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

DETECCIÓN DE PRENDAS DE VESTIR UTILIZANDO MODELOS DE DETECCIÓN DE OBJETOS BASADOS EN *DEEP LEARNING*

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO

SIMÓN RENATO SEPÚLVEDA OSSES

PROFESOR GUÍA:
JOSÉ MANUEL SAAVEDRA RONDO

MIEMBROS DE LA COMISIÓN:
JORGE FELIPE SILVA SÁNCHEZ
JUAN MANUEL BARRIOS NÚÑEZ

SANTIAGO DE CHILE
2020

B

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE INGENIERO CIVIL ELÉCTRICO
POR: SIMÓN RENATO SEPÚLVEDA OSSES
FECHA: 2020
PROF. GUÍA: JOSÉ MANUEL SAAVEDRA RONDO

DETECCIÓN DE PRENDAS DE VESTIR UTILIZANDO MODELOS DE DETECCIÓN DE OBJETOS BASADOS EN *DEEP LEARNING*

El uso de herramientas de visión computacional ha tenido un fuerte impacto en la industria dadas sus variadas aplicaciones y su capacidad de automatización de complejos y demandantes procesos. Una de las industrias en donde se ha requerido este tipo de herramientas es la industria de la moda, la cual ha tenido un fuerte crecimiento en los últimos años llegando a generar billones de dólares. Estas aplicaciones van desde la recomendación de vestimentas en compras *online*, hasta la caracterización y generación de estadísticas de vestimentas en tiendas físicas.

Dado una inmensa cantidad de posibles aplicaciones, se considera que en su mayoría es necesario realizar detecciones de vestimentas en imágenes, lo cual resulta en una potente y primordial herramienta para su uso ulterior. Es por esto, que en el presente trabajo se evalúan modelos de detección de objetos en el contexto de detección de vestimentas. Para este fin se escogen modelos de *object detection* del área de *deep learning*, debido su increíble capacidad y alta precisión demostrada en la última década. Los modelos escogidos resultan en: YOLOv3, RetinaNet, Faster R-CNN, Mask R-CNN y TridentNet. Los *datasets* utilizados para su entrenamiento son DeepFashion2 y ModaNet, los cuales contienen imágenes de vestimentas en ambientes de interior y exterior.

Luego de la preparación de los datos y *frameworks* se da inicio al entrenamiento, en donde se utiliza una máquina virtual en la nube para acelerar el proceso. Se utiliza ResNet-50 como *backbone* de los modelos, exceptuando YOLOv3 donde se utiliza la red por defecto Darknet-53. Luego del entrenamiento, se utiliza la métrica *average precision* de COCO para su comparación. Además de esto, y con el fin de evaluar los modelos en un ambiente real, es que se utiliza un *dataset* privado proporcionado por el profesor guía.

Se tiene como resultado que TridentNet obtiene el mejor desempeño en todos los *datasets*, a costa de un mayor tiempo de inferencia. Se realizan pruebas para analizar el comportamiento de los modelos en un ambiente real, utilizando el *dataset* privado. Se concluye que dentro de los *datasets* de entrenamiento ModaNet posee mejores resultados en general debido a tener clases más generales y no tan específicas como si tiene DeepFashion2. Como principal resultado se tiene que los modelos poseen alta capacidad de detectar vestimentas de amplio uso, como lo son pantalones, poleras, calzado, lentes, faldas y vestidos, mientras que vestimentas que suelen estar ocluidas por otras vestimentas o las cuales no poseen forma definida, como bufandas y corbatas, poseen bajo índice de detección.

Finalmente, se utiliza la detección de vestimentas aplicándose para su uso en *clothing retrieval*, en donde dada la detección del modelo se genera un vector de características de la vestimenta detectada utilizando una red pre-entrenada. Se utiliza DeepFashion2 para evaluar el resultado. Se obtienen buenos resultados a pesar de lo simple de la estrategia utilizada.

Agradecimientos

Partiré agradeciendo a mi familia, pilar fundamental en mi crecimiento como persona y que sin duda no sería el mismo sin cada uno de sus integrantes. En especial agradecer a mi mamá por estar siempre preocupada de mi educación, donde a pesar de lo adversa de las condiciones y del sacrificio que implicara me entregó la mejor educación que pude haber tenido tanto académica como interpersonal, y también por mantenerme siempre alimentado. A mi hermana por apañarme siempre y cuidarme desde pequeño.

Agradecer a mis amigos, a esos amigos de básica con los cuales no hablas muy a menudo pero sin embargo sabes que puedes contar con ellos y ellos contigo. A los amigos del liceo, los que ya son parte de tu familia y disfrutas con su compañía. A los amigos y amigas de la U con los cuales compartí las primeras experiencias universitarias e hicieron más agradable los primeros años, con los que comparto hasta el día de hoy y espero seguir haciéndolo. A los compañeros y compañeras de eléctrica, que hicieron que ir a clases se hiciera entretenido y hasta lúdico, y a los amigos que gané para toda la vida.

A Jade y Evita, que a pesar de la tarea extra que implica su cuidado, y a veces sus destrozos, te alegran los días con su compañía. A Bronquito que me acompañó gran parte de mi vida y ahora yace en un lugar mejor.

A la persona que me ha acompañado en este último año y que ha hecho mi vida más entretenida, agradable y por sobre todo más feliz, gracias por ser parte de mi vida. A su familia por acogerme y alimentarme.

Y finalmente un agradecimiento especial a las personas que luchan día a día por hacer de este país un país más justo y un mejor lugar para vivir.

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

Introducción

Motivación

El uso de visión computacional en la industria ha tenido un fuerte impacto en la automatización de complejos y demandantes procesos. Se podría mencionar, por ejemplo, la primera utilización de redes neuronales convolucionales para un sistema de reconocimiento de cheques bancarios [4]. Hoy en día las aplicaciones de la inteligencia artificial son variadas, donde la tecnología se encuentra en constante crecimiento y con ello su área de influencia.

Una de las industrias que ha tenido un fuerte crecimiento en la última década es la industria de la moda, no solo por la industrialización en la creación de vestimentas sino que también debido a las distintas modas que imponen las celebridades de televisión, cine y redes sociales. Solamente en Estados Unidos los consumidores gastaron cerca de US$380 billones[1] en apariencia y calzado en el año 2017, cifra que asciende a US$2.5 trillones[2] a nivel global[3]. En el caso de las ventas vía internet (*e-commerce*), la industria generó un total de US$533 billones en 2018, donde se prevee un crecimiento de hasta US$827 para el 2023[4].

Es en este escenario donde se encuentra una posibilidad de aplicar técnicas de *machine learning*, y en particular de *deep learning*, que entreguen herramientas para facilitar procesos de venta o para facilitar al cliente la búsqueda y compra de vestimentas. Un ejemplo de esta aplicación es, dada una imagen, detectar vestimentas y sugerir similares, lo que ayuda al usuario a encontrar de manera rápida el producto deseado, como se muestra en la Figura 1. Estas aplicaciones suponen una gran ventaja para el *e-commerce*, ya que disminuye los tiempos de búsqueda de las personas al comprar, como también entrega una manera sencilla de recomendar vestimentas basado en el producto requerido por el cliente.

Dado esta inmensa cantidad de posibles aplicaciones, se considera que en su mayoría es necesario realizar detecciones de vestimentas en imágenes, lo cual resulta en una potente y primordial herramienta para su uso ulterior. Dentro de estas aplicaciones, no solamente se consideran aplicaciones para el *e-commerce*, si no que también para su uso en tiendas físicas, lo que permite por ejemplo, caracterizar a los clientes que ingresan a ella según su vestimenta,

[1] Del anglosajón *billion* equivalente a 10^9.

[2] Del anglosajón *trillion* equivalente a 10^{12}.

[3] `https://www.jec.senate.gov/public/_cache/files/39201d61-aec8-4458-80e8-2fe26ee8a31e/economic-impact-of-the-fashion-industry.pdf`

[4] `https://www.salecycle.com/blog/featured/online-fashion-retail-11-essential-statistics/`

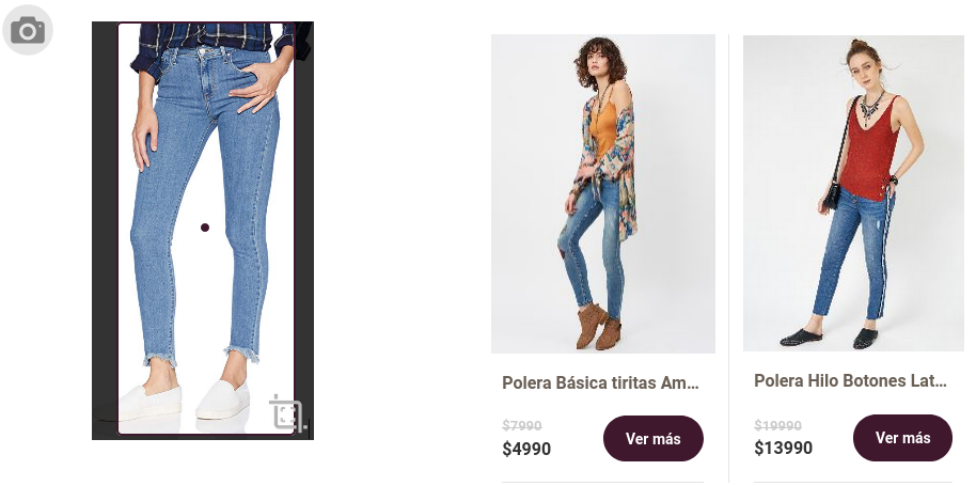


Figura 1: Ejemplo obtenido desde `umbrale.cl`. Dada la imagen, se detectan la prenda y se sugieren imágenes en donde se encuentren prendas similares, en este caso el pantalón.

o la utilización de pantallas interactivas donde se recomienden vestimentas similares para el cliente estando frente a una pantalla.

En consecuencia de esto, surge la problemática de tener un detector de vestimentas capaz de realizar detecciones bajo distintos ambientes, y que además tenga un alto desempeño. Sin embargo, la correcta detección de vestimentas no es una tarea fácil; la gran cantidad de tipos y su gran variedad en formas, tamaños, colores y texturas, conlleva una considerable dificultad para los modelos de detección.

Este trabajo tendrá como fin evaluar distintos métodos de detección de objetos basados en *deep learning* para la detección de vestimentas en imágenes tanto de usuarios como de catálogos de tiendas. Sumado a esto, se utilizará la detección entregada por los modelos para una sencilla aplicación de *clothing retrieval*.

Objetivos

En este trabajo se estudiarán y aplicarán distintos modelos de detección de objetos en el contexto particular de detección de vestimentas. Se realizará el entrenamiento de estos y su posterior comparación utilizando métricas de evaluación.

Objetivos Generales

El objetivo general del presente trabajo es la evaluación de diferentes modelos convolucionales de detección de objetos basados en *deep learning* en el contexto de detección de vestimentas.

Objetivos Específicos

Los objetivos específicos se presentan a continuación:

- Definir modelos de detección de objetos a utilizar.
- Definir *datasets* de vestimenta para el correcto entrenamiento de los modelos.
- Dado los modelos de detección y *datasets* escogidos, realizar el entrenamiento utilizando librerías y *frameworks open source*.
- Evaluar los modelos utilizando métricas apropiadas para el contexto de detección de objetos.
- Comparar los modelos evaluados, tanto cuantitativa como cualitativamente.
- Realizar una sencilla aplicación de *clothing retrieval* dada las detecciones entregadas por un modelo a elección.

Estructura de la memoria

La memoria se encuentra compuesta por la siguiente estructura:

- **Introducción** : Esta sección.
- **Marco Teórico**: Introducción a las redes neuronales convolucionales y posteriormente el detalle de los modelos de detección de objetos que se utilizan en este trabajo. Se detalla también las métricas a utilizar para la comparación de los modelos.
- **Metodología**: Explicación de metodología de trabajo que hizo posible el entrenamiento de los modelos y la obtención de resultados. Se muestra la preparación de los datos y de los *frameworks* de entrenamiento.
- **Resultados**: Se muestran los resultados tanto cuantitativos como cualitativos del entrenamiento de los modelos.
- **Análisis**: Se realiza el análisis de los resultados, en donde se realizan comparaciones de los modelos dadas distintas situaciones de detección. Se analiza también el porqué los modelos aciertan o erran en las distintas clases.
- **Conclusiones**: Se concluye el trabajo realizado de manera concisa, considerando también trabajo futuro.

Capítulo 1

Marco Teórico

En esta sección se introducirán los conceptos necesarios para el completo entendimiento del trabajo realizado. Para ello se detallarán las principales características de una red neuronal convolucional, lo que comprende las operaciones que realiza sobre las entradas como también la arquitectura general de esta.

Luego se detallarán los modelos de detección de objetos que se utilizan en el presente trabajo. Se explicará, para cada modelo, su principio de funcionamiento.

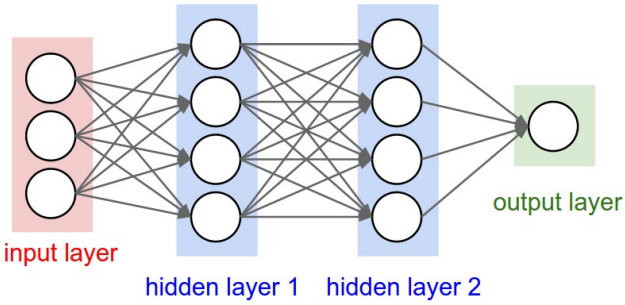
Finalmente se destallarán las métricas de desempeño utilizada para los modelos de detección de objetos, las cuales se utilizan para su comparación.

1.1. Red neuronal Convolucional (CNN)

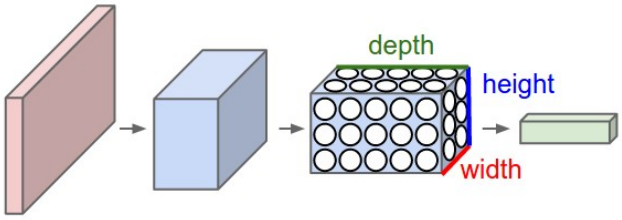
Una red neuronal convolucional (*convolutional neural network*) es un algoritmo de *deep learning* capaz de realizar tareas complejas sobre imágenes, como lo son la clasificación, detección y segmentación de objetos, entre otras. Su funcionamiento es similar a una red neuronal ordinaria (*multilayer perceptron*) al poseer neuronas y pesos entrenables, con la salvedad que utiliza la operación matemática de convolución para realizar el proceso de manera eficiente reduciendo el número de parámetros, como también preservando la información espacial. La comparación con una red neuronal ordinaria y una red neuronal convolucional se presenta en la Figura 1.1.

Anterior a la era del *deep learning* se utilizaban algoritmos de extracción de características clásicos, tales como HOG [5] o LBP [6]. Si bien estas características funcionan bien bajo ciertas condiciones, su elección era arbitraria y dependían mucho del problema a abordar. Con la llegada de las redes convolucionales la elección de estas se hizo innecesaria ya que la red es capaz de aprender las características importantes a extraer generando una mejor representación latente, y en consecuencia mejores resultados.

Para profundizar en las redes convolucionales primero es necesario conocer la operación de



(a) Red MLP.



(b) Red convolucional.

Figura 1.1: Diferencias entre red MLP y una red neuronal convolucional. Mientras que una MLP recibe como entrada un vector, una red convolucional recibe una imagen de uno (escala de grises) o más canales (RGB).

convolución. El cálculo de la convolución para una imagen I y un filtro k de tamaño $L_x \times L_y$ está dado por:

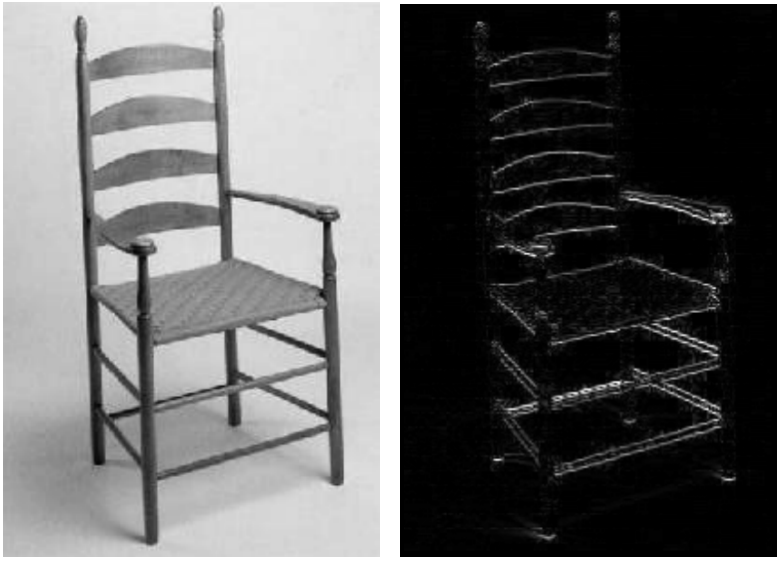
$$y[\mathrm{i}, j] = \sum_{u=0}^{L_x-1} \sum_{v=0}^{L_x-1} k[u, v] \cdot I[\mathrm{i} - u, j - v] \tag{1.1}$$

Donde $y[\mathrm{i}, j]$ es el píxel resultante en la posición (i, j), con $(0, 0)$ las coordenadas de la esquina superior izquierda de la imagen.

Para notar las consecuencias del cálculo de la convolución, se puede ejemplificar su aplicación utilizando el filtro de Sobel en la dirección $\hat{j}$, el cual corresponde a una aproximación del gradiente en esta dirección. Este filtro se define como:

$$S_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ +1 & +2 & +1 \end{bmatrix} \tag{1.2}$$

Se puede aplicar este filtro sobre una imagen como muestra la Figura 1.2. Se observa como la imagen conserva las componentes en la dirección contraria a la del filtro aplicado. La gran ventaja de una red neuronal convolucional es su capacidad de obtener estos filtros de manera automática al ser entrenada en un conjunto de datos pertinente al problema a abordar.



(a) Imagen de silla en escala de grises.

(b) Imagen luego de aplicación de filtro de Sobel.

Figura 1.2: Ejemplificación de aplicación de filtro de Sobel en la direción $\hat{j}$.

Una CNN se encuentra compuesta de diversas capas, donde las principales corresponden a capas convolucionales, capas de *pooling* y capas *fully connnected*.

1.1.1. Capa convolucional

Una capa convolucional es el núcleo principal de una red convolucional. Gracias a la naturaleza de la convolución es que una capa convolucional es capaz de aprender relaciones espaciales dentro de la imagen, obteniendo un mejor entendimiento de la imagen a nivel global. Su salida está dada por un tensor tridimensional el cual recibe el nombre de ***feature map***. Un diagrama de una capa convolucional se presenta en la Figura 1.3.

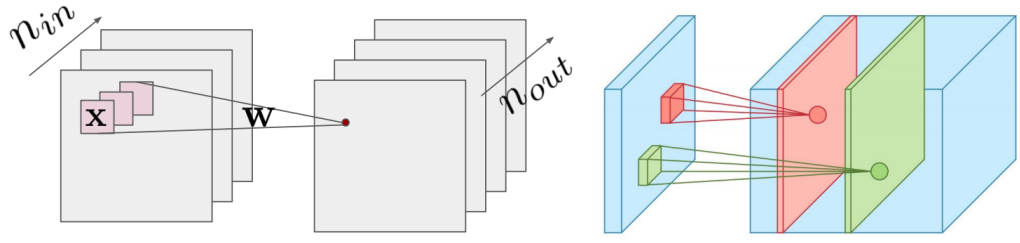


Figura 1.3: Capa convolucional

Cada capa convolucional posee los siguientes hiper parámetros:

- **Tamaño del filtro:** Corresponde a las dimensiones de los filtros utilizados en la capa, y es lo que define el campo receptivo observado como también el número de parámetros entrenables. Generalmente se suelen utilizar filtros cuadrados.

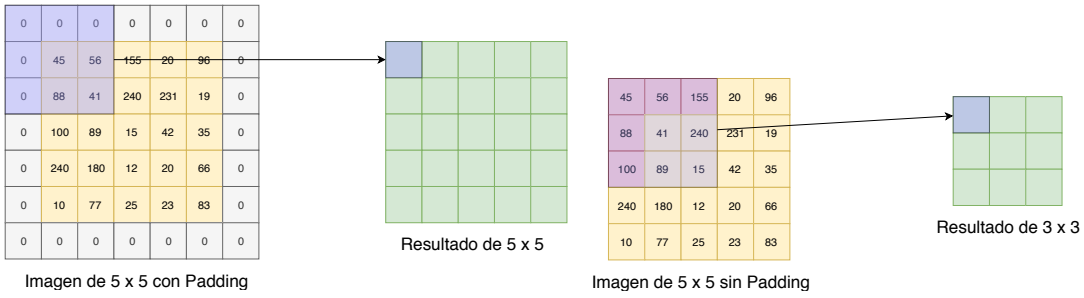
- **Profundidad de la capa:** Corresponde al número de filtros a utilizar, donde cada uno de ellos se encarga de extraer características particulares de la entrada. Estos filtros poseen como parámetros su tamaño los cuales determinarán el número de parámetros entrenables. Una capa de tamaño $W \times H \times K$, ancho, alto y número de filtros (profundidad) respectivamente, tendrá un total de $(C_{in} \times W \times H + 1) \times K$ parámetros entrenables, donde C_{in} corresponde al número de canales de la entrada.
- ***Stride:*** Corresponde al desplazamiento del filtro al realizar la convolución.
- ***Padding:*** Se puede aplicar *zero padding* a la imagen con tal de no perder bordes de esta al realizar la convolución, como se muestra en la Figura 1.4. La cantidad de *padding* añadido, que corresponden a filas y columnas de ceros, se escogen arbitrariamente y suele ser en función del *stride* escogido.

  Se puede calcular entonces el tamaño del *feature map* obtenido luego de una capa de convolución de dimensiones $W_f \times H_f \times K$, dado por las fórmulas:

$$W_f = \frac{W - F + 2P}{S} + 1 \tag{1.3}$$

$$H_f = \frac{H - F + 2P}{S} + 1 \tag{1.4}$$

  Donde $W \times H$ son las dimensiones de la entrada, F corresponde al tamaño del filtro (en este caso cuadrado), P a la cantidad de *padding* añadido y S al *stride* escogido.



(a) Resultado de convolución con *padding* $P = 1$. La imagen resultante conserva las mismas dimensiones que la imagen de entrada.

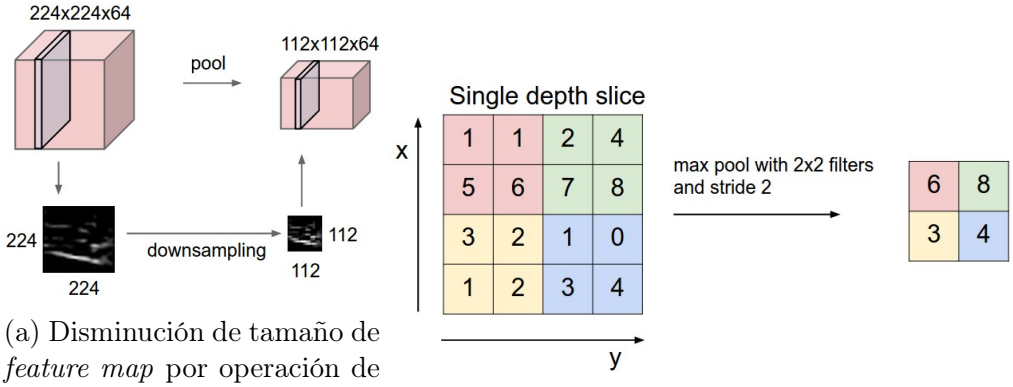
(b) Resultado de convolución sin *padding*. La imagen disminuye su tamaño perdiendo dos filas y columnas.

Figura 1.4: Resultado de convolución con y sin *padding*. Para ambos ejemplos se utiliza *stride* $S = 1$.

1.1.2. Capa de *Pooling*

Una capa de *pooling* se encarga de reducir el tamaño de los *feature maps* generados con tal de disminuir el número de parámetros y tiempo de cómputo de la red. Estas capas suelen ubicarse seguidas de la capa de convolución sin embargo no poseen pesos entrenables. El esquema de aplicación se presenta en la Figura 1.5.

Los hiperparámetros que la definen son:



(a) Disminución de tamaño de *feature map* por operación de *pooling*.

(b) Ejemplo de *max pooling* de 2×2.

Figura 1.5: Aplicación de *pooling* luego de capa convolucional.

- **Tipo de *pooling*:** Corresponde a la función utilizada en su aplicación. La más utilizada es *max pooling*, la cual entrega el valor máximo de la región observada cuyo tamaño se define arbitrariamente. También se utiliza *average pooling*, la cual entrega el promedio de los valores observados.
- **Tamaño de la región:** Tamaño de la región a la cual se le aplica el *pooling*.
- ***Stride:*** Al igual que en la capa convolucional corresponde a los saltos de píxeles al mover la región observada.

1.1.3. Capa *Fully connected*

En una red CNN clásica se utiliza una red neuronal MLP para realizar la tarea de clasificación luego de la extracción de características realizadas por las capas convolucionales. Las neuronas en esta capa se conectan con todas las salidas de la capa anterior, para luego generar una salida de tamaño igual al número de clases de entrenamiento.

1.2. *Residual Network* (ResNet)

Con la llegada de AlexNet [7] hubo un antes y un después en lo que redes neuronales convolucionales se trata. Obtuvo el primer lugar en la competencia de ImageNet en 2012, versus algoritmos clásicos de extracciones de características seleccionados manualmente. Seguido de esto surgieron redes como VGG [8], la cual era más profunda y obtenía mejores resultados. La intuición detrás de esto es que al añadir progresivamente más capas, la red era capaz de profundizar su conocimiento y por lo tanto aprender características cada vez más complejas. Esto si bien es cierto hasta cierto punto, acarrea otros problemas los cuales imposibilitan una red "ultra profunda" con resultados magníficos. Algunos de estos son el sobreajuste de la red (*overfitting*) y el problema de *vanishing gradients*. He et. al [3] demuestra empíricamente que al añadir una cantidad significativa de capas el error en entrenamiento y test incluso empeora, como se ve en la Figura 1.6.

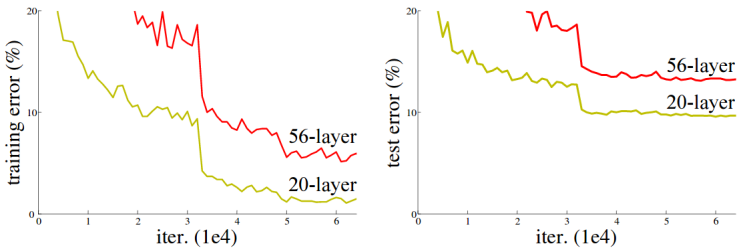


Figura 1.6: Error de entrenamiento (izquierda) y de test (derecha) en CIFAR-10 [2], en redes "planas" de 20 y 56 capas.

Es aquí donde He et. al [3] introduce los llamados bloques residuales (ver Figura 1.7). Los bloques residuales, cuya salida esta dada por $\mathcal{F}(x) + x$, ayudan en la creación de redes más profundas, evitando en su mayoría los problemas de las redes "planas". La hipótesis e intuición detrás de esto es que un bloque residual es más fácil de optimizar que uno "no referenciado", en efecto, si un mapeo de identidad fuera óptimo bastaría con aproximarse a $\mathcal{F}(x) = 0$. Una comparación de una red "plana" con una con bloques residuales se muestra en la Figura 1.8.

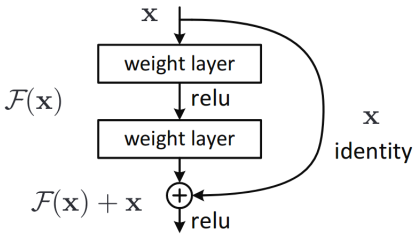


Figura 1.7: Bloque residual [3].

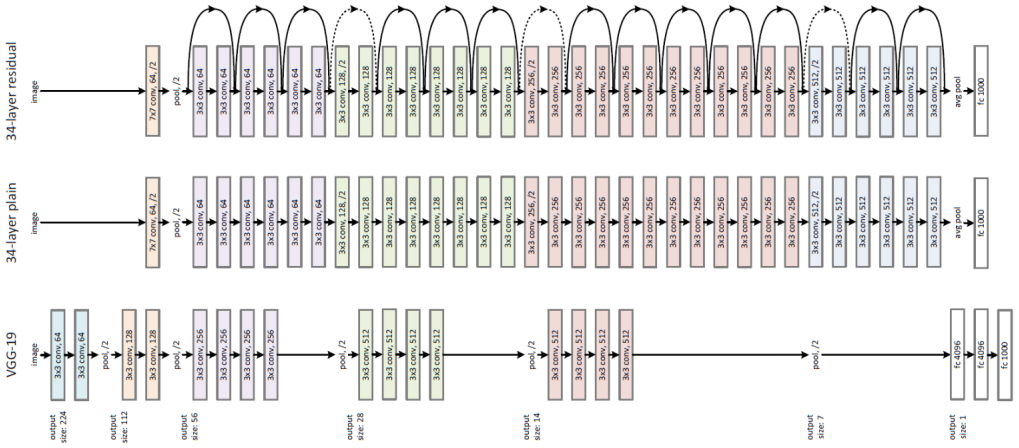


Figura 1.8: De arriba hacia abajo: Red residual con 34 capas, red "plana" de 34 capas, red VGG-19 de 19 capas.

Las redes con bloques residuales fueron denominadas ResNet, donde destacan ResNet-18, ResNet-50 y ResNet-101 donde el número indica la cantidad de capas. Los resultados demostraron que He et. al estaba en lo correcto, al ganar la competencia de ImageNet en 2015 obteniendo un *top-1 error* de solo %5.71 con ResNet-101 frente al anterior ganador

VGG-19 con %9.33 . Desde ese entonces ResNet ha sido ampliamente utilizada siendo la red convolucional por defecto en muchas aplicaciones tales como clasificación y detección, donde en esto último es utilizada como extractora de características.

1.3. Detección de objetos

En la sección anterior se introdujeron las redes neuronales convolucionales para realizar clasificación sobre imágenes, esto con el supuesto de tener un único objeto por imagen. Sin embargo, en las imágenes suelen existir más de un objeto. Es entonces que surge lo que se conoce como *object detection*, área la cual encierra los métodos y algoritmos que realizan detección (y clasificación) de múltiples objetos en una imagen.

Se detallarán los algoritmos de *object detection* los cuales serán utilizados posteriormente. Estos corresponden a Faster R-CNN [9], Mask R-CNN [10], YOLO [11], RetinaNet [12] y TridentNet [13]. Estos modelos se utilizarán ya que son representantes de distintas estrategias de solución para el problema de *object detection*: mientras que YOLO y RetinaNet realizan la detección en una única pasada por la red; Faster R-CNN, Mask R-CNN y TridentNet realizan la detección en dos fases, donde se primero se obtienen posibles objetos dentro de la imagen (*proposals*), y luego se realiza la clasificación de estos objetos. Del mismo modo, estos modelos realizan aproximaciones a la detección multiescala de distinta manera, lo cual se detallará más adelante.

1.3.1. Faster R-CNN

Faster R-CNN [9] es un sistema de detección de objetos el cual tiene su origen en R-CNN [14], evolucionando a Fast R-CNN [15] para finalmente aterrizar en Faster R-CNN. Este modelo consta de tres etapas fundamentales en su funcionamiento.

La primera etapa corresponde a una red convolucional pre-entrenada la cual servirá como extractor de características para las imágenes de entrada.

Seguido a esto se tiene una *Region Proposal Network* (RPN), la cual tiene como objetivo utilizar las características entregadas por la red convolucional para generar regiones de interés en donde se podrían contener objetos. A estas regiones de interés se le aplica la operación de *Region of Interest Pooling* (RoI) reutilizando las características extraídas de la CNN para acotar la cantidad de regiones y generar vectores de caracteristicas de los posibles objetos.

Finalmente entra en acción la red R-CNN, la cual utiliza esta información para clasificar (o descartar) el contenido de cada *bounding box*, y adaptar sus dimensiones de manera de ajustarse mejor al objeto. Un diagrama de la arquitectura simplificada de Faster R-CNN se muestra en la Figura 1.9.

A continuación se explicará en detalle cada una de estas etapas.

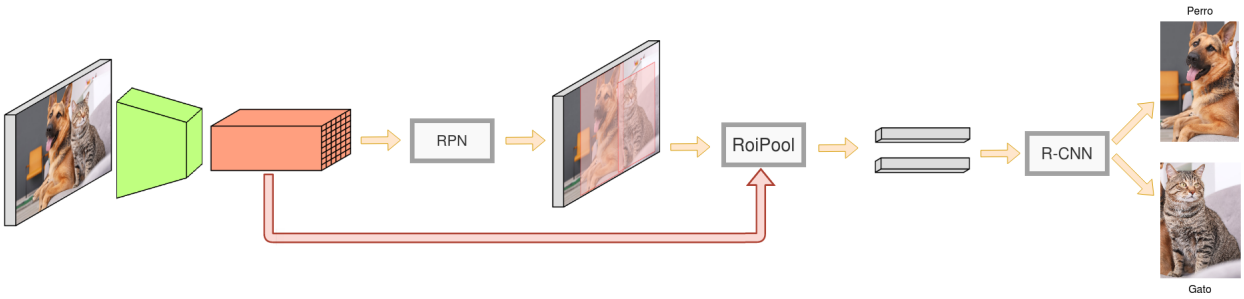


Figura 1.9: Arquitectura de Faster R-CNN. Utilizando la red extractora de características (verde) se genera el *feature map* de la imagen (naranjo) con el cual la RPN genera las regiones de interés, a las cuales se le aplica la operación de *RoIPooling* generando un *feature map* específico para cada una de ellas, para luego ser utilizado como entrada a la red R-CNN y obtener tanto la clasificación como el *bounding box* del objeto.

Red convolucional base (*backbone*)

Para la extracción de características se utiliza una red convolucional pre-entrenada (denominada *backbone*) usualmente en grandes conjuntos de datos como ImageNet, utilizando las características entregadas por alguna capa de la red anterior a la de clasificación. Originalmente Faster R-CNN utiliza VGG [16] como red extractora de características, sin embargo esta elección es arbitraria y suele utilizarse alguna con un buen *trade-off* entre rapidez y precisión. Una de las redes más utilizadas como *backbone* es ResNet en todas sus variantes.

Region Proposal Network (RPN)

La RPN genera regiones de interés en las cuales podría existir un objeto. Para definir los *bounding boxes* de estas regiones se utilizan *anchors*, los cuales corresponden a *bounding boxes* de diferentes tamaños y proporciones que se utilizan como referencia para definir el tamaño de la región.

Con los *anchors* generados la RPN genera regiones con posibles objetos llamados *proposals*. Cada uno de estos *proposals* generan dos salidas, donde la primera corresponde a la probabilidad de ser un objeto de interés (*objectness score*) y la segunda corresponde a su posición relativa respecto al *anchor*. Estas salidas se obtienen gracias a la operación de convolución de 1×1. Se tiene como resultado una capa de *box-regression* y una de *box-classification*, de tamaño $4k$ y $2k$ respectivamente donde k es el número de *anchors*. Esto es ilustrado en la Figura 1.10.

Para el entrenamiento de la RPN se asigna una etiqueta binaria indicando existencia para cada *anchor*. Esta etiqueta es asignada como positiva a el o los *anchors* con el mayor IoU (*Intersection over Union*) con un *ground truth box*, o a un *anchor* con un IoU mayor a 0.7 con algún *ground truth*. Además se asigna una etiqueta negativa si el IoU es menor que 0.3. Los *anchors* que no quedan en ninguna categoría no aportan al entrenamiento.

Luego la función de *loss* queda definida por:

$$L(\{p_\mathrm{i}\},\{t_\mathrm{i}\}) = \frac{1}{N_{cls}}\sum_\mathrm{i} L_{cls}(p_\mathrm{i}, p_\mathrm{i}^*) + \lambda\frac{1}{N_{reg}}\sum_\mathrm{i} p_\mathrm{i}^* L_{reg}(t_\mathrm{i}, t_\mathrm{i}^*) \tag{1.5}$$

Donde i es el índice del *anchor* en cada *mini batch* y p_i es la probabilidad predicha de ser un objeto. La etiqueta de *ground-truth* p_i^* toma el valor de 1 si el *anchor* es positivo y cero en el caso contrario, t_i corresponde a un vector representando las 4 coordenadas parametrizadas del *bounding box* predicho y t_i^* a las del *bounding box* de *ground-truth* asociado al *anchor* positivo. El *loss* L_{cls} corresponde al *log loss* y el *loss* L_{reg} queda definido por $L_{reg}(t_\mathrm{i}, t_\mathrm{i}^*) = R(t_\mathrm{i} - t_\mathrm{i}^*)$ donde R es la función de *smooth loss* definida en [15]. Las constantes N_{cls} y N_{reg} toman el valor del tamaño del *mini batch* y de la cantidad total de *anchors* respectivamente, y λ corresponde a un parámetro de balance.

El vector con las coordenadas parametrizadas de los *bounding box* quedan definidas por:

$$t_x = (x - x_a)/w_a \qquad t_y = (y - y_a)/h_a \tag{1.6}$$
$$t_w = \log(w/w_a) \qquad t_h = \log(h/h_a) \tag{1.7}$$
$$t_x^* = (x^* - x_a)/w_a \qquad t_y^* = (y^* - y_a)/h_a \tag{1.8}$$
$$t_w = \log(w^*/w_a) \qquad t_h^* = \log(h^*/h_a) \tag{1.9}$$

Donde x, y, w, h corresponde al centro del *bounding box* y sus dimensiones, y las variables x, x_a y x^* corresponden a predicho, *anchor* y *ground-truth* respectivamente, al igual que el resto de variables.

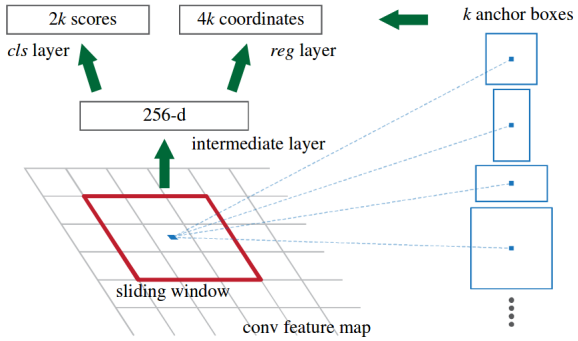


Figura 1.10: *Region Proposal Network*. Para cada ventana deslizante, producto de la operación de convolución, se generan vectores de salida de tamaño $2k$ y $4k$ utilizando convolución de 1×1.

Region of Interest Pooling (RoiPooling)

Con el resultado de la RPN se tiene una serie de posibles objetos y sus *bounding boxes* pero sin clase asignada. Faster R-CNN utiliza RoI (*Region of Interest Pooling*), introducido en [15], el cual reutiliza el *feature map* de la imagen completa obtenido desde la red base para generar un *feature map* independiente para cada uno de los objetos detectados. Para esto sigue como:

1. Se divide el *feature map* de tamaño $h \times w$ en una grilla de tamaño $H \times W$.
2. A cada una de las celdas, de tamaño aproximado $h/H \times w/W$, se le aplica la operación de *max pooling*.

Se obtiene entonces un nuevo *feature map* de tamaño fijo independiente del tamaño del *bounding box*, de tamaño $H \times W \times D$ con D la profundidad del *feature map*. Para Faster R-CNN se utiliza una grilla de tamaño 7×7. Esta operación se ilustra en la Figura 1.11.

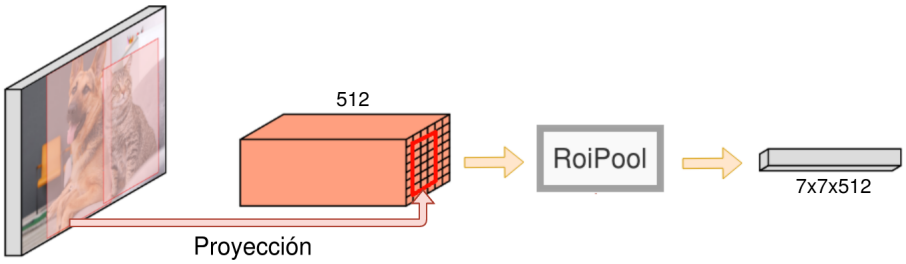


Figura 1.11: Se obtiene un *feature map* para la región de interés. Se tiene una salida de la red base con profundidad 512, por lo que el *feature map* de la región resultante queda de tamaño $7 \times 7 \times 512$.

R-CNN (*head*)

La etapa final de Faster R-CNN es utilizar la red R-CNN y el *feature map* obtenido desde la operación de *RoIPooling* para clasificar el objeto y obtener el *bounding box* final . Para esto, se realiza un estiramiento del *feature map* de cada objeto y se utilizan dos capas *fully connected* para la clasificación. En el caso de utilizar ResNet como *backbone*, con la cuarta (C4) etapa como salida, se utiliza la quinta etapa (C5) como *head* suprimiendo así la necesidad de utilizar capas *fully connected* y utilizando *average pooling* para obtener la salida de la red.

1.3.2. Mask R-CNN

Mask R-CNN [10] surge como una extensión a Faster R-CNN al añadir una nueva rama de predicción de segmentaciones de objetos para cada región de interés, esto en paralelo con la rama existente de clasificación y regresión, como se observa en la Figura 1.12.

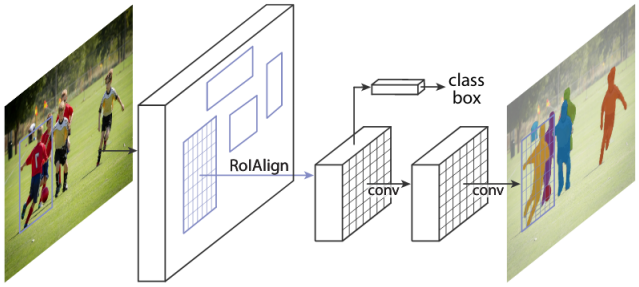


Figura 1.12: Arquitectura en general de Mask R-CNN.

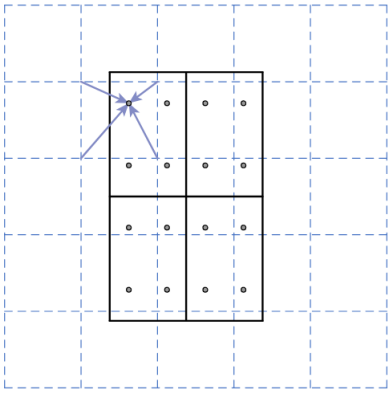


Figura 1.13: *RoiAlign*, donde las lineas punteadas representan el *feature map* y las lineas sólidas un *RoI*.

Para lograr el objetivo de la segmentación es que Mask elimina la operación de *RoIPooling* de Faster R-CNN reemplazándola por *RoiAlign*, esto ya que *RoIPooling* cuantizaba (generaba valores enteros a partir de números en punto flotante) las coordenadas de la región de interés sobre el *feature map* de características, lo que generaba una importante perdida de información, la cual si bien para detección puede considerarse despreciable, no lo es en segmentación debido a su necesaria exactitud a nivel de píxel. Esto puede verse ejemplificado al cuantizar una región de interés de ancho 665 sobre un *feature map* de tamaño 32, generando la coordenada correspondiente $[665/32] = [20.78] = 20$, perdiendo un total de $0.78 \cdot 32$ de la imagen original. Además esto ocurre nuevamente al generar el *feature map* de cada *RoI* con la grilla utilizada por Faster R-CNN de tamaño 7×7.

Para solucionar esto es que *RoIAlign* no realiza ninguna cuantización de las coordenadas, si no que utiliza interpolación bilineal para encontrar el valor correspondiente de estas en el nuevo espacio, sin ningún tipo de aproximación. Para esto utiliza los puntos más cercanos de la grilla en el *feature map*, como muestra la Figura 1.13. Esto le permite a Mask R-CNN generar segmentaciones de los objetos con gran exactitud, al mismo tiempo que entrega el *bounding box* asociado. El detalle de como realiza esta segmentación no se cubrirá en este trabajo debido a que solo se utilizará la detección entregada por el *bounding box*, por lo que para mayor detalle se sugiere revisar su documento técnico [10].

1.3.3. YOLOv3 (*You Only Look Once*)

YOLO es un sistema de detección de objetos en donde su principal característica y ventaja es que realiza tanto la detección como la clasificación en un único proceso (*one-stage detector*), sin necesidad de una *region proposal network* la cual le indique donde se encuentran los posibles objetos.

Para su comprensión se detallarán su primera versión YOLO y su sucesora YOLOv2, antecesora de YOLOv3.

YOLO

YOLO fue introducido por Redmon et. al [11] como un sistema de detección de objetos simple y rápido, el cual alcanza grandes porcentajes de precisión pudiendo ser utilizado en tiempo real.

Detección

YOLO predice en una sola pasada todas las clases junto con sus *bounding boxes*. Para esto YOLO divide la imagen de entrada en una grilla de tamaño $S \times S$ donde cada una de las celdas de esta es responsable de predecir sólo un objeto, el cual es asignado si el centro de éste cae en la celda.

Cada celda predice B *bounding boxes* junto con su *confidence score*, el cual indica la probabilidad de existencia de un objeto. Ese *score* se define formalmente como $\mathbb{P}(Object) \cdot IoU_{pred}^{truth}$. Si no existe objeto en la celda se asigna $\mathbb{P}(Object) = 0$, en caso contrario se asigna este valor igual a 1 y el *score* queda definido por el IoU (*intersection over union*) entre el *bounding box* predicho y el *ground truth*.

Cada predicción consiste en un *bounding box* $[x, y, w, h]$ y su *confidence score*. La posición (x, y) es relativa a la celda donde cae el objeto por lo cual sus valores quedan definidos entre 0 y 1. Del mismo modo el tamaño (w, h) es relativo al tamaño de la imagen.

Además de las predicciones mencionadas, cada celda predice C probabilidades condicionales $\mathbb{P}(Class_i|Object)$, una para cada clase. La cantidad de probabilidades predichas depende únicamente de la cantidad de clases y no de los B *bounding boxes*. En tiempo de test se multiplica la probabilidad condicional de la clase junto con el *confidence score* para asignar un *score* a la clase.

Finalmente esto genera un tensor de salida de la red de tamaño $S \times S \times (B * 5 + C)$. Para *Pascal VOC* [17] se utilizó $S = 7, B = 2$ y $C = 20$ la cantidad de clases lo que da un tensor de salida de $7 \times 7 \times 30$ como se muestra en la Figura 1.14. En la Figura 1.15 se muestra una ilustración de la detección realizada.

Arquitectura de la red

La arquitectura de YOLO inspirada en GoogleNet [18], corresponde a una red neuronal

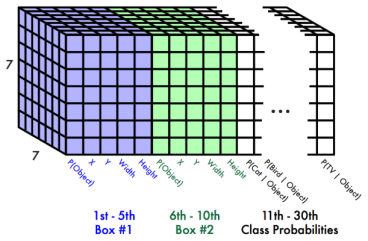


Figura 1.14: Tensor de salida de YOLO en Pascal VOC.

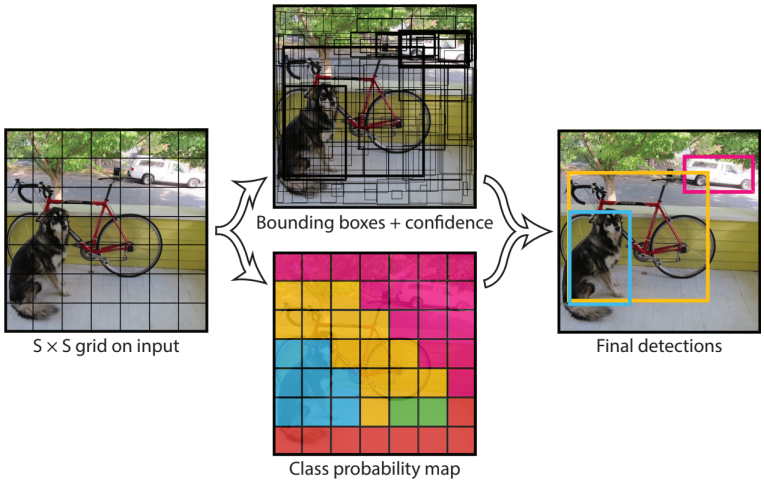


Figura 1.15: Detecciones realizadas por YOLO.

convolucional compuesta de 24 capas convolucionales seguida de dos capas *fully connected* como muestra la Figura 1.16. Además se entrenó una versión ligera de YOLO denominada Fast YOLO la cual solo cuenta con 9 capas convolucionales.

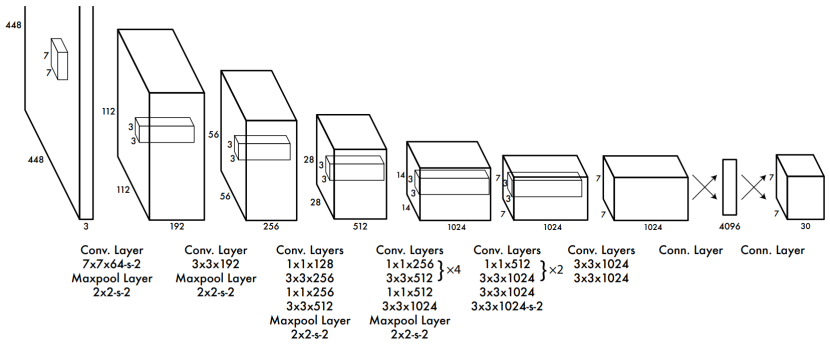


Figura 1.16: Arquitectura de YOLO.

Entrenamiento

Se realiza un pre entrenamiento en el *dataset* ImageNet para las primeras 20 capas convolucionales seguidas de una capa de *average pooling* y una *fully connected*. Luego, para utilizarla para detección, se quitan las dos capas añadidas y se le añaden cuatro capas convolucionales y dos *fully connected* además de aumentar el tamaño de entrada a la red de 224×224 a 448×448, quedando la arquitectura que se muestra en la Figura 1.16.

Se utiliza error cuadrático medio como función de *loss*, la cual se divide en tres componentes principales:

- ***Classification loss*:** si un objeto es detectado el *classification loss* por cada celda es el error cuadrático de la probabilidad condicional por clase. Luego se suma para todas las celdas.

$$\sum_{\mathrm{i}=0}^{S^2} \mathbb{1}_{\mathrm{i}}^{obj} \sum_{c \in \text{ classes}} (p_{\mathrm{i}}(c) - \hat{p}_{\mathrm{i}}(c))^2 \tag{1.10}$$

  Donde $\mathbb{1}_{\mathrm{i}}^{obj} = 1$ si un objeto aparece en la celda i y 0 en caso contrario, y $\hat{p}_{\mathrm{i}}(c)$ la probabilidad condicional para la clase c en la celda i.

- ***Localization loss:*** Corresponde al *loss* de las predicciones de la posición y tamaño del *bounding box*. Nuevamente solo es relevante si el objeto se encuentra en la celda correspondiente.

$$\lambda_{\text{coord}} \sum_{\mathrm{i}=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{\mathrm{i}j}^{obj} [(x_{\mathrm{i}} - \hat{x_{\mathrm{i}}}^2) + (y_{\mathrm{i}} - \hat{y_{\mathrm{i}}}^2)] + \tag{1.11}$$

$$\lambda_{\text{coord}} \sum_{\mathrm{i}=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{\mathrm{i}j}^{obj} [(\sqrt{w_{\mathrm{i}}} - \sqrt{\hat{w_{\mathrm{i}}}})^2 + (\sqrt{h_{\mathrm{i}}} - \sqrt{\hat{h_{\mathrm{i}}}})^2] \tag{1.12}$$

  Se calcula la raíz cuadrada sobre las dimensiones del *bounding box* con el fin de no asignar igual importancia a iguales diferencias entre *bounding boxes* pequeños y *bounding boxes* grandes. Además se multiplica por el factor λ_{coord} a las celdas con objetos para evitar inestabilidad en el entrenamiento debido a la cantidad considerablemente mayor de celdas sin objetos.

- ***Confidence loss:*** corresponde al *loss* para el *confidence score* para objetos predichos y no predichos:

$$\sum_{\mathrm{i}=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{\mathrm{i}j}^{obj} (C_{\mathrm{i}} - \hat{C}_{\mathrm{i}})^2 + \lambda_{\text{noobj}} \sum_{\mathrm{i}=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{\mathrm{i}j}^{noobj} (C_{\mathrm{i}} - \hat{C}_{\mathrm{i}})^2 \tag{1.13}$$

  Donde $\mathbb{1}_{\mathrm{i}j}^{noobj}$ es el complemento de $\mathbb{1}_{\mathrm{i}j}^{obj}$ y λ_{noobj} es un factor para disminuir importancia de las celdas sin objeto.

Finalmente el *loss* de la red queda definido por la suma de estos:

$$\sum_{\mathrm{i}=0}^{S^2} \mathbb{1}_{\mathrm{i}}^{obj} \sum_{c\in \text{ classes}} (p_{\mathrm{i}}(c) - \hat{p}_{\mathrm{i}}(c))^2 + \lambda_{\text{coord}} \sum_{\mathrm{i}=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{\mathrm{ij}}^{obj} [(x_{\mathrm{i}} - \hat{x_{\mathrm{i}}}^2) + (y_{\mathrm{i}} - \hat{y_{\mathrm{i}}}^2)] \tag{1.14}$$

$$+\lambda_{\text{coord}} \sum_{\mathrm{i}=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{\mathrm{ij}}^{obj} [(\sqrt{w_{\mathrm{i}}} - \sqrt{\hat{w}_{\mathrm{i}}})^2 + (\sqrt{h_{\mathrm{i}}} - \sqrt{\hat{h}_{\mathrm{i}}})^2] \tag{1.15}$$

$$+ \sum_{\mathrm{i}=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{\mathrm{ij}}^{obj} (C_{\mathrm{i}} - \hat{C}_{\mathrm{i}})^2 + \lambda_{\text{noobj}} \sum_{\mathrm{i}=0}^{S^2} \sum_{j=0}^{B} \mathbb{1}_{\mathrm{ij}}^{noobj} (C_{\mathrm{i}} - \hat{C}_{\mathrm{i}})^2 \tag{1.16}$$

YOLOv2

Redmon et. al [19] introduce YOLOv2 como una mejora al YOLO original con el fin de lidiar con las limitaciones de este. Las mejoras se definen como *"Better*, *"Faster"* y *"Stronger"*.

Better

YOLO original poseía algunas falencias con respecto a los sistemas de detección en el estado del arte. En particular poseía un mayor número de localizaciones erróneas además de menor *recall* que los sistemas basados en *region proposal*. Para lidiar con esto se introducen las siguientes mejoras:

- ***Batch Normalization*:** Logra mejorar la convergencia además de eliminar la necesidad de otras formas de normalización. Se añade a todas las capas convolucionales lo que genera una mejora del 2 % en mAP. Con esto además se puede remover el *dropout* sin temor a generar sobreajuste.
- **Alta resolución:** YOLO original se entrenaba en resolución 224×224 para luego aumentar a 448×448 para la detección. Con YOLOv2 se hace un *fine tuning* en la red clasificadora en resolución 448×448 por 10 épocas en ImageNet. Esto le da a la red tiempo de ajustar sus filtros para trabajar en una mayor resolución. Esto genera una mejora de al menos 4 % en mAP.
- ***Anchor boxes*:** YOLO predecía los *bounding boxes* directamente desde las capas *fully connected*. En YOLOv2 se eliminan estas capas y se introducen *anchor boxes* para predecir los *bounding boxes*. Esto conlleva a que en vez de predecir el *bounding box* exacto este se predice a través de transformaciones de los *anchor boxes*. Con esto, la predicción de la clase se mueve desde la celda (YOLO) hacia los *anchor boxes*, como también la probabilidad de ser objeto, lo que elimina la limitación de YOLO de predecir sólo una clase por celda.
- ***Dimension clusters**:* Con la introducción de los *anchor boxes* surge la necesidad de elegir éstos de manera adecuada. Es por esto que en YOLOv2 se utiliza un método automático de elección de *anchors* basado en *k-means*. Esto se hace eligiendo un k tal que el IoU de los centroides con los *ground-truth boxes* de los datos de entrenamiento sea el mayor posible. Para la generación de los *clusters* se utiliza la siguiente distancia:

$$\mathrm{d}(box, centroid) = 1 - IoU(box, centroid) \tag{1.17}$$

- ***Direct location prediction**:* Al igual que en YOLO, se predice la ubicación del *bounding box* de manera directa y relativo a la celda. Para las dimensiones del *bounding box* se utiliza una transformación del *anchor box* asociado, con lo que la red predice 5 coordenadas parametrizadas para cada *bounding box*, t_x, t_y, t_w, t_h, t_o. Luego, si la ubicación de la celda con respecto a la esquina superior izquierda es (c_x, c_y) y el *anchor* asociado posee dimensiones p_w y p_h, ancho y largo respectivamente, entonces la predicción será:

$$b_x = \sigma(t_x) + c_x \tag{1.18}$$
$$b_y = \sigma(t_y) + c_y \tag{1.19}$$
$$b_w = p_w \mathrm{e}^{t_w} \tag{1.20}$$
$$b_h = p_h \mathrm{e}^{t_h} \tag{1.21}$$
$$Pr(object) \cdot IoU(b, object) = \sigma(t_o) \tag{1.22}$$

  Donde $\sigma(\cdot)$ corresponde a la función logística, cuyo fin es restringir la predicción entre 0 y 1.

- ***Fine grained features**:* Mientras que YOLO realiza una grilla de 7×7, YOLOv2 utiliza una de 13×13, lo que si bien ayuda en la detección de objetos grandes no necesariamente lo hace para objetos pequeños. Para lidiar con esto YOLOv2 utiliza una capa *passthrough* la cual conecta el *feature map* de salida con una capa anterior de tamaño 26×26. La capa a concatenar es un mapa de tamaño $26 \times 26 \times 512$ la cual se ajusta para calzar con las dimensiones de la última capa quedando en un mapa de tamaño $13 \times 13 \times 2048$.

- **Entrenamiento multi escala:** Debido a la eliminación de las capas *fully connected* YOLOv2 resulta en un modelo *fully convolutional* [20], lo que le permite una entrada de tamaño variable. Esta ventaja se utiliza en YOLOv2 al variar la entrada a la red cada 10 *batches* eligiendo una al azar entre tamaños múltiplos de 32: {320,352,...,608}. Esto fuerza a la red a aprender a predecir objetos a través de múltiples tamaños.

Faster

En YOLOv2 se introduce la arquitectura de red **Darknet-19**, la cual posee 19 capas convolucionales y 5 de *max pooling*, la que se presenta en la Tabla 1.1. Similar a la red VGG se utiliza en su mayoría filtros de tamaño 3×3 y la operación de *global average pooling* en vez de capas *fully connected* para realizar las predicciones. Además, se usan capas convolucionales de tamaño 1×1 para comprimir la representación entre convoluciones de 3×3.

Esta nueva arquitectura requiere solo 5.85 billones (5.85×10^9) de operaciones para procesar una imagen obteniendo 72.9 % *top-1 accuracy* y 91.2 % *top-5 accuracy* en ImageNet.

Capa	#Filtros	Tamaño/Stride	Salida
Convolucional	32	3×3	224×224
Max pool		$2 \times 2/2$	112×112
Convolucional	64	3×3	112×112
Max pool		$2 \times 2/2$	56×56
Convolucional	128	3×3	56×56
Convolucional	64	1×1	56×56
Convolucional	128	3×3	56×56
Max pool		$2 \times 2/2$	28×28
Convolucional	256	3×3	56×56
Convolucional	128	1×1	56×56
Convolucional	256	3×3	56×56
Max pool		$2 \times 2/2$	14×14
Convolucional	512	3×3	14×14
Convolucional	256	1×1	14×14
Convolucional	512	3×3	14×14
Convolucional	256	1×1	14×14
Convolucional	512	3×3	14×14
Max pool		$2 \times 2/2$	7×7
Convolucional	1024	3×3	7×7
Convolucional	512	$\times 1$	7×7
Convolucional	1024	3×3	7×7
Convolucional	512	1×1	7×7
Convolucional	1024	3×3	7×7
Convolucional	1000	1×1	7×7
Avg. pool		Global	1000
Softmax			

Tabla 1.1: Arquitectura de Darknet-19 entrenada con ImageNet-1000.

YOLOv3

YOLOv3 [21] se presenta como una serie de actualizaciones a su predecesor, como cambios en la arquitectura y en las predicciones. Estas se enuncian a continuación:

- **Predicción de *bounding box*:** YOLOv3 predice un *score* de existencia de objeto (*objectness score*) para cada *bounding box* utilizando regresión logística. Este debe tomar el valor 1 si el *anchor* objetivo se superpone al *ground truth* más que cualquier otro *anchor*. Para el resto de los *anchors* si estos poseen un $IoU > 0.5$ no incurre en costo en la función de *loss*.
- **Predicción de clase:** YOLOv3 reemplaza la capa de clasificación *softmax* por clasificadores logísticos independientes por clase, lo que le permite realizar clasificación multi etiqueta.
- **Predicción multi escala:** YOLOv3 predice *bounding boxes* en 3 diferentes escalas utilizando un procedimiento similar a las *feature pyramid networks* [1]. La última de las capas convolucionales corresponde a un tensor tridimensional conteniendo tanto el *bounding box*, el *objectness* y la predicción de las clases, de tamaño $N \times N \times [3 * (4 + 1 + N_{cls})]$, donde se tienen 4 *bounding box offsets*, 1 *objectness prediction* y N_{cls} el número de clases. Luego de esto se vuelve 2 capas hacia atrás y esta se escala con factor de 2 para luego tomar un *feature map* de mayor resolución y concatenarlo con el *feature map* escalado, resultado al cual se le aplican capas convolucionales para finalmente obtener

el segundo tensor con predicciones. Este procedimiento se vuelve a realizar una vez más para predecir los *bounding boxes* en la última escala.

- ***Feature extractor:*** Se introduce la red Darknet-53, cuya arquitectura es similar a su antecesora pero más profunda y con *shorcuts connections*. Al compararse con ResNet-101 obtiene mejor desempeño además de ser 1.5 veces más rápida. En el caso de ResNet-152 posee un desempeño similar pero al doble de velocidad. La arquitectura de la red se muestra en la Tabla 1.2.

Capa	#Filtros	Tamaño/Stride	Salida
Convolucional	32	3×3	256×256
Convolucional	64	$3 \times 3/2$	128×128
Convolucional	32	1×1	
Convolucional	64	3×3	
Residual			128×128
Convolucional	128	$\times 1$ $3 \times 3/2$	64×64
Convolucional	64	1×1	
Convolucional	128	3×3	
Residual			64×64
Convolucional	256	$\times 2$ $3 \times 3/2$	32×32
Convolucional	128	1×1	
Convolucional	256	3×3	
Residual			32×32
Convolucional	512	$\times 8$ $3 \times 3/2$	16×16
Convolucional	256	1×1	
Convolucional	512	3×3	
Residual			16×16
Convolucional	1024	$\times 8$ $3 \times 3/2$	8×8
Convolucional	512	1×1	
Convolucional	1024	3×3	
Residual			8×8
Avg. pool		$\times 4$ Global	
Dense		1000	
Softmax			

Tabla 1.2: Arquitectura de Darknet-53 entrenada con ImageNet-1000.

1.3.4. RetinaNet

RetinaNet es un detector de objetos introducido por Lin et al. [12] en 2017 y que al igual que YOLO realiza las predicciones en una única pasada por la red (*one-stage detector*). Esta red se compone de dos bloques principales: una *Feature Pyramid Network* [1] y su función de *loss* propuesta (*focal loss*). Un diagrama simplificado de su arquitectura se presenta en la Figura 1.17.

Focal loss

Focal loss es una función de costo diseñada para lidiar con los problemas de desequilibrio entre clases de *background* y *foreground* durante el entrenamiento. Este desequilibrio sucede

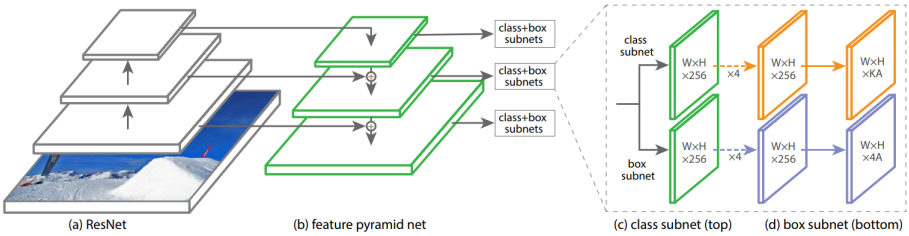


Figura 1.17: Bloques principales de arquitectura de RetinaNet.

debido a que en los sistemas de detección como YOLO o YOLOv2 (a la fecha YOLOv3 no había sido presentado) se da el hecho que en las imágenes de entrenamiento existen una gran cantidad de *anchors* por imagen (98 en el caso de YOLO y $\sim 1k$ en YOLOv2) pero muy poca cantidad de objetos. Esto si bien suele atenuarse con el uso de pesos para las clases no soluciona el problema de que el *loss* para objetos bien clasificados ($p \geq 0.5$) sea de una magnitud considerable al utilizar *cross-entropy*.

Focal loss aborda este problema de desbalanceo de clases proponiendo una función de *loss* que maximice su valor para ejemplos mal clasificados (*hard negatives*) y lo minimice para los ejemplos bien clasificados (*easy examples*). Esta función de *loss* se define formalmente como:

$$FL(p_t) = \alpha_t(1 - p_t)^\gamma \log(p_t) \tag{1.23}$$

Donde: $p_t = \begin{cases} p & \text{, si } y = 1 \\ 1 - p & \text{, si no} \end{cases}$, con p la probabilidad estimada de la clase con etiqueta $y = 1$, y α_t, γ factores de balance.

En la Figura 1.18 se muestra el comportamiento del *cross entropy* tradicional y de *focal loss* a distintos valores del factor γ. Se observa como el *loss* para muestras mal clasificadas mantienen un valor de *loss* alto en ambos casos, mientras que las muestras bien clasificadas mantienen un *loss* bajo utilizando *focal loss*.

Feature Pyramid Network

La FPN (*Feature Pyramid Network*) [1] es la columna vertebral de RetinaNet y la que realiza el trabajo de extracción de características. La FPN se aprovecha de la naturaleza piramidal de las redes convolucionales, dada por cada una de sus capas, para realizar predicciones a distintas escalas sin sacrificar rendimiento al tener una única imagen de entrada.

La idea principal de la FPN se ilustra en el Figura 1.19. La estructura base contiene una secuencia de niveles en la pirámide, donde cada nivel corresponde a una etapa de la red. Cada etapa es de la mitad del tamaño de la etapa anterior y contiene múltiples capas convolucionales de igual tamaño.

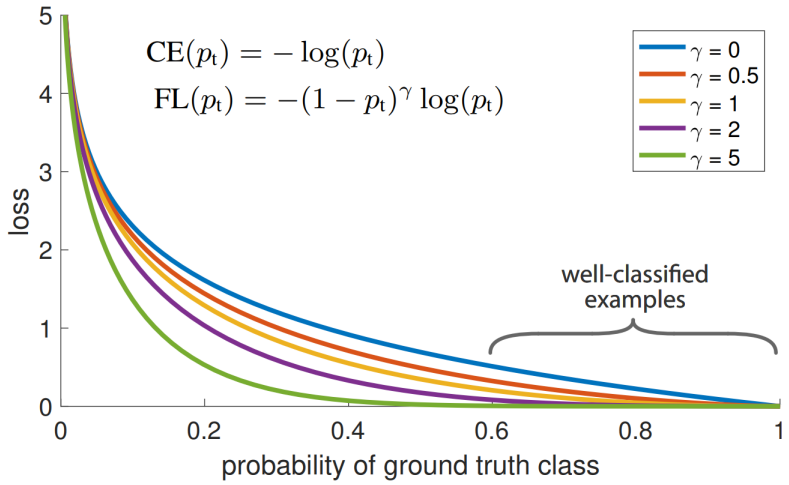


Figura 1.18: *Focal loss* y *cross-entropy* versus probabilidad.

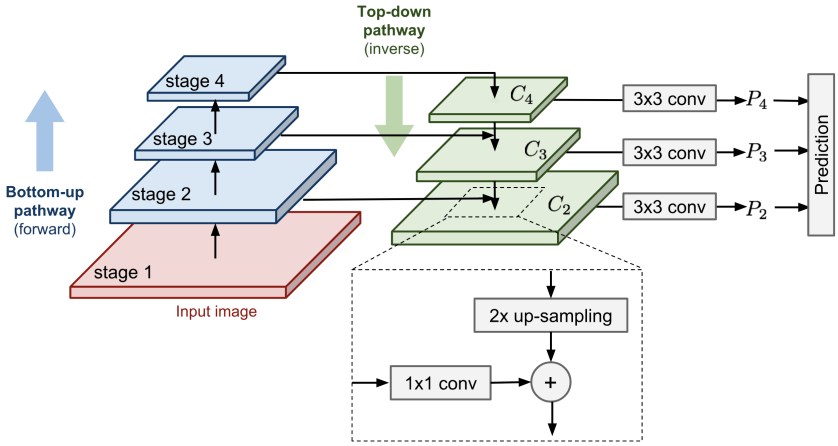


Figura 1.19: Módulo de FPN, donde C_i corresponde a la última capa convolucional de la etapa i.

Estas conexiones entre etapas se describen a continuación:

- ***Bottom-up*:** Corresponde a la conexión típica de una red convolucional entre sus capas (*forward computation*). La última capa C_i de cada etapa i es tomada como la referencia para esta. Para el caso de la red ResNet se toma esta capa como la última de cada bloque residual.
- ***Top-down and lateral connections*:** Va en el sentido inverso y es la responsable de añadir mayor información semántica entre capas. Se obtienen de la siguiente forma:
  1. La última capa es aumentada al doble de su tamaño.
  2. A la última capa de la etapa anterior se le reduce su dimensión aplicando una convolución de 1×1.
  3. Ambos *feature maps* obtenidos son sumados elemento a elemento para obtener el *feature* resultante.

Arquitectura

La arquitectura general de RetinaNet se encuentra compuesta por la red extractora de características (*backbone*) seguida de dos sub-redes cada una encargada de la clasificación y de la predicción de *bounding* boxes respectivamente (ver Figura 1.17). De *backbone* se utiliza FPN construida sobre una ResNet al igual que en [1].

Similar a Faster R-CNN y YOLOv3, RetinaNet utiliza *anchors* donde se escogen un total de $A = 9$ por cada nivel de la pirámide. Cada uno de los *anchors* es asignado a un vector *one hot* de largo K igual al número de clases y un vector de tamaño 4 para el *bounding box regressor*. Se utiliza una regla similar a la red RPN para asignar la pertenencia de los *anchors*, donde el *anchor* con $IoU \geq 0.5$ con algún *ground truth box* es asignado a esa clase mientras que es asignado como *background* si posee un IoU en el intervalo $[0, 0.4)$. En el caso de tener IoU en el intervalo $[0.4, 0.5)$ es ignorado.

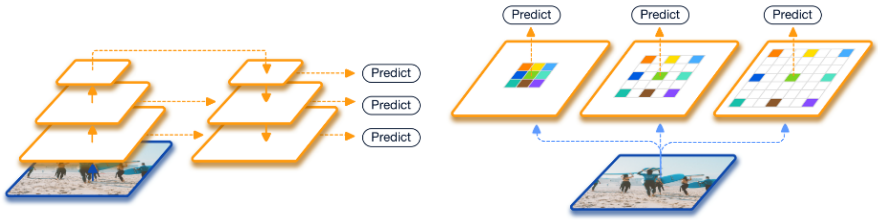
Para la sub-red de clasificación se utiliza una red convolucional del tipo FCN la cual predice a cada nivel de la FPN, donde los parámetros son compartidos a través de estos. Tomando como entrada un *feature map* con C canales (profundidad) de algún nivel de la FPN, la red aplica cuatro capas convolucionales de 3×3 de C filtros seguido de la función de activación ReLU. La salida final esta dada por una capa convolucional con $K \cdot A$ filtros con función de activación sigmoide. Se utiliza $C = 256$ y $A = 9$ como parámetros para los experimentos.

Para la sub-red de *box regression* se utiliza una red con igual arquitectura que la sub-red de clasificación salvo por la capa de salida la cual está dada por $4 \cdot A$ canales. Esta salida entrega, para cada *anchor*, el vector con las 4 coordenadas parametrizadas utilizando la misma parametrización que en Faster R-CNN.

1.3.5. TridentNet

TridentNet [13] surge como una respuesta a la problemática de las variaciones de escalas de los objetos en los modelos de detección, lo que genera que estos tengan problemas en detectar objetos pequeños o muy grandes. Si bien esto había sido abordado con FPN y con detecciones a múltiple escala como en YOLOv3, los *features* eran extraídos en diferentes niveles de la red y por lo tanto con diferentes parámetros, lo que genera una alternativa aunque de bajo costo computacional, insatisfactoria a las pirámides de imágenes. Es por esto que al contrario de FPN, TridentNet genera predicciones a distintas escalas con iguales parámetros pero distintos campos receptivos, como se muestra en la Figura 1.20.

La operación utilizada por TridentNet y la cual le permite generar *features* a distintas escalas corresponde a la *dilated convolution* (Atrous *convolution*) [22], la cual es a una adaptación de la convolución clásica donde el filtro es "dilatado" generando un mayor campo receptivo de este. Este comportamiento se ejemplifica en la Figura 1.21 y se formaliza con la fórmula:

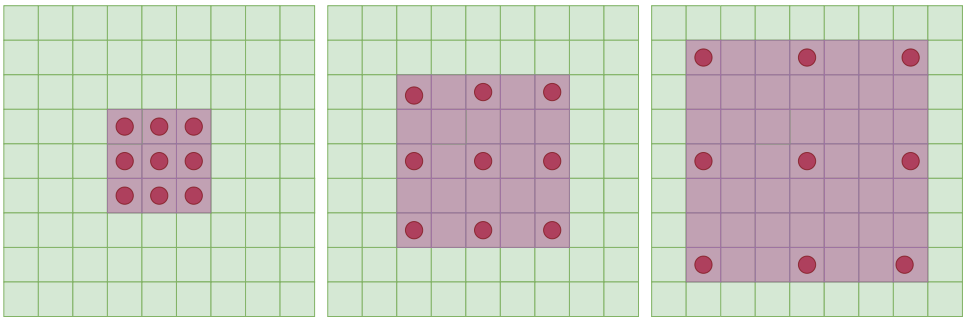


(a) Arquitectura de FPN. (b) Bloque de TridentNet.

Figura 1.20: Comparacion TridentNet con FPN.

$$(F *_l k)(\mathbf{p}) = \sum_{s+l\mathbf{t}=\mathbf{p}} F(s)k(\mathbf{t}) \tag{1.24}$$

Donde con $l = 1$ se obtiene la convolución clásica.



(a) Factor de dilatación $l = 1$. (b) Factor de dilatación $l = 2$. (c) Factor de dilatación $l = 3$.

Figura 1.21: *Dilated convolution*. La imagen es representada por los cuadros verdes, mientras que el filtro se representa por los cuadros y puntos rosas. Los lugares sin puntos son reemplazados por ceros.

Utilizando esta operación se construyen los *trident blocks*, los cuales corresponden a bloques compuestos de tres ramas cuyos parámetros son compartidos pero poseen distintos factores de dilatación, uno para cada escala objetivo, como se muestra en la Figura 1.22. Esto tiene como fin generar *feature maps* específicos para cada escala, donde los parámetros también son compartidos entre las redes RPN y R-CNN de cada rama.

Para evitar que objetos sean detectados por la rama incorrecta (objetos pequeños en la rama con con alta dilatación) lo que degrada el desempeño del detector, es que se genera un rango de aceptación $[l_\text{i}, u_\text{i}]$ para cada rama i. Para cada *RoI* de tamaño $w \times h$ en la imagen original, este es válido para la rama i cuando:

$$l_\text{i} \leq \sqrt{wh} \leq u_\text{i} \tag{1.25}$$

Esto se utiliza en el entrenamiento y es aplicado tanto para la red RPN como para la red R-CNN. En inferencia en cambio, se generan resultados para todas las ramas para luego filtrar los *bounding boxes* que caen fuera del rango de la rama.

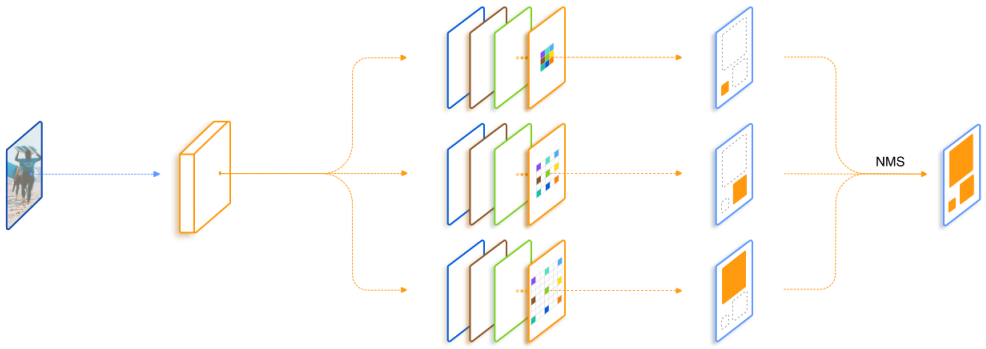


Figura 1.22: Ilustración de TridentNet. Las múltiples ramas en los *trident blocks* comparten los mismos parámetros pero con distinto factor de dilatacion. Luego las predicciones realizadas por cada rama se evalúan en conjunto utilizando *non maximum supression*.

Con esto se construye una red TridentNet utilizando como base otro detector. En este caso particular se construye TridentNet en base a Faster R-CNN, con una red ResNet como *backbone* en donde se añaden un total de 10 *trident blocks* en la etapa 4 de la red. La cantidad de bloques y la etapa elegida fue escogida en base a experimentación, donde esta combinación obtuvo el mejor desempeño. Para la construcción del *trident block* en el bloque residual se utiliza el esquema que se presenta en la Figura 1.23.

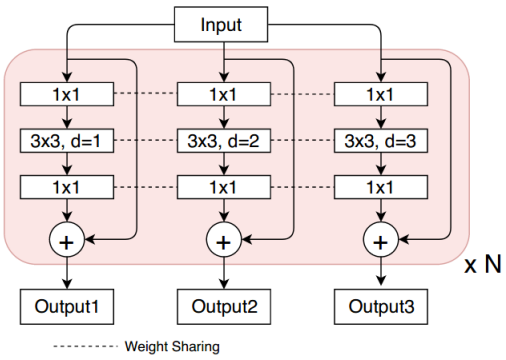


Figura 1.23: *Trident block* aplicado sobre bloque residual.

1.4. Métricas de desempeño

Para evaluar el desempeño de los modelos de detección suele utilizarse la métrica introducida por COCO [23], que corresponde al cálculo de *average precision* a distintos valores de

IoU. Para explicar esta métrica es necesario introducir algunos conceptos:

- **Intersection Over Union (IoU):** IoU es la proporción entre el área de la intersección de los *bounding boxes* (predicho y *ground truth*) sobre el área de la unión de estos. Se utiliza para determinar que tan precisa es la detección y para el posterior cálculo del *average precision*. Se calcula como se indica en la Figura 1.24.

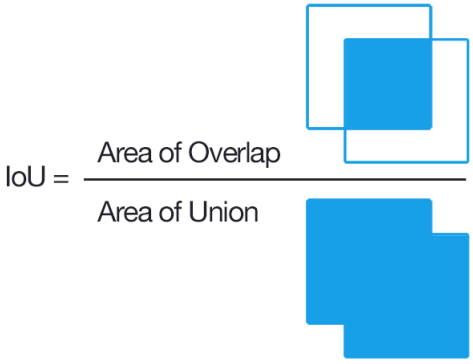


Figura 1.24: Cálculo de *Intersection Over Union*

- ***Precision*:** Mide qué tan precisas son las detecciones, es decir el porcentaje de predicciones que son correctas. Se calcula como:

$$Precision = \frac{TP}{TP + FP} = \frac{TP}{\text{total de detecciones}} \tag{1.26}$$

- ***Recall*:** Mide qué tan bien se predicen las instancias relevantes a las consultas. Se calcula como:

$$Recall = \frac{TP}{TP + FN} = \frac{TP}{\text{total de } ground\ truths} \tag{1.27}$$

Para el cálculo de los valores de TP,FP, y FN, se utiliza un umbral para el valor de IoU asociado entre los *bounding boxes*. Con este umbral IoU^{tr} se define:

- ***True Positive* (TP):** Verdadero positivo, ocurre cuando el IoU entre los *bounding boxes* (predicho y *ground truth*) es mayor a IoU^{tr} y las clases concuerdan.
- ***False Positive* (FP):** Falso positivo, ocurre cuando el IoU entre los *bounding boxes* (predicho y *ground truth*) es menor a IoU^{tr}; o este es mayor, pero existe una detección con mayor *confidence score* y $\text{IoU} > \text{IoU}^{tr}$ anotada com TP.
- ***False Negative* (FN):** Falso negativo, ocurre cuando el IoU entre los *bounding boxes* (predicho y *ground truth*) es mayor a IoU^{tr} pero las clases no concuerdan.
- ***True Negative* (TN):** No es utilizado ya que se asume que las imágenes tienen al menos un objeto en ellas.

Con estos conceptos introducidos es posible definir el cálculo del *average precision*. En el caso tradicional este cálculo esta dado por la fórmula:

$$AP = \frac{1}{\#\text{resultados relevantes}} \sum_{k=1}^{n} P(k) \cdot \text{rel}(k) \tag{1.28}$$

Donde la cantidad de resultados relevantes son los documentos recuperados que corresponden a la misma clase de la consulta, n es la cantidad de resultados requeridos por consulta, $P(k)$ es el valor de *precision* en la posición $k \in [1, n]$, y $\text{rel}(k)$ es una función binaria indicando si el resultado en la posición k es o no relevante para la consulta.

En el caso del AP utilizado por COCO se reemplaza *precision* en la posición k por *interpolated precision*, definida para el *PASCAL VOC Challenge* [17] como:

$$p_{interp} = \max_{\tilde{r}:\tilde{r} \geq r} p(\tilde{r}) \tag{1.29}$$

Además, en vez de la utilización de K consultas relevantes, se utilizan 101 valores de *recall* en el intervalo $[0, 1]$. Finalmente, el cálculo de *average precision* queda definido por:

$$AP^{\text{COCO}} = \frac{1}{101} \sum_{r \in \{0, 0.01, 0.02, .., 1\}} p_{interp}(r) \tag{1.30}$$

Para la obtención de cada uno de los valores de *precision* en el cálculo de AP se procede como sigue:

- Se obtienen las detecciones de todas las imágenes en el conjunto a testear, las cuales se ordenan de mayor a menor según el valor de *confidence score* asociado.
- A cada detección se le calcula el IoU con las anotaciones *ground truth* disponibles por imagen. Si el valor de IoU cumple que $\text{IoU} \geq \text{IoU}^{tr}$ y la clase concuerda es anotado como TP. Cabe destacar que solo existe un TP por *bounding box*.
- A cada detección se le calcula su *recall* asociado utilizando *interpolated precision*, según su posición en la lista de detecciones.

Con estos pasos se puede obtener la curva de *precision-recall*, la cual es de bastante utilidad para evaluar la calidad de las detecciones. Un ejemplo de esta curva, utilizando 11 puntos en el *recall*, se observa en la Figura 1.25.

Es con este cálculo de AP que se definen las siguientes métricas utilizadas por COCO:

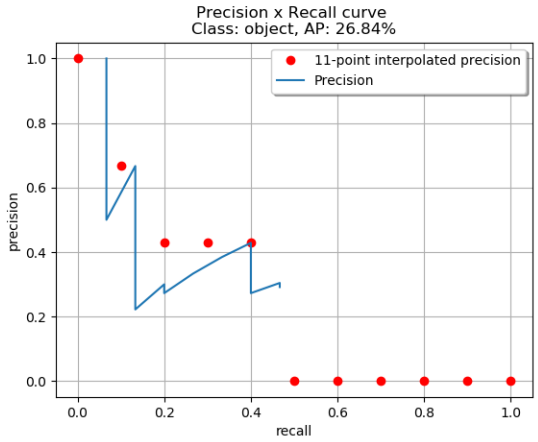


Figura 1.25: Ejemplo de curva *precision-recall*, obtenida de `https://github.com/rafaelpadilla/Object-Detection-Metrics`. Se utilizan 11 puntos en *recall*.

- AP: Utilizando la definición presentada en la Ecuación 1.30, se calcula el *average precision* a distintos valores de IoU^{tr}, para luego ser promediados. Estos valores de IoU^{tr} se consideran en el intervalo $[0.5, 0.95]$ a un paso de 0.05. Esta es la métrica por defecto para la comparación entre modelos.
- AP_{50}: Se calcula el *average precision* utilizando $\text{IoU}^{tr} = 0.5$.
- AP_{75}: Se calcula el *average precision* utilizando $\text{IoU}^{tr} = 0.75$.

Como consecuencia de estos cálculos, es preferible tener detecciones a todo nivel de *recall* para obtener un alto valor de AP. Esto incurre en un problema al preferenciar modelos con alta cantidad de detecciones con valores bajos de *confidence score*, en desmedro de modelos con pocas pero precisas detecciones.

Debe considerarse que para COCO, AP (*average precision*) y mAP (*mean average precision*) poseen el mismo significado en el contexto de detección de objetos[1]. No obstante, y para evitar confusiones, en este trabajo se utilizará mAP para referirse al cálculo de *average precision* a distintos valores de IoU y a través de todas las clases, AP para referirse al caso de una única clase, y AP_{50} y AP_{75} para referirse al cálculo de *average precision* a un valor fijo de IoU^{tr} (0.5 y 0.75 en este caso) a través de las distintas clases.

1.5. Estado del arte

El problema de detección de vestimenta ha sido abordado a lo largo del tiempo tanto con algoritmos clásicos de extracción de características, como también en la última década con redes neuronales profundas (*deep learning*).

Para el caso de algoritmos clásicos se puede mencionar el trabajo de Ming Yang et. al [24], en donde se utilizan algoritmos de extracción de características, tales como HOG [5] o SIFT

[1] `http://cocodataset.org/#detection-eval`

[25], para clasificar vestimentas utilizando una máquina de soporte vectorial (SVM), mientras que para la detección de estas se utiliza una aproximación dada la detección del rostro de la persona. Esta detección carece de sofisticación al depender completamente de la detección del rostro, por lo cual pierde sentido en imágenes sin personas. Sumado a esto, asume que la vestimenta de la persona se compone por superior e inferior, ignorando las vestimentas largas como los vestidos.

Por el lado de los algoritmos de *deep learning*, se puede mencionar el trabajo de Camila Álvarez [26], en donde entrena modelos de detección de objetos como Faster R-CNN y YOLOv2, para su posterior comparación y utilización para el auto-etiquetado de imágenes de vestimenta y para su uso en *clothing retrieval*. En el trabajo de Camila se entrenan los modelos con alrededor de 150000 imágenes obtenidas mediante *crowd sourcing*. Su trabajo presenta una similitud con el presente trabajo, debido a que compara el desempeño de modelos de *object detection* en el contexto de vestimenta, sin embargo esta comparación se limita a Faster R-CNN y YOLOv2.

En la misma línea de lo anterior, puede mencionarse el trabajo de Kucer et. al [27], en donde se entrena Mask R-CNN en el *dataset* ModaNet para su posterior uso en *clothing retrieval*, en donde se utiliza una red convolucional para la búsqueda por similitud. Esta aplicación puede verse como una de las posibles extensiones al presente trabajo, donde se utiliza la detección obtenida como información vital para la posterior recuperación de información.

Una interesante aplicación realiza Wang-Cheng Kang et. al [28], en donde se recomiendan vestimentas o productos de moda dado la vestimenta de la persona y la escena general de la imagen. Para esto, utilizan una red convolucional con el fin de extraer características globales de la imagen, para luego generar la recomendación. Si bien en este trabajo no se realiza detección, esta podría ayudar en las recomendaciones al localizar en forma precisa las prendas utilizadas en la escena.

Capítulo 2

Metodología

En este capitulo se detallará la metodología de trabajo aplicada para llevar a cabo el presente trabajo. En ella se especificará la configuración experimental utilizada, la que contempla los modelos y sus configuraciones, el *hardware utilizado*, los *datasets* utilizados y sus propiedades, y finalmente el uso de los *frameworks* de entrenamiento de los modelos. Se presenta también la metodología llevada a cabo para la aplicación de *clothing retrieval*.

Primeramente se especifican los modelos que se utilizarán en el entrenamiento y su posterior evaluación, donde se especifica el *backbone* a utilizar por cada uno de ellos. Seguido de esto, se especifican los *frameworks* en donde se entrenarán estos modelos, para luego precisar el entorno de pruebas (*hardware*) en donde se desarrolla íntegramente el presente trabajo.

Una vez introducidos los modelos y su *frameworks* de entrenamiento, se caracterizan los conjuntos de datos (*datasets*) a utilizar para este. Estos conjuntos de datos resultan clave ya que de ellos dependen el desempeño final de los modelos en un ambiente real.

Para un correcto uso de los *datasets*, deben prepararse de acuerdo al *framework* de entrenamiento a utilizar, por lo que se detalla el cómo se preparan cada uno de ellos. También se detallan los pasos necesarios para el correcto entrenamiento de los modelos.

Finalmente se especifica la estrategia utilizada para la aplicación de *clothing retrieval*.

2.1. Configuración experimental

2.1.1. Modelos de detección

Se entrenarán los modelos presentados en las Sección 1.3. La elección de estos modelos radica en la variedad de técnicas utilizadas para la detección: mientras que RetinaNet y YOLOv3 realizan la detección del *bounding box* y de la clase en la misma pasada; Faster R-CNN, Mask R-CNN y TridentNet la realizan en dos pasos, donde se generar primero las detecciones y posteriormente se clasifica el objeto.

En la Tabla 2.1 se muestran los *backbones* utilizados para los distintos modelos.

Backbones modelos	
Modelo	Backbone
YOLOv3	Darknet-53
RetinaNet	ResNet-50-FPN
Faster R-CNN	ResNet-50-FPN
Mask R-CNN	ResNet-50-FPN
TridentNet	ResNet-50-C4

Tabla 2.1: *Backbones* de modelos utilizados. ResNet-50-FPN indica una ResNet-50 construida con una *Feature Pyramid Network* [1], mientras que ResNet- 50-C4 indica que las características se extraen de la cuarta etapa de la red.

2.1.2. *Frameworks* de entrenamiento

Para el entrenamiento de los modelos de detección de objetos se utilizan *frameworks* que entregan las herramientas necesarias para este fin. Se utilizan librerías *open source*.

Para los modelos Faster R-CNN, Mask R-CNN y RetinaNet, pertenecientes al área de investigación de inteligencia artificial de Facebook (Facebook AI Research's, FAIR), se utiliza el *framework* Detectron [29] (originalmente en *Caffe* [30]) en su adaptación a Pytorch [31] llamada `maskrcnn-benchmark` [32].

Para YOLOv3 se utiliza Darknet [33], librería de redes neuronales en lenguaje `C` creada por los mismos autores, utilizándola mediante un *fork* de *AlexeyAB* [1].

En el caso de TridentNet se utiliza SimpleDet [34], *framework* de entrenamiento de modelos de detección de objetos, donde sus principales contribuyentes son los mismos autores de TridentNet.

2.1.3. Entorno de pruebas

Para los experimentos y resultados obtenidos se utiliza la configuración de *hardware* que se muestra en la Tabla 2.2, donde el *hardware* de entrenamiento es utilizado para el entrenamiento de los modelos y corresponde a una instancia de maquina virtual en la nube, y el *hardware* de inferencia es utilizado para la obtención de los resultados e imágenes. Es importante recalcar la importancia de utilizar GPU debido a su capacidad de paralelismo que permite entrenar los modelos hasta 100 veces más rápido que en CPU. Cabe destacar que cada uno de los entrenamiento de los modelos demora alrededor de 20 horas.

[1] `https://github.com/AlexeyAB/darknet`

Hardware utilizado		
Componente	Entrenamiento	Inferencia
CPU	Intel Skylake 2-vCPU	AMD Ryzen 5 1600 3.2 GHz
GPU	Nvidia K80 12GB	Nvidia GTX 1070 8GB
RAM	12GB Ram DDR4	16GB Ram DDR4
Almacenamiento	Disco SSD	Disco SSD

Tabla 2.2: *Hardware* utilizado para la realización de experimentos.

2.2. Conjunto de datos

Para realizar el entrenamiento de los modelos de detección de objetos es necesaria la utilización de un conjunto de datos (*dataset*) apropiado al problema, que contenga una variada cantidad de imágenes junto con sus anotaciones correspondientes. Se decide utilizar en el contexto de detección de vestimenta los *datasets DeepFasion2* [35] y ModaNet [36].

2.2.1. DeepFashion2

Conjunto de datos de moda que contiene 491000 imágenes con un total de 801.000 prendas repartidas entre 13 categorías distintas, entre fotografías de catálogo de moda y de fotos comunes de usuarios. Estas categorías corresponden a: *short sleeve top, long sleeve top, short sleeve outwear, long sleeve outwear, vest, sling, shorts, trousers, skirt, short sleeve dress, long sleeve dress, vest dress, sling dress*. El *dataset* se encuentra dividido en conjunto de entrenamiento (391k imágenes), conjunto de validación (34k imágenes), y conjunto de test (67k imágenes). Para el caso particular del conjunto de entrenamiento es posible acceder únicamente a la mitad de los datos disponibles [2].

Para acceder a los datos es necesario enviar un formulario[3] de petición por el cual facilitan la contraseña de los archivos a descomprimir. Estos archivos contienen tanto las imágenes como las anotaciones y tienen un tamaño de 10Gb para el conjunto de entrenamiento, 2Gb para el conjunto de validación y 2Gb para el conjunto de test.

Cada imagen del conjunto de datos posee un `ID` único de 6 dígitos el cual identifica la imagen y el archivo `.json` que posee las anotaciones correspondientes a esta. Así, una imagen con `ID` = 000001 tiene como imagen el archivo `000001.jpg` y un archivo de anotaciones `000001.json`, con la siguiente información:

- **source:** `String` que indica el origen de la imagen. Puede ser `shop` o `user`, lo que indica si es de comercio o tomada por usuarios respectivamente.
- **pair_id:** Número que es igual en imágenes de `shop` y `user` que corresponden a la misma vestimenta.
- **item_1:** Ítem de vestimenta con la siguiente información:

[2] `https://github.com/switchablenorms/DeepFashion2/issues/29`
[3] `https://bit.ly/35ncyHU`

  - **category_name:** `string` que indica la categoría del ítem.
  - **category_id:** número correspondiente con su categoría. Van desde el 1 al 13, donde el 1 corresponde a *short sleeve top* y el 13 a *sling dress*, pasando por todas las categorías.
  - **style:** número usado para distinguir vestimentas con igual **pair_id** pero con diferencias en color, estampado o logo. En consecuencia un ítem cuya imagen posea igual **pair_id** e igual **style** corresponde a la misma vestimenta en distintas imágenes.
  - **bounding box:** corresponde al *bounding box* que encierra el ítem. El formato utilizado es $[x_1, y_1, x_2, y_2]$, donde (x_1, y_1) corresponden a las coordenadas de la esquina superior izquierda, mientras que (x_2, y_2) a la esquina inferior derecha.
  - **landmarks:** puntos de referencia del ítem.
  - **segmentation:** polígono que encierra el ítem.
  - **scale:** número indicador de pequeña escala, mediana escala y larga escala.
  - **occlusion:** número indicador de nivel de oclusión.
  - **zoom_in:** número indicador de nivel de zoom.
  - **viewpoint:** número indicador de punto de vista del ítem.
- item_n: ...

Es importante señalar que en este trabajo de detección de objetos solamente se utiliza el *bounding box* del ítem, ignorando los *landmarks* y la segmentación. En la Tabla 2.3 se muestran el número de instancias por clase tanto en el conjunto de entrenamiento como en el de validación.

Número de instancias DeepFashion2		
Clase	#Train	#Valid
short sleeve top	71645	12556
long sleeve top	36064	5966
short sleeve outwear	543	142
long sleeve outwear	13457	2011
vest	16095	2113
sling	1985	322
shorts	36616	4167
trousers	55387	9586
skirt	30835	6522
short sleeve dress	17211	3127
long sleeve dress	7907	1477
vest dress	17949	3352
sling dress	6492	1149
Total	312186	52490

Tabla 2.3: Estadísticas *dataset* DeepFashion2.

En la Figura 2.1 se muestran algunas imágenes del *dataset*.

Figura 2.1: Ejemplos de imágenes DeepFashion2.

2.2.2. ModaNet

Conjunto de datos de moda que contiene 55176 imágenes con anotaciones útiles para aplicaciones en detección y segmentación de vestimentas. Las imágenes se encuentran repartidas en 13 clases correspondientes a: *bag, belt, boots, footwear, outer, dress, sunglasses, pants, top, shorts, skirt, headwear, scarf/tie*. Estas anotaciones están basadas en el *dataset* PaperDoll [4], donde la contribución realizada por ModaNet corresponde a añadir segmentaciones a las vestimentas.

Para acceder a la base de datos es necesario descargar el *dataset* de PaperDoll el cual tiene un tamaño aproximado de 40Gb, sin embargo las anotaciones solo están disponibles para un subconjunto de imágenes con un tamaño aproximado de 2Gb. Las anotaciones se descargan desde el repositorio de ModaNet en formato `.json` con un tamaño de 100Mb. Por razones desconocidas las anotaciones de validación y test nunca fueron subidas a la web, por lo que solo se dispone de las anotaciones de entrenamiento.

Las anotaciones se encuentran en formato COCO [23], donde la información importante utilizada en el presente trabajo corresponde al ítem `bbox`, el cual se utiliza para el entrenamiento de los modelos de detección de objetos. En la Tabla 2.4 se muestran el número de instancias por clase del *dataset* y en la Figura 2.2 se muestran algunos ejemplos de imágenes.

[4] `https://github.com/kyamagu/paperdoll/tree/master/data/chictopia`

Número de instancias ModaNet		
Clase	#Train	#Valid
bag	18747	2128
belt	12933	1423
boots	12562	1397
footwear	69398	7759
outer	22680	2528
dress	13114	1478
sunglasses	7913	884
pants	20886	2287
top	32701	3588
shorts	6361	714
skirt	12277	1377
headwear	4933	515
scarf/tie	4590	526
Total	239095	26604

Tabla 2.4: Estadísticas *dataset* ModaNet.

Figura 2.2: Ejemplos de imágenes ModaNet.

2.3. Preparación de los datos

Para el correcto uso de los *frameworks* de entrenamiento de los modelos es necesario preparar los datos en el formato adecuado para cada uno de ellos. Se muestra a continuación el procedimiento para ello.

2.3.1. Darknet

La preparación de los datos en el *framework* Darknet consta de una serie de pasos para su correcta lectura. Para empezar, por cada imagen en la base de datos se debe crear un archivo `.txt` con la información de los objetos presentes la imagen. Estos archivos deben estar en el mismo directorio de las imágenes y su contenido debe estar en el formato:

```
    <cls_id>  <x_mid>  <y_mid>  <widht>  <height>
```

Donde:

- `cls_id` corresponde al `id` asociado a la clase, el cual va desde 0 hasta `num_cls - 1`.
- `<x_mid> <y_mid>` corresponde al centro del *bounding box* relativo a las dimensiones de la imagen, por lo que toma valores en el conjunto $(0, 1]$. Se calculan como:

$$x_{\text{mid}} = \frac{x_2 + x_1}{2} \cdot \frac{1}{W} \qquad\qquad y_{\text{mid}} = \frac{y_2 + y_1}{2} \cdot \frac{1}{H} \tag{2.1}$$

- `<widht> <height>` dimensiones del *bounding box* relativos a la imagen.

Luego, si una imagen `img1.jpg` de tamaño 600×400 posee dos objetos en las posiciones $(100, 100, 200, 200)$ y $(100, 300, 400, 400)$, de la clase 0 y 1 respectivamente, el archivo `img1.txt` contendrá lo siguiente:

```
0 0.25 0.375 0.16667 0.25
1 0.83333 0.875 0.5 0.25
```

Sumado a esto, debe crearse un archivo `train.txt` que indique los directorios a todas las imágenes que se utilizarán en el entrenamiento. Si las imágenes se encuentran en el directorio `data`, este archivo debería lucir como:

```
data/img1.jpg
data/img2.jpg
data/img3.jpg
```

Este mismo procedimiento debe realizarse con el conjunto de test o de validación.

2.3.2. Detectron y SimpleDet

Para los *frameworks* `Detectron` y `SimpleDet` es necesario tener los anotaciones en el formato COCO, el cual para detección de objetos debe seguir la siguiente estructura escrita en un archivo `.json`:

```
{
"info": info, "images": [image], "annotations": [annotation], "licenses": [license],
}

info{
"year": int, "version": str, "description": str,
"contributor": str, "url": str, "date_created": datetime,
}

image[{
"id": int, "width": int, "height": int, "file_name": str,
"license": int, "flickr_url": str, "coco_url": str, "date_captured": datetime,
}]

license{
"id": int, "name": str, "url": str,
}

annotation[{
"id": int, "image_id": int, "category_id": int, "segmentation": RLE or [polygon],
"area": float, "bbox": [x,y,width,height], "iscrowd": 0 or 1,
}]

categories[{
"id": int, "name": str, "supercategory": str,
}]
```

2.4. Entrenamiento de los modelos

Se presenta en esta sección los pasos para el entrenamiento de los modelos utilizando los *frameworks* mencionados.

2.4.1. Darknet

Para el uso del *framework* Darknet es necesario generar tres archivos de configuración tanto para el *dataset* como para la red. Estos archivos corresponden a:

- `yolov3.cfg`: Corresponde al archivo que instancia la red YOLOv3 en el *framework* Darknet. En este archivo se detallan todas las capas y filtros de la red junto con los parámetros de entrenamiento. Es aquí donde deben configurarse el número de filtros en las capas `[yolo]`, según la cantidad de clases del *dataset*.
- `obj.names`: Archivo que contiene las etiquetas de las clases de entrenamiento. Debe tener el siguiente formato:

```
class_1
class_2
class_3
```

- `obj.data`: Contiene los directorios de los archivos utilizados para entrenar junto con el directorio donde se guardarán los pesos del entrenamiento. Su formato es:

```
classes = <num_classes>
train  = data/train.txt
valid  = data/test.txt
names = data/obj.names
backup = weights/
```

Una vez configurado estos archivos es necesario descargar los pesos pre-entrenados en ImageNet para su utilización como extractor de características de YOLOv3. Obtenido esto, se da inicio al entrenamiento con el comando por `bash`:

```
./darknet detector train <data_file> <cfg_file> <pretrained_weights>
```

2.4.2. Detectron

Antes de empezar con el entrenamiento de los modelos es necesario editar algunos archivos del *framework* para adaptarlo al *dataset* a utilizar. El primer paso corresponde a crear una clase del *dataset* que extienda desde la clase `torchvision.datasets.coco.CocoDetection`, lo que permite utilizar la API de COCO para Python [5] para evaluar los modelos. Esta clase debe crearse en el directorio `maskrcnn-benchmark/maskrcnn_benchmark/data/datasets` y ser agregada al archivo `__init__.py` del mismo directorio. Sumado a esto deben añadirse los directorios del *dataset* y del archivo `.json` con anotaciones al archivo `paths_catalogs.py`, ubicado en `maskrcnn_benchmark/config`.

Para la creación de la instancia del modelo es que se utilizan archivos de configuración en formato `.yaml`. Estos archivos vienen pre-configurados para los distintos modelos presentes en el *framework*, y deben ser configurados para establecer los parámetros de entrenamiento.

Uno de los parámetros importantes en el entrenamiento corresponde a la cantidad de imágenes por *batch*, el cual esta dado por la variable `IMS_PER_BATCH`, donde debe tomar un valor igual a 1 o 2 si se entrena el modelo con una única GPU. Esto se debe a que el proceso de entrenamiento requiere una gran cantidad de memoria de la GPU, tanto para almacenar las salidas intermedias de la red, como también los pesos y el cálculo del gradiente para cada uno de ellos.

Uno de los últimos pasos es descargar los pesos pre-entrenados en COCO y utilizarlos para el nuevo *dataset* con la técnica de *transfer learning*. Para esto es necesario cargar el modelo pre-entrenado y reemplazar las capas de clasificación y regresión por unas adecuadas al *dataset* y a su cantidad de clases. Esto se realiza con el siguiente fragmento de código:

```
def removekey(d, listofkeys):
    r = dict(d)
    for key in listofkeys:
```

[5] https://github.com/cocodataset/cocoapi/tree/master/PythonAPI

```
        r.pop(key)
    return r

pretrained_path = 'path_to_pretrained_weights'
DETECTRON_PATH = os.path.expanduser(pretrained_path)

_d = torch.load(DETECTRON_PATH)
_d.pop('optimizer')
_d.pop('scheduler')
_d.pop('iteration')
newdict = _d

newdict['model'] = removekey(_d['model'],
                             ['module.roi_heads.box.predictor.cls_score.bias',
                              'module.roi_heads.box.predictor.cls_score.weight',
                              'module.roi_heads.box.predictor.bbox_pred.bias',
                              'module.roi_heads.box.predictor.bbox_pred.weight'])

torch.save(newdict, path_to_save_new_weights)
```

Finalmente la red puede ser entrenada con el comando por bash:

```
python tools/train_net.py --config-file "configs/config_file.yaml"
```

Donde en el argumento `--config-file` se indica el directorio del archivo de configuración de la red.

2.4.3. SimpleDet

Teniendo el *dataset* junto con sus anotaciones, el primer paso para la correcta utilización del *framework* `SimpleDet` corresponde en editar el archivo de configuración del modelo a conveniencia. Este archivo viene en formato `.py` y corresponde a un serie de clases que definen al modelo y sus parámetros de entrenamiento.

Realizado esto, el siguiente paso es descargar los pesos pre-entrenados para realizar *transfer learning*. Esto se realiza re-inicializando las capas de clasificación y regresión con el siguiente fragmento de código:

```
import mxnet as mx
params = mx.nd.load("pretrained_coco_weights-0000.params")
del params["arg:bbox_cls_logit_weight"]
del params["arg:bbox_cls_logit_bias"]
del params["arg:bbox_reg_delta_weight"]
del params["arg:bbox_reg_delta_bias"]
mx.nd.save("pretrained_weights_to_train-0000.params", params)
```

Finalmente es posible iniciar el entrenamiento de la red con el siguiente comando por `bash`:

```
python detection_train.py --config config/config_file.py
```

Donde en el argumento `--config` se indica el directorio del archivo de configuración de la red.

2.5. Aplicación en *Retrieval*

Se desea aplicar *clothing retrieval* sobre las vestimentas detectadas en base a un modelo de detección de objetos a elección. Para esto, se utiliza el *dataset* DeepFashion2 ya que posee fotos de vestimentas tanto en imágenes capturadas por tiendas como en imágenes capturadas por usuario vistiendo las prendas. Sumado a esto, DeepFashion2 pone a disposición un código para evaluar la calidad del *retrieval*. En la Figura 2.3 se muestra un ejemplo de *clothing retrieval*.

Figura 2.3: Ejemplo de aplicación de *retrieval* en DeepFashion2.

La estrategia a utilizar consiste en obtener un vector de características de cada una de las prendas detectadas por el modelo de detección de objetos, para luego realizar una búsqueda sobre el conjunto de vectores, previamente calculados, con el fin de encontrar al más similar según un criterio arbitrario. Estos pasos se representan gráficamente en la Figura 2.4 y se detallan a continuación:

1. Se utiliza un modelo de detección de objetos para detectar las vestimentas presentes en la imagen.
2. Para cada una de estas vestimentas se recorta la imagen utilizando su *bounding box* correspondiente. Este recorte se utiliza como entrada a una red neuronal convolucional la cual se encarga de extraer un vector de características de la prenda.
3. Utilizando este vector de características se realiza una consulta sobre los vectores presentes en el conjunto, los cuales fueron previamente obtenidos y almacenados en disco. En esta consulta se obtienen los vectores más similares (más cercanos) mediante una función de similitud.

4. Obtenidos los vectores más similares se utiliza su `img_id` y su *bounding box (ground truth)* correspondiente para generar el resultado final de la consulta, en donde se indiquen las prendas encontradas.

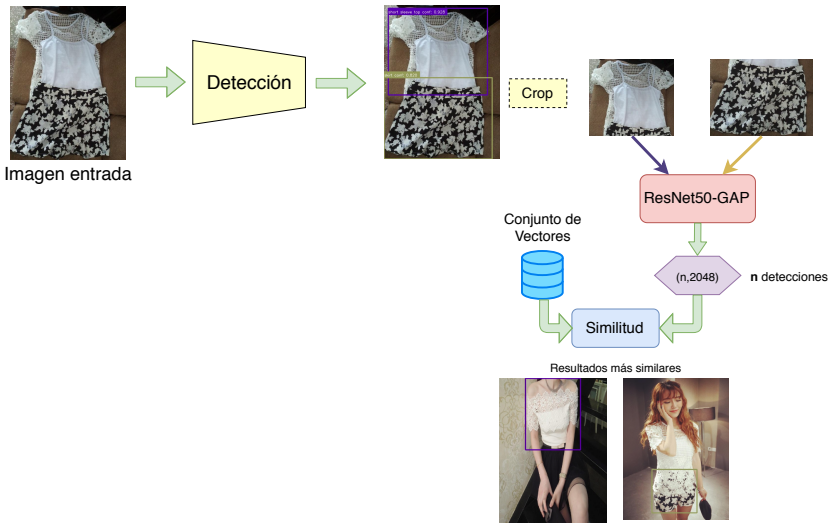


Figura 2.4: Esquema de aplicación de *retrieval*.

La red neuronal utilizada para la extracción de características corresponde a una ResNet-50 pre-entrenada en ImageNet, utilizando la capa anterior a la capa de salida y realizando *global average pooling*, en donde se promedian los valores de cada canal de un *feature map* de tamaño $H \times W \times D$, generando un vector de tamaño D. En este caso los vectores generados tienen tamaño 2048.

Para la comparación entre vectores se utiliza la función de similitud coseno. Esta función entrega valores en el intervalo $[-1, 1]$, donde 1 significa muy similares ($cos(0) = 1$, misma dirección) , y -1 muy disímiles ($cos(\pi) = -1$, dirección opuesta). Dado dos vectores u, v, se calcula como:

$$similitudCoseno(u, v) = cos(\theta) = \frac{u \cdot v}{||u||||v||} \tag{2.2}$$

Para la evaluación del *retrieval* se utilizara el código dispuesto por DeepFashion2 [6], el cual lee los resultados obtenidos desde un archivo `.json` creado con anterioridad con las imágenes de `user` como consulta. Estos resultados deben obtenerse utilizando únicamente las detecciones encontradas tanto en las fotos de usuarios (`user`), como en las fotos de tiendas (`shop`), sin utilizar los *ground truth* del conjunto de validación. Debe tenerse en cuenta que un resultado se considera correcto solo si además de pertenecer a la misma prenda (mismo `pair_id`) la clase detectada concuerda con el *ground truth* y posee un valor de IoU mayor o igual a 0.5.

[6] `https://github.com/switchablenorms/DeepFashion2/blob/master/deepfashion2_api/PythonAPI/deepfashion2_retrieval_test.py`

Capítulo 3

Resultados

Se presentan resultados del entrenamiento de los modelos. Estos son obtenidos luego de evaluar sobre el conjunto de validación de los *datasets* y aplicar las métricas detalladas en la Sección 1.4. Se presentan también resultados de tiempos de inferencia de los modelos, y de la aplicación realizada en *retrieval*.

3.1. Tiempos de inferencia

Se presenta en la Figura 3.1 los tiempos promedio de inferencia de los modelos utilizando el *hardware* presentado en la Sección 2.2.

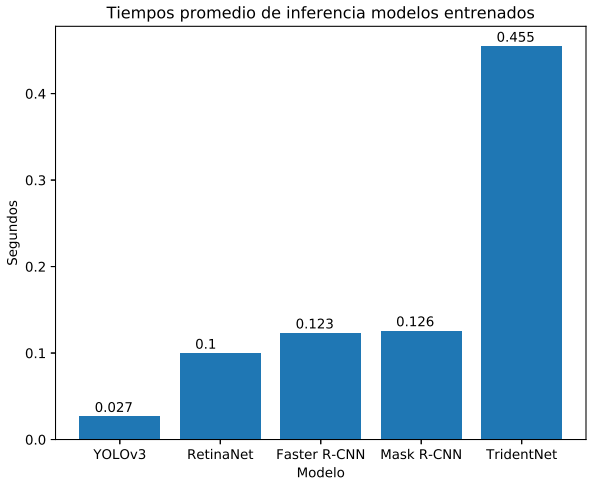


Figura 3.1: Tiempos promedio de inferencia.

3.2. Detección

En la Tabla 3.1 se presentan los resultados de *Average Precision*, en sus variantes mAP, AP_{50} y AP_{75}, de los modelos entrenados en los conjuntos ModaNet y DeepFashion2, mientras que en el Figura 3.2 se presenta el AP por cada una de las clases. Del mismo modo, en las Tablas 3.2 y 3.3 se muestran los resultados de AP por clase para ambos *datasets*.

Evaluación ModaNet				Evaluación DeepFashion2			
Modelo	mAP	AP_{50}	AP_{75}	Modelo	mAP	AP_{50}	AP_{75}
YOLOv3	0.391	0.681	0.406	YOLOv3	0.314	0.517	0.353
RetinaNet	0.475	0.691	0.532	RetinaNet	0.322	0.451	0.371
Faster R-CNN	0.489	0.736	0.566	Faster R-CNN	0.354	0.518	0.419
Mask R-CNN	0.49	0.744	0.56	Mask R-CNN	0.376	0.534	0.455
TridentNet	**0.54**	**0.775**	**0.622**	TridentNet	**0.509**	**0.708**	**0.616**

Tabla 3.1: *Average precision* en ModaNet y DeepFashion2. Se resaltan en negrita los mejores resultados.

	YOLOv3			RetinaNet			Faster R-CNN			Mask R-CNN			TridentNet		
Clase	AP	AP_{50}	AP_{75}	AP	AP_{50}	AP_{75}	AP	AP_{50}	AP_{75}	AP	AP_{50}	AP_{75}	AP	AP_{50}	AP_{75}
short sleeve top	**0.55**	**0.846**	**0.662**	**0.573**	**0.765**	**0.655**	**0.635**	**0.859**	**0.753**	**0.602**	**0.844**	**0.712**	**0.679**	**0.906**	**0.821**
long sleeve top	0.376	0.629	0.417	0.398	0.585	0.454	0.445	0.665	0.521	0.486	0.718	0.588	**0.595**	0.808	**0.72**1
short sleeve outwear	0.072	0.129	0.089	0.01	0.014	0.011	0.0	0.0	0.0	0.0	0.0	0.0	0.222	0.333	0.262
long sleeve outwear	0.259	0.481	0.25	0.365	0.536	0.4	0.412	0.661	0.458	0.446	0.663	0.527	0.581	0.77	0.704
vest	0.381	0.62	0.445	0.328	0.469	0.393	0.361	0.532	0.444	0.335	0.498	0.41	0.463	0.716	0.59
sling	0.064	0.119	0.067	0.019	0.033	0.018	0.029	0.051	0.03	0.027	0.047	0.029	0.275	0.446	0.317
shorts	0.389	**0.701**	0.389	**0.519**	**0.734**	**0.608**	**0.484**	**0.78**	0.554	**0.535**	**0.784**	**0.638**	0.567	**0.865**	0.674
trousers	0.**504**	**0.859**	**0.543**	**0.559**	**0.838**	**0.623**	**0.589**	**0.89**	**0.686**	**0.611**	**0.894**	**0.707**	**0.631**	**0.941**	**0.742**
skirt	**0.419**	0.691	**0.476**	0.444	0.597	0.521	0.462	0.656	**0.561**	0.504	0.676	0.61	**0.591**	0.801	**0.731**
short sleeve dress	0.341	0.506	0.434	0.334	0.422	0.391	0.392	0.53	0.489	0.416	0.551	0.525	0.579	0.726	0.698
long sleeve dress	0.178	0.277	0.212	0.197	0.253	0.229	0.223	0.304	0.268	0.275	0.357	0.335	0.417	0.529	0.503
vest dress	0.368	0.568	0.439	0.388	0.52	0.451	0.428	0.6	0.513	0.417	0.577	0.52	0.559	0.746	0.681
sling dress	0.163	0.293	0.163	0.057	0.091	0.062	0.138	0.204	0.171	0.235	0.336	0.308	0.456	0.622	0.561

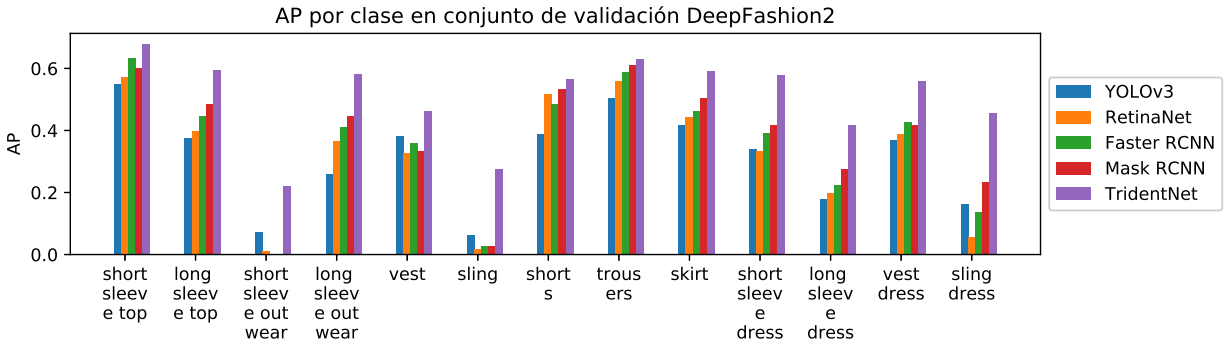
Tabla 3.2: Desempeño por clase en DeepFashion2. Se resalta en negrita las clases con mejores resultados en cada métrica.

Se presentan en la Figura 3.3 las curvas de *precision-recall* obtenidas para ambos *datasets* y para cada modelo. Del mismo modo, en las Figuras 3.4 y 3.5 se presentan las curvas de *precision-recall* por clase. Estas curvas se obtuvieron con la métrica AP_{50}.

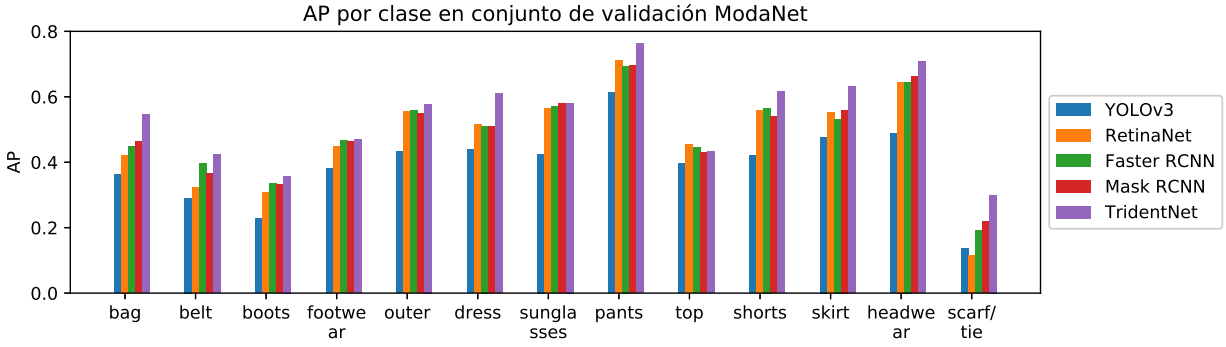
En las Figuras 3.6 y 3.7 se muestran resultados de detecciones en DeepFashion2 y ModaNet respectivamente, junto con sus *ground truth* correspondientes.

	YOLOv3			RetinaNet			Faster R-CNN			Mask R-CNN			TridentNet		
Clase	AP	AP_{50}	AP_{75}	AP	AP_{50}	AP_{75}	AP	AP_{50}	AP_{75}	AP	AP_{50}	AP_{75}	AP	AP_{50}	AP_{75}
bag	0.362	0.752	0.288	0.422	0.692	0.437	0.448	0.748	0.505	0.464	0.771	0.522	0.547	0.816	0.621
belt	0.291	0.663	0.2	0.322	0.618	0.301	0.396	0.73	0.376	0.365	0.736	0.307	0.423	0.766	0.443
boots	0.227	0.379	0.266	0.308	0.442	0.355	0.334	0.503	0.39	0.332	0.484	0.386	0.358	0.523	0.412
footwear	0.382	0.649	0.422	0.447	0.668	0.51	0.466	0.703	0.54	0.464	0.704	0.54	0.47	0.707	0.54
outer	0.434	0.714	0.49	0.556	0.796	0.615	0.559	0.816	0.65	0.548	0.824	0.647	0.578	0.828	0.672
dress	0.438	0.669	0.511	0.516	0.733	0.57	0.511	0.761	0.592	0.509	0.766	0.592	0.611	0.834	0.71
sunglasses	0.425	**0.862**	0.335	**0.565**	**0.888**	**0.695**	**0.571**	**0.894**	**0.704**	**0.58**	**0.904**	0.688	0.581	**0.906**	0.705
pants	**0.612**	**0.868**	**0.735**	**0.712**	**0.896**	**0.81**	**0.694**	**0.905**	**0.81**	**0.695**	**0.911**	**0.817**	**0.762**	**0.937**	**0.873**
top	0.396	0.654	0.438	0.454	0.7	0.485	0.446	0.702	0.496	0.429	0.702	0.473	0.433	0.668	0.481
shorts	0.421	0.733	0.432	0.557	0.766	0.634	0.566	0.804	0.673	0.539	0.793	0.641	0.616	0.838	0.723
skirt	**0.477**	0.735	**0.572**	0.552	0.714	0.64	0.532	0.727	0.646	0.56	0.741	**0.673**	**0.632**	0.811	**0.744**
headwear	**0.489**	**0.851**	**0.513**	**0.645**	**0.853**	**0.756**	**0.645**	**0.872**	**0.8**	**0.661**	**0.89**	**0.8**	**0.708**	**0.921**	**0.843**
scarf/tie	0.137	0.322	0.083	0.114	0.216	0.103	0.192	0.404	0.17	0.22	0.448	0.196	0.3	0.534	0.313

Tabla 3.3: Desempeño por clase en ModaNet. Se resaltan en negrita las clases con mejores resultados en cada métrica.

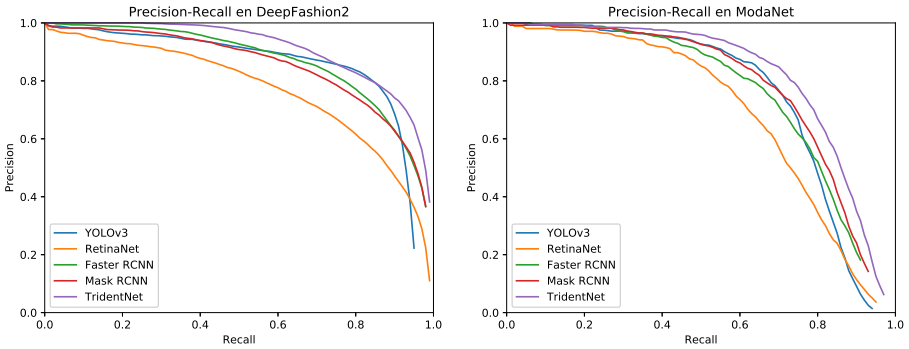


(a) DeepFashion2



(b) ModaNet

Figura 3.2: AP por clase en conjunto de validación.



(a) DeepFashion2. (b) Modanet.

Figura 3.3: Curvas *precision-recall* en conjuntos de validación.

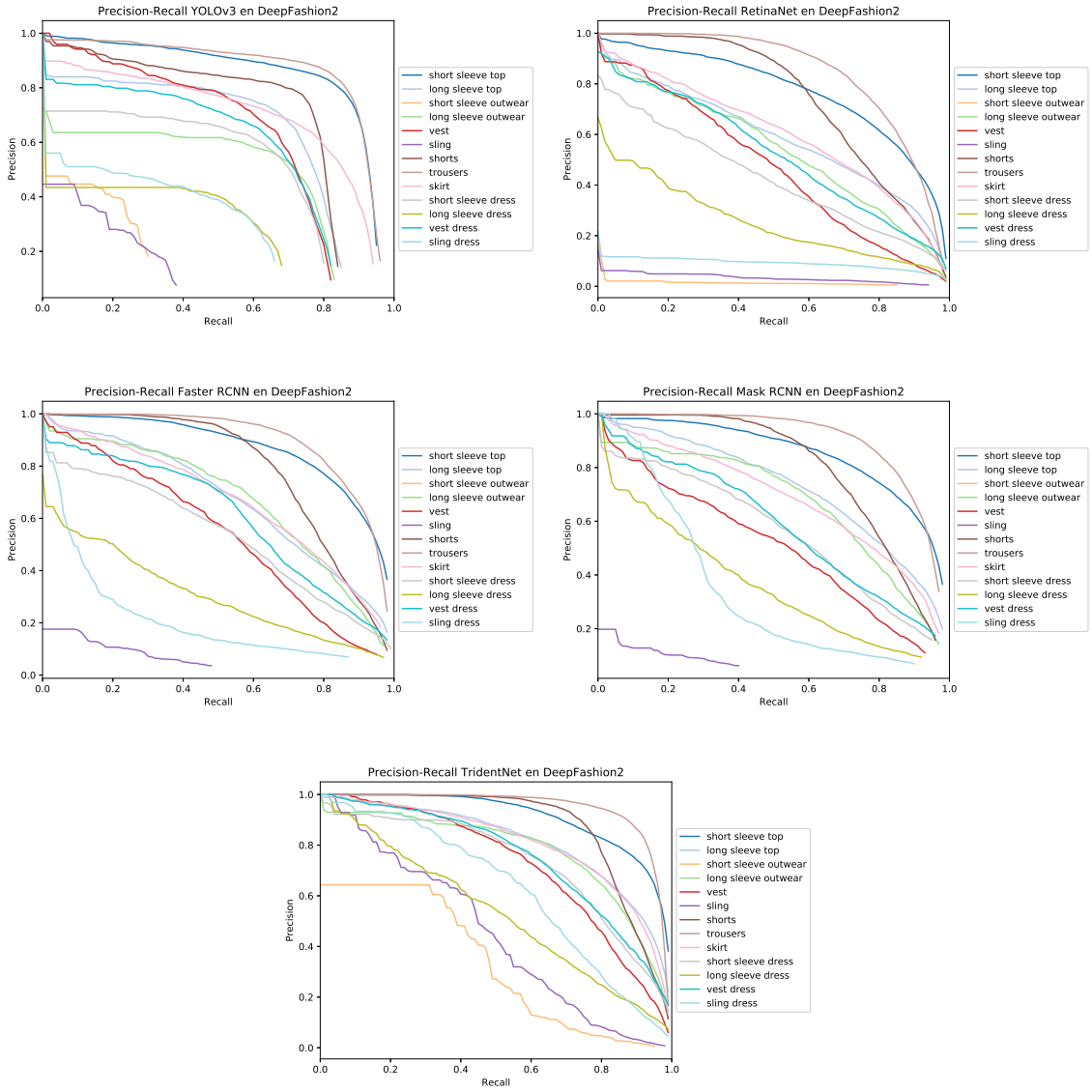


Figura 3.4: Curvas *precision-recall* con métrica AP_{50} para cada unas de las clases y modelos en *dataset* DeepFashion2.

