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

RECONOCIMIENTO RÁPIDO DE OBJETOS USANDO OBJECT PROPOSALS Y DEEP LEARNING

MEMORIA PARA OPTAR AL TÍTULO DE INGENIERA CIVIL ELÉCTRICA

CLAUDIA NAIOMI SOTO BARRA

PROFESOR GUÍA:
JAVIER RUIZ DEL SOLAR SAN MARTIN

MIEMBROS DE LA COMISIÓN:
PATRICIO LONCOMILLA ZAMBRANA
FELIPE TOBAR HENRIQUEZ

SANTIAGO DE CHILE
2017

RESUMEN DE LA MEMORIA PARA OPTAR
AL TÍTULO DE INGENIERA CIVIL ELