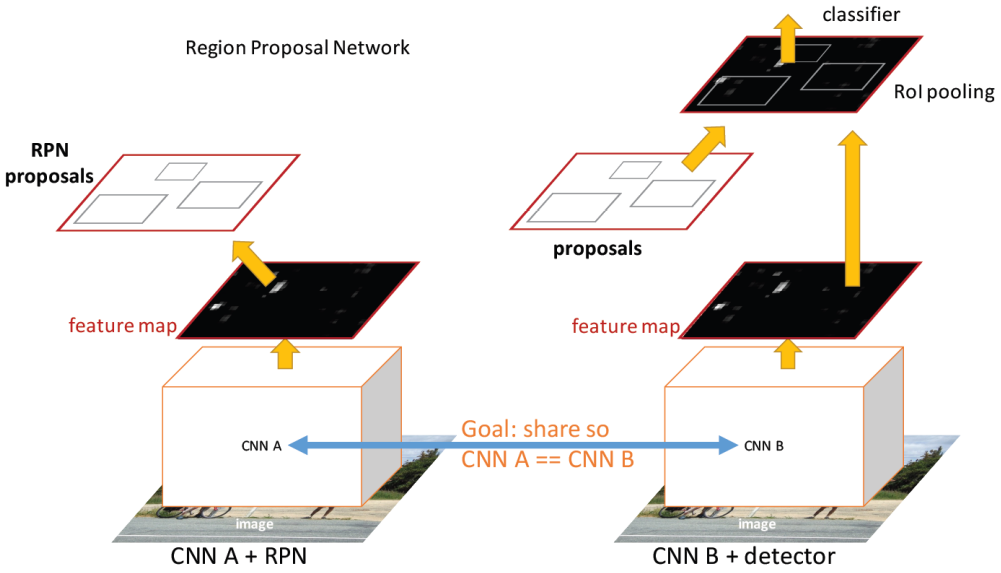


Figura 2.12: A la izquierda se muestra el módulo RPN y a la derecha el módulo detector Fast R-CNN, los cuales al compartir los pesos de las capas convolucionales CNN A y CNN B forman el sistema Faster R-CNN. Imagen obtenida desde: Tools for Efficient Object Detection, ICCV 2015 Tutorial.

El segundo método de entrenamiento consiste en unir la parte RPN y Fast R-CNN en una sola red. En cada iteración en el paso hacia delante se generan *proposals* los cuales son tratados como fijos para el entrenamiento del detector Fast R-CNN. Este método resulta más rápido que el primero y con similares resultados. Esta forma de entrenar es aproximada porque no considera el gradiente con respecto a las coordenadas del *bounding box*.

[2]Implementación en Matlab `https://github.com/ShaoqingRen/faster_rcnn`. Implementación en Python `https://github.com/rbgirshick/py-faster-rcnn`

2.1.3.3.3. Dimensiones de las Salidas del Sistema Faster R-CNN

Una consideración práctica sobre este detector es que en la etapa de la red RPN (*region proposal network*), se producen cientos y hasta miles de *proposals*. Sin embargo, se establece una selección de a los más 300 del total generado en la etapa de *test* por cada imagen, es por esta razón que la salida desde el detector, específicamente la encargada de clasificar, es una matriz de dimensiones $\sim 300x21$, es decir, para cada *proposal* se genera un vector de largo 21, con puntajes que representan la confianza en la clasificación. Un gran número de filas corresponden a predicciones de la clase fondo, las cuales son descartadas posteriormente. Para esto se utiliza el vector de puntajes, si este indica que un *proposal* pertenece a la clase fondo, entonces se descarta, de esta forma se escogen las detecciones que corresponden a algún objeto según la red.

En la Figura 2.13 se presenta un diagrama genérico para el sistema Faster R-CNN, en el cual se señala el tamaño máxima que puede tener cada arreglo que sale desde alguna de las capas completamente conectadas. Se debe destacar que la dimensión de los vectores, es decir, el número de columnas de los arreglos es siempre la misma y viene dada por la arquitectura de la red convolucional (ZF o VGG16), lo que varía es el número de filas, siendo el máximo 300.

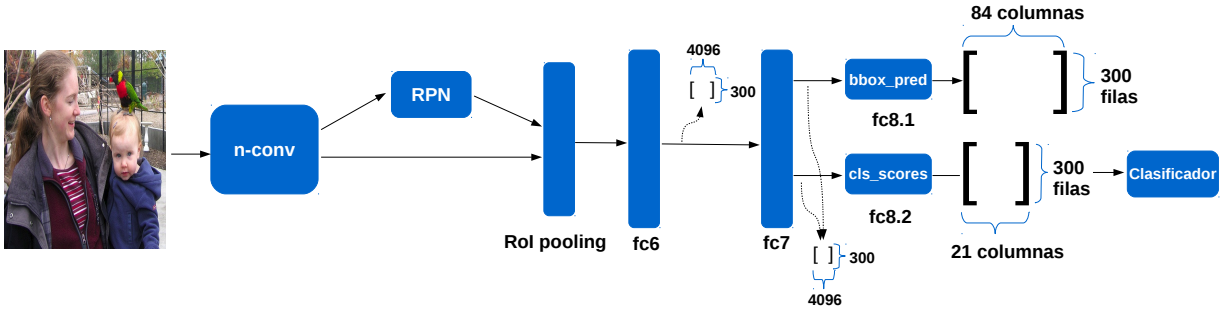


Figura 2.13: Diagrama general sistema Faster R-CNN, indicando las dimensiones de los arreglos de salida.

2.2. Clasificadores Estadísticos

2.2.1. Support Vector Machines

Support Vector Machines (SVMs) [5] son un conjunto de algoritmos de aprendizaje supervisado que se pueden utilizar para tareas como clasificación y regresión [4]. En términos simples SVM encuentra un plano que separa a dos clases de acuerdo a sus características. Lo anterior corresponde a dividir las características/patrones de forma lineal. Esto se puede extender para casos más difíciles donde los patrones no son linealmente separables.

Una idea central de esta máquina es mapear un vector de entrada de forma no lineal a

un espacio de características de una dimensión mayor. Luego en este espacio se construye un hiperplano de separación óptima. A continuación se describe el método para los casos de patrones linealmente y no linealmente separables, basado en lo expuesto en [19].

2.2.1.1. Patrones Linealmente Separables

Dado un problema de dos clases, representadas por $d_i = +1$ y $d_i = -1$, para separar los patrones por una superficie de decisión, se puede definir la ecuación del hiperplano de la siguiente forma:

$$\mathbf{w}^t \cdot \mathbf{x} + b = 0 \tag{2.23}$$

Donde $\mathbf{w}$ es un vector normal al hiperplano (vector de pesos), $\mathbf{x}$ es un vector de entrada y b es el *bias* que corresponde a un valor constante. Para un cierto $\mathbf{w}$ y b, la separación entre el hiperplano y el conjunto de datos más cercano se denomina *margen*. Cuando los datos son separables se pueden seleccionar dos planos tal que entre ellos no se encuentre ningún dato, para seleccionar toda la porción de los datos de una clase o la otra, estos dos planos se pueden definir como:

$$\mathbf{w}^t \cdot x + b \geq 1 \text{ para } d_i = +1 \tag{2.24}$$

$$\mathbf{w}^t \cdot x + b \leq -1 \text{ para } d_i = -1 \tag{2.25}$$

Luego por geometría se puede determinar que la distancia entre los dos planos definidos por las igualdades de las ecuaciones (2.24) y (2.25) está dada por $\frac{2}{||w||}$. Por lo tanto, para maximizar el margen se tiene que minimizar $\mathbf{w}$, además se puede obtener que la distancia desde el origen al hiperplano óptimo, definido por la ecuación (2.23), es $\frac{b}{||w||}$. En la Figura 2.29refplanos se pueden observar los hiperplanos con línea segmentada que definen el margen y el hiperplano óptimo, para un problema de dos clases.

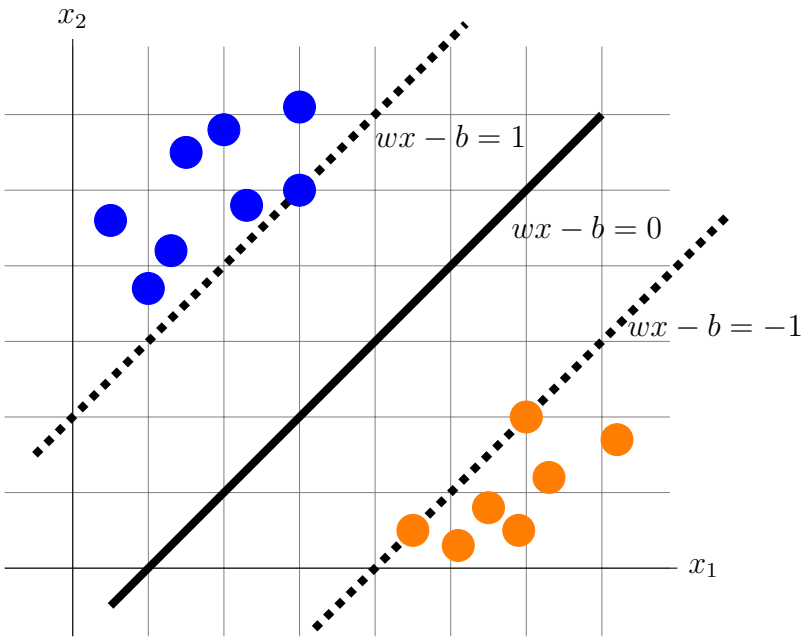


Figura 2.14: Separación de dos clases (círculos grises y naranjos) mediante un hiperplano. Los círculos de colores en el plano (x_1, x_2) representan los datos de entrada al clasificador.

En el problema de minimización antes expuesto se puede cambiar sin alterar el resultado $||w||$ por $\frac{||w||^2}{2}$, donde el $\frac{1}{2}$ se agrega por conveniencia matemática, asimismo la minimización de la expresión $\frac{||w||^2}{2}$ está restringida a cumplir con $\mathrm{d_i}(\mathbf{w}^t\mathbf{x_i} + b) \geq 1$, donde $\mathrm{d_i}$ puede tomar valores de +1 o -1 dependiendo a qué clase pertenezca. En virtud de lo antes expresado este es un problema de optimización con restricciones y se puede resolver utilizando los multiplicadores de Lagrange. Entonces la función lagrangeana para el problema, se puede expresar como:

$$\mathscr{L}(\mathbf{w}, b, \alpha_i) = \frac{\mathbf{w}^t\mathbf{w}}{2} - \sum_{i=1}^{N} \alpha_i[\mathrm{d_i}(\mathbf{w}^t\mathbf{x_i} + b) - 1] \tag{2.26}$$

Luego aplicando las condiciones de optimalidad se pueden obtener los parámetros del clasificador ($\mathbf{w}$ y b) de la siguiente forma:

$$\mathbf{w} = \sum_{i=1}^{N} \alpha_i \mathrm{d_i}\mathbf{x_i} \quad con \quad \alpha_i \neq 0 \tag{2.27}$$

$$b = 1 - \sum_{i=1}^{N} \alpha_i \mathrm{d_i}\mathbf{x}_i^t\mathbf{x} \tag{2.28}$$

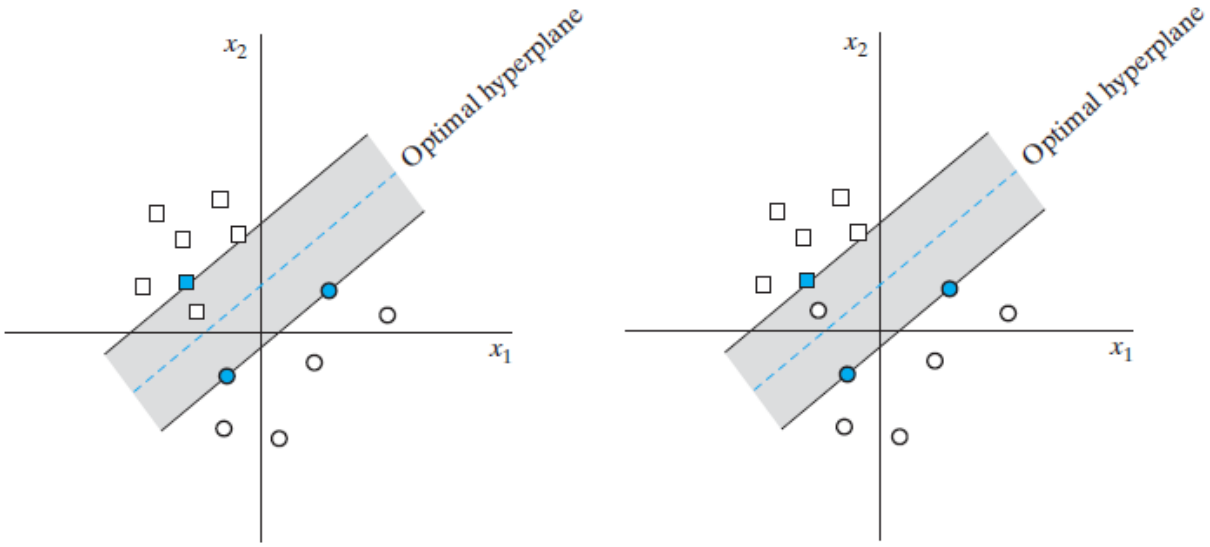
2.2.1.2. Patrones No Linealmente Separables

En el caso en que los patrones no sean linealmente separables, no se podrá encontrar un hiperplano que separe de forma exacta los datos, por lo tanto, ante esta situación se busca

encontrar un hiperplano tal que se minimice la probabilidad de clasificación errónea. Para tratar con estos datos mal clasificados, se introduce en la ecuación del hiperplano de decisión, una variable escalar no negativa denominada ξ, que recibe el nombre de *slack variable*, entonces la ecuación de la superficie de decisión se expresa de la siguiente forma:

$$\mathrm{d_i}(\mathbf{w^t x_i} + b) \geq 1 - \xi_\mathrm{i} \qquad \mathrm{i} = 1, 2, ..., N \tag{2.29}$$

Las variables ξ_i miden el grado de clasificación errónea, para $0 < \xi_\mathrm{i} \leq 1$ los datos caen dentro de la región de separación y en el lado correcto, como se aprecia en la Figura 2.15 (a), por el contrario cuando $\xi_\mathrm{i} > 1$ los datos caen en el lado incorrecto de la separación (Figura 2.15 (b)).



(a) Datos mal clasificados dentro del margen pero en el lado correcto.

(b) Datos mal clasificados dentro del margen pero en el lado incorrecto.

Figura 2.15: Región de decisión caso no separable. Imágenes obtenidas desde [19].

En este caso los vectores de soporte son aquellos que satisfacen la ecuación (2.29), incluso pueden existir vectores que cumplan con $\xi_\mathrm{i} = 0$. El funcional que se desea minimizar con respecto a $\mathbf{w}$ se puede escribir como sigue:

$$\Phi(\mathbf{w}, \xi_\mathrm{i}) = \frac{\mathbf{w^t w}}{2} + C \sum_{\mathrm{i}=1}^{N} \xi_\mathrm{i} \tag{2.30}$$

En la ecuación (2.30) el parámetro C controla la compensación entre la maximización del margen y la correcta clasificación de los datos. Si C se escoge como un valor grande quiere decir que existe una mayor penalización de los errores, mencionar que el segundo término de la ecuación (2.30) se relaciona con el error de clasificación. La forma dual del problema se

puede expresar de la siguiente manera:

$$Q(\alpha) = \sum_{\mathrm{i}=1}^{N} \alpha_{\mathrm{i}} - \frac{1}{2} \sum_{\mathrm{i}=1}^{N} \sum_{j=1}^{N} \alpha_{\mathrm{i}} \alpha_j \mathrm{d_i} \mathrm{d}_j \mathbf{x_i^t x_j} \tag{2.31}$$

$$\text{Sujeto a:} \quad \sum_{\mathrm{i}=1}^{N} \alpha_{\mathrm{i}} \mathrm{d_i} = 0 \tag{2.32}$$

$$0 \leq \alpha_{\mathrm{i}} \leq C \tag{2.33}$$

En el problema dual anterior, lo que se busca determinar son los multiplicadores de Lagrange $\{\alpha_{\mathrm{i}}\}_{\mathrm{i}=1}^{N}$, tal que se maximice la función objetivo $Q(\alpha)$. El conjunto de α_{i} definen el vector de soporte buscado.

2.2.1.3. Aumento de Dimensionalidad Utilizando un Kernel

Cabe destacar otro aspecto presente en SVM el cual se relaciona con la idea que el espacio original de características puede ser trasladado a uno de mayor dimensión, donde el problema sí sea separable. Este traspaso de una dimensión a otra se realiza por una función denominada *Kernel*. Dentro de las más comunes destacan la polinomial, gaussiana (*radial basis function*) y tangente hiperbólica (sigmoide).

Si $\mathbf{x}$ es el vector que representa los datos de entradas al clasificador, que pertenecen a un espacio de dimensión m_0, entonces se denota $\varphi(\mathbf{x})$ como una función no lineal que aumenta la dimensión del espacio de entrada. Dada esta transformación se puede definir un hiperplano actuando como la superficie de decisión, lo cual se escribe de la siguiente forma:

$$\sum_{j=1}^{\infty} w_j \varphi_j(\mathbf{x}) = 0 \tag{2.34}$$

El término $\{w_j\}_{j=1}^{\infty}$ en la ecuación (2.34), denota un conjunto de vectores pesos infinitamente largos, que transforman las características del espacio de entrada al espacio de salida. La ecuación (2.34) se puede escribir matricialmente como:

$$\mathbf{w}^T \phi(\mathbf{x}) = 0 \tag{2.35}$$

La ecuación (2.27) se puede escribir de la siguiente forma:

$$\mathbf{w} = \sum_{\mathrm{i}=1}^{N_s} \alpha_{\mathrm{i}} \mathrm{d_i} \phi(\mathbf{x}_{\mathrm{i}}) \tag{2.36}$$

En la ecuación anterior N_s es el número de vectores de soporte y el vector de características es expresado como $\phi(\mathbf{x}_{\mathrm{i}}) = [\varphi_1(\mathbf{x}_1), \varphi_2(\mathbf{x}_2), ...]^T$. Usando las ecuaciones (2.35) y (2.36), se obtiene lo siguiente:

$$\sum_{\mathrm{i}=1}^{N_s} \alpha_{\mathrm{i}} \mathrm{d_i} \phi^T(\mathbf{x}_{\mathrm{i}}) \phi(\mathbf{x}_{\mathrm{i}}) = 0 \tag{2.37}$$

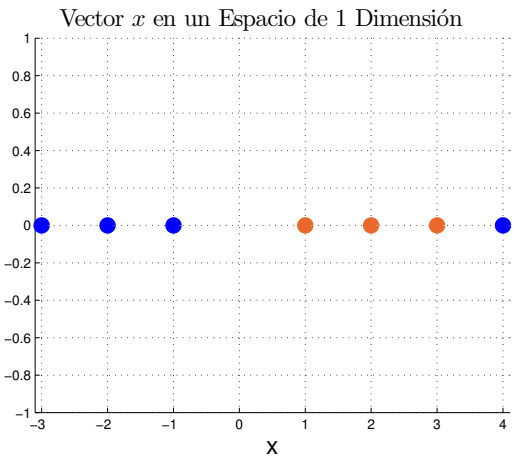
El término $\phi^T(\mathbf{x}_\mathrm{i})\phi(\mathbf{x}_\mathrm{i})$ define un producto punto, que es el denominado Kernel de SVM. En virtud de lo anterior, se tiene que el Kernel está dado por:

$$k(\mathbf{x}, \mathbf{x}_\mathrm{i}) = \phi^T(\mathbf{x}_\mathrm{i})\phi(\mathbf{x}_\mathrm{i}) \tag{2.38}$$

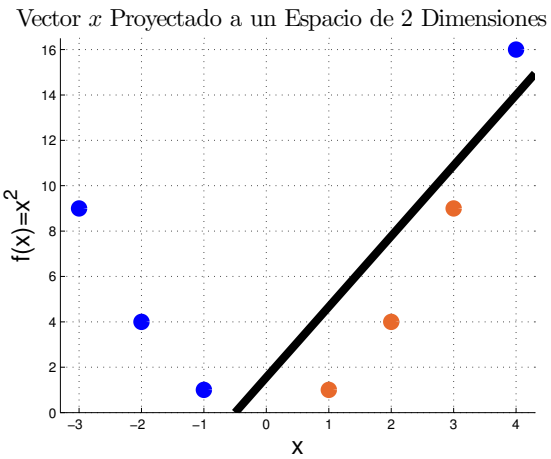
$$= \sum_{j=1}^{\infty} \varphi(\mathbf{x}_\mathrm{i})\varphi(\mathbf{x}) \quad \text{con i} = 1, 2, ..., N_s \tag{2.39}$$

Finalmente, para encontrar los elementos del vector de soporte que permite maximizar el margen de separación entre clases, se resuelve el problema dual de la sección anterior, dado por las ecuaciones (2.31), (2.32) y (2.33). La única diferencia es que en la ecuación (2.31) el término $\mathbf{x}_\mathrm{i}^t\mathbf{x}_j$, se reemplaza por $\mathbf{k}(\mathbf{x}_\mathrm{i}, \mathbf{x}_j)$.

Para ilustrar el beneficio de un aumento de la dimensión de los datos de entrada, se muestra en la Figura 2.16 (a) datos de dos clases, puntos azules y verdes que se ubican sobre una recta, donde no es posible que sean separados por un hiperplano lineal, sin embargo, al trasladarse el vector x a un espacio de dos dimensiones (Figura 2.16 (b)), mediante un kernel cuadrático, se obtiene que las clases son separables fácilmente por el mencionado hiperplano.



(a) Datos de entrada (vector x) para un problema de clasificación de dos clases (puntos azules y verdes). Caso no linealmente separable.



(b) Datos de entrada (vector x) trasladados a un espacio de dos dimensiones (x, x^2). Caso linealmente separable.

Figura 2.16: Efecto del aumento de dimensionalidad para lograr trabajar en un espacio donde las clases sean linealmente separables. Se usa la función $\phi : x \to (x, x^2)$ para aumentar las dimensiones del problema.

En la Tabla 2.1 se resumen algunos de los *kernels* más comunes [57], junto con los parámetros que se requieren definir para su uso, típicamente se realiza validación cruzada y búsqueda en una grilla lo más amplia posible, para encontrar el o los parámetros óptimos según corresponda.

Kernel	$k(x, x_\mathrm{i}) =$	Parámetros
Lineal	$x^T x_\mathrm{i}$	No posee
Polinomial	$(\gamma x^t x_\mathrm{i} + r)^\mathrm{d}$	$\gamma > 0$, r, d
RBF	$\exp(-\gamma \|\|\mathrm{x} - \mathrm{x_i}\|\|^2)$	$\gamma = \frac{1}{2\sigma^2}$
Sigmoide	$\tanh(\gamma \mathrm{x}^\mathrm{t} \mathrm{x_i} + \mathrm{r})$	γ, r

Tabla 2.1: Resumen de *kernels* más comunes.

2.2.1.4. SVM Multiclase

Se han propuesto diversos métodos para utilizar SVM en clasificación multiclase [25], entre los cuales destacan aquellos que proponen combinar varios clasificadores binarios para lograr el objetivo. Entre estos se pueden distinguir dos métodos: *one-vs-one* (o one-vs-rest) y *one-vs-all* [4], [25].

2.2.1.4.1. One-vs-One (ovo)

Este método construye un clasificador binario por cada par de clases distintas, de esta forma si el número total de clase es k, entonces el total de clasificadores entrenados es $k(k-1)/2$. Se advierte que el número de modelos crece de forma cuadrática con la cantidad de clases, por lo que, para conjuntos de datos con muchas clases puede resultar muy lento el desarrollo de este método.

La estrategia adoptada por LIBSVM [4] para determinar a qué categoría pertenece un nuevo ejemplo x que entra al clasificador *ovo*, es asignar a este la clase que resulte más votada, para esto cada SVM binario vota por una de sus dos clases y finalmente la que tenga más votos es considerada como la predicha. En el caso en que dos categorías tengan los mismos votos se escoge la primera que aparece en el arreglo que almacena los nombres de las clases.

2.2.1.4.2. One-vs-All (ova)

Esta implementación para clasificación multiclase con SVM, construye k modelos, con k el número de clases del problema que se busca afrontar. El i-ésimo modelo considera las muestras de la clase i como positivas y todas las restantes como negativas, de aquí el otro nombre con el que se conoce *one-vs-rest*.

Para obtener la salida desde el clasificador multiclase *ova* dado una muestra x, se trabaja con las probabilidaes que entrega indiviualmente cada modelo, se tiene que para cada clasificador i se consigue una probabilidad asociada a que x pertenezca a dicha clase, esto se escribe como $p_\mathrm{i}(x)$. Luego se construye un arreglo con las k probabilidades (una para cada clase) y se normaliza para que los puntajes finales sean válidos, es decir, sumen 1.

2.2.1.5. Algoritmo para la Gestión Eficiente de Memoria Durante el Entrenamiento de SVM

En el entrenamiento de un modelo para detección de objetos habitualmente se tiene un gran número de ejemplos de fondo, comparado con la cantidad de muestras positivas (i.e. que contienen un objeto). Por ejemplo en el caso de *deformable parts model* (DPM) [10] en una sola imagen se pueden obtener del orden de 10^5 ejemplos de fondo por cada objeto. En trabajos basados en *object proposal* [15], [54], esta diferencia puede ser del orden de 70:1.

Producto del desequilibrio entre las clases y el gran número de ejemplos de fondo, se requiere utilizar técnicas que hagan frente a estos desafíos, ya que, puede resultar impracticable considerar todas las muestras negativas al mismo tiempo. La solución estándar para lo anterior se conoce como *bootstrapping* [51], [52], la cual esencialmente consiste en agregar a un subconjunto inicial de entrenamiento, los ejemplos negativos que resulten mal clasificados. Actualmente al algoritmo de *bootstrapping* propuesto para SVM en [10] se le conoce como *hard negative mining* (HNM), que es ampliamente utilizado en los métodos de detección de objetos que ocupan CNN [15],[20]. El término *hard example mining* (HEM) se usa porque esta técnica aplica a todas las clases, no solamente a las negativas [47].

En el algoritmo de HEM para SVM, se pueden definir las muestras *easy* y *hard* para un conjunto de entrenamiento D, dado un modelo m.

$$E(m, D) = \{(x, y) \in D \mid (y \cdot w^t \cdot \phi(x) + b) > 1\} \quad (2.40)$$

$$H(m, D) = \{(x, y) \in D \mid (y \cdot w^t \cdot \phi(x) + b) < 1\} \quad (2.41)$$

A continuación se detallan las ecuaciones (2.41) y (2.40):

- $(\mathbf{x}, \mathbf{y})$: repesentan al conjunto de entrenamiento D, particularmente son las muestras y sus etiquetas ($y \in \{-1, 1\}$).

- $\mathbf{E}(\mathbf{m}, \mathbf{D})$: Son las muestras correctamente clasificadas y fuera del margen definido por el modelo m.

- $\mathbf{H}(\mathbf{m}, \mathbf{D})$: Son las muestras incorrectamente clasificadas o que están dentro del margen definido por el modelo m.

En el Algoritmo 1 se describen los cuatro pasos fundamentales que se desarrollan cuando se entrena un modelo usando HEM. Primero se comienza con un *caché* inicial de ejemplos, que es un pequeño subconjunto del set de entrenamiento completo D ($C_1 \subseteq D$). Luego se entrena el modelo inicial y se retiran desde el *caché* muestras e*asy*, agregando ejemplos *har*d, esto se itera hasta que todos los ejemplos *har*d encontrados en D esten contenidos en el *caché* ($H(m_\mathrm{i}, D) \subseteq C_\mathrm{i}$).

Algoritmo 1: Algoritmo de entrenamiento HEM.

Input : $\mathbf{C_1}$: Caché inicial de ejemplos, tal que $C_1 \subseteq D$
Output: **m**: Modelo de SVM final

1 Entrenar un modelo m usando C_i
2 Si $H(m_\mathrm{i}, D) \subseteq C_\mathrm{i}$ terminar y retornar m_i
3 Si 2 no se cumple retirar los ejemplos e*asy* desde el *caché*, $C_\mathrm{i}^{'} = C_\mathrm{i} \setminus X$ para X tal que $X \subseteq E(m_\mathrm{i}, C_\mathrm{i})$
4 Agregar ejemplos *har*d al *caché*, $C_{\mathrm{i}+1} = C_\mathrm{i}^{'} \cup X$ para X tal que $X \subseteq D$ y $X \cap H(m_\mathrm{i}, D) \setminus C_t \neq \emptyset$. Iterar hasta que se cumpla la condición 2

El caso particular cuando en el Algoritmo 1 se mantienen en el *caché* todos los ejemplos positivos iniciales y se actualizan solo los negativos, correponde al método de *hard negative mining*. Destacar que en el cuarto paso siempre se encuentran datos *har*d, de lo contrario el algoritmo habría terminado en el paso 2. En la implementación de HNM efectuada en [15] se menciona que con una sola iteración es suficiente para obtener buenos resultados. Otro aspecto importante es que en [10] se demuestra que este algoritmo termina en un número finito de pasos y que cuando lo hace el clasificador converge al modelo óptimo definido en términos del conjunto de entrenamiento completo.

2.2.2. Random Forest

Random Forests (RFs) son una combinación de árboles de decisión (Figura 2.17), tal que, cada árbol se construye con un set de datos muestreados aleatoriamente y con reemplazo desde el conjunto de entrenamiento, además en la división de cada nodo se selecciona de forma aleatoria un número fijo de características [3], [33]. Con este método se pueden realizar las tareas de clasificación y regresión. Este trabajo se centra en RF actuando como clasificador. Para la tarea antes mencionada cada árbol realiza una predicción para una muestra de entrada, se dice que cada árbol vota por una clase. Luego la clase de salida desde el *forest* será la que posea más votos.

En la siguiente Sección 2.2.2.1 se describen los aspectos principales de los árboles de decisión, luego en la Sección 2.2.2.2 se explica el algoritmo de *Random Forest*, finalmente en la Sección 2.2.2.3 se indican algunas aplicaciones de RF en tareas de visión computacional.

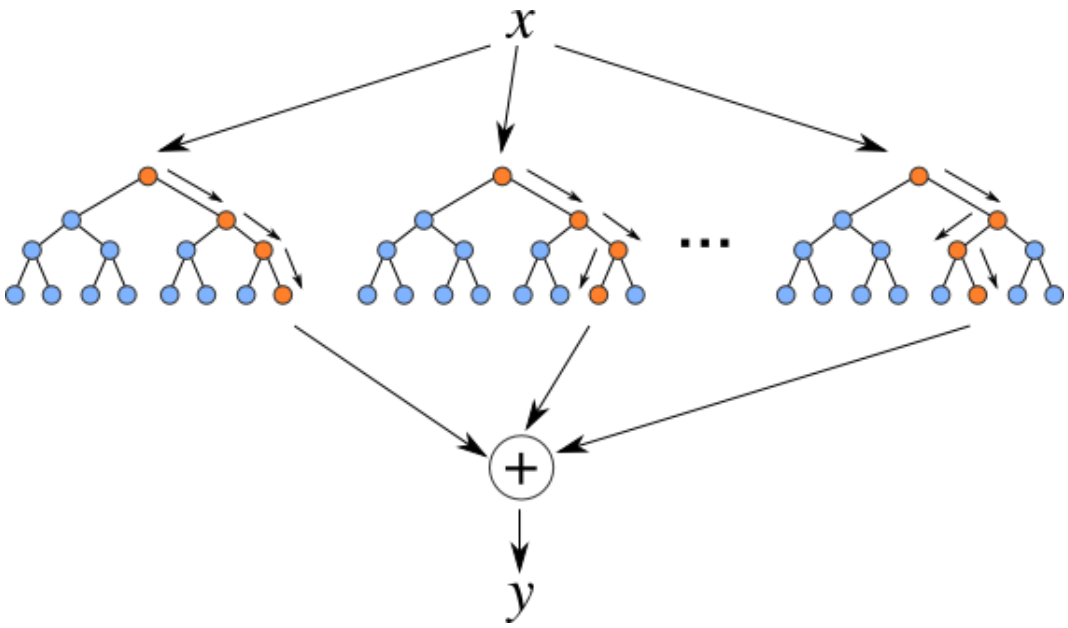


Figura 2.17: Grupo de n árboles de decisión formando el *Random Forest*. Se ejemplifica el proceso de clasificiación de un dato x, para el cual cada árbol vota por una clase y la que obtiene más votos es la predicción final y del clasificador.

2.2.2.1. Árboles de Decisión

Los árboles de decisión son sistemas de predicción en múltiples pasos, en este la clasificación se realiza mediante una secuencia de preguntas aplicadas a características individuales de un vector de entrada [8], [53].

La secuencia de preguntas son organizadas en un árbol el cual está compuesto por nodos. El primero es el nodo raíz, el cual se conecta por ramas a otros nodos hasta que se alcanzan los nodos terminales (nodos hojas), estos no poseen más ramificaciones y son los que contienen la predicción de salida desde el árbol. En cada nodo interno (no hoja) se pregunta por una característica del vector de entrada. Un tipo de pregunta es la siguiente: *es la característica* $x_{\mathrm{i}} \leq \alpha$, donde α representa un valor umbral que se conoce como punto de división. Mediante este tipo de preguntas los datos que llegan a cada nodo son divididos en dos subconjuntos. La Figura 2.18 muestra un árbol de decisión binario con sus nodos internos y terminales.

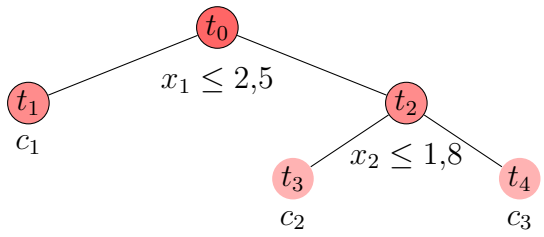


Figura 2.18: Árbol de decisión binario. Dado un vector de características como entrada al árbol, en el nodo raíz (t_0) se pregunta por la característica x_1 y en el nodo interno t_2 se pregunta por la característica x_2. Los nodos t_1, t_2, t_3 son los nodos terminales y predicen las clases c_1, c_2, c_3.

En la etapa de entrenamiento de un árbol de decisión se determina qué característica x_i del vector de entrada se usará en cada nodo, además se selecciona el valor umbral α (punto de división) utilizado en las preguntas que dividen el espacio de entrada. Para determinar estos dos elementos se prueba en cada nodo todas las características y todos los posibles valores que estas pueden tomar según los datos de entrenamiento. Luego se determina la mejor división del nodo (mejor par característica, punto de división) en base a alguna métrica que cuantifique la pureza de la separación, se usa por ejemplo el Índice de Gini o la ganancia de información. Este procedimiento se efectúa para cada nodo del árbol, hasta alcanzar los nodos terminales.

En el algoritmo de entrenamiento de los árboles se establece que se detiene la división de un nodo si en este caen muestras que solo pertenecen a una misma clase. Además se puede controlar la división de los nodos por medio de dos hiper-parámetros que se seleccionan por el usuario:

1. **Máxima profundidad del árbol:** Máximo número de nodos desde el nodo raíz, una vez que se alcanza este valor se para la división de los nodos.

2. **Mínimo de muestras en nodos terminales:** Se refiere al mínimo número de muestras que está encargado de dividir un nodo, si este valor se alcanza o se tiene un número inferior de muestras, se detiene la división y el respectivo nodo se declara como terminal.

2.2.2.2. Descripción del Algoritmo de Random Forest

En el algoritmo de entrenamiento de *Random Forest* [3] se construyen múltiples árboles de decisión. Cada árbol de decisión es entrenado en un conjunto de datos que se elabora seleccionando ejemplos de forma aleatoria desde los datos de entrenamiento, esta selección se realiza con reemplazo, por lo que, en los datos de entrenamiento de cada árbol es probable que existan muestras repetidas. Esta etapa en la cual se generan árboles con el mismo algoritmo, pero con distintos conjuntos de entrenamiento se conoce con el nombre de *bagging* [2].

El RF propuesto en [3] además de la aleatoriedad proveniente de la formación de los conjuntos de entrenamiento para cada árbol, se seleccionan aleatoriamente en cada nodo una

cantidad m de características para determinar con cuál de estas se obtiene la mejor división del nodo respectivo. A diferencia de los árboles de decisión típicos en los cuales cada nodo es dividido usando la mejor características entre todas [33]. En el Algoritmo 2 se resumen de forma general los pasos que se efectuan para entrenar un clasificador RF.

Algoritmo 2: Algoritmo de entrenamiento de *Random Forest* [18]. La clase predicha (c_i) para un dato de entrada es la que cuenta con la mayoría de votos, entre todos lo árboles del clasificador.

Input : $(\mathbf{X_i}, \mathbf{y_i})$: X_i es el vector de características de entrada de dimensión p. Por su parte y_i es la etiqueta.
B: Número de árboles en el *forest*
m: Número de características seleccionadas aleatoriamente en cada nodo.
$\mathbf{n_{min}}$: Mínimo número de muestras que está encargado de dividir un nodo.

Output: $\mathbf{c_i}$: Clase predicha por el RF

1 **for** $b = 1 : B$ **do**
2 **(a)** Extraer un conjunto de muestras Z de tamaño N desde los datos de entrenamiento.
3 **(b)** Construir un árbol T_b para el conjunto Z. Recursivamente repetir los siguientes pasos para cada nodo terminal del árbol, hasta que el mínimo tamaño del nodo (n_{min}) es alcanzado.
(i) Seleccionar aleatoriamente m variables desde las p características de los vectores de entrenamiento de entrada X_i.
(ii) Seleccionar la mejor variable/punto de división entre las m.
(iii) Dividir el nodo en dos nodos hijos.
4 **end**
5 Retornar el conjunto de árboles $\{T_b\}_1^B$

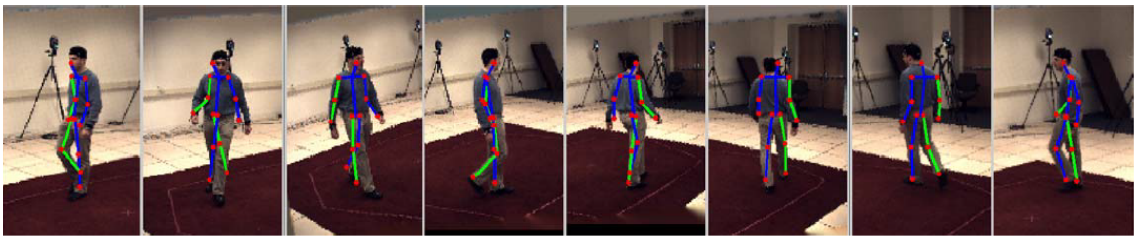
Algunas características de RF son que resulta robusto a *outliers* y ruido, posee tan buena precisión como otros clasificadores (SVM y redes neuronales), es rápido en la predicción y al ser completamente paralelizable resulta más rápido aún. Además tiene principalmente dos parámetros que son el número de características que se prueban en cada nodo y el número de árboles, aunque usualmente el clasificador no resulta muy sensible a estos valores [33].

2.2.2.3. Aplicación de Random Forest en Tareas de Visión Computacional

La utilización de RF en tareas de visión computacional brinda algunas particularidades como lo son la rapidez con la cual es capaz de ejecutarse y lo robusto que resulta al clasificar. Además se caracteriza por ser una herramienta flexible en distintas labores de visión.

RF se ha empleado por ejemplo en reconocimiento de puntos de interés en tiempo real [32]. También se ha utilizado en tareas de segmentación de imágenes, como en [46], trabajo en el cual se destaca este clasificador por su precisión, rápido entrenamiento y evaluación. En [42] se usa este clasificador para la detección de pose humana, donde señalan que para un *bounding box* dado, la clasificación con un *forest* de 100 árboles toma alrededor de 15 ms en su implementación en Matlab.

(a) Segmentación de imágenes. Imagen obtenida desde [46].



(b) Detección de pose humana. Imagen obtenida desde [42].

Figura 2.19: Ejemplo de dos tareas de visión computacional en las cuales se ha utilizado RF.

Capítulo 3

Metodología e Implementación del Sistema de Detección de Objetos Desarrollado

3.1. Descripción General del Desarrollo del Sistema de Detección de Objetos

Como se ha mencionado a lo largo de este trabajo, el sistema de detección de objetos desarrollado se basa en el detector conocido como Faster R-CNN [41]. La idea principal es mejorar este detector usando RF y SVM, para clasificar los objetos detectados. Los mencionados clasificadores son entrenados usando como vectores de características las salidas desde las capas completamente conectadas del sistema Faster R-CNN.

Se desarrollan fundamentalmente dos sistemas de detección denominados Faster R-CNN + RF y Faster R-CNN+SVM. Las primeras tareas que se realizan para desarrollarlos es entrenar de forma *offline* los clasificadores RF y SVM. El primero de estos se entrena solamente con vectores de características provenientes desde la capa fc8 del sistema Faster R-CNN. Por su parte, para SVM se utilizan los vectores extraídos desde las capas fc8, fc7 y fc6, de forma separa.

Para formar el conjunto de entrenamiento, se utilizan las imágenes del set de *train* de la base de datos PASCAL Visual Object Classes (VOC) 2007 [9]. Cada una de estas imágenes entran al sistemas Faster R-CNN (entrenado previamente en la misma base de datos) y generan como salida una cantidad máxima de 300 vectores de características, cada uno de estos vectores representan las características extraídas a un *bounding box* distinto en una misma imagen.

A priori no se sabe a qué objeto pertenecen los vectores de salida mencionados en el párrafo anterior. Por lo que, usando el *bounding box* al cual pertenecen (predicho por la salida de localización del sistema Faster R-CNN) y los *bounding boxes ground truth*, se etiquetan estos

vectores calculando el solapamiento entre ambos. Si este supera un cierto umbral para algunos de los *ground truth* se etiqueta con la respectiva clase, en caso contrario se le asigna la clase fondo. En las secciones 3.3.3 y 3.3.2, se detallan los dos métodos de etiquetado programados.

El primer sistema que se describirá en las próximas secciones es el Faster R-CNN+RF, el cual se entrena con los vectores de características obtenidos desde la capa fc8 del sistema base [41]. Para implementar este sistema se usan dos librerías de RF, una desarrollada en el lenguaje de programción R (randomForest [33]) y otra en Python (RF de *scikit-learn* [39]). Para el sistema que usa la implementación de RF en R se hace uso de la interface rpy2 [13], para unir el código base (Faster R-CNN) escrito en Python, con el código del clasificardor RF.

El segundo sistema implementado es el Faster R-CNN+SVM. La librería usada para SVM es la conocida como LIBSVM, desarrollada en C y que cuenta con una interfaz para trabajar en Python. El entrenamiento se realiza con los vectores que se obtienen de las capas completamente conectadas fc8, fc7 y fc6. Para el caso de las dos últimas, dado que la dimensión de los vectores es de 4.096 (aproximadamente 195 veces mayor que la dimensión de los vectores de fc8), se implementa el algoritmo conocido como *Hard Example Mining* (HNM) [10] para entrenar SVM.

El algoritmo antes citado permite entrenar SVM con una fracción de los datos totales de entrenamiento, sin perjudicar los resultados. HNM básicamente consiste en entrenar el modelo iterativamente comenzando con un pequeño conjunto de entrenamiento e ir agregando a este únicamente las muestras que resulten difíciles de clasificar, además se retiran aquellos datos que son fáciles de clasificar.

En la siguiente sección se describe la base de datos, con la cual se entrenan los dos sistemas planteados y además fue usada previamente por el sistema Faster R-CNN. Adicionalmente esta base de datos, establece una forma estándar de medir los resultados para los sistemas de detección, esto se describe en la sección 3.2.1.

3.2. Base de Datos

La base de datos estándar con la cual se efectúan las distintas pruebas en el presente trabajo es la que provee el desafío PASCAL Visual Object Classes (VOC) 2007 [9]. En adelante se denota simplemente como VOC07. Esta base de datos proporciona un conjunto de imágenes y sus anotaciones, principalmente para dos tareas. La primera es clasificación (determinar si una imagen contiene alguna muestra de un objeto de una clase en particular) y la segunda es detección (determinar donde está un objeto de una clase en particular en una imagen).

Los datos se encuentran divididos en tres conjuntos: entrenamiento (*train*), validación (*val*) y prueba (*test*), los cuales están compuestos por 2.501, 2.510 y 4.952 imágenes. Además, habitualmente para entrenar se unen los conjuntos de entrenamiento y validación (*trainval*), formando un set de 5.011 imágenes. En la Tabla 3.1 se detalla el número de imágenes y

objetos por cada clase, para cada conjunto.

El número de clases en VOC07 es 20: *aeroplane, bicycle, bird, boat, bottle, bus, car, cat, chair, cow, dining table, dog, horse, motorbike, person, potted plant, sheep, sofa, train, TV/-monitor*. La Figura 3.1 muestra dos ejemplos aleatorios por cada clases de la base de datos, en estas se puede apreciar que la categoría *person* aparece frecuentemente junto con otros objetos. Por ejemplo se logra distinguir junto con las clases *bicycle, bottle, dinning table, horse, motorbike, sofa* y *TV/Monitor*, lo anterior también se ve reflejado en las estadísticas expuestas en la Tabla 3.1, donde *person* es la única clase que en todos los conjuntos supera las 1.000 apariciones.

Clase	train		val		trainval		test	
	img	obj	img	obj	img	obj	img	obj
Aeroplane	112	151	126	155	23	306	204	285
Bicycle	116	176	127	177	243	353	239	337
Bird	180	243	150	243	330	486	282	459
Boat	81	140	100	150	181	290	172	263
Bottle	139	253	105	252	244	505	212	469
Bus	97	115	89	114	186	229	174	213
Car	376	625	337	625	713	1250	721	1.201
Cat	163	186	174	190	337	376	322	358
Chair	224	400	221	398	445	798	417	756
Cow	69	136	72	123	141	259	127	244
Dining table	97	103	103	112	200	215	190	206
Dog	203	253	218	257	421	510	418	489
Horse	139	182	148	180	287	362	274	348
Motorbike	120	167	125	172	245	339	222	325
Person	1.025	2.358	983	2.332	2.008	4.690	2.007	4.528
Potted plant	133	248	112	266	245	514	224	480
Sheep	48	130	48	127	96	257	97	242
Sofa	111	124	118	124	229	248	223	239
Train	127	145	134	152	261	297	259	282
Tv/monitor	128	166	128	158	256	324	229	308
	2.501	6.301	2.510	6.307	5.011	12.608	4.952	12.032

Tabla 3.1: Estadísticas de la base de datos VOC07. La última fila representa el valor total, el cual para las columnas i*mg* no coincide con la suma de sus elementos, porque, una imagen puede contener objetos de más de una clase.

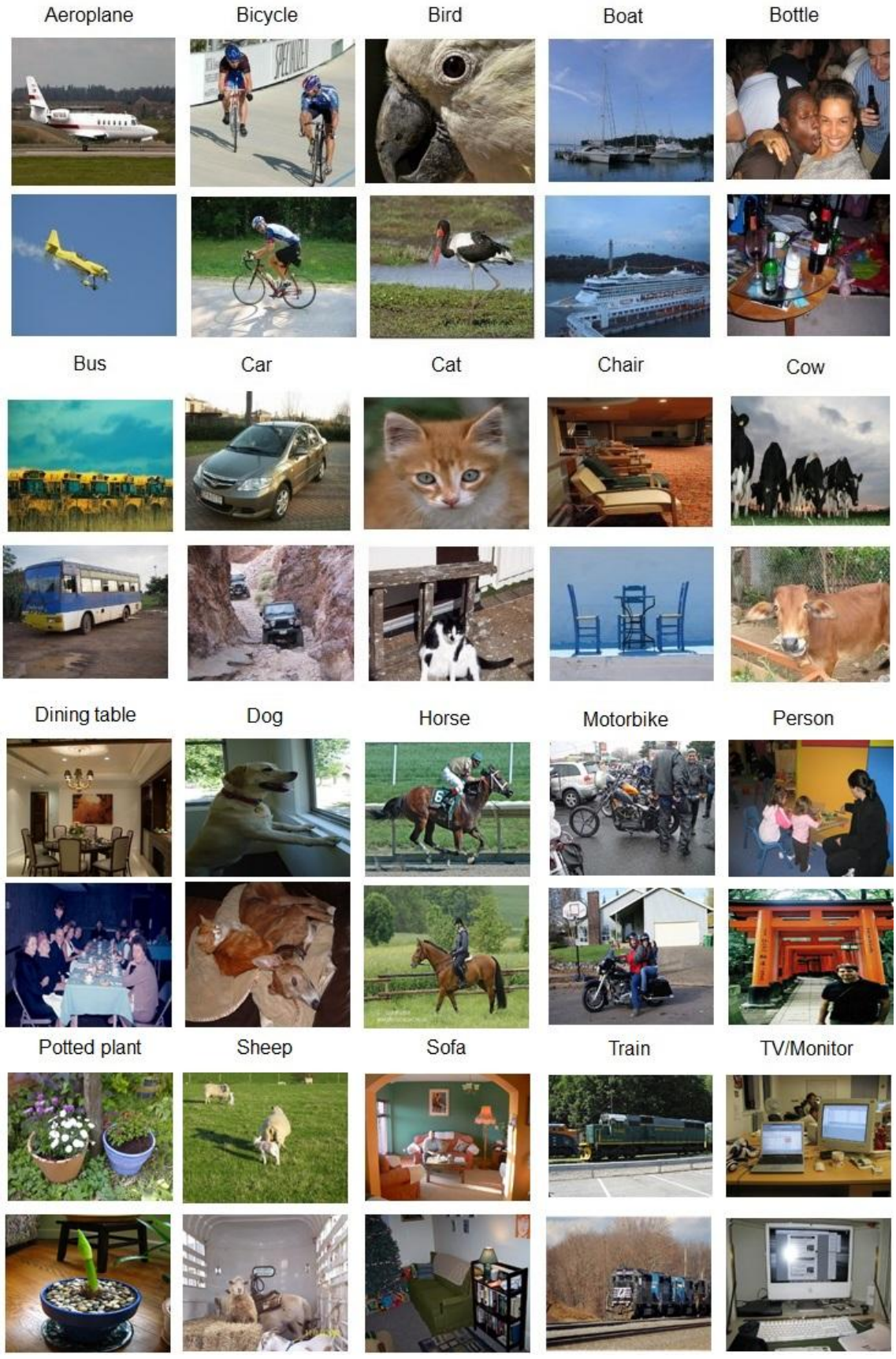


Figura 3.1: Ejemplos de imágenes con las clases de la base de datos VOC07.

3.2.1. Métrica Estándar para Medir el Rendimiento de una Detección

La competencia PASCAL VOC07 además de la base de datos provee una forma estándar de medir los resultados de los sistemas de clasificación y detección. La métrica utilizada recibe el nombre de Average Precision (AP) [9], la cual refleja en un sólo número la forma de la curva precision/recall.

Antes de definir el *recall* y *precision*, se explican brevemente los términos *true positive* (TP), *true negative* (TN)), *false positive* (FP) y *false negative* (FN) (Tabla 3.2):

- ***True Positive (TP):*** Cuando la clase positiva es la verdadera y el sistema la clasifica como positiva
- ***True Negative (TN):*** Cuando la clase negativa es la verdadera y el sistema la clasifica como negativa
- ***False Positive (FP):*** Cuando la clase negativa es la verdadera y el sistema la clasifica como positiva
- ***False Negative (FN):*** Cuando la clase positiva es la verdadera y el sistema la clasifica como negativa

		Clase Predicha	
		+	-
Clase Real	+	**True Positive (TP)**	**False Negative (FN)**
	-	**False Positive (FP)**	**True Negative (TN)**

Tabla 3.2: Matriz de confusión con los términos TP, TN, FP y FN.

En una tarea de clasificación el *recall* o *true positive rate* (TPR) indica cuántos ejemplos de la clase positiva son clasificados correctamente, se puede expresar como la relación entre los *true positive* y la suma de *true positive* y *false negative*:

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

Por su parte el valor de *precision* refleja cuántos de las predicciones de clase positiva son efectivamente correctas, es la razón entre los *true positive* y la suma de *true positive* y *false positive*:

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

En el caso de un sistema de detección para definir las ecuaciones (3.1) y (3.2), se debe recurrir al cálculo de la métrica *Intersection over Union* (IoU), la que cuantifica la calidad de la predicción de un *bonding box* b_{ij}, conociendo previamente el *bounding box ground truth* B_{gt}. Formalmente esta se expresa como una división, donde el numerador representa la intersección de las áreas del *bounding box ground truth* y el predicho por el sistema, por su parte el denominador es la unión de las áreas de ambos *bounding box* (Figura 3.2), lo anterior se expresa matemáticamente como sigue:

$$IoU(B_{gt}, b_{ij}) = \frac{Area(B_{gt} \cap b_{ij})}{Area(B_{gt} \cup b_{ij})} \tag{3.3}$$

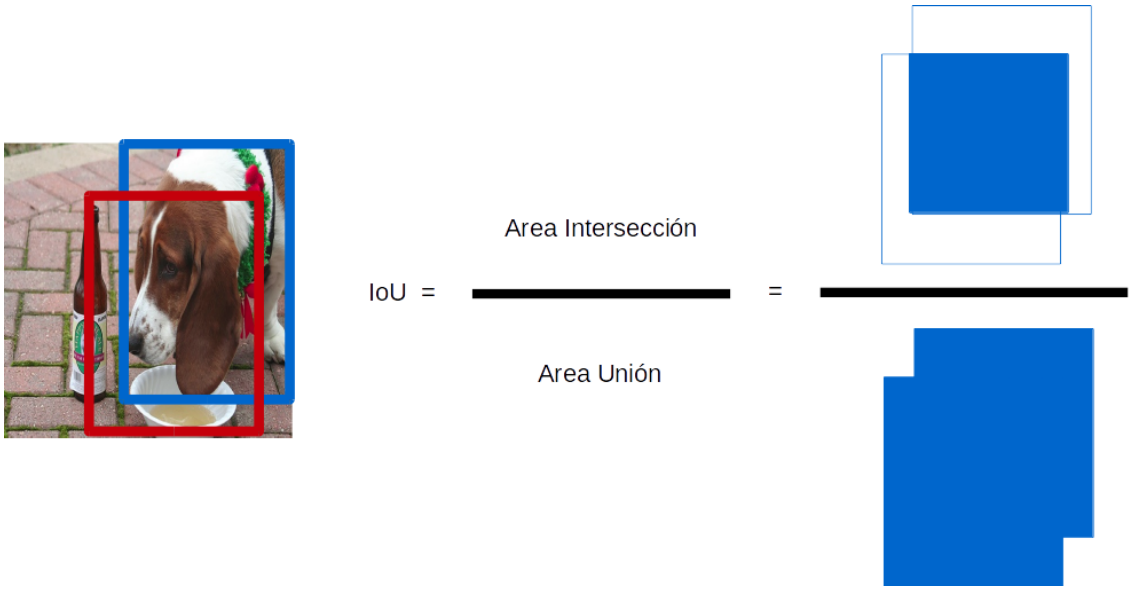


Figura 3.2: Ejemplo determinación de IoU, en la imagen del perro el *bounding box* azul es el *ground truth* y el rojo es la predicción.

A continuación se explica la forma en que se construye la curva *precision/recall* para un sistema de detección, basado en la notación que se utiliza en [45]. Para cada clase de objeto y para cada imagen, un método de detección debe entregar la ubicación predicha b_{ij} y una confianza s_{ij} asociada a esta. Por cada detección j en una imagen i, $z_{ij} = 1$ si $IoU(B_{gt}, b_{ij}) > 0{,}5$ con B_{gt} representando el *ground truth* de una Imagen I_i para una clase en particular, por el contrario, si $IoU(B_{gt}, b_{ij}) \leq 0{,}5$ entonces $z_{ij} = 0$. Si para una clase de objeto en particular, N representa al total de *ground truth*, a través, de todas las imágenes, entonces para un umbral t el *recall* y *precision* se definen como:

$$Recall(t) = \frac{\sum_{ij} 1[s_{ij} \geq t] z_{ij}}{N} \tag{3.4}$$

$$Precision(t) = \frac{\sum_{ij} 1[s_{ij} \geq t] z_{ij}}{\sum_{ij} 1[s_{ij} \geq t]} \tag{3.5}$$

Variando el umbral t se consigue formar la curva *precision/recall*, y luego se calcula el *average precision* que corresponde en términos simples a determinar el área bajo esta curva. Más detalles sobre el cómputo de este se pueden encontrar en [9]. Cabe destacar que los valores de *precision* y *recall* se encuentran entre 0 y 1, por lo que, la máxima área que encierra la curva construida con estas métricas tiene valor 1.

3.3. Métodos de Etiquetado Conjunto de Entrenamiento

3.3.1. Extracción Vectores de Características

En la Sección 2.1.2 se detallan las redes convolucionales ZF y VGG16, estas son las que se utilizan para desarrollar los distintos sistemas propuestos en este trabajo. Las tres últimas capas de ambas redes son del tipo completamente conectadas, las cuales se denominan $fc8$, $fc7$ y $fc6$. En la Figura 3.3 se presenta un diagrama general de la arquitectura desarrollada en el sistema Faster R-CNN. En esta se aprecian las tres capas completamente conectadas, destacando que existen dos capas $fc8$ en paralelo, esto es porque la superior (*fc8.1*) se encarga de predecir el *bounding box* para una clase en particular y la capa $fc8$ inferior (*fc8.2*) es la responsable de entrar a la función *softmax* para producir los puntajes de confianza para la localización predicha.

La extracción de los vectores de características se efectúa desde la capa *fc8.2* para la mayoría de las pruebas que se van a presentar en el Capítulo 4. La dimensión de los vectores de esta capa es de 21, por las 20 clases de la base de datos más la clase fondo (*background*). En la Sección 4.6.1.1 se explora el rendimiento del sistema Faster R-CNN+SVM cuando se entrena con vectores obtenidos desde las capas $fc7$ y $fc6$, notar que estos tienen una dimensión de 4.096.

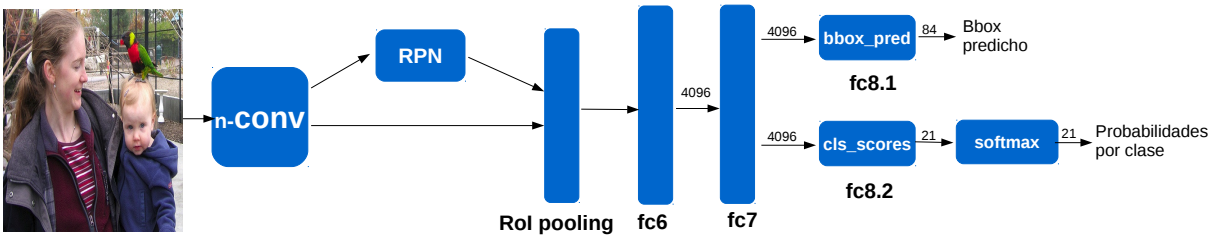


Figura 3.3: Diagrama general del sistema Faster R-CNN.

Como fue explicado en la sección anterior, la red RPN genera a lo más 300 *proposals* en la etapa de *test*, y las salidas desde $fc6$, $fc7$ y *fc8.2*, son ordenadas como matrices de dimensiones $\sim 300x4096$ (salidas desde $fc6$ y $fc7$) y $\sim 300x21$ (salida desde *fc8.2*). En estas las filas representan el número de *proposals* y las columnas son las características o el largo

de los vectores de salida. Son estas matrices las que se etiquetan para formar el conjunto de entrenamiento. Se debe mencionar que la dimensión de los vectores de salida desde la capa *fc8.1* es de 84 por las 4 coordenadas del *bounding box* para las 21 clases.

3.3.2. Método de Etiquetado 1

Los nuevos sistemas propuestos para Faster R-CNN contemplan el uso de los clasificadores SVM y RF. El ajuste de estos se hace de manera *offline*, para lo cual, primero se forma el conjunto de entrenamiento usando el sistema original ya entrenado, para esto se extraen los vectores de características como se explica en la sección anterior (3.3.1).

Para fomar el conjunto de entrenamiento se deben etiquetar los vectores de características de salida desde la capa fc que se utilice. *A priori* no se sabe a qué clase pertenecen, para determinar esto se cálcula la *Intersection over Union* (IoU) entre, los *bounding box ground truth* B_{gt} y los predichos por la red b_{i}. Se establece que si $IoU(B_{gt}, b_{\mathrm{i}}) \geq 0{,}5$ se etiqueta el *proposal* con la clase del B_{gt} respectivo, por el contrario si $IoU(B_{gt}, b_{\mathrm{i}}) \in [0{,}1; 0{,}5)$ entonces se considera la predicción como fondo (clase 0) [14].

La salida desde las capas fc son arreglos, para los cuales el número de filas está determinado por la cantidad de *proposals* que genera la red RPN. De ahora en adelante se considera este como el máximo posible, es decir, 300.

Sin importar la capa que se utilice para extraer los vectores de características, el presente método de etiquetado solo requiere el arreglo de salida desde la capa *fc8.1*, el cual posee como máximo 300 filas y 84 columnas. Estas últimas representan 21 *bounding box* distintos (84 dividido por las 4 coordenadas que definen a un *bounding box*).

A continuación se explican los pasos seguidos para etiquetar el arreglo de *bounding boxes* predichos (b_{i}) de dimensión $300x84$. Se etiqueta siguiendo los pasos del Algoritmo 3, en este se recorren las 300 filas de b_{i} ($pp = 0 : 299$), las 21 clases ($c = 0 : 20$) y todos los *bounding boxes ground truth* (B_{gt}).

En la línea 4 (Algoritmo 3) se calcula la IoU entre un *ground truth* específico y el c-ésimo *bounding box* predicho para el *proposal pp*. Como el cálculo de IoU para una clase y un *proposal* se realiza con todos los *ground truth* estos valores se almacenan en *array_iou* (línea 5) para luego encontrar el máximo, que corresponde, al *ground truth* con el que se obtiene mayor coincidencia para el *bounding box* c-ésimo. Este valor se almacena en *array_iou_max*. Esto se realiza para las 21 clases, siendo este el largo final del arreglo de i*ou* máximos (*array_iou_max*). Con esto se tiene que para un solo *proposal* hay 21 valores de i*ou* y como este únicamente puede pertenecer a una clase, entonces se selecciona el máximo del arreglo *array_iou_max* (línea 9). Finalmente, en base al valor de *iou_max* se etiqueta cada uno de los 300 *bounding box*, almacenando la clase a la que pertenecen en el vector llamado *clases* y formando dos matrices con vectores de características, una para las muestras positivas ($featuresPos$) y otra para las negativas ($featuresNeg$), lo anterior se desarrolla desde la línea 10 a la 15 del Algoritmo 3.

Algoritmo 3: Método de Etiquetado 1

Input : $\mathbf{B_{gt}}$: Arreglo con las coordenadas de todos los *ground truth*
$\mathbf{b_i}$: Arreglo de salida desde alguna de las capas fc del sistema, la cantidad de filas representa el número de *proposals*
scores: Arreglo para el cual sus filas representan los vectores de características, es la salida desde una capa fc (*fc6,fc7 o fc8.2*)

Output: **clases**: Arreglo con las etiquetas negativas
mfeatures: Matriz de características

```
for pp = 0 : 299 do
    for c = 0 : 20 do
        for k = 0 : num total de gt − 1 do
            iou = IoU(B_gt(k), b_i(pp, 4 · c : 4 · (c + 1) − 1)
            array_iou.append(iou)
        end
        array_iou_max = max(array_iou)
    end
    iou_max = max(array_iou_max)
    if iou_max ≥ 0,5 then
        clases(pp) =clase de B_gt con el cual se calculó iou_max
        featuresPos = scores(pp)
    else if 0,1 ≤ iou_max < 0,5 then
        clases(pp) = clase fondo
        featuresNeg = scores(pp)
end
mfeatures = featuresPos + featuresNeg
return mfeatures, clases
```

Una vez etiquetados todos los vectores de características y formada la matriz de entrenamiento ($mfeatures$), se procede a redistribuir la proporción entre ejemplos de la clase fondo y el resto (ejemplos positivos), tal como se propone en [14]. Esto conduce a una disminución de ejemplos positivos ruidosos. El umbral inferior de 0,1 para etiquetar la clase fondo actúa como una buena heurística según lo planteado en [20]. Especificar que este método de etiquetado es usado en [14] para entrenar una red desde las capas completamente conectadas en adelante, esto cuando el sistema de detección depende de un algoritmo de *proposals* externo como ocurre en [14],[15].

3.3.3. Método de Etiquetado 2

Se implementa un segundo método de etiquetado, conforme a lo planteado en el material suplementario de [15], en donde se señala que el algoritmo expuesto en la sección anterior resulta más óptimo en el caso del entrenamiento (o *fine-tuning*) de las capas completamente conectadas, que para entrenar un clasificador SVM. Lo anterior debido principalmente a que las capas fc requieren una mayor cantidad de datos para lograr una buena convergencia, no

así para SVM, donde los ejemplos positivos de baja calidad resultan más desfavorables para el desempeño del clasificador.

El método de etiquetado 2 se basa en lo sugerido por [15]. Para formar los ejemplos positivos se toman solamente los *ground truth*, que son 12.608 objetos para el conjunto de *trainval*. Por su parte, los *bounding box* de la red que se catalogan como instancias negativas, son aquellas para las cuales la IoU con los *ground truth* no supera en ningún caso el valor de 0,3. Luego siguiendo lo planteado en [20], se retira cualquier muestra negativa que se solape más de un 70 % con otra de la misma clase.

El Algoritmo 4 describe los pasos seguidos para implementar la segunda forma de etiquetado, este es semejante al Algoritmo 3, pero se diferencia en los siguientes aspectos: la condición sobre iou_max (línea 10), la cual ahora es que su valor pertenezca al intervalo $[0; 0{,}3]$. Además en este caso el arreglo de las etiquetas ($clasesNeg$) está compuesto exclusivamente por la clase fondo, otro elemento distinto es que el Algoritmo 4 genera como salida el arreglo $bbNeg$, el que contiene los *bounding boxes* para los cuales se etiquetó la clase negativa.

El último paso para completar el presente método de etiquetado, consiste en usar el arreglo $bbNeg$, que se obtiene como salida del Algoritmo 4, para determinar los *bounding boxes* negativos que se solapan más de un 70 % y descartar los respectivos vectores de características desde *featuresNeg*. Luego el arreglo con los vectores de características negativos se une con el de las muestras positivas (*ground truth*) y se forma la matriz de entrenamiento.

Algoritmo 4: Método de Etiquetado 2

Input : $\mathbf{B_{gt}}$: Arreglo con las coordenadas de todos los *ground truth*
$\mathbf{b_i}$: Arreglo de salida desde alguna de las capas fc del sistema, la cantidad de filas representa el número de *proposals*
scores: Arreglo para el cual sus filas representan los vectores de características, es la salida desde una capa fc (*fc6,fc7 o fc8.2*)

Output: **clasesNeg**: Arreglo con las etiquetas negativas
bbNeg: Arreglo con los *bounding boxes* para los cuales se etiquetó la clase negativa
featuresNeg: Arreglo con los vectores de características etiquetados como fondo

```
for pp = 0 : 299 do
    for c = 0 : 20 do
        for k = 0 : num total de gt − 1 do
            iou = IoU(B_gt(k), b_i(pp, 4 · c : 4 · (c + 1)))
            array_iou.append(iou)
        end
        array_iou_max = max(array_iou)
    end
    iou_max = max(array_iou_max)
    if 0 ≤ iou_max ≤ 0,3 then
        clasesNeg(pp) =clase fondo
        bbNeg = bounding box para el cual se calculó iou_max
        featuresNeg = scores(pp)
    end
end
```

3.4. Sistema Faster R-CNN + RF

La idea fundamental propuesta en este trabajo es combinar las redes neuronales convolucionales (CNN) con algunos métodos clásicos de *machine learning* como en este caso lo es con Random Forest. En particular se conecta este clasificador a la salida de la última capa completamente conectada del sistema Faster R-CNN, específicamente a *fc8.2*, como se aprecia en la Figura 3.4.

Se plantea que el clasificador RF es una buena opción para trabajar junto con las CNN, debido a algunas de sus características, por ejemplo que es multiclase, por lo que no hay que recurrir a implementar una cascada como sí sucede con los clasificadores binarios. Además es un algoritmo paralelizable, lo cual acelera su proceso de entrenamiento, algo deseado cuando desde una imagen se pueden obtener miles de ejemplos para entrenar. Por último se tiene que en [11] se señala como el mejor clasificador disponible para trabajar en un amplio número de bases de datos.

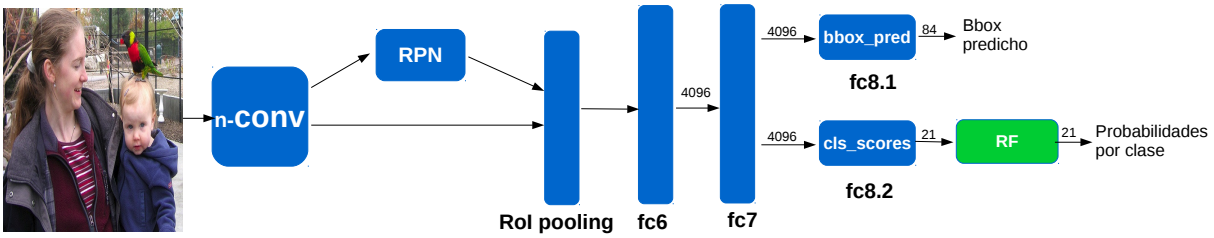


Figura 3.4: Diagrama general del sistema Faster R-CNN+RF.

En la implementación del sistema Faster R-CNN+RF, el clasificador final se entrena de forma *offline* mediante los vectores de características extraídos desde la capa *fc8.2*, los cuales se etiquetan según los métodos explicados en las Secciones 3.3.2 y 3.3.3. Siguiendo lo planteado en [11] y [12], la primera librería que se utiliza para RF es la llamada randomForest [33], escrita en el lenguaje de programación R. Esto porque según [11] es la tercera mejor librería de la familia RF y la quinta al compararla con otras 178 implementaciones de clasificadores (SVM, NN, Bayesiano, entre otros métodos).

Una segunda librería empleada es la que provee *scikit-learn* [39]. Esta se utiliza porque está desarrollada en Python, que es el mismo lenguaje en que se encuentra escrito el sistema Faster R-CNN, por lo que, resulta más rápida la implementación del sistema final. Además permite efectuar fácilmente el entrenamiento de RF en múltiples *cores*. Por otra parte su rendimiento la ubica en el séptimo lugar para la familia de RF y en el puesto 18 a nivel global [12].

Poner en funcionamiento el sistema Faster R-CNN+RF con la librería randomForest, requiere comunicar el código del RF escrito en R con el resto del sistema desarrollado en Python. Para esto se utiliza la interface rpy2 [13], que permite trabajar en Python con la sintaxis y librerías de R.

En [33] se plantea un código de ejemplo para ilustrar el funcionamiento de RF. El Algoritmo 5 presenta una implementación de este con pequeños cambios exclusivamente en la sintaxis sin afectar los resultados. Principalmente lo que se realiza es cargar la librería MASS que contiene la base de datos Forensic Glass (fgl) [33], luego se asigna a la variable x el conjunto de entrenamiento, la variable y representa las etiquetas con las que se entrena. Finalmente, se usa randomForest, para ajustar un modelo con 500 árboles. El clasificador resultante queda almacenado en $myrf$ (línea 6).

El Algoritmo 6 presenta la implementación en Python del código en R expuesto en el Algoritmo 5, para esto se usa la interface rpy2 [13] y la librería *pandas* [36] (líneas 1-3). La instrucción *ro.r()* permite trabajar dentro de esta con la sintaxis de R. En las líneas 5 a la 9 se escribe lo mismo que en el Algoritmo 5, hasta aquí las varibles x e y no pueden ser utilizadas directamente en Python, ya que, están en un formato que solo lee R. Para llevarlas a Python hay que convertirlas explícitamente como se hace en las líneas 10 y 12. En estos puntos xpy es un *DataFrame* de *pandas* (estructura de datos 2D de pandas) e ypy es una lista de Python. Para poder clasificar con randomForest en Python las variables xpy e ypy se

deben transformar tal que sus estructuras sean del tipo rpy2.DataFrame y rpy2.FactorVector, respectivamente.

El entrenamiento en Python usando la librería de RF que provee R se realiza con la instrucción ***ro.r.randomForest(x,y,ntree=500)*** (línea 14 Algoritmo 6). La variable x representa la matriz de entrenamiento y es del tipo rpy2.Dataframe la cual pertenece a la estructura de datos *data frame* de R. Por su parte y (etiquetas del conjunto de entrenamiento) es del tipo rpy2.FactorVector que corresponde al grupo de variables *factors* en R. Es importante que se respeten estas estructuras de datos, puesto que son las requeridas por la librería para poder ser ejecutada.

Algoritmo 5: Código de ejemplo en R.

```
1 library(randomForest)
2 library(MASS)
3 data(fgl)
4 set.seed(17)
5 x=fgl[1:214,1:9]
6 y=fgl[1:214,10]
7 myrf=randomForest(x,y,ntree=500)
```

Algoritmo 6: Código de ejemplo en Python.

```
1  import rpy2.robjects as ro
2  import pandas.rpy.common as com
3  from rpy2.robjects.packages
   import importr
4  MASS=importr('MASS')
5  ro.r.library('randomForest')
6  ro.r('data(fgl)')
7  ro.r('set.seed(17)')
8  ro.r('x=fgl[1:214,1:9]')
9  ro.r('y=fgl[1:214,10]')
10 xpy = com.load_data('x')
11 x =
   com.convert_to_r_dataframe(xpy)
12 ypy = com.load_data('y')
13 y = ro.FactorVector(ypy)
14 myrf = ro.r.randomForest(x,
   y,ntree=500)
```

En la etapa de *test* la salida desde la capa *fc8.2* (Figura 3.4) se debe convertir a un *data frame* de R para poder generar las probabilidades asociadas a cada *bounding box* predicho por la capa *fc8.1*. Lo anterior es realizado por la función que se esboza en el Algoritmo 7. Esta recibe como entradas *scores* que es la salida desde *fc8.1* y rf que es el modelo entrenado previamente y que se lee desde el disco. Para usar *scores* con rpy2, este se transforma a la estructura de datos rpy2.DataFrame, y luego las predicciones de los puntajes de confianza (probabilidades) se obtienen con la instrucción **ro.r.predict(rf,x,type="prob")** (línea 7 Algoritmo 7). Los puntajes antes mencionados se almacenan en el arreglo denominado *proba*, que es finalmente la salida desde el bloque RF que se aprecia en la Figura 3.4.

Algoritmo 7: Función para obtener vectores de probabilidades con RF de R en Python.

Input : **rf**: Modelo de RF almacenado en el disco duro
scores: Arreglo para el cual sus filas representan los vectores de características, es la salida desde una capa fc (*fc6, fc7 o fc8.2*)
Output: **proba**: Arreglo de probabilidades

```
1 Function randomforest(scores,rf)
2     import pandas.rpy.common as com
3     import pandas as pd
4     df =pd.DataFrame(data=scores)
5     x = com.convert_to_r_dataframe(df)
6     proba=ro.r.predict(rf,x,type = "prob")
7     return proba
```

En este trabajo solamente se analiza el efecto del número de árboles en el RF, para el resto de parámetros que se pueden controlar se usan los valores recomendados por la librería [33], el detalle se expone a continuación:

- ***mtry***: Número de características aleatoriamente muestreadas para entrenar cada nodo, su valor es $\sqrt{p}$ donde p representa la dimensión de los vectores de características.

- ***nodesize***: Mínimo número de muestras requeridas para dividir un nodo, su valor por defecto es 1.

La profundidad de los árboles no se limita explícitamente y solo se detendrá el crecimiento cuando se alcance la condición impuesta por *nodesize*. Se debe destacar que se usan los mismos parámetros para las dos librerías ocupadas, randomForest en R y *scikit-learn* en Python.

Para implementar el sistema con el RF de *scikit-learn* se requiere poco desarrollo, ya que, como se ha mencionado antes esta librería está disponible en Python, que es el mismo lenguaje ocupado en Faster R-CNN. Un esquema simplificado para entrenar con el mencionado RF se muestra en el Algoritmo 8. Aquí la primera línea establece los parámetros para el entrenamiento, en este caso $n\_estimators$ es el número de árboles y $n\_jobs = -1$ indica que se usen todos los *cores* disponibles para el entrenamiento. Por último con la instrucción **myrf.fit**(**x**, **y**) se entrena el clasificador. Se debe agregar que para obtener un vector de probabilidades se recurre a la siguiente instrucción **myrf.predict_proba**(**scores**), con *scores* un arreglo con vectores de características para los cuales se realiza la predicción.

Algoritmo 8: Seudocódigo de entrenamiento para RF de *scikit-learn*.

Input : **x**: Matriz de entrenamiento
y: Etiquetas de x
Output: **myrf**: Modelo de RF entrenado

```
1 myrf =RandomForestClassifier(n_estimators = 160, n_jobs = −1)
2 myrf.fit(x, y)
3 save(myrf)
```

3.5. Sistema Faster R-CNN + SVM

En la presente sección se describen los detalles de la implementación del sistema Faster R-CNN+SVM. Siguiendo la misma idea planteada en la Sección 3.4 se conecta a la capa *fc8.2* de la red de detección Faster R-CNN un clasificador SVM (Figura 3.5), pasando a ser este modelo el encargado de generar como salida vectores de probabilidades de dimensión 21.

El uso de CNN junto con SVM para desarrollar las tareas de clasificación y detección de objetos se ha abordado en variados trabajos [7], [14], [15], [16], [20], [40], [59]. Los resultados obtenidos varían caso a caso, en algunos se logra un aumento en el rendimiento, pero no se puede generalizar. Se debe destacar que en Fast R-CNN [14] (el trabajo previo a Faster R-CNN [41]) se comparan los rendimientos del sistema de detección implementado con softmax y con SVM, obteniendo con este último una disminución de 0.1 puntos en el mAP [14].

Se realizan pruebas del sistema de detección con SVM puesto que corresponde a una alternativa evaluada habitualmente junto con CNN, por lo que, se considera un punto de referencia para comparar con los resultados obtenidos al emplear RF, que es una opción que se ha planteado en menor medida para sistemas de clasifciación y detección que usen CNN. Otro punto importante por el cual se emplea SVM es que existen técnicas de entrenamiento que permiten ajustar los modelos con pequeñas porciones del conjunto de entrenamiento, consiguiendo los mismos o incluso mejores resultados, tal como se detalla en la Sección 2.2.1.5.

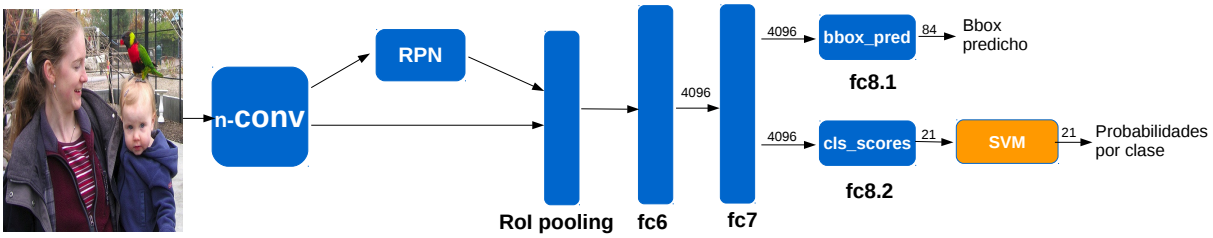


Figura 3.5: Diagrama general del sistema Faster R-CNN+SVM.

En la mayoría de las pruebas el entrenamiento de SVM se realiza usando los vectores de características de la capa *fc8.2* (Figura 3.5). El entrenamiento de SVM con los vectores extraídos desde las capas *fc7* y *fc6* se explica en la Sección 3.5.1.

La librería empleada en este caso es LIBSVM [4], la cual se encuentra desarrollada en C y que provee una interfaz para poder trabajar directamente en Python. En [11] califican esta librería como la mejor implementación de SVM y la tercera entre todas las familias.

Como se señala en la Sección 2.2.1.4, para implementar SVM multiclase destacan los métodos *one-vs-one* (*ovo*) y *one-vs-all* (*ova*). En el primero de estos es el que provee la librería LIBSVM, dado que para VOC07 se tienen 21 clases, entonces bajo esta metodología se entrenan 210 clasificadores binarios, por lo que además se implementa la versión *ova*, para acelerar la obtención de resultados, ya que, con esta se ajustan solamente 21 modelos.

Se prueban para los SVM multiclase (*ovo* y *ova*) dos *kernels*, lineal y RBF (*radial basis function*). Para ambos el valor de c es fijado en 1, en la mayoría de los análisis. Además para el *kernel* RBF el valor de γ se escoge como $1/21$, estos son los valores por defecto de la librería LIBSVM [4]. En el siguiente párrafo se indican los procedimientos seguidos para buscar el valor de c óptimo.

Cuando se entrena con el conjunto etiquetado por el método 2 se realiza una búsqueda del parámetro c óptimo para ovo y ova utilizando *kernel* lineal. Específicamente se busca el c que maximice globalmente el rendimiento y no para cada modelo por separado. El método propuesto en [4] para optimizar c es hacer una búsqueda en una grilla y utilizar *v-fold cross-validation*, dividiendo el conjunto de entrenamiento en v subconjuntos de igual tamaño (más detalles en [57]). Lo anterior no se lleva a cabo, debido al consumo de recursos computacionales. Por esto, se cambia la metodología por la búsqueda en una grilla entrenando los clasificadores (SVM ovo y ova) en el conjunto de *train* y midiendo el rendimiento en el set de *validation* para distintos valores de c. Luego para el que se obtiene el mayor mAP se escoje como óptimo (resultados expuestos en la Sección 4.6.1).

Antes de ajustar el SVM los datos de entrenamiento son escalados al intervalo [-1,1]. Lo anterior cuando se trabaja con las salidas desde la capa *fc8.2*. Posteriormente se entrena y se guarda el modelo en el disco. En la etapa de *test* también se escalan los vectores de características. Esto se efectúa con los mismos valores usados en el escalado del conjunto de entrenamiento. Luego se lee el modelo desde el disco y se predicen los repectivos vectores de probabilidades para los *proposals* que se obtienen desde una imagen.

En los Algoritmos 9 y 10, se observan seudo códigos que representan el entrenamiento y *test* para SVM. Se debe destacar que en el primero se seleccionan los parámetros del modelo en la línea 2, donde $-t$ 0 fija un kernel lineal, $-b$ 1 se utiliza para poder obtener probabiliadedes desde SVM y $-c$ 1 es el valor del parámetro de costo.

Algoritmo 9: Seudo código de entrenamiento para SVM.

Input : **x**: Matriz de entrenamiento
y: Etiquetas de x
Output: **mysvm**: Modelo de SVM entrenado
amin: Valores mínimos para cada columna de características de x
amax: Valores máximos para cada columna de características de x

```
1 x,amin,amax=svmScaletrain(x,-1,1)
2 param_train_linear=' − t 0 − b 1 − c 1'
3 mysvm= svm_train(y, x, param_train_linear)
4 save(mysvm)
```

Algoritmo 10: Seudo código de *test* para SVM.

Input : **scores**: Arreglo para el cual sus filas representan los vectores de características, es la salida desde fc8.2
mysvm: Modelo de SVM previamente entrenado
amin: Valores mínimos para cada columna de características de x
amax: Valores máximos para cada columna de características de x

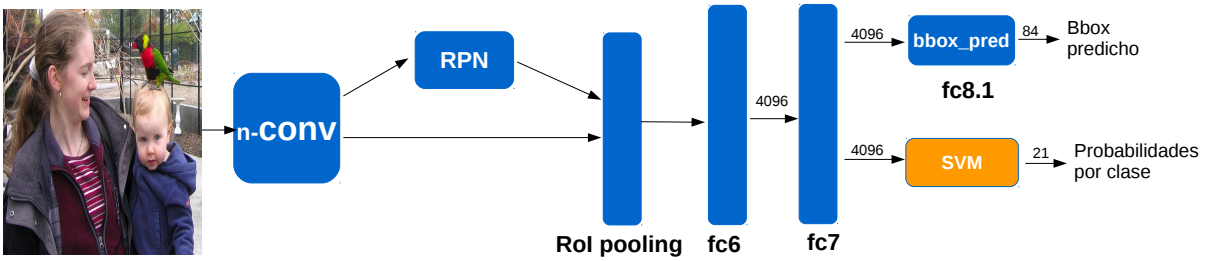
Output: **proba**: Arreglo de probabilidades

```
1 load(mysvm)
2 scores=svmScaletest(scores,-1,1,amin,amax)
3 proba=svm_predict(scores,mysvm,'-b 1')
```

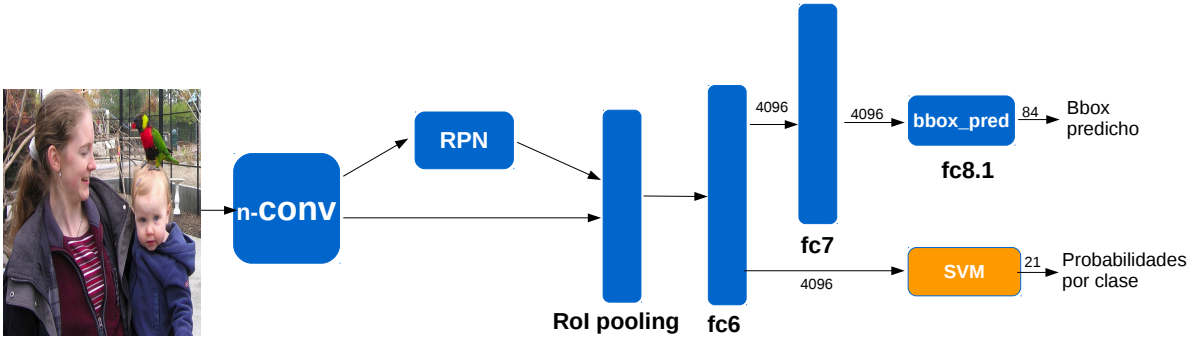
3.5.1. Detalles de la Implementación de Hard Example Mining para SVM

Además de la extracción de características desde la capa *fc8.2* se evalúa si las capas $fc7$ y $fc6$, pueden generar atributos que obtengan mejores resultados que la última capa completamente conectada, ya que, como son capas menos profundas pueden sufrir un sobreajuste inferior.

En la Figura 3.6 (a) y (b) se presentan los esquemas del sistema Faster R-CNN+SVM que trabajan con las características que generan las capas $fc7$ y $fc6$ repectivamente. Se debe resaltar que para ambas capas sus vectores de salida poseen una dimensión de 4.096, por lo que, dado el gran número de ejemplos que se pueden extraer desde una sola imagen (miles de muestras de fondo por cada objeto), se vuelve necesario recurrir al método de entrenamiento para SVM *hard example mining*, puesto que, permite un ajuste eficiente en el uso del tiempo y de memoria.



(a) Vector de características extraído desde $fc7$.



(b) Vector de características extraído desde $fc6$.

Figura 3.6: Esquemas sistema Faster R-CNN+SVM.

Se debe recordar que se está trabajando con la base de datos VOC07 la cual contempla 20 clases distintas. Para poder efectuar la tarea de detección se agrega la clase 0 que representa las muestras de fondo. Entonces las clases desde la 1 a la 20 son las categorías de objetos en la base de datos. Debido a la dimensión de los vectores de características de salida desde $fc7$ y $fc6$ resulta más costoso el entrenamiento, por lo cual, solamente se utiliza la versión *ova* de SVM multiclase, ajustando 21 modelos y no 210 como ocurre con el método *ovo*.

Cuando se ajustan los modelos para las clases positivas (1-20) se utiliza para entrenar cada SVM la metodología de HNM, que corresponde al caso en que se aplica *data-mining* solo sobre los ejemplos negativos. Específicamente dado un clasificador j el *caché* inicial C_{j1} considera todos los ejemplos positivos de la clase j y la misma cantidad de muestras negativas seleccionadas desde el conjunto de entrenamiento completo D, respetando las proporciones de cada clase. Luego se aplican los pasos presentados en el Algoritmo 1.

Un caso especial surge cuado se ajusta el modelo 0, que clasifica como clase positiva la clase 0 y todas las restantes como clase negativa, ya que, los ejemplos de fondo son mucho mayores a los de todas las otras clases (12.608 para el conjunto de *trainval*). Bajo estas condiciones se aplica *hard positive mining*. El *caché* inicial C_{01} considera 6.304 (12.608/2=6.304) muestras de la clase 0 y las mismas para el resto de las clases. No se puede trabajar con los 12.608 ejemplos, porque, en la etapa de entrenamiento no se logran ajustar en memoria. Enseguida se aplica el Algoritmo 1, pero limitando en 5.000 la máxima cantidad de ejemplos negativos

que se pueden agregar al *caché*.

En la Sección 2.2.1.5 se indica que según [15] una iteración de HEM es suficiente para obtener buenos resultados, luego en este trabajo cuando se entrena el modelo 0 se itera en una sola oportunidad y para los clasificadores del 1 al 20 se realizan máximo 4 iteraciones. Un detalle de la implementación es que para lograr trabajar con el conjunto de datos completo D, este se dividió en 6 archivos, uno con los ejemplos de los 12.608 objetos y los otros 5 con las muestras negativas obtenidas en intervalos de 1.000 imágenes. De esta forma cuando se aplica *data-mining* se accede de un archivo a la vez, puesto que de lo contrario surgen problemas por falta de espacio en la memoria.

3.6. Algoritmo para Eliminar Múltiples Detecciones de un Mismo Objeto

En los procedimientos de detección usualmente se obtienen múltiples detecciones cercanas para un mismo objeto, tal como se aprecia en la Figura 3.7. La forma más clásica para tratar con estas es utilizar *non-maximum suppression* (NMS), técnica que es empleada en numerosos sistemas de detección [10], [14], [15], [20], [41].

Las detecciones son caracterizadas por un *bounding box* y un puntaje, NMS lo que realiza es ordenar estas detecciones de mayor a menor puntaje y eliminar aquellas que se solapen más de un cierto umbral con alguna detección de más alto puntaje. Esta técnica tiene un gran impacto positivo en competencias que penalizan las dobles detecciones, tales como [9], [45].

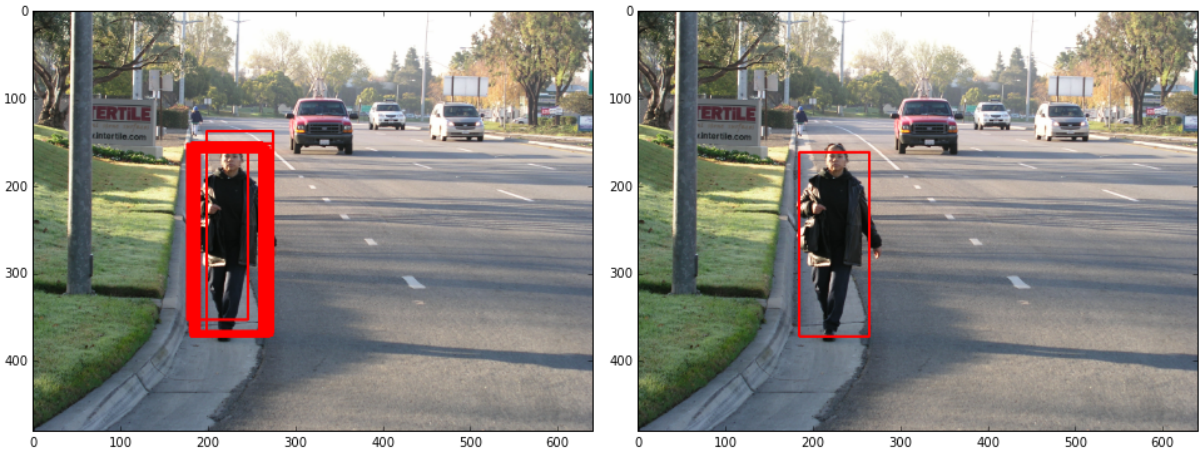


Figura 3.7: Ejemplo de múltiples detecciones y su posterior eliminación con NMS.

Capítulo 4

Resultados y Análisis

4.1. Sistema Faster R-CNN

4.1.1. Resultados Usando la Red ZF

Una de las redes neuronales convolucionales que se utiliza en [41] para implementar el sistema Faster R-CNN es la red ZF, con la cual obtienen la mayor rapidez en la detección. Es por esto que en el presente trabajo esta red se emplea para realizar la mayoría de las pruebas para determinar los parámetros y umbrales óptimos.

Para utilizar la red ZF el sistema Faster R-CNN se entrena con el método aproximado en el conjunto de *trainval*, alcanzando un rendimiento del 60.0 % (Tabla 4.2), que es muy similar al 59.9 % obtenido en [41] para la misma red, pero con el entrenamiento alternado, el cual es más lento y complejo.

Sistema	Train	mAP
Faster R-CNN Replicado	Approx. Join.	60.0
Faster R-CNN [paper]	Alt. Opt.	59.9

Tabla 4.1: Rendimiento sistema Faster R-CNN replicado y el expuesto en [41], medido en el conjunto de *test* VOC07 para la red ZF.

4.1.2. Resultados Usando la Red VGG16

La red más precisa que ocupan en [41] es la conocida como VGG16. Con esta logran un mAP de 69.9 %, usando el entrenamiento alternado, además en la presentación realizada en la ICCV15 sobre el mismo trabajo reportan un rendimiento de 70.0 %, con el entrenamiento aproximado. Al replicar este resultado se alcanza un mAP de 69.3 % (Tabla 4.2). Se debe Recalcar que en los casos mencionados anteriormente el entrenamiento se realiza en el conjunto de *trainval* y los resultados se miden en el conjunto de *test* de VOC07

La red VGG16 se utiliza en el presente trabajo, solamente en los casos en que el sistema implementado con la red ZF obtiene los mejores resultados, considerándose los mismos puntos óptimos, tanto para los umbrales que se emplean en el etiquetado del conjunto de entrenamiento, como para los parámetros de los clasificadores.

Sistema	Train	mAP
Faster R-CNN Replicado	Approx. Join.	69.3
Faster R-CNN [ICCV15]	Approx. Join	70.0
Faster R-CNN [paper]	Alt. Opt.	69.9

Tabla 4.2: Rendimiento sistema Faster R-CNN replicado, el presentado en ICCV15 y el expuesto en [41], medido en el conjunto de *test* VOC07 para la red VGG16.

4.2. Estadísticas Conjuntos de Entrenamiento

Complementariamente a lo expuesto en la Tabla 3.1, la Figura 4.1 muestra un histograma en escala logarítmica de la cantidad de objetos (barras negras) e imágenes (barras rojas),para el conjunto *trainval*.

Como fue indicado en la Sección 3.2, la clase *person* es por lejos la más frecuente. Para el conjunto *trainval* la categoría señalada está presente en 2008 imágenes, llegando a los 4690 ejemplos. La segunda categoría con más apariciones es *car* con 713 imágenes y 1250 objetos, en el extremo opuesto, se encuentra *dining table* con 200 imágenes y 215 apariciones.

Se debe notar que en el histograma mostrado en la Figura 4.1 no está presente la clase *fondo*, la cual debe ser etiquetada siguiendo los métodos propuestos en las Secciones 3.3.2 y 3.3.3. El resultado de estos etiquetados se presentan en los histogramas de las Secciones 4.2.1 y 4.2.2.

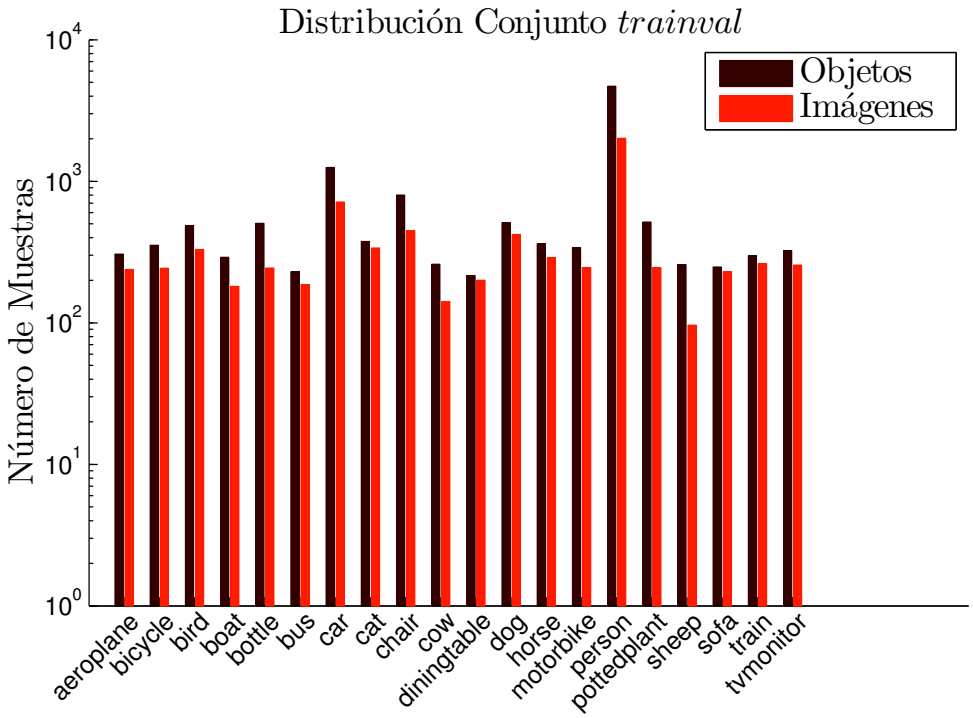


Figura 4.1: Histograma por clase para los *ground truth* del conjunto de entrenamiento *trainval*.

4.2.1. Conjunto de Entrenamiento para Método de Etiquetado 1

En la Sección 3.3.2 se detalla el método de etiquetado 1, el que se fundamenta en los trabajos [14], [15], [20]. En el siguiente cuadro se describen los aspectos esenciales de este método, el cual utiliza la métrica *Intersection over Union* para determinar en qué casos un *bounding box* pertenece a una de las 20 clases de la base de datos o al fondo.

Método de Etiquetado 1

1 Para cada imagen en el conjunto de entrenamiento, usando el bounding box ground truth (B_{gt}) y el bounding box predicho por el sistema (b_{ij}), etiquetar cada proposal de la siguiente forma:

- **1.1** Si $IoU(B_{gt}, b_{ij}) \geq 0.5$ para alguno de los bbox del ground truth considerar al proposal como perteneciente a una de las 20 clases.
- **1.2** $IoU(B_{gt}, b_{ij}) \in [0.1, 0.5) \Rightarrow$ se considera como fondo

2 Redistribuir los ejemplos, tal que la clase fondo corresponda al 90 % de las muestras.

Una vez implementado el método 1 para las imágenes del conjunto de entrenamiento *trainval* (5.011 imágenes), se obtiene la distribución entre clases que se aprecia en la Figura 4.2. Este histograma muestra lo obtenido para las dos redes usadas, es decir, ZF (barras color rojo) y VGG16 (barras color negro). Para la primera de estas se etiquetan 552.182 ejemplos, mientras que para VGG16 432.641

Se puede advertir que la cantidad de ejemplos etiquetados para las clases *bottle, chair* y *potted plan* son las que presentan una menor diferencia entre las dos redes. Se tienen las siguientes cantidades para ZF y VGG16: 747 y 775 para la clase *bottle*, 2.438 y 2.245 para *chair*, 1.287 y 1.196 para *potted plant*.

A continuación se describe la distribución para la red VGG16. La mayor cantidad de ejemplos los concentra la clase *fondo* con 389.367. Para el resto de las categorías se mantiene globalmente la forma de la distribución del set *trainval* (Figura 4.1). La clase *person* sigue siendo la más frecuente con 13.187 muestras, seguida de *car* con 4.725. Por otro lado, la clase con menos ejemplos cambia, la original era *dining table*, y ahora pasa a ser la clase *sheep* con 710 ejemplos.

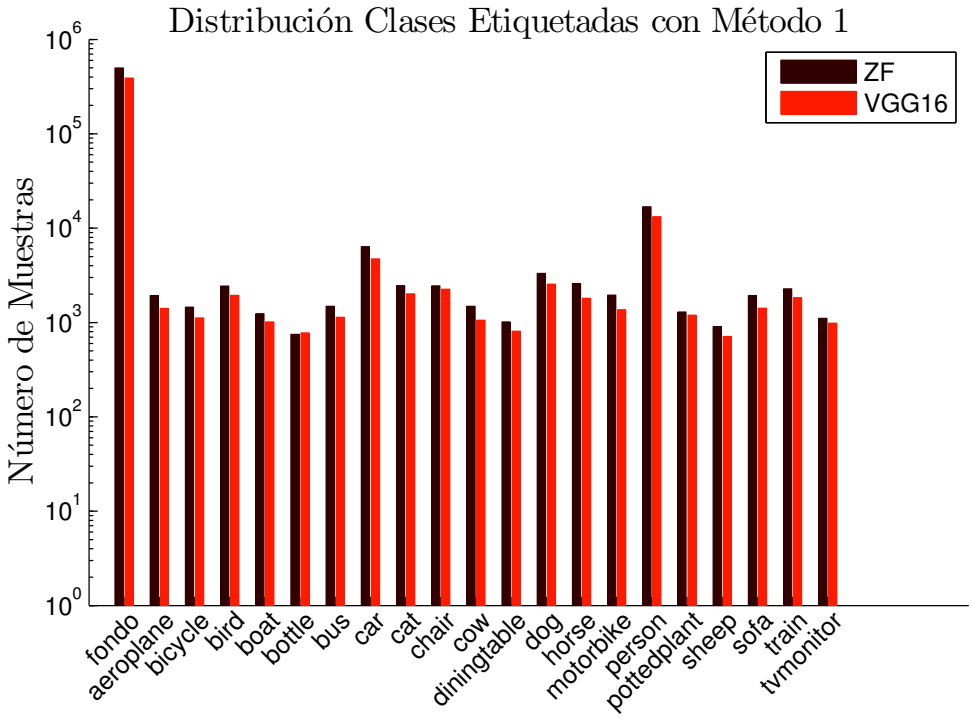


Figura 4.2: Histograma por clase para el conjunto de entrenamiento *trainval* etiquetado con el método 1.

4.2.2. Conjunto de Entrenamiento para Método de Etiquetado 2

El método de etiquetado 2 se describe en la Sección 3.3.3, el cual se sustenta en los trabajos [15], [20]. Este método es similar al primero, en el sentido que los dos usan *Intersection over Union* para etiquetar los *bounding box* predichos por la red. En el siguiente cuadro se describe de forma general el presente método.

Método de Etiquetado 2

1 Para cada imagen en el conjunto de entrenamiento, usando el bounding box ground truth (B_{gt}) y el bounding box predicho por el sistema (b_{ij}), etiquetar cada proposal de la siguiente forma:

1.1 Si $IoU(B_{gt}, b_{ij}) \in [0.0\ ;\ 0.3] \Rightarrow$ se considera como fondo

2 Retirar las muestras negativas que se solapen entre ellas más de un 70 %

3 Considerar como ejemplos positivos solamente a los *bounding boxes ground truth*

Cabe mencionar que en este caso se distinguen dos umbrales, uno se denomina de etiquetado (u_1) y el otro de solapamiento (u_2), que inicialmente son 0.3 y 0.7, tal como se expresa en el cuadro anterior. En la Sección 4.4 se varían estos umbrales para encontrar los valores óptimos, resultando que 0.2 y 0.8 pueden generar mejores resultados, aunque muy cercanos. En consecuencia, para este método de etiquetado, se exponen dos histogramas (Figuras 4.3 y 4.4).

Dado que este método etiqueta solo los ejemplos de la clase *fondo*, la cantidad de muestras positivas es la misma que posee originalmente el set de *trainval*. Es por esto que la distribución de los ejemplos positivos es idéntica en los histogramas expuestos en las Figuras 4.1, 4.3 y 4.4.

En la Figura 4.3 se muestra el histograma para el primer par de umbrales ($u_1 = 0.3$, $u_2 = 0.7$). Se tiene que la cantidad de muestras de fondo etiquetados por ZF es de 347.947 y para VGG16 es de 533.583. Por su parte, para los umbrales $u_1 = 0.2$, $u_2 = 0.8$ (Figura 4.4) la cantidad de muestras negativas es de 360.380 para ZF y 576.369 para VGG16.

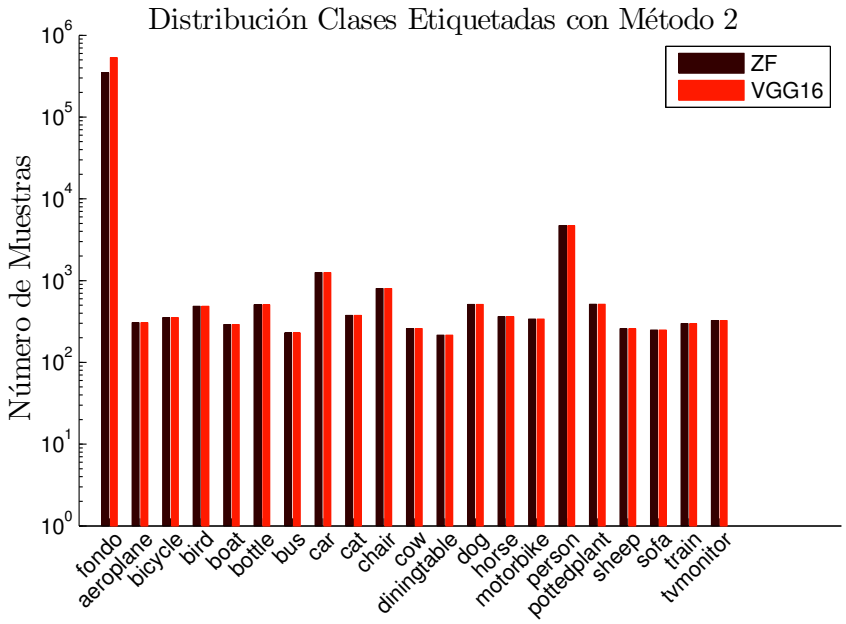


Figura 4.3: Histograma por clase para el conjunto de entrenamiento *trainval* etiquetado con el método 2. Con los umbrales 0.7 y 0.3.

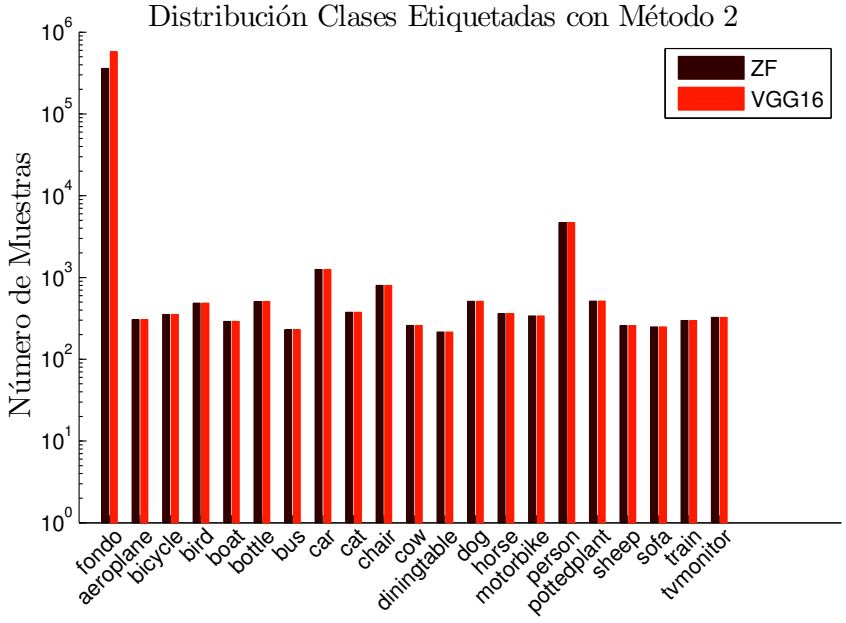


Figura 4.4: Histograma por clase para el conjunto de entrenamiento *trainval* etiquetado con el método 2. Con los umbrales 0.8 y 0.2.

4.3. Sistema Faster R-CNN + RF Método de Etiquetado 1

4.3.1. Resultados Usando la Red ZF

4.3.1.1. Librería randomForest

Se implementa el sistema Faster R-CNN + RF usando la librería randomForest en R [33]. Primero se procede a buscar la cantidad de árboles en el clasificador con la cual se obtiene el mayor rendimiento usando el método de etiquetado 1. Para realizar este análisis se entrena el clasificador en el conjunto de *train* y se mide el mAP del sistema Faster R-CNN+RF en el conjunto *val*. Además se busca la redistribución óptima del conjunto de entrenamiento. Se prueban 6 proporciones distintas (50 %, 60 %, 70 %, 75 %, 80 % y 90 %) entre la clase fondo y el resto de las categorías.

Los resultados de las pruebas antes mencionadas se muestran en la Figura 4.5, donde se aprecia que la mejor redistribución del conjunto de entrenamiento es la que considera un 90 % de ejemplos de la clase fondo (curva roja). Para este caso el mAP promedio es de 84.78 %, alcanzando un valor máximo de 85.2 % con 120 árboles en el clasificador (Tabla 4.3).

Se debe recalcar que para redistribuir las clases se eliminan ejemplos positivos, ya que, estos son inicialmente superiores en número a los ejemplos negativos o de la clase fondo. Dado que los ejemplos positivos se consideran como tal cuando $IoU \in [0{,}5; 1]$, existen muestras que resultan confusas para el sistema debido al umbral de 0.5 empleado. Es por esto que al redistribuir y por ende eliminar ejemplos positivos de baja calidad el rendimiento del sistema aumenta. Similar efecto se podría lograr cambiando el umbral de 0.5 por uno más restrictivo.

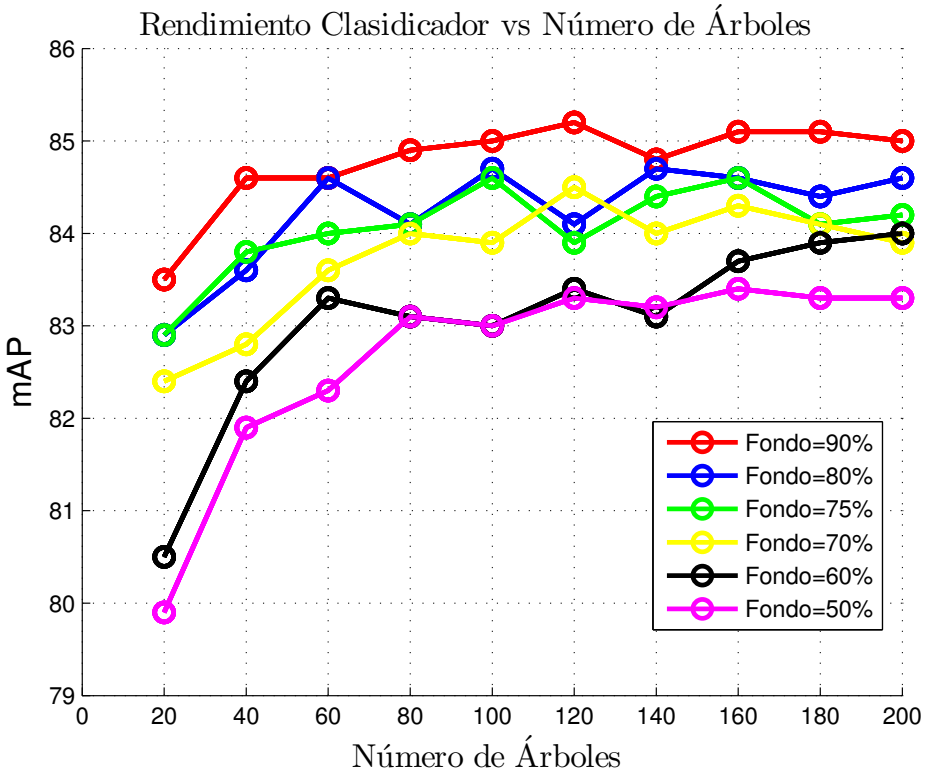


Figura 4.5: Gráfico análisis del número óptimo de árboles en el clasificador.

% clase fondo	mAP Promedio	Número de árboles									
		20	40	60	80	100	120	140	160	180	200
90 %	84.78	83.5	84.6	84.6	84.9	85.0	85.2	84.8	85.1	85.1	85.0

Tabla 4.3: Rendimiento del sistema Faster R-CNN+RF versus el número de árboles, para el conjunto de entrenamiento con un 90 % de los ejemplos pertenecientes a la clase fondo.

Luego de encontrar la redistribución óptima y el número de árboles con los cuales se maximiza el mAP, se procede a entrenar el clasificador en el conjunto de *trainval* y se mide el rendimiento en el conjunto de *test*. Se entrenan tres clasificadores RF con 120, 160 y 180 árboles, ya que, son con los que se obtienen los valores más altos de mAP en el conjunto de validación (Tabla 4.3), los resultado se exponen en la Tabla 4.4. En esta se aprecia que el mayor mAP se consigue con 180 árboles (mAP=57.5 %)

Sistema	mAP
Faster R-CNN+RF120	56.7
Faster R-CNN+RF160	57.4
Faster R-CNN+RF180	57.5

Tabla 4.4: Rendimiento sistema Faster R-CNN + RF medido en el conjunto de prueba, con el número óptimo de árboles.

4.3.1.2. Librería scikit-learn

En esta sección se muestran los resultados del sistema Faster R-CNN+RF, usando el clasificador RF que provee la librería de *machine learning* en Python *scikit-learn* [39].

Se entrena el clasificador en el conjunto de *trainval* etiquetado con el método 1 y se evalúa el rendimiento del sistema Faster R-CNN+RF en el conjunto de *test*. Se realiza el entrenamiento considerando el mismo número de árboles óptimo obtenido con la librería randomForest en R. Los resultados se exponen en la Tabla 4.5.

A pesar que según el estudio sobre la precisión de distintas librerías hecho en [12], *scikit-learn* se encuentra en el séptimo lugar para la familia de *Random Forests* (puesto 18 entre todas las familias), se obtienen para este caso similares resultados al compararlos con los conseguidos para la librería randomForest, la cual según [12] es la tercera mejor librería de la familia y la quinta entre los 179 clasificadores evaluadas.

Sistema	mAP
Faster R-CNN+RF120	57.2
Faster R-CNN+RF160	57.6
Faster R-CNN+RF180	57.5

Tabla 4.5: Rendimiento sistema Faster R-CNN+RF medido en el conjunto de prueba, utilizando la librería *scikit-learn*.

4.3.2. Resultados Usando la Red VGG16

4.3.2.1. Librería scikit-learn

Parar desarrollar el sistema Faster R-CNN+RF con la red VGG16 se usa el RF que provee la librería *scikit-learn*, puesto que, en las secciones anteriores se obtuvo similar performance al comparar los resultados conseguidos con la librería de nombre *randomForest* escrita en R.

Se entrena el sistema para tres casos, con 160, 180 y 200 árboles, consiguiendo el mayor rendimiento para la segunda cantidad mencionada, el mAP para este caso fue de 67.8 % (Tabla 4.6) que es 1.5 puntos inferior al mAP del sistema replicado (Tabla 4.2).

Para los sistemas implementados con RF usando el método de etiquetado 1, se tiene que la red ZF baja 2.4 puntos el rendimiento del sistema base replicado (Tabla 4.1), que solo usa la función softmax en la capa de salida. Por su parte el mejor resultado para VGG16 (Tabla 4.6) disminuye 1.5 puntos el mAP del sistema base, lo cual es menor a lo realizado por ZF, puesto que la red VGG16 es más precisa y al utilizar sus capas para extraer características estas aportan mayor separabilidad entre las clases que se buscan clasificar.

Sistema	mAP
Faster R-CNN+RF160	67.3
Faster R-CNN+RF180	67.8
Faster R-CNN+RF200	67.7

Tabla 4.6: Rendimiento sistema Faster R-CNN+RF medido en el conjunto de prueba, utilizando la librería *scikit-learn* para la red VGG.

4.4. Sistema Faster R-CNN + RF Método de Etiquetado 2

4.4.1. Resultados Usando la Red ZF

4.4.1.1. Librería randomForest

En este caso se realiza en primera instancia un análisis similar al efectuado en la Sección 4.3, es decir, para el método de etiquetado 2 se busca la cantidad de árboles óptimo, entrenando el clasificador en el conjunto de entrenamiento y midiendo el resultado en el conjunto de validación. Además se analiza la mejor redistribución del conjunto de entrenamiento, se prueban 6 proporciones distintas entre el fondo y el resto de las clases. Adicionalmente se estudia el caso que no contempla ninguna redistribución.

El rendimiento del sistema Faster R-CNN + RF para las distintas redistribuciones consideradas se exhibe en la Figura 4.6 donde se advierte que los dos menores rendimientos en promedio se alcanzan al considerar un 50 % y 60 % de la clase fondo. Por su parte, al no efectuar ninguna redistribución (curva celeste) se obtiene el máximo mAP promedio (84.94 % Tabla 4.7).

La disminución en el mAP al desarrollar la redistribución se debe principalmente a que se eliminan ejemplos negativos para generar las proporciones deseadas, por lo que, se entrena con menos muestras, lo cual empobrece el conjunto de entrenamiento.

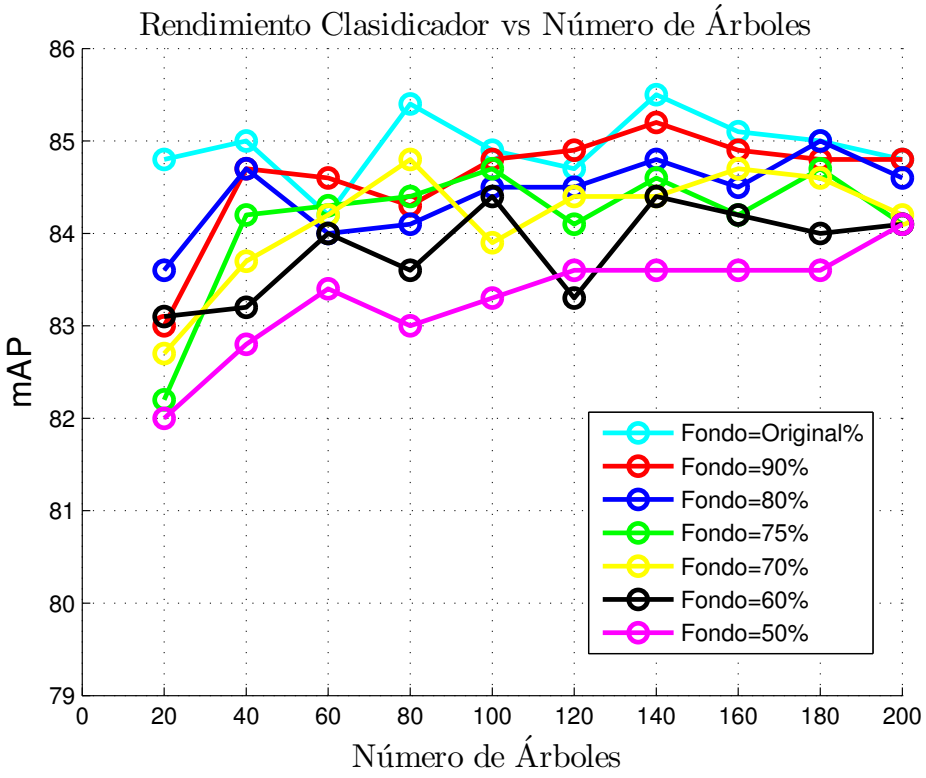


Figura 4.6: Gráfico análisis del número óptimo de árboles en el clasificador.

% clase fondo	mAP Promedio	Número de árboles									
		20	40	60	80	100	120	140	160	180	200
96.45 %	84.94	84.8	85.0	84.2	85.4	84.9	84.7	85.5	85.1	85.0	84.8

Tabla 4.7: Rendimiento del sistema Faster R-CNN+RF versus el número de árboles, para el conjunto de entrenamiento original (sin redistribuir).

Para el conjunto de entrenamiento sin redistribuir las clases, se obtiene que los tres mejores resultados son con 80 (mAP=85.4 %), 140 (mAP=85.5 %) y 160 (mAP=85.1 %) árboles, entonces con estas cantidades se entrena el clasificador RF en el set de *trainval*. Los resultados se detallan en la Tabla 4.8, lo cual es levemente inferior a lo obtenido con el método 1 de etiquetado para la misma librería (Tabla 4.4). Por ejemplo, para 160 árboles se alcanza con el primer método un mAP de 57.4 % y para el segundo método el mAP es de 57.1 %.

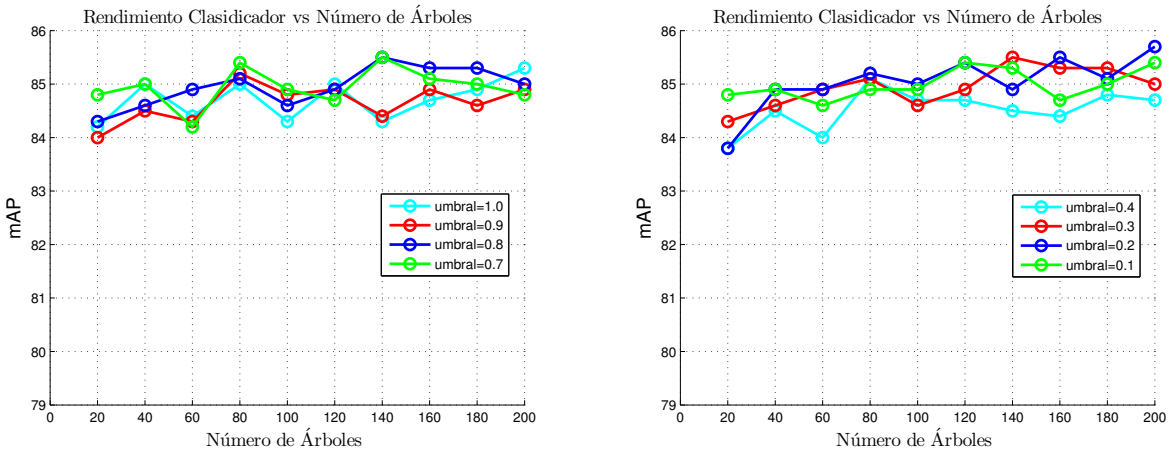
Sistema	mAP
Faster R-CNN+RF80	56.5
Faster R-CNN+RF140	56.8
Faster R-CNN+RF160	57.1

Tabla 4.8: Rendimiento sistema Faster R-CNN+RF medido en el conjunto de prueba, con el número óptimo de árboles, para los umbrales originales del método 2 de etiquetado.

Como fue mencionado en la Sección 3.3.3, el segundo método de etiquetado considera fundamentalmente dos umbrales, uno es el umbral de solapmiento de 0.7, que se usa para eliminar los ejempos negativos que se intersectan en una proporción mayor o igual al mencionado umbral. El segundo es el umbral que se utiliza para definir la cota superior del intervalo de etiquetado de los ejemplos negativos según el criterio de intersección sobre la unión, entre los *bounding boxes* predichos y los de *ground truth*, el valor de este es de 0.3.

Se investiga el efecto de variar los umbrales señalados en el párrafo anterior, para aquello se procede de forma análoga al análisis hecho sobre la redistribución de las clases. Es decir, se entrenan los clasificadores en el conjunto de entrenamiento y se mide el mAP en el conjunto de validación, variando la cantidad de árboles en el *forest* desde 20 hasta 200 con paso de 10, esto se repite para cada umbral distinto probado.

Primero se examinan, para el umbral de solapamiento, los valores 0.7, 0.8, 0.9 y 1.0, obteniéndose una mínima mejora de 0.1 puntos en el mAP promedio con el umbral 0.8 al comparar con los segundos mejores resultados, que se logran con 0.7 (Figura 4.7 (a)). Luego considerando el valor 0.8 se analiza el umbral de etiquetado, explorando los valores 0.1, 0.2, 0.3 y 0.4. El mayor rendimiento se consigue con 0.2 (Figura 4.7 (b) curva azul). Sin embargo, la diferencia en el mAP promedio con los umbrales 0.1 y 0.3 es de 0.05 y 0.09 puntos porcentuales respectivamente, por lo cual, se esperan similares resultados al usar alguno de los tres umbrales mencionados (0.1, 0.2 o 0.3).



(a) Análisis número óptimo de árboles variando el umbral de solapamiento.

(b) Análisis número óptimo de árboles variando el umbral de etiquetado.

Figura 4.7: Análisis número óptimo de árboles, variando los umbrales del método de etiquetado 2.

El análisis anterior refleja que el rendimiento del sistema es insensible ante la variación de los umbrales del método de etiquetado 2, lo cual se ratifica al realizar el entrenamiento en el conjunto de *trainval* con 160 y 200 árboles, que son las cantidades que resultan con mayor mAP (Tabla 4.9) para los nuevos umbrales (0.8 umbral de solapamiento y 0.2 umbral de etiquetado). Los resultados para los nuevos límites se resumen en la Tabla 4.10, estos resultan muy similares a los obtenidos sin realizar el estudio de los umbrales (Tabla 4.8),

para 160 árboles la diferencia es de 0.4 puntos en el mAP, a favor de los umbrales iniciales.

% clase fondo	mAP Promedio	Número de árboles									
		20	40	60	80	100	120	140	160	180	200
96.62 %	85.04	83.8	84.9	84.9	85.2	85.0	85.4	84.9	85.5	85.1	85.7

Tabla 4.9: Rendimiento del sistema Faster R-CNN+RF versus el número de árboles, para el conjunto de entrenamiento con umbral de solapamiento 0.8 y umbral de etiquetado 0.2 .