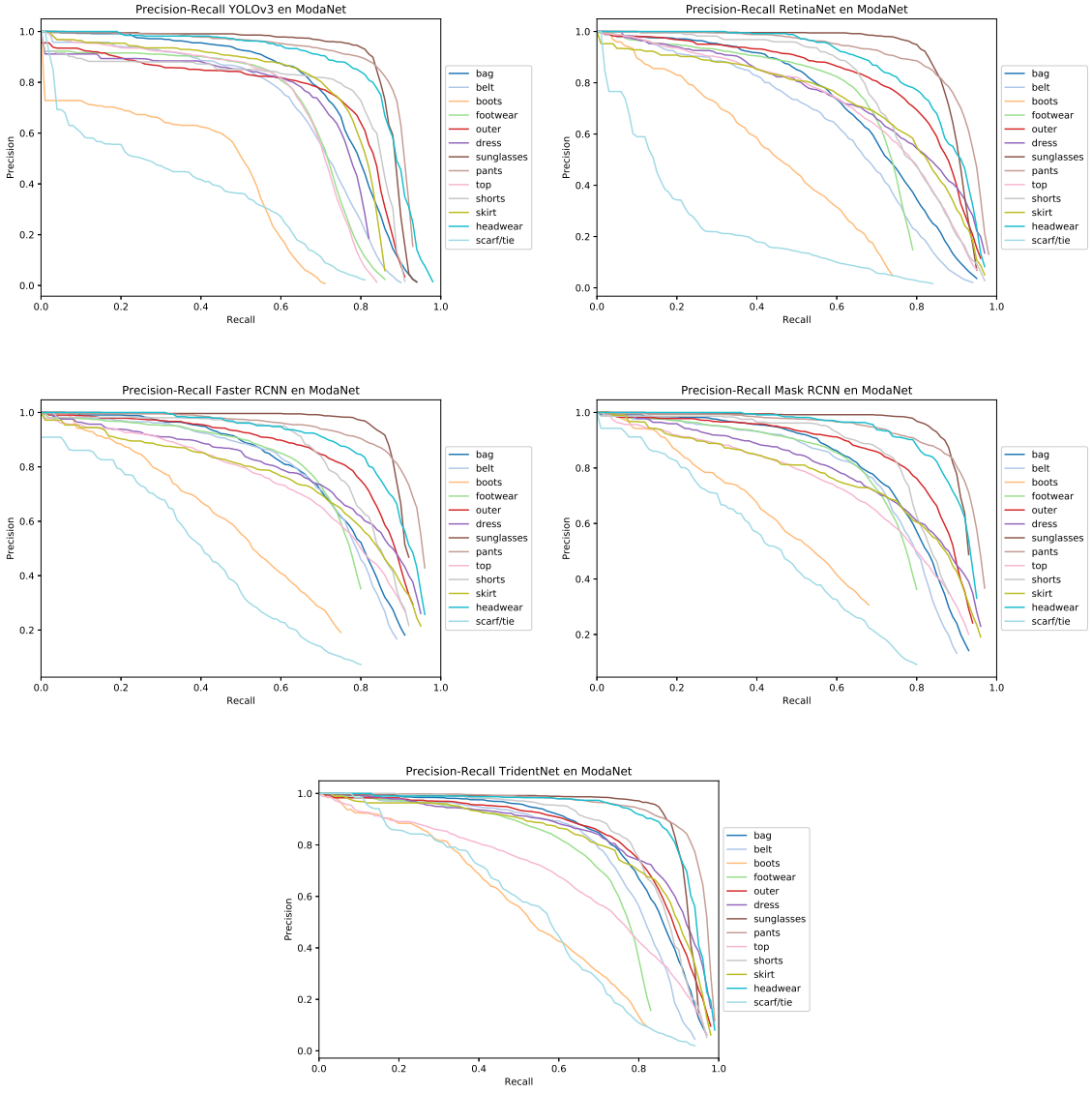


Figura 3.5: Curvas *precision-recall* con métrica AP_{50} para cada unas de las clases y modelos en *dataset* ModaNet.

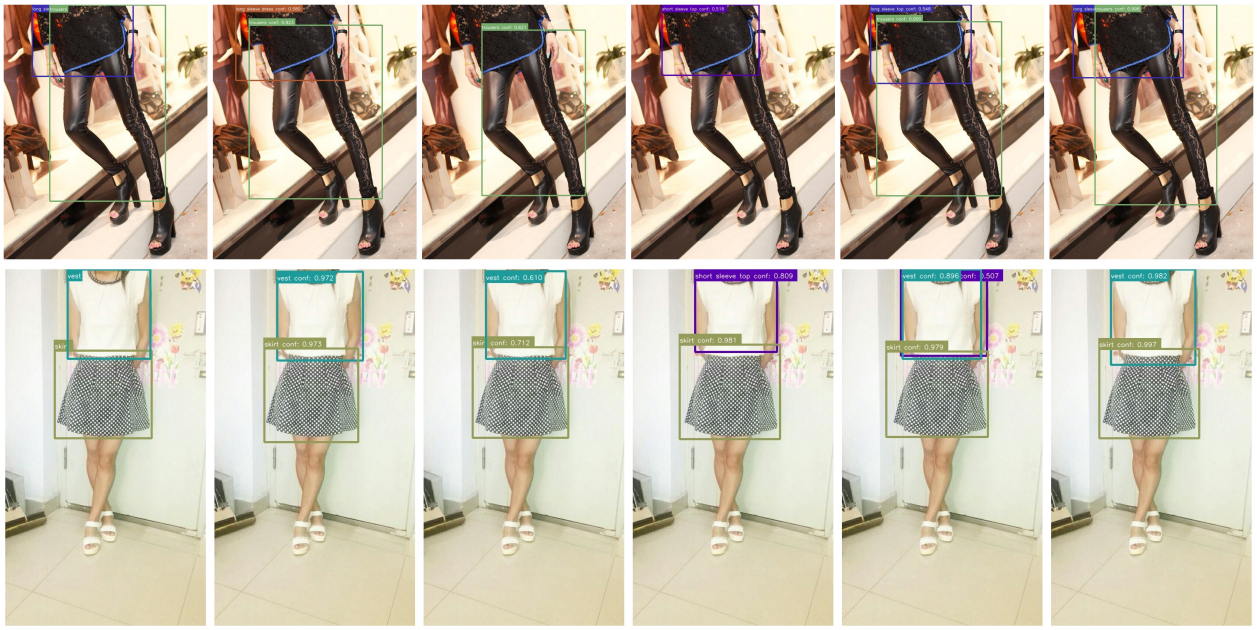


Figura 3.6: Resultados de detección en DeepFashion2. De izquierda a derecha: *Ground-truth*, YOLOv3, RetinaNet, Faster R-CNN, Mask R-CNN, TridentNet.

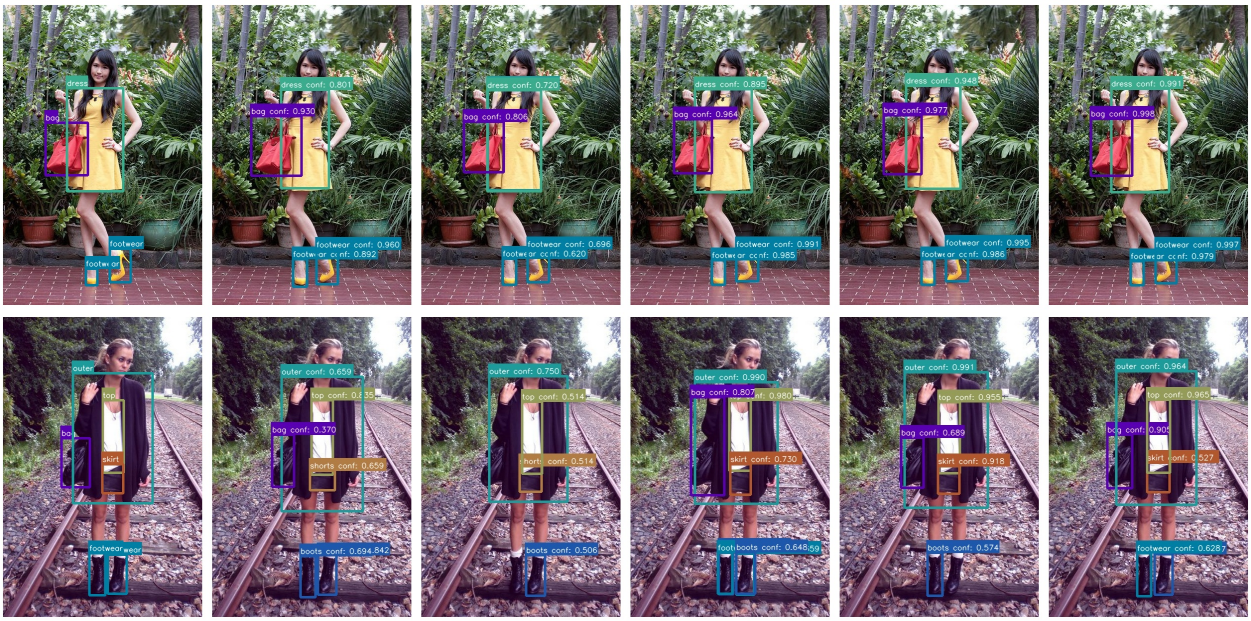


Figura 3.7: Resultados de detección en ModaNet. De izquierda a derecha: *Ground-truth*, YOLOv3, RetinaNet, Faster R-CNN, Mask R-CNN, TridentNet.

3.2.1. Conjunto de datos externo

Para una validación extra de los modelos, se utiliza un *dataset* privado proporcionado por profesor guía. Este se encuentra compuesto de imágenes extraídas de diferentes fuentes, por lo cual posee una gran variedad de ambientes, y de tamaños. Se realizó una correspondencia con las clases de ModaNet, debido a su gran similaridad. El calzado se engloba únicamente en la clase *footwear*, por lo que no existen instancias de la clase *boots*, la cual es eliminada de los cálculos

En la Tabla 3.4 se muestran los resultados de mAP, AP_{50} y AP_{75} para cada modelo, mientras que los resultados de AP por clase se presentan en la Tabla 3.5 y en la Figura 3.8. Se presentan también, la curva de *precision-recall* en la Figura 3.9, y del mismo modo las curvas por clase en la Figura 3.10.

Evaluación *dataset* privado			
Modelo	mAP	AP_{50}	AP_{75}
YOLOv3	0.206	0.466	0.147
RetinaNet	0.208	0.461	0.169
Faster R-CNN	0.209	0.480	0.151
Mask R-CNN	0.242	0.523	0.199
TridentNet	**0.276**	**0.561**	**0.251**

Tabla 3.4: *Average precision dataset* privado. Se resaltan en negrita los mejores resultados.

	YOLOv3			RetinaNet			Faster R-CNN			Mask R-CNN			TridentNet		
Clase	AP	AP_{50}	AP_{75}	AP	AP_{50}	AP_{75}	AP	AP_{50}	AP_{75}	AP	AP_{50}	AP_{75}	AP	AP_{50}	AP_{75}
bag	0.214	0.603	0.089	0.14	0.476	0.019	0.202	0.581	0.053	0.187	0.589	0.032	0.254	0.65	0.127
belt	0.047	0.191	0.009	0.024	0.121	0.002	0.019	0.097	0.001	0.037	0.193	0.002	0.037	0.192	0.002
footwear	0.026	0.128	0.001	0.027	0.143	0.001	0.023	0.122	0.001	0.035	0.173	0.001	0.032	0.163	0.001
outer	0.296	0.606	0.233	**0.389**	**0.683**	**0.435**	**0.368**	0.703	**0.343**	**0.441**	0.74	**0.507**	**0.469**	0.762	**0.55**
dress	**0.315**	**0.609**	**0.267**	**0.381**	**0.716**	**0.357**	**0.408**	**0.741**	**0.421**	**0.415**	**0.769**	**0.412**	**0.502**	**0.828**	**0.574**
sunglasses	0.01	0.059	0.001	0.014	0.09	0.0	0.004	0.028	0.0	0.011	0.071	0.0	0.031	0.161	0.001
pants	**0.423**	**0.742**	**0.447**	**0.453**	**0.776**	**0.495**	**0.398**	**0.764**	**0.367**	**0.479**	**0.809**	**0.526**	**0.541**	**0.85**	**0.633**
top	0.278	0.564	0.238	0.288	0.623	0.229	0.259	0.609	0.172	0.31	0.632	0.255	0.286	0.587	0.24
shorts	0.243	0.598	0.119	0.218	0.548	0.118	0.214	0.589	0.082	0.255	0.601	0.156	0.296	0.676	0.199
skirt	**0.303**	0.602	**0.252**	0.313	0.575	0.312	0.316	0.591	0.299	0.359	0.615	0.382	0.444	0.7	0.527
headwear	0.25	**0.722**	0.08	0.212	0.663	0.055	0.234	**0.722**	0.064	0.278	**0.773**	0.092	0.271	**0.767**	0.087
scarf/tie	0.066	0.171	0.026	0.035	0.115	0.01	0.058	0.21	0.01	0.096	0.309	0.02	0.145	0.397	0.071

Tabla 3.5: Desempeño por clase en *dataset* privado. Se resalta en negrita las clases con mejores resultados en cada métrica

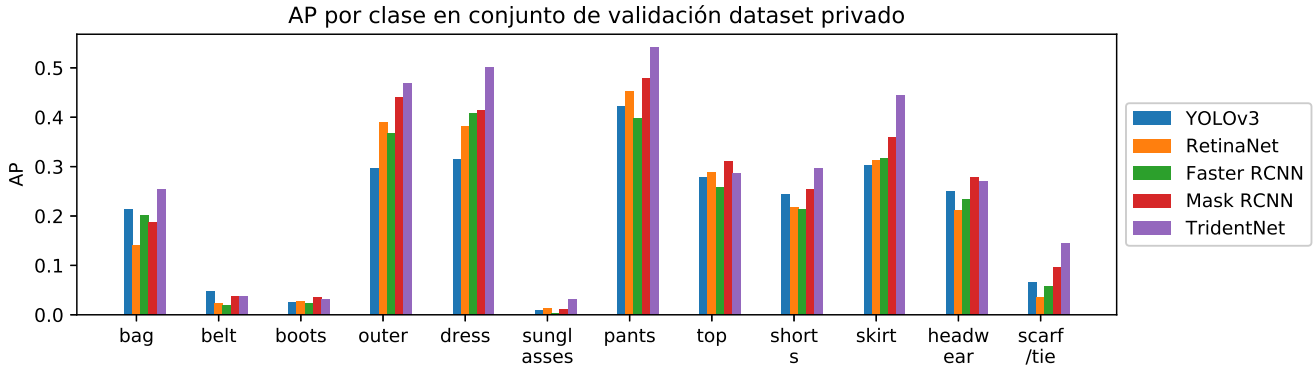


Figura 3.8: AP por clase *dataset* privado.

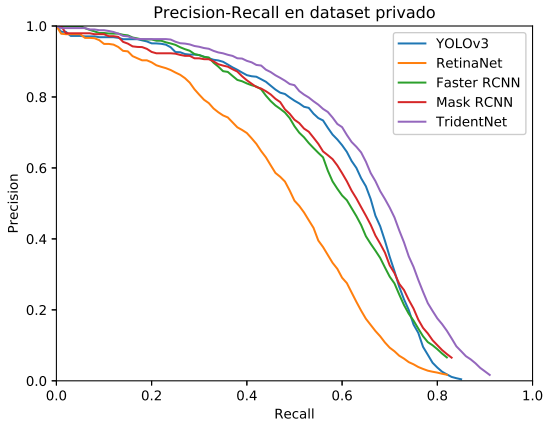


Figura 3.9: Curva *precision-recall* obtenida de conjunto de validación de *dataset* privado.

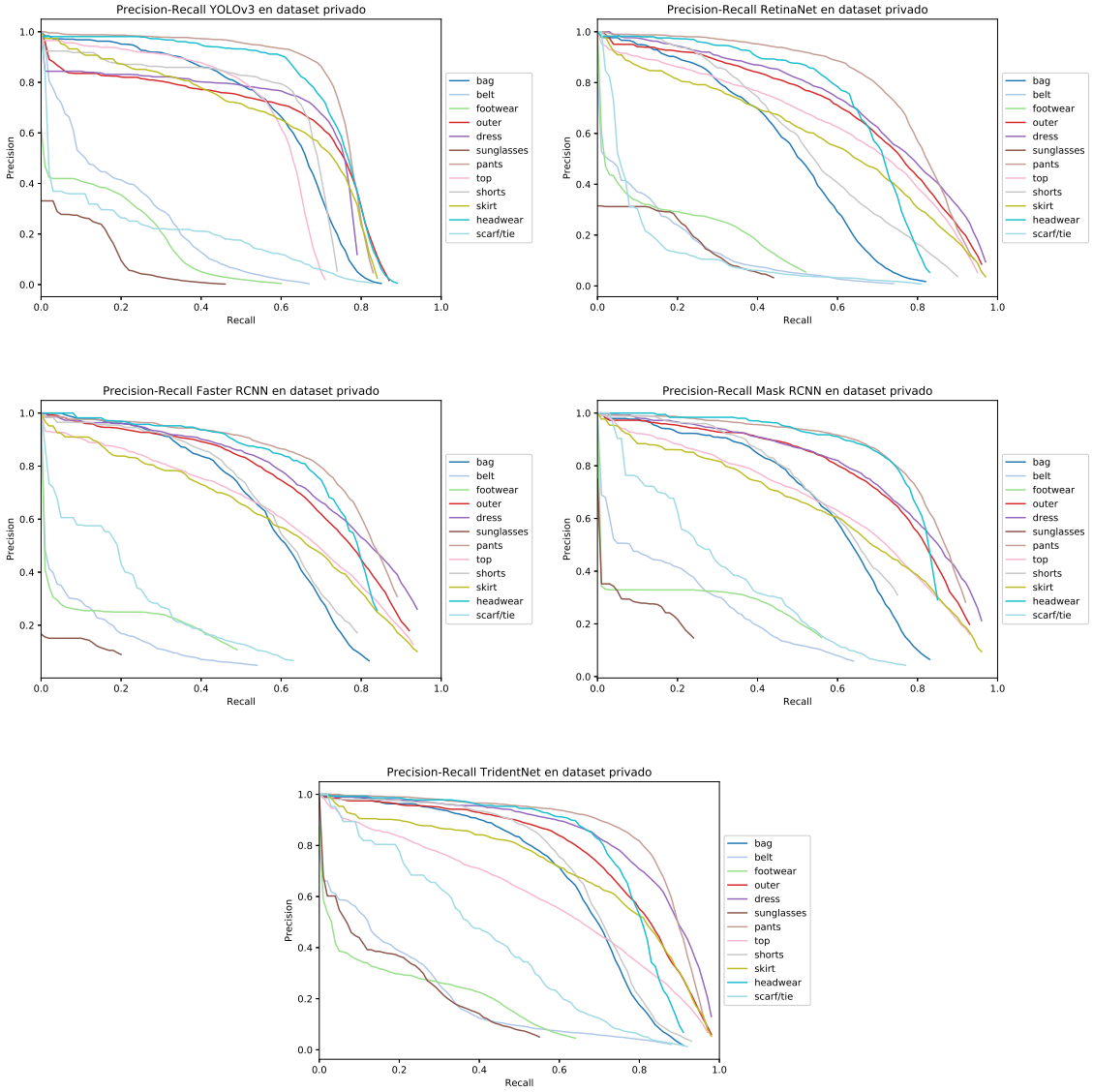


Figura 3.10: Curvas *precision-recall* para cada unas de las clases y modelos en *dataset* privado.

3.3. *Retrieval*

En la Tabla 3.6 se muestran los resultados de aplicar el test dispuesto por DeepFashion2 para evaluar la calidad del *retrieval*, comparándose con los resultados presentados en [35]. En la Figura 3.11 se muestran resultados exitosos en la recuperación de imagenes mientras que en la Figura 3.12 se muestran resultados incorrectos.

Evaluación *retrieval*		
Métrica	DeepFashion2	Este trabajo
top-1	0.079	0.046
top-5	0.198	0.105
top-10	0.273	0.174

Tabla 3.6: Resultados *retrieval* versus resultados de DeepFashion2.

Figura 3.11: Resultados favorables en *clothing retrieval*. En primera columna la consulta y en el resto de columnas los primeros cuatro resultados de la búsqueda.

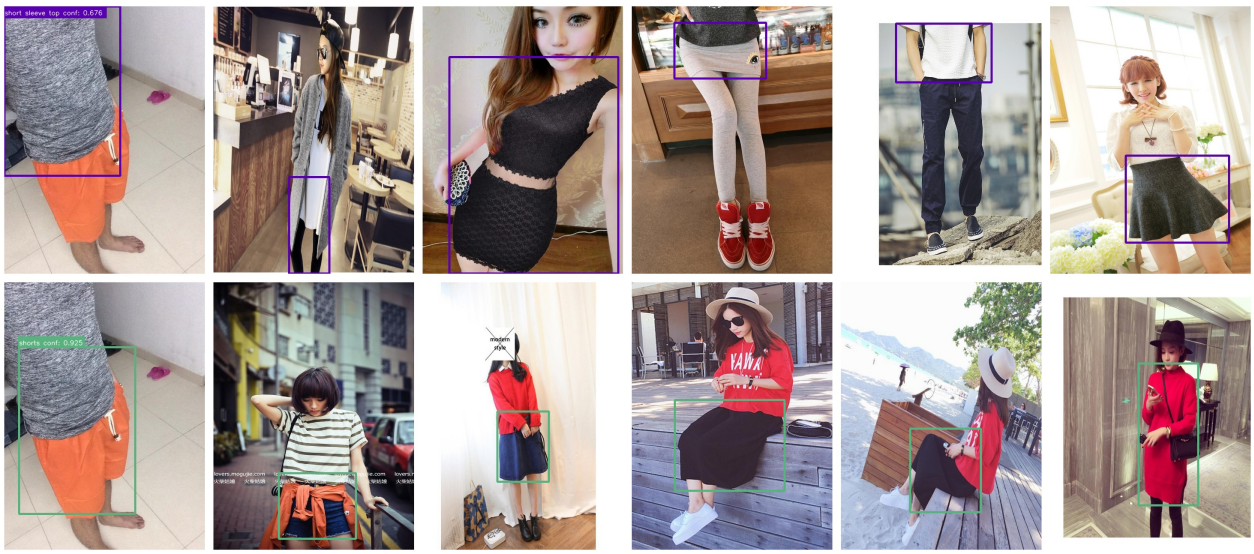


Figura 3.12: Resultados desfavorables en *clothing retrieval*. En primera columna la consulta y en el resto de columnas los primeros cuatro resultados de la búsqueda.

Capítulo 4

Análisis

En cuanto a los tiempos de inferencia promedio, se observa en la Figura 3.1 como YOLOv3 obtiene el menor valor, lo cual era lo esperado dada su arquitectura, la cual le permite obtener velocidades cuatro veces superiores [21] que por ejemplo, RetinaNet. En el caso de Faster R-CNN y Mask R-CNN se observan tiempos casi idénticos, lo cual es esperado debido a su idéntica arquitectura de detección de objetos, salvo la operación de *RoiAlign* utilizada en Mask en reemplazo de *RoiPool*, lo cual no agrega tiempo considerable en el procesamiento. El tiempo de inferencia considerablemente mayor utilizado por TridentNet ocurre por la gran cantidad de *trident blocks* utilizados en su arquitectura, los cuales incurren en mayor requerimiento computacional.