Sistema	mAP
Faster R-CNN+RF160	56.7
Faster R-CNN+RF200	56.9

Tabla 4.10: Rendimiento sistema Faster R-CNN+RF medido en el conjunto de prueba, con el número óptimo de árboles, para los nuevos umbrales (0.8 umbral de solapamiento y 0.2 umbral de etiquetado).

4.4.1.2. Librería scikit-learn

Al igual que en la Sección 4.3.1.2 se obtiene el rendimiento del sistema, usando la librería *scikit-learn*. Se realizan pruebas para los dos pares de umbrales probados anteriormente, con las mismas cantidades de árboles expuestas en las Tablas 4.8 y 4.10.

En la Tabla 4.11 se exponen los resultados de medir el mAP en el conjunto de prueba y entrenar los clasificadores en el conjunto de *trainval*. Se observa que los rendimientos son igual de competitivos que los obtenidos con la librería randomForest en R para el método de etiquetado 2 (Tablas 4.8 y 4.10).

Comparando lo obtenido con el método 1 y 2 de etiquetado, se tiene que para el primero existe una leve ventaja en lo que respecta al rendimiento. Por ejemplo para el máximo mAP de ambos métodos existe una diferencia de 0.3 puntos, esto considerando hasta el momento solamente lo expuesto para la red ZF (Tablas 4.4, 4.5 para el método 1 y Tablas 4.8, 4.10, 4.11 para el método 2).

Sistema	Umbral 1	Umbral 2	mAP
Faster R-CNN+RF80	0.7	0.3	56.7
Faster R-CNN+RF140	0.7	0.3	56.8
Faster R-CNN+RF160	0.7	0.3	57.3
Faster R-CNN+RF160	0.8	0.2	56.6
Faster R-CNN+RF200	0.8	0.2	57.0

Tabla 4.11: Rendimiento sistema Faster R-CNN+RF, usando el número de árboles óptimo. Umbral 1 es el umbral de etiquetado y umbral 2 es el umbral de solapamiento.

4.4.2. Resultados Usando la Red VGG16

4.4.2.1. Librería scikit-learn

Las últimas pruebas que se efectúan para el sistema Faster R-CNN+RF consideran las cantidades de 160, 180 y 200 árboles, usando para el entrenamiento los conjuntos etiquetados con los dos pares de umbrales expuestos en las secciones anteriores.

Al observar los resultados expuestos en la Tabla 4.12 y compararlos con los de la Tabla 4.6, se distingue que el rendimiento para el método de etiquetado 1 es superior al método 2 en al menos 0.9 puntos, con lo que la diferencia resulta más evidente que la expresada por el mismo sistema pero usando la red ZF.

Sistema	Umbral 1	Umbral 2	mAP
Faster R-CNN+RF160	0.7	0.3	66.2
Faster R-CNN+RF180	0.7	0.3	66.4
Faster R-CNN+RF200	0.7	0.3	66.4
Faster R-CNN+RF160	0.8	0.2	66.2
Faster R-CNN+RF180	0.8	0.2	66.3
Faster R-CNN+RF200	0.8	0.2	66.2

Tabla 4.12: Rendimiento sistema Faster R-CNN+RF, usando la red VGG16. Umbral 1 es el de etiquetado y umbral 2 es el de solapamiento.

4.5. Sistema Faster R-CNN + SVM Método de Etiquetado 1

4.5.1. Resultados Usando la Red ZF

En este caso se utiliza el método de etiquetado 1 redistribuido, considerando el 90 % de los ejemplos pertenecientes a la clase fondo, para entrenar en el conjunto de *trainval* un clasificador SVM, empleando el método *one-versus-one* para clasificación multiclase.

Como se ha mencionado en secciones anteriores para todas las pruebas efectuadas con SVM se utiliza la librería *libsvm*. En la Tabla 4.13 se exhiben los resultados de medir el mAP del sistema Faster R-CNN + SVM en el conjunto de *test*, para dos pruebas distintas. La primera considera el entrenamiento del SVM con kernel RBF y parámetros $c = 1$ y $\gamma = 1/21$, por su parte, la segunda prueba difiere en que se trabaja con kernel lineal con $c = 1$. El sistema que utiliza kernel RBF resulta con una precisión mayor, alcanzando un mAP de 56.7 %, comparado con el que emplea un kernel lineal, el cual logra un mAP igual a 53.4 %.

Sistema	Parámetros	mAP ovo
Faster R-CNN+SVM RBF	c=1,γ=1/21	56.7
Faster R-CNN+SVM Lineal	c=1	53.4

Tabla 4.13: Resultados sistema Faster R-CNN+SVM método de etiquetado 1, desarrollado usando la red ZF

4.5.2. Resultados Usando la Red VGG16

Para la red VGG16 se realizan las mismas pruebas efectuadas para la red ZF en la sección anterior, obteniéndose resultados similares entre los dos sistemas probados. Específicamente para kernel RBF se consigue un rendimiento de 68.9 y para el kernel lineal el mAP es de 68.7 (Tabla 4.14).

Los valores expuesto en la Tabla 4.14 son los que más se acercan al rendimiento del sistema base replicado que es de 69.3 % (Tabla 4.2). Sin embargo, siguen resultando inferiores. Al comparar los presentes valores con los alcanzados para el mismo método de etiquetado y la misma red (Tabla 4.6) pero usando RF, se tiene que el sistema con SVM lo supera en 1.1 puntos, al contrastar los mejores resultados. Esta mejora en 1.1 puntos se considera importante, inclinando la balanza en favor de utilizar SVM en vez de RF.

Sistema	Parámetros	mAP ovo
Faster R-CNN+SVM RBF	c=1,γ=1/21	68.9
Faster R-CNN+SVM Lineal	c=1	68.7

Tabla 4.14: Resultados sistema Faster R-CNN+SVM método de etiquetado 1, desarrollado usando la red VGG16

4.6. Sistema Faster R-CNN + SVM Método de Etiquetado 2

4.6.1. Resultados Usando la Red ZF

En este apartado se utiliza el conjunto de entrenamiento etiquetado de la forma en que se explica en la Sección 3.3.3, con los umbrales de solapamiento y etiquetado iguales a 0.8 y 0.2 respectivamnete. Se emplean estos valores dado que el rendimiento es similar a cuando se utilizan los umbrales iguales a 0.7 y 0.3, como se puede apreciar en los resultados expuestos en la Sección 4.4.

Usando el etiquetado antes señalado se procede a realizar cuatro pruebas distintas para el entrenamiento del clasificador SVM, estas se detallan a continuación:

1. **Prueba 1:** Entrenamiento en set de *trainval* sin escalar y con kernel RBF ($c = 1$ y $\gamma = 1/21$)

2. **Prueba 2:** Entrenamiento en set de *trainval* escalado al intervalo $[-1; 1]$ y con kernel RBF ($c = 1$ y $\gamma = 1/21$)

3. **Prueba 3:** Entrenamiento en set de *trainval* escalado al intervalo $[-1; 1]$ y con kernel lineal ($c = 1$)

4. **Prueba 4:** Entrenamiento en set de *trainval* escalado al intervalo $[-1; 1]$, con kernel lineal y buscando un valor de c que optimice el rendimiento de todos los clasificadores en conjunto.

En la prueba número 4 se busca el valor del parámetro c óptimo en el intervalo $[2^{-6}, 2^6]$, considerando 7 valores distintos $(2^{-6}, 2^{-4}, 2^{-2}, 2^0, 2^2, 2^4, 2^6)$. El procedimiento seguido considera primero el ajuste del clasificador SVM en el conjunto de entrenamiento y luego se determina el rendimiento en el conjunto de validación, para cada uno de los 7 modelos entrenados.

En la Figura 4.8 se grafica el mAP determinado en el conjunto de *test* para los 7 valores de c considerados. Esto se efectúa para dos métodos de clasificación multiclase, uno es el *one-vs-one* (ovo) y el otro el *one-vs-all* (ova). Se aprecia que para el primer método (curva azul) $c = 2^4$ maximiza el mAP del sistema, no obstante, los demás valores alcanzan un rendimiento similar, exeptuando los casos de $c = 2^{-6}$ y $c = 2^{-4}$, donde el mAP más bajo ($mAP_{c_{ovo}=2^{-6}} = 83{,}0\,\%$) es hasta 3.1 puntos menor que el máximo ($mAP_{c_{ovo}=2^4} = 86{,}1\,\%$). Para el caso ova (curva roja) el valor $c = 2^{-2}$ es el que optimiza el mAP, sin embargo, la diferencia con el resto de los resultados no supera un punto.

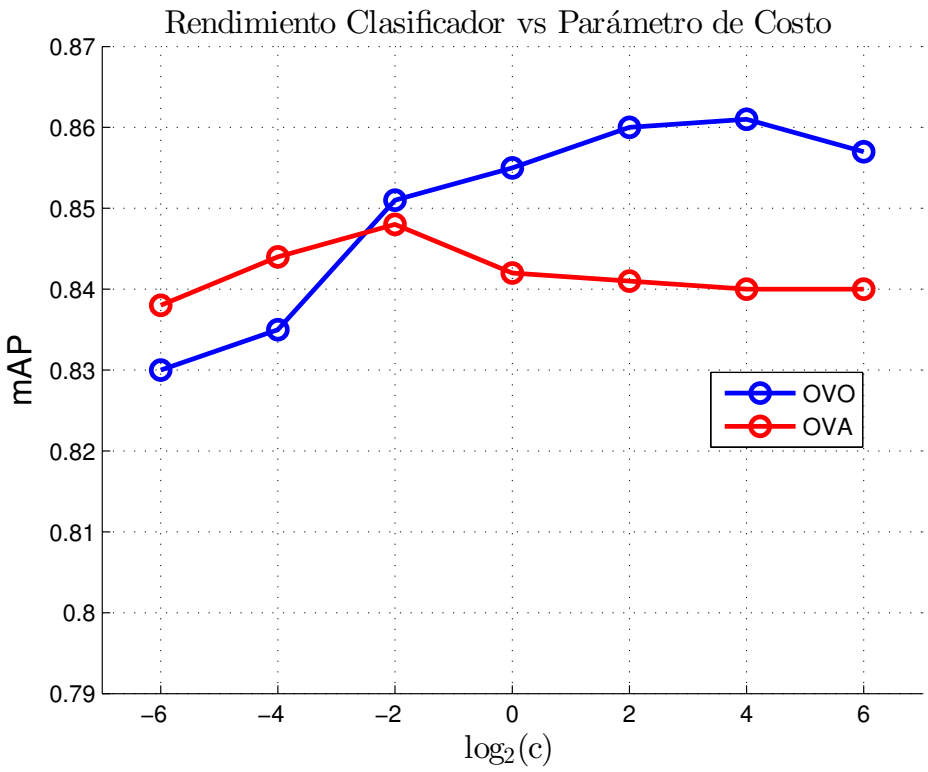


Figura 4.8: Análisis parámetro c que maximiza el rendimeinto de SVM.

Para las cuatro pruebas se contrastan los métodos de clasificación multiclase *one-vs-one* (ovo) y *one-vs-all* (ova), resultando ambos tener una gran similitud en sus rendimientos (Tabla 4.15). Los mejores resultados se obtienen en la prueba 3 (mAP=57.6), es decir, al entrenar un SVM con kernel lineal con $c = 1$ haciendo uso del método de entrenamiento multiclase ovo (Tabla 4.15). El anterior correponde al mejor resultado para la red ZF, al comparar con lo alcanzado por el método 1 de etiquetado (Tabla 4.13).

En lo que respecta al sistema provisto con los valores de c óptimos, se refleja en el resultado final la insensibilidad a este, al menos en el intervalo en que se realizó la búsqueda. Vale la pena señalar que los resultados sin efectuar la optimización de c fueron iguales o levemente superiores (Tabla 4.15). Una eventual solución a esto sería llevar a cabo una búsqueda en un intervalo más grande, o bien buscar el c óptimo para cada clasificador y no uno global. Sin embargo, cualquiera de los dos casos resulta poco práctico dado el tiempo de cómputo que requieren.

Sistema	Parámetros	mAP	
		ovo	ova
Faster R-CNN+SVM RBF [1]	c=1, $\gamma = 1/21$	50.3	50.7
Faster R-CNN+SVM RBF	c=1, $\gamma = 1/21$	56.0	56.7
Faster R-CNN+SVM Lineal	c=1	57.6	54.0
Faster R-CNN+SVM Lineal	$c_{ovo} = 2^4$, $c_{ova} = 2^{-2}$	56.5	54.0

[1] Entrenamiento con set de *trainval* sin escalar

Tabla 4.15: Resultados sistema Faster R-CNN+SVM

4.6.1.1. Entrenamiento con Hard Example Mining

En esta sección se explora el comportamiento del sistema Faster R-CNN+SVM cuando el clasificador final se entrena con vectores de características extraídos desde las distintas capas completamente conectadas (fc). En todas las pruebas de las secciones anteriores se ha utilizado exclusivamente la capa fc8, que corresponde a la capa de salida. En este caso, además se hace uso de $fc7$ y $fc6$.

Cabe señalar que las dimensiones de los vectores de características que se obtienen desde $fc8$ es de 21. Por su parte, los vectores que se extraen desde $fc7$ y $fc6$ son de dimensión 4096, por lo cual, no resulta posible entrenar con estos vectores, ya que, la matriz de características que se forma excede la cantidad de memoria RAM con la cual se cuenta. La solución a este problema es entrenar usando el algoritmo denominado *hard example mining* (HNM) [10], puesto que, con este se logra incluso entrenar los modelos con menos del 1 % de los datos sin sacrificar la precisión del clasificador.

Se usa HNM para el método multiclase *ova*, ya que, con este se entrenan 21 clasificadores, lo cual es 10 veces menor que el número de clasificadores entrenados con el método *ovo*. El resultado obtenido al aplicar HNM para el entrenamiento desde la capa $fc8$ resulta similar al conseguido con el método de ajuste clásico para SVM (Tabla 4.16). Además, se comprueba que para este caso a medida que se usa una capa menos profunda para extraer características el mAP decae, es así como desde $fc7$ se obtiene un rendimiento de 50.8 % y para $fc6$ uno de 49.2 %.

Sistema	mAP ova	mAP ova HNM		
	fc8	fc8	fc7	fc6
Faster R-CNN+SVM RBF	56.7	56.5	-	-
Faster R-CNN+SVM Lineal	54.0	53.7	50.8	49.2

Tabla 4.16: Resultados sistema Faster R-CNN+SVM. Tabla construida con el método 2 de etiquetado con la distribución original y con umbrales de solapamineto de 0.8 y umbral de etiquetado de 0.2.

4.6.2. Resultados Usando la Red VGG16

La última prueba que se realiza para el sistema que considera el clasificador SVM, es el entrenamiento efectuado usando la red VGG16. Se utilizan los conjuntos de entrenamiento (*trainval*) etiquetados con los dos pares de umbrales expuestos para el caso de RF, reflejándose para este sistema, que con ambos etiquetados se obtienen similares valores de mAP (Tabla 4.17).

El método de etiquetado que resulta con un más alto mAP es el 1 (Tabla 4.14) para el caso en que se usa la red VGG16 para el sistema desarrollado con SVM. Esto es similar a lo que ocurre al emplear RF para la misma red (Tabla 4.6). Entre los dos clasificadores (SVM y RF) con el que se obtiene un mayor rendimiento es con SVM.

Sistema	Umbral 1	Umbral 2	mAP	
			ovo	ova
Faster R-CNN+SVM RBF	0.7	0.3	66.8	67.5
Faster R-CNN+SVM Lineal	0.7	0.3	67.5	65.2
Faster R-CNN+SVM RBF	0.8	0.2	66.6	67.4
Faster R-CNN+SVM Lineal	0.8	0.2	66.9	65.2

Tabla 4.17: Resultados sistema Faster R-CNN+SVM para el método de etiquetado 2, desarrollado usando la red VGG16.

4.7. Resumen Mejores Resultados

4.7.1. Rendimiento con red ZF

El resumen de los mejores resultados obtenidos para los sistemas desarrollados con la red ZF se presentan en la Tabla 4.18, en esta además se aprecia el detalle de los AP por cada clase, la columna *Etq* indica el método de etiquetado del respectivo sistema.

El mejor mAP obtenido en este trabajo para la red ZF es de 57.6 (FRCNN+RF160 y FRCNN+SVM Lineal), lo cual es 2.4 puntos menor que el conseguido con el sistema base Faster R-CNN (Tabla 4.18 primera fila). Destacar que para el sistema que utiliza RF los rendimientos más altos se consiguen con la librería *scikit-learn* y con randomForest en R que es citada como la mejor según [11]. Esto se debe a que no hay una gran diferencia estadística en los resultados obtenidos por estas dos librerías [12], en consecuencia la elección de estas tiene poca trascendencia en los resultados finales.

En relación al AP específico para cada clase, existe una clara evidencia que para *bottle, chair* y *potted plant* resulta más compleja la detección, ya que, en todos los sistemas son las categorías que poseen un AP más bajo. Para *bottle y potted plant* esto se debe principalmente a que son objetos pequeños en comparación al resto y además se suelen encontrar ocluidos por otros elementos. Por su parte, la clase *chair* también sufre de oclusiones y por lo general

se encuentra acompañada de la clase *person*, ocurriendo en algunos casos que solo esta es detectada.

El sistema Faster R-CNN lidera en la detección de 16 de las 20 clases (Tabla 4.18), de las 4 restantes en dos no se aprecian diferencias significativas, 0.1 puntos para *aeroplane* y 0.7 para *cow*, esto al comparar con el máximo de la respectiva clase, por otro lado, para la clase *bus* existe una diferencia de 4.7 puntos, y para *motorbike* la distancia es de 4.4 puntos.

Sistema	Etq	Librería	mAP	aero	bike	bird	boat	bott	bus	car	cat	chr	cow	dtbl	dog	horse	mbk	prsn	plnt	shp	sofa	train	tv
FRCNN	-	Caffe	**60.0**	65.0	**72.2**	**55.8**	**46.1**	**36.3**	65.9	**74.9**	**73.1**	**36.1**	63.1	**60.6**	**66.9**	**76.3**	64.6	**67.2**	**33.1**	**57.4**	**55.3**	**69.0**	**60.5**
FRCNN+RF160	m1	sklearn	**57.6**	63.0	69.7	52.6	41.0	32.5	**70.6**	72.6	66.6	35.8	59.2	57.3	64.1	72.3	**69.0**	65.9	28.9	55.0	49.9	66.0	59.4
FRCNN+RF160	m2 [2]	sklearn	57.3	60.0	71.6	55.5	41.1	30.6	65.6	69.3	71.2	34.5	63.7	54.8	65.0	73.2	66.7	65.6	27.8	56.8	52.0	65.3	56.4
FRCNN+SVM RBF [1]	m1	libsvm	56.7	63.5	71.4	53.0	41.3	28.4	63.2	72.9	69.0	33.1	**63.8**	56.5	64.6	73.7	64.2	64.8	25.7	51.9	49.6	64.4	59.2
FRCNN+SVM Lineal[1]	m2 [3]	libsvm	**57.6**	**65.1**	72.1	53.4	41.6	31.7	64.3	69.6	67.1	34.8	61.0	59.1	66.0	71.7	65.1	66.6	31.5	54.9	53.0	65.1	57.1

[1] svm multiclase ovo
[2] umbral solapamiento=0.7, umbral etiquetado=0.3
[3] umbral solapamiento=0.8, umbral etiquetado=0.2

Tabla 4.18: Resumen mejores resultados para los sistemas implementados con la red ZF.

4.7.2. Rendimiento con red VGG16

La utilización de la red VGG16 permite disminuir la brecha que existe entre los sistemas planteados y el Faster R-CNN original propuesto en [41]. Esto debido a que la red VGG16 es más precisa que la ZF, por lo que, con la primera se consiguen extraer mejores características desde sus capas completamente conectadas.

El sistema desarrollado que consigue el más alto mAP es el FRCNN+SVM con *kernel* RBF, para SVM multiclase ovo y entrenado en el conjunto de *trainval* etiquetado con el método 1, alcanzando un mAP de 68.9 (Tabla 4.19), lo cual es 0.4 puntos inferior al sistema base Faster R-CNN [41]. Este es finalmente el mejor resultado obtenido entre todos los sistemas probados.