Al observar el resultado del cálculo de mAP presentado en la Figura 3.1, se observa como los modelos entrenados en ModaNet obtienen un mejor desempeño general. Este desempeño se explica de mejor manera observando los gráfico de AP por clase en la Figura 3.2. Aquí se observa como las clases que menor AP poseen en DeepFashion2 corresponden a clases bastante difíciles de diferenciar incluso para el ojo humano, debido a su elevada similitud y alta granularidad, lo cual a vez genera ambigüedades en los etiquetados. Sin embargo, al fijarnos en las clases comunes entre ambos *datasets*, y que son al mismo tiempo clases bien diferenciadas entre si, se obtienen valores bastante similares de AP, tal como se observa en la Tabla 4.1.

	YOLOv3		RetinaNet		Faster R-CNN		Mask R-CNN		TridentNet	
Clase	DF2	MN	DF2	MN	DF2	MN	DF2	MN	DF2	MN
trousers/pants	0.504	0.612	0.559	0.712	0.589	0.694	0.611	0.695	0.631	0.762
skirt	0.419	0.477	0.444	0.552	0.462	0.532	0.504	0.56	0.591	0.632
short sleeve top/top	0.55	0.396	0.573	0.454	0.635	0.446	0.602	0.429	0.679	0.433
short sleeve dress/dress	0.341	0.438	0.334	0.516	0.392	0.511	0.416	0.509	0.579	0.611

Tabla 4.1: Clases comunes a ambos *datasets* y su respectivo AP. DF: DeepFashion2; MN: ModaNet.

En ModaNet se observa que las clases con mejor desempeño corresponden a *sunglasses*, *pants* y *headwear* (ver Figura 3.2b y Tabla 3.3), esto debido a que son clases que usualmen-

te tienen formas definidas y no son ocluidas por otros objetos, por lo que resulta de fácil detección para los modelos. Por otro lado *scarf/tie* (bufanda/corbata) es la clase con peor desempeño, debido a ser objetos altamente deformables y los cuales suelen ser ocluidos por otras vestimentas.

En lo que concierne a los resultados del *dataset* privado, se observa en la Tabla 3.5 que al igual que ModaNet, posee bueno resultados en las clases *pants* y *headwear*, siendo esta ultima notoria únicamente en métrica AP_{50}. Sin embargo, se observa también su bajo desempeño en la clase *sunglasses*, la cual resultó ser una con el mejor desempeño en ModaNet. Este resultado se debe a que en esta clase las anotaciones se encuentran poco ajustadas a la vestimenta, causando que la detección sea ignorada al tener un valor de IoU menor al umbral mínimo ($\text{IoU}^{tr} = 0.5$), lo cual no sucede en ModaNet al tener *bounding boxes* bien ajustadas a las vestimentas. Esto sucede también en la clase *footwear*, donde a eso se suma la inconsistencia en el etiquetado del calzado, encontrando en algunos casos que se etiquetan por separado y en otros juntos como una sola prenda. Estos casos problemáticos se evidencian en la Figura 4.1, donde se muestran un par de ejemplos de detecciones de las clases *sunglasses* y *footwear*, junto con sus respectivos *ground truth* y valores de IoU.

Figura 4.1: Ejemplos de resultados para clases *sunglasses* y *footwear* obtenidos con TridentNet. Se muestra azul el *ground truth* junto con el IoU calculado con la detección (verde y celeste respectivamente).

4.1. Deficiencias en métrica AP

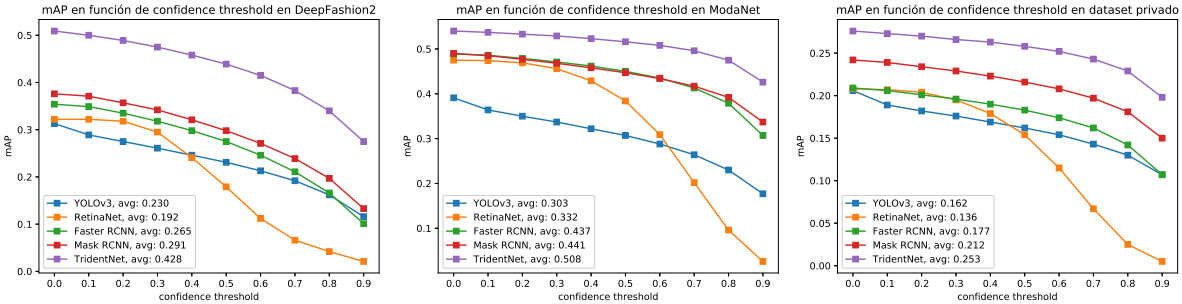
La métrica de AP, tal como se menciona en la Sección 1.4, está construida de tal manera que para obtener buenos resultados es deseable tener valores de *precision* a lo largo de todos los puntos de *recall*. Esto provoca que modelos que entreguen bajos valores de *confidence*

score y una gran cantidad de detecciones, puedan seguir teniendo valores altos en el AP, en desmedro de modelos que entreguen pocas detecciones pero con alta precisión. En efecto, dos modelos con igual cantidad y dimensiones de *bounding boxes* detectados, con *confidence scores* distintos pero ordenados de igual manera, conllevan en iguales resultado de AP. Esto se suma al hecho de que para el cálculo de esta métrica se utiliza *confidence threshold* igual a cero, lo cual podria llenar de resultados indeseables en una aplicación real.

Para evidenciar este comportamiento, es que se calcula el valor de mAP a distintos valores de *confidence threshold*. Esto permite evaluar como se comporta realmente el modelo en una aplicación real, y que tan estable es a los cambios de este umbral. Este comportamiento se muestra en la Figura 4.2. Como se observa, los modelos disminuyen su mAP conforme se aumenta el *confidence threshold* aplicado, sin embargo esto es más notorio en RetinaNet, donde a umbrales altos el mAP disminuye a valores cercanos a cero. Esto representa como modelos que en un principio entregan resultados similares en AP, se ven afectados de distinta manera al cambiar el umbral de detección, lo que se evidencia notoriamente entre los modelos YOLOv3 y RetinaNet. Sumado a esto, se observa como YOLOv3 entrega resultados de mAP similares a Faster R-CNN a altos valores de *confidence threshold*, para los *datasets* DeepFashion2 y privado.

Este comportamiento también se puede evidenciar en las curvas de *precision-recall* al variar el valor de *confidence threshold*, como se observa en la Figura 4.3 en el caso de ModaNet. Se observa como se ven afectados los modelos a medida que se aumenta este umbral, disminuyendo *precision* a valores altos de *recall*.

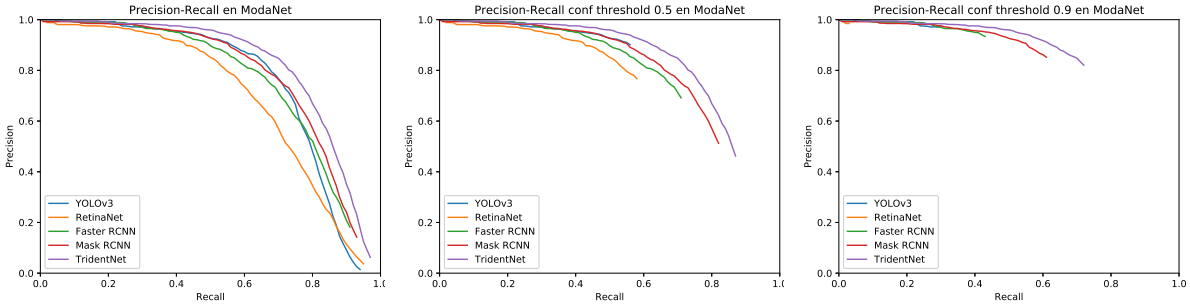
Se evidencia también, la gran estabilidad mostrada por TridentNet, donde su promedio de detecciones se encuentra ligeramente por debajo del mAP con umbral igual a cero. Esto se debe a que TridentNet entrega por lo general altos valores de *confidence score* a sus detecciones. En la Figura 4.4 se presenta un ejemplo de detecciones con distintos umbrales para los modelos TridentNet y RetinaNet.



(a) DeepFashion2. (b) ModaNet. (c) *Dataset* privado.

Figura 4.2: mAP en función de *confidence threshold* para los *datasets* utilizados. Se muestra el promedio de los mAP obtenidos para cada modelo.

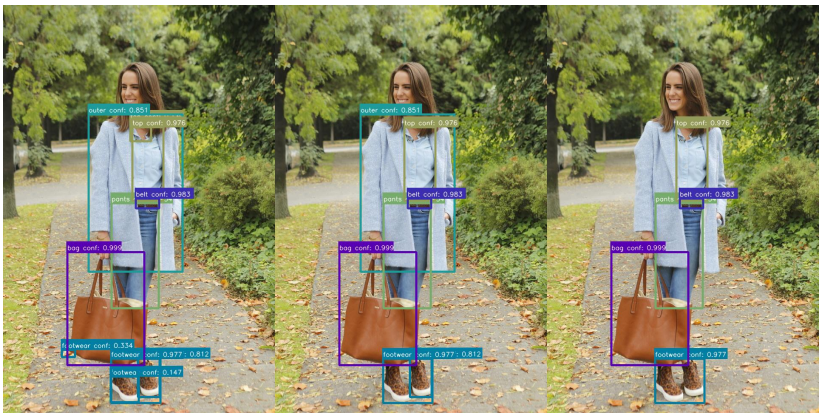
En cuanto a los *datasets* utilizados, se observa como el más afectado en el cambio del umbral de *confidence score* resulta ser DeepFashion2. Esto se debe a la elevada dificultad de sus datos, debido a la alta granularidad de sus clases. En el caso de ModaNet y de *dataset*



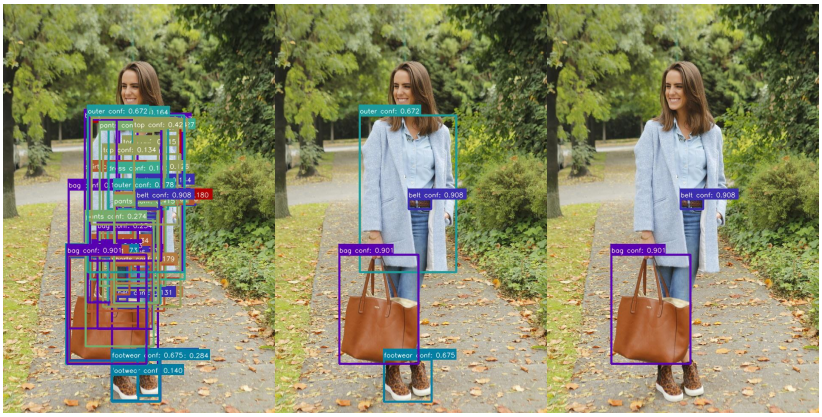
(a) *Conf. threshold* 0. (b) *Conf. threshold* 0.5. (c) *Conf. threshold* 0.9.

Figura 4.3: Curvas *precision-recall* a distintos valores de *confidence threshold* en *dataset* ModaNet.

privado se observan curvas bastante similares, a pesar de que en ModaNet se mantienen valores mayores de mAP.



(a) TridentNet.



(b) RetinaNet.

Figura 4.4: Resultados de detección utilizando distintos valores de *confidence threshold* con TridentNet y RetinaNet. Valores utilizados de izquierda a derecha: $0.1, 0.5, 0.9$.

4.1.1. Retrieval

En cuanto a los resultados presentados en *retrieval*, se observa que a pesar de realizar una estrategia carente de sofisticación, se obtienen resultados bastante prometedores. Cabe recordar que el extractor de características utilizado corresponde a una red ResNet-50 pre-entrenada en ImageNet, a la cual no se le realiza ningún tipo de ajuste ya sea *transfer learning* o *fine tuning*, y además las características extraídas se comparan únicamente por métricas de distancia, sin realizar ningún tipo de post-procesamiento. Esto abre variadas posibilidades de sofisticación para un trabajo futuro.

Conclusión

Los modelos de detección de objetos han sido una parte importantísima en el desarrollo del *deep learning*, debido a sus múltiples aplicaciones en todo tipo de áreas. En el presente trabajo se aplicó la detección de objetos en el contexto de detección de vestimenta, donde se utilizaron modelos en el estado del arte entrenados en *datasets* apropiados los cuales son de libre acceso y de gran popularidad en este contexto.

Para el entendimiento del trabajo se realizó una detallada descripción de los modelos de detección de objetos utilizados, como también de las métricas utilizadas para su comparación. Seguido de esto, se explicó la metodología utilizada para la preparación de los datos con el fin de ser utilizados para el proceso de entrenamiento.

Para el entrenamiento fue necesario el uso de distintos *frameworks* de redes neuronales los cuales entregaron herramientas que facilitaron el proceso, sin embargo estas herramientas son de uso general, por lo que fue necesario el entendimiento íntegro del código fuente y su posterior adaptación al problema particular, lo que conlleva un arduo trabajo. Este entrenamiento consume una gran cantidad de tiempo y recursos computacionales, debido a la gran cantidad de datos a procesar y a la gran cantidad de parámetros de los modelos, tardando incluso un día completo en realizarse. A causa de esto, fue necesario el uso de una instancia de una máquina virtual en la nube para el aceleramiento de este.

Los resultados obtenidos dan por satisfecho el objetivo principal de este trabajo, ya que el entrenamiento de los modelos resultó exitoso y fue posible una comparación objetiva tanto cuantitativa como cualitativa de estos, en ambos *datasets* utilizados. Sumado a esto, se validó el entrenamiento de los modelos en un *dataset* de vestimentas externo, el cual ayudó a caracterizar el comportamiento de los modelos de una manera más completa y a verificar la universalidad de los resultados.

Para la comparación de los modelos se utilizaron métricas estandarizadas para detección de objetos, las cuales permitieron analizar los modelos bajo ciertas condiciones de uso. En vista de que estas métricas por si solas no son capaces de caracterizar el desempeño de los modelos, se realizaron ajustes a parámetros de la detección para evaluar el comportamiento de los modelos bajo distintas situaciones, cuyo objetivo era contextualizar su uso en aplicaciones reales.

Los resultados indican como los modelos tienen alta capacidad de detectar vestimentas bien definidas y sin ambigüedades en su definición, como son el caso de las clases *pants*, *skirt*, *dress*, *headwear* y *sunglasses*, mientras que clases con alta granularidad y de bastante

ambigüedad en su etiquetado les resulta difícil, como es el caso de las clases presentes en DeepFashion2. Del mismo modo, le resulta difícil detectar objetos que poseen demasiada variedad en sus formas, y que además suelen encontrarse ocluidos por otras vestimentas, como es el caso de la clase *scarf/tie* (bufanda/corbata).

En cuanto a los resultados por modelo, estos señalan el gran desempeño y precisión de TridentNet versus el resto de los modelos, el cual entrega detecciones de alto *confidence score*, y que además localizan de excelente forma a los objetos. Sin embargo, este desempeño se hace a costa de alta necesidad de cómputo, por lo que puede no ser viable para una aplicación en tiempo real (cámaras de vigilancia). Por otro lado, modelos como Mask R-CNN entregan resultados aceptables para una aplicación real, utilizando casi un cuarto de tiempo de cómputo comparado con TridentNet.

Se considera también como satisfactoria la realización del objetivo secundario de este trabajo, el cual correspondía a utilizar la detección de vestimentas en cierta aplicación de interés. Esto fue llevado a cabo utilizando las detecciones de vestimentas para su uso en *clothing retrieval*, herramienta capaz de buscar vestimentas, dada las detectadas en una imagen, similares en una base de datos. Los resultados obtenidos no superaron a los publicados por DeepFashion2, sin embargo la estrategia utilizada resultó en una primera aproximación al problema, la cual a pesar de su simplicidad entregó resultados bastante prometedores, además de ser visualmente aceptables.

Como contribución del trabajo se tiene el repositorio de `GitHub` en donde se encuentran los modelos entrenados junto con sus archivos de configuración, como también el código fuente de la mayoría de los experimentos realizados en este trabajo. El repositorio se encuentra disponible en `https://github.com/simaiden/Clothes-Detection`.

Trabajo Futuro

Debido a que ModaNet resulta un *dataset* más aplicable a casos reales de uso, se podría aumentar su cantidad de imágenes de entrenamiento utilizando las disponibles en DeepFashion2, realizando la correspondencia de clases cuando corresponda, y utilizando etiquetado manual o semi-automático donde no exista esta correspondencia. Un etiquetado semi-automático consistiría en utilizar los modelos entrenados para detectar las vestimentas, y dada esta detección realizar el ajuste corresponde del *bounding box* y de la clase.

La continuación natural de este trabajo es la creación de un sistema sofisticado de *clothing retrieval*, el cual utilice la detección realizada por un modelo a elección. Para este fin se podrían utilizar un trabajo similar a lo realizado por Hermans et. al [37], en donde se utiliza una red convolucional combinada con *triplet loss* para la re-identificación de personas. Esta estrategia es bastante similar a lo utilizado en *retrieval*, debido a que se busca la misma instancia del objeto, o una muy similar, sobre un conjunto de datos dado.

Bibliografía

[1] T. Lin, P. Dollár, R. B. Girshick, K. He, B. Hariharan, and S. J. Belongie, "Feature pyramid networks for object detection," *CoRR*, vol. abs/1612.03144, 2016.

[2] A. Krizhevsky, V. Nair, and G. Hinton, "Cifar-10 (canadian institute for advanced research),"

[3] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015.

[4] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," in *Proceedings of the IEEE*, pp. 2278–2324, 1998.

[5] N. Dalal and B. 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, (USA), p. 886–893, IEEE Computer Society, 2005.

[6] T. Ojala, M. Pietikäinen, and D. Harwood, "Performance evaluation of texture measures with classification based on kullback discrimination of distributions," *Proceedings of 12th International Conference on Pattern Recognition*, vol. 1, pp. 582–585 vol.1, 1994.

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25* (F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, eds.), pp. 1097–1105, Curran Associates, Inc., 2012.

[8] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014.

[9] S. Ren, K. He, R. B. Girshick, and J. Sun, "Faster R-CNN: towards real-time object detection with region proposal networks," *CoRR*, vol. abs/1506.01497, 2015.

[10] K. He, G. Gkioxari, P. Dollár, and R. B. Girshick, "Mask R-CNN," *CoRR*, vol. abs/1703.06870, 2017.

[11] J. Redmon, S. K. Divvala, R. B. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *CoRR*, vol. abs/1506.02640, 2015.

[12] T. Lin, P. Goyal, R. B. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," *CoRR*, vol. abs/1708.02002, 2017.

[13] Y. Li, Y. Chen, N. Wang, and Z. Zhang, "Scale-aware trident networks for object detection," *CoRR*, vol. abs/1901.01892, 2019.

[14] R. B. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *CoRR*, vol. abs/1311.2524, 2013.

[15] R. B. Girshick, "Fast R-CNN," *CoRR*, vol. abs/1504.08083, 2015.

[16] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *CoRR*, vol. abs/1409.1556, 2014.

[17] M. Everingham, L. Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *Int. J. Comput. Vision*, vol. 88, p. 303–338, June 2010.

[18] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Computer Vision and Pattern Recognition (CVPR)*, 2015.

[19] J. Redmon and A. Farhadi, "YOLO9000: better, faster, stronger," *CoRR*, vol. abs/1612.08242, 2016.

[20] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," *CoRR*, vol. abs/1411.4038, 2014.

[21] J. Redmon and A. Farhadi, "Yolov3: An incremental improvement," *CoRR*, vol. abs/1804.02767, 2018.

[22] F. Yu and V. Koltun, "Multi-scale context aggregation by dilated convolutions," 11 2015.

[23] T. Lin, M. Maire, S. J. Belongie, L. D. Bourdev, R. B. Girshick, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: common objects in context," *CoRR*, vol. abs/1405.0312, 2014.

[24] M. Yang and K. Yu, "Real-time clothing recognition in surveillance videos," in *2011 18th IEEE International Conference on Image Processing*, pp. 2937–2940, Sep. 2011.

[25] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *Int. J. Comput. Vision*, vol. 60, pp. 91–110, Nov. 2004.

[26] C. Álvarez Inostroza, "Using automatic clothing labeling to improve the quality of clothing retrieval systems," 2018.

[27] M. Kucer and N. Murray, "A detect-then-retrieve model for multi-domain fashion item retrieval," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2019.

[28] W.-C. Kang, E. Kim, J. Leskovec, C. Rosenberg, and J. McAuley, "Complete the look:

Scene-based complementary product recommendation," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2019.

[29] R. Girshick, I. Radosavovic, G. Gkioxari, P. Dollár, and K. He, "Detectron." `https://github.com/facebookresearch/detectron`, 2018.

[30] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *arXiv preprint arXiv:1408.5093*, 2014.

[31] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, "Automatic differentiation in PyTorch," in *NeurIPS Autodiff Workshop*, 2017.

[32] F. Massa and R. Girshick, "maskrcnn-benchmark: Fast, modular reference implementation of Instance Segmentation and Object Detection algorithms in PyTorch." `https://github.com/facebookresearch/maskrcnn-benchmark`, 2018. Accessed: December 2019.

[33] J. Redmon, "Darknet: Open source neural networks in c." `http://pjreddie.com/darknet/`, 2013–2016.

[34] Y. Chen, C. Han, Y. Li, Z. Huang, Y. Jiang, N. Wang, and Z. Zhang, "Simpledet: A simple and versatile distributed framework for object detection and instance recognition," *Journal of Machine Learning Research*, vol. 20, no. 156, pp. 1–8, 2019.

[35] Y. Ge, R. Zhang, L. Wu, X. Wang, X. Tang, and P. Luo, "A versatile benchmark for detection, pose estimation, segmentation and re-identification of clothing images," *CVPR*, 2019.

[36] S. Zheng, F. Yang, M. H. Kiapour, and R. Piramuthu, "Modanet: A large-scale street fashion dataset with polygon annotations," *CoRR*, vol. abs/1807.01394, 2018.

[37] A. Hermans, L. Beyer, and B. Leibe, "In defense of the triplet loss for person re-identification," *CoRR*, vol. abs/1703.07737, 2017.