En lo que respecta al AP por cada clase, se tiene que el sistema Faster R-CNN en este caso es el máximo para 11 categorías (Tabla 4.19), con la red ZF son 16. Para las restantes 9 clases la diferencia con los máximos oscila entre 0.2 para la clase *bike* y 3.1 para la clase *tv/monitor*. En general cuando el Faster R-CNN no logra ser el máximo la diferencia es mínima a favor de otro sistema (menor a 1.0 punto). Por otro lado, el sistema Faster R-CNN+SVM con *kernel* RBF lidera en 5 clases; *bus, chair, cow, dog* y *tv/monitor*

Sistema	Etq	Librería	mAP	aero	bike	bird	boat	bott	bus	car	cat	chr	cow	dtbl	dog	horse	mbk	prsn	plnt	shp	sofa	train	tv
FRCNN	-	Caffe	**69.3**	**69.1**	78.0	**68.5**	**58.1**	**53.5**	76.1	**79.9**	**78.6**	48.9	75.8	**67.8**	77.4	80.5	**76.4**	**76.7**	42.0	68.0	**65.0**	**76.4**	68.6
FRCNN+RF180	m1	sklearn	67.8	67.1	77.3	66.7	54.8	51.7	75.5	79.1	78.3	47.5	71.4	63.4	77.9	**82.1**	75.1	75.9	37.8	**68.4**	62.6	73.7	69.1
FRCNN+RF180	m2 [2]	sklearn	66.4	65.2	**78.2**	66.7	53.6	47.5	75.0	79.0	77.9	43.9	73.5	61.1	76.4	80.3	73.6	73.5	34.4	67.5	61.8	75.0	63.8
FRCNN+SVM RBF [1]	m1	libsvm	**68.9**	68.7	78.0	67.3	56.2	51.9	**76.5**	79.6	77.8	**49.2**	**76.5**	63.6	**79.5**	79.9	76.1	76.1	41.2	68.3	64.4	76.2	**71.7**
FRCNN+SVM Lineal[1]	m2 [2]	libsvm	67.5	68.7	77.1	67.4	52.9	48.4	75.6	79.3	77.6	45.5	74.9	65.7	76.0	80.4	74.2	70.1	**42.2**	67.3	61.1	76.1	68.6

[1] svm multiclase ovo
[2] umbral solapamiento=0.7, umbral etiquetado=0.3

Tabla 4.19: Resumen mejores resultados para los sistemas implementados con la red VGG16.

4.8. Ejemplos de Detecciones de Objetos

Se procede a mostrar algunos ejemplos de detecciones hechas por los sistemas Faster R-CNN y Faster R-CNN+SVM, usando la red VGG16. Para el sistema con SVM se considera la versión multiclase *one-vs-one (ovo)* con *kernel* RBF, puesto que, entre los sistemas desarrollados en este trabajo resulta el con mayor mAP (Tabla 4.19).

En las Figuras 4.9 y 4.10 se muestran ejemplos de detecciones hechas en imágenes de la base de datos PASCAL VOC07, específicamente en el conjunto de *test*. Algunas de las imágenes expuestas se escogieron especialmente, porque, también son mostradas en [41], como por ejemplo el primer y tercer par de imágenes de la Figura 4.9. En cada par de ejemplos, el de la izquierda siempre representa al resultado del sistema Faster R-CNN, mientras que los de la derecha son las salidas del sistema Faster R-CNN+SVM. Para observar el detalle de los *bounding boxes* y los puntajes asociados a estos, se recomienda dirigirse a la versión digital.

Las primeras dos imágenes de la Figura 4.9 corresponden a la clase *cat*, el sistema Faster R-CNN+SVM entrega un puntaje de 0.999, lo cual es 0.001 puntos mayor al entregado por Faster R-CNN. En los ejemplos de la segunda fila se identifican claramente 7 personas. Sin embargo, los sistemas no logran localizar a la persona que está sentada en el extremo de la mesa. Además hay objetos de las clases *potted plant* y *bottle* que tampoco consiguen ser detectados. Se debe destacar que estas son dos de las tres categorías más complejas, además de la clase *chair*. Finalmente en el tercer grupo no se identifica el segundo monitor, ni la silla en la que está sentado el niño. En [41] se muestran detecciones para esta misma imagen, logrando en aquel caso identificar los dos objetos que aquí no. Esto se debe principalmente a las diferencias que se obtuvieron al replicar el sistema original (Tabla 4.2), además los resultados de [41] son para un sistema entrenado en los conjuntos *trainval* de VOC07 y VOC12.

En la Figura 4.10 se presentan variados ejemplos de detecciones, de distintas escalas y relaciones de aspecto, al igual como se hace en [41]. Se pueden apreciar algunas muestras de las clases *bottle, chair* y *potted plant* que son las más complejas de detectar con altos puntajes, debido a varias razones como por ejemplo, que son objetos por lo general pequeños (*bottle* y *potted plant*) y además se pueden encontrar mayoritariamente ocluidos por otros elementos.

Figura 4.9: Ejemplos de detecciones en imágenes de VOC07 *test*, para los sistemas Faster R-CNN y Faster R-CNN+SVM con *kernel* RBF, usando la red VGG16. Se recomienda dirigirse a la versión digital para la correcta apreciación de las detecciones.

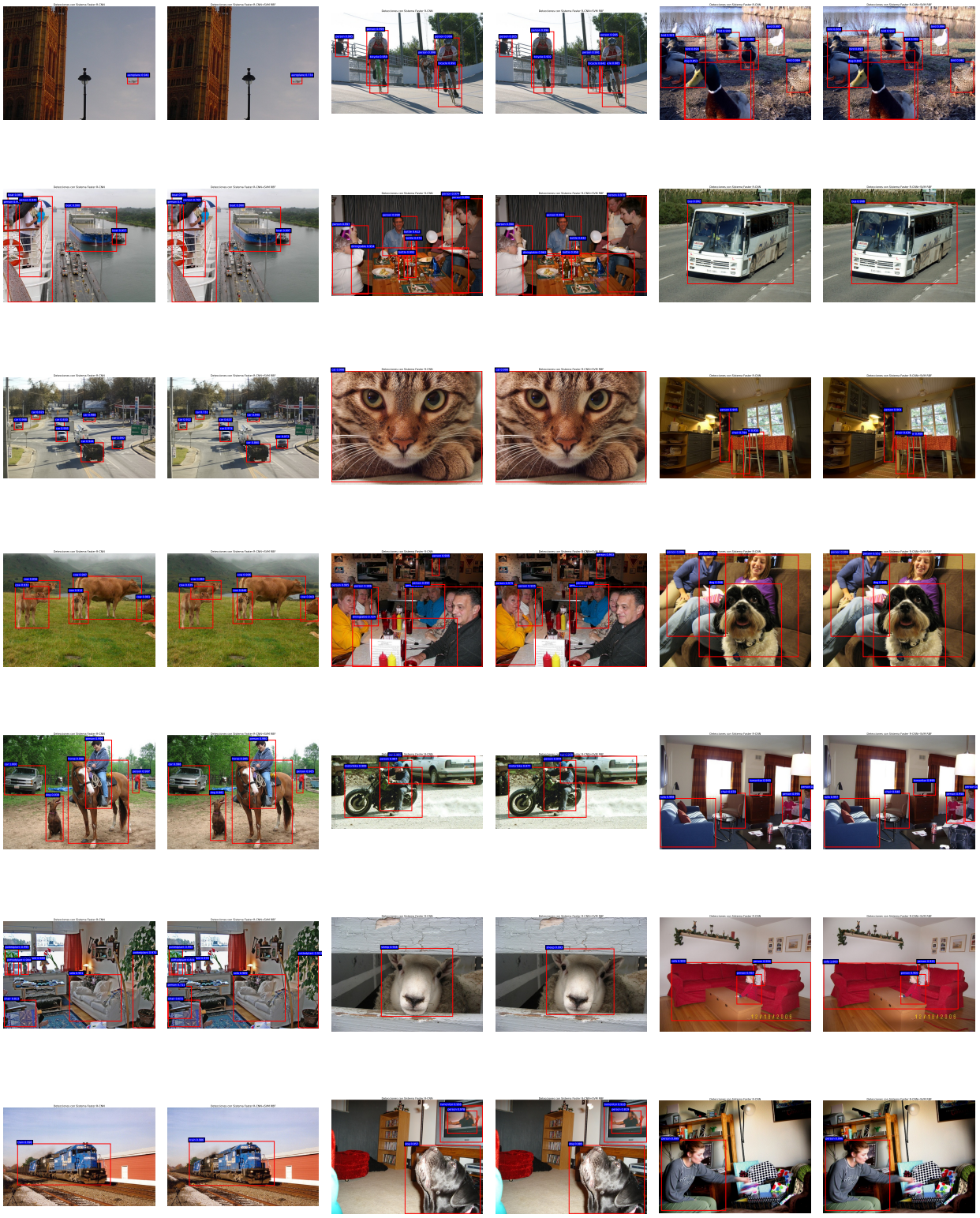


Figura 4.10: Ejemplo de detecciones en imágenes de VOC07 *test*, para los sistemas Faster R-CNN y Faster R-CNN+SVM con *kernel* RBF, usando la red VGG16. Se recomienda dirigirse a la versión digital para la correcta apreciación de las detecciones.

Capítulo 5

Conclusiones y Trabajo Futuro

5.1. Conclusiones

En el presente trabajo se utilizó como base el sistema de detección de objetos conocido como Faster R-CNN [41]. Este sistema utiliza a su vez fundamentalmente 2 arquitecturas de trabajos previos para definir, entre otras cosas, el número de capas convolucionales y el tamaño de los filtros, estas arquitecturas son las redes conocidas como ZF [59] y VGG16 [48]. Utilizando una de las implementaciones disponibles en Github del sistema Faster R-CNN, se replicaron los resultados para las dos redes antes señaladas, obteniéndose un rendimiento de 60.0 % y 69.3 % de mAP, para ZF y VGG16.

Los sistemas desarrollados en el presente trabajo alcanzaron mayor rendimiento cuando se empleó como arquitectura base la red VGG16. Los dos mejores resultados conseguidos fueron con el sistema de detección Faster R-CNN+SVM con *kernel* RBF, logrando un mAP de 68.9 % y con el sistema Faster R-CNN+RF180, con un 67.8 % de mAP. Con ambos sistemas no se logró superar el rendimiento del sistema base (69.3 % de mAP).

En el trabajo realizado se logró apreciar que los dos métodos de etiquetado de los vectores de características permitían obtener similares resultados. El método 1 obtuvo una leve ventaja en el mAP, motivada fundamentalmente, porque, con este se obtienen más datos de entrenamiento para las clases positivas, lo cual favoreció la respuesta de los clasificadores. Por otra parte, se comprobó que los umbrales de etiquetado propuestos originalmente en los trabajos [14], [15], [20], resultaban ser los óptimos o estar en torno a estos en un rango de ± 0.1 puntos de mAP.

Con respecto a la capa completamente conectada desde la cual se extraen los vectores de características, resultó mejor hacerlo desde *fc8.2*, que desde una menos profunda, como *fc7* o *fc6*, ya que, se obtuvo que mientras más lejana de la salida era la extracción de los vectores, menor era el rendimiento del sistema. El mAP para *fc8.2*, $fc7$ y $fc6$ fue de 53.7, 50.8 y 49.2 respectivamente.

Relacionado con el párrafo anterior anterior, cabe destacar la efectividad de la técnica *hard*

negative mining (HNM), que permitió entrenar los sistemas cuando se utilizaban los vectores de características desde $fc7$ y $fc6$, ya que, estos al ser de dimensión 4096, provocaban que el conjunto de entrenamiento creciera por sobre la cantidad de memoria disponible. Con HNM se consiguió entrenar con subconjuntos del set de entrenamiento completo, estos en algunas oportunidades resultaban contener menos del 1 % de los datos, sin perjudicar el rendimiento del clasificador final.

En un principio se probó el efecto de utilizar la mejor librería de RF para implementar el sistema de detección Faster R-CNN+RF, esto siguiendo lo planteado en [11], [12]. Sin embargo, se llegó a la conclusión que la diferencia en rendimiento no logra ser estadísticamente significativa, al menos entre las 20 mejores implementaciones de clasificadores evaluadas en [11], [12].

Un aspecto que se logró comprobar fue que RF resultó más rápido en la etapa de entrenamiento y prueba, comparado con los modelos de SVM. No obstante, con este último se obtuvieron los mejores resultados. Además para RF se obtuvo que el rendimiento del sistema de detección de objetos aumentaba con la cantidad de árboles.

5.2. Trabajo Futuro

Los sistemas evaluados buscan intervenir lo menos posible en la arquitectura original del Faster R-CNN [41], esto para conservar su simplicidad. Manteniendo como fundamento el principio anterior, mejorar el rendimiento de los sistemas planteados no resulta una tarea sencilla.

Se sugiere que para Random Forest existen más posibilidades de mejorar sus resultados, debido a que se ha evaluado en menor medida comparado con SVM, además es un algoritmo paralelizable y que resulta rápido en la etapa de *test*. Una buena idea es seguir lo planteado en [27], que desarrolla una red convolucional profunda en la cual su predicción final se realiza por medio de un árbol de decisión y además implementa un algoritmo para entrenar de extremo a extremo el sistema, incluyendo las CNN y el árbol.

Bibliografía

[1] Yoshua Bengio. Learning deep architectures for ai. *Foundations and Trends in Machine Learning*, 2(1):1–127, 2009.

[2] Leo Breiman. Bagging predictors. *Machine Learning.*, 24(2):123–140, August 1996.

[3] Leo Breiman. Random forests. *Machine Learning.*, 45(1):5–32, October 2001.

[4] Chih-Chung Chang and Chih-Jen Lin. Libsvm: A library for support vector machines. *ACM Trans. Intell. Syst. Technol.*, 2(3):27:1–27:27, May 2011.

[5] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine Learning.*, 20(3):273–297, September 1995.

[6] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 1 - Volume 01*, CVPR '05, pages 886–893, Washington, DC, USA, 2005. IEEE Computer Society.

[7] Jeff Donahue, Yangqing Jia, Oriol Vinyals, Judy Hoffman, Ning Zhang, Eric Tzeng, and Trevor Darrell. Decaf: A deep convolutional activation feature for generic visual recognition. In *International Conference in Machine Learning (ICML)*, 2014.

[8] Richard O. Duda, Peter E. Hart, and David G. Stork. *Pattern Classification (2Nd Edition)*. Wiley-Interscience, 2000.

[9] Mark Everingham, Luc Van Gool, C. K. I. Williams, J. Winn, and Andrew Zisserman. The pascal visual object classes (voc) challenge, 2009.

[10] Pedro F. Felzenszwalb, Ross B. Girshick, David McAllester, and Deva Ramanan. Object detection with discriminatively trained part-based models. *IEEE Trans. Pattern Anal. Mach. Intell.*, 32(9):1627–1645, September 2010.

[11] Manuel Fernández-Delgado, Eva Cernadas, Senén Barro, and Dinani Amorim. Do we need hundreds of classifiers to solve real world classification problems? *J. Mach. Learn. Res.*, 15(1):3133–3181, January 2014.

[12] Manuel Fernández-Delgado, Eva Cernadas, Senén Barro, and Dinani Amorim. Presentation of the paper : Do we need hundreds of classifiers to solve real world classification

problems? https://albahnsen.shinyapps.io/presentation_jmlr_15_delgado14a/, March 2015.

[13] Laurent Gautier. rpy2 2.8.5 : Python interface to the r language. https://pypi.python.org/pypi/rpy2, December 2016.

[14] Ross Girshick. Fast r-cnn. In *International Conference on Computer Vision (ICCV)*, 2015.

[15] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. In *Computer Vision and Pattern Recognition*, 2014.

[16] Georgia Gkioxari, Bharath Hariharan, Ross B. Girshick, and Jitendra Malik. R-cnns for pose estimation and action detection. *CoRR*, abs/1406.5212, 2014.

[17] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org.

[18] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference and prediction*. Springer, 2 edition, 2009.

[19] Simon S. Haykin. *Neural Networks and Learning Machines*. Number v. 10 in Neural networks and learning machines. Prentice Hall, 2009.

[20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Spatial pyramid pooling in deep convolutional networks for visual recognition. In *ECCV*, 2014.

[21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, 2015.

[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[23] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. *CoRR*, abs/1603.05027, 2016.

[24] Geoffrey E. Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *CoRR*, abs/1207.0580, 2012.

[25] Chih-Wei Hsu and Chih-Jen Lin. A comparison of methods for multiclass support vector machines. *Trans. Neur. Netw.*, 13(2):415–425, March 2002.

[26] Andrej Karpathyr. Stanford university cs231n: Convolutional neural networks for visual recognition. http://cs231n.github.io, Septiembre 2015.

[27] Peter Kontschieder, Madalina Fiterau, Antonio Criminisi, and Samuel Rota Bulo . Deep neural decision forests. June 2016.

[28] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[29] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553):436–444, 5 2015.

[30] Yann LeCun, Bernhard E. Boser, John S. Denker, Donnie Henderson, R. E. Howard, Wayne E. Hubbard, and Lawrence D. Jackel. Advances in neural information processing systems 2. chapter Handwritten Digit Recognition with a Back-propagation Network, pages 396–404. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1990.

[31] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. In *Proceedings of the IEEE*, pages 2278–2324, 1998.

[32] Vincent Lepetit, Pascal Lagger, and Pascal Fua. Randomized trees for realtime keypoint recognition. In *In Proc. Int. Conf. on Computer Vision and Pattern Recognition (CVPR 2005)*, pages 775–781, 2005.

[33] Andy Liaw and Matthew Wiener. Classification and regression by randomforest. *R News*, 2(3):18–22, 2002.

[34] David G. Lowe. Object recognition from local scale-invariant features. In *Proceedings of the International Conference on Computer Vision-Volume 2 - Volume 2*, ICCV '99, pages 1150–, Washington, DC, USA, 1999. IEEE Computer Society.

[35] Andrew L. Maas, Awni Y. Hannun, and Andrew Y. Ng. Rectifier nonlinearities improve neural network acoustic models. *ICML*, 2013.

[36] Wes McKinney. pandas: a foundational python library for data analysis and statistics.

[37] M. Oquab, L. Bottou, I. Laptev, and J. Sivic. Learning and transferring mid-level image representations using convolutional neural networks. In *CVPR*, 2014.

[38] Sinno Jialin Pan and Qiang Yang. A survey on transfer learning. *IEEE Trans. on Knowl. and Data Eng.*, 22(10):1345–1359, October 2010.

[39] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[40] Ali Sharif Razavian, Hossein Azizpour, Josephine Sullivan, and Stefan Carlsson. CNN features off-the-shelf: An astounding baseline for recognition. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR Workshops 2014, Columbus, OH,*

USA, June 23-28, 2014, pages 512–519, 2014.

[41] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in Neural Information Processing Systems (NIPS)*, 2015.

[42] Grégory Rogez, Jonathan Rihan, Srikumar Ramalingam, Carlos Orrite, and Philip HS Torr. Randomized trees for human pose detection. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.

[43] Sebastian Ruder. An overview of gradient descent optimization algorithms. *CoRR*, abs/1609.04747, 2016.

[44] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.

[45] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael S. Bernstein, Alexander C. Berg, and Fei-Fei Li. Imagenet large scale visual recognition challenge. *CoRR*, abs/1409.0575, 2014.

[46] Jamie Shotton, Matthew Johnson, and Roberto Cipolla. Semantic texton forests for image categorization and segmentation. In *Proc. IEEE CVPR*, June 2008.

[47] Abhinav Shrivastava, Abhinav Gupta, and Ross B. Girshick. Training region-based object detectors with online hard example mining. In *2016 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2016, Las Vegas, NV, USA, June 27-30, 2016*, pages 761–769, 2016.

[48] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[49] Suraj Srinivas, Ravi Kiran Sarvadevabhatla, Konda Reddy Mopuri, Nikita Prabhu, Srinivas S. S. Kruthiventi, and R. Venkatesh Babu. A taxonomy of deep convolutional neural nets for computer vision. *Frontiers in Robotics and AI*, 2:36, 2016.

[50] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.

[51] Kah Kay Sung. *Learning and Example Selection for Object and Pattern Detection*. PhD thesis, Cambridge, MA, USA, 1996. AAI0800657.

[52] Kah Kay Sung and Tomaso Poggio. Example-based learning for view-based human face detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 20:39–51, 1998.

[53] Sergios Theodoridis and Konstantinos Koutroumbas. *Pattern Recognition, Third Edition*. Academic Press, Inc., Orlando, FL, USA, 2006.

[54] Jasper R. R. Uijlings, Koen E. A. van de Sande, Theo Gevers, and Arnold W. M. Smeulders. Selective search for object recognition. *International Journal of Computer Vision*, 2013.

[55] Li Wan, Matthew Zeiler, Sixin Zhang, Yann L. Cun, and Rob Fergus. Regularization of neural networks using dropconnect. In Sanjoy Dasgupta and David Mcallester, editors, *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, volume 28, pages 1058–1066. JMLR Workshop and Conference Proceedings, May 2013.

[56] Sida Wang and Christopher Manning. Fast dropout training. In Sanjoy Dasgupta and David Mcallester, editors, *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, volume 28, pages 118–126. JMLR Workshop and Conference Proceedings, May 2013.

[57] Chih wei Hsu, Chih chung Chang, and Chih jen Lin. A practical guide to support vector classification, 2010.

[58] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Proceedings of the 27th International Conference on Neural Information Processing Systems*, NIPS'14, pages 3320–3328, Cambridge, MA, USA, 2014. MIT Press.

[59] Matthew D. Zeiler and Rob Fergus. Visualizing and understanding convolutional networks. In David Fleet, Tomas Pajdla, Bernt Schiele, and Tinne Tuytelaars, editors, *Computer Vision – ECCV 2014*, volume 8689 of *Lecture Notes in Computer Science*, pages 818–833. Springer, 2014.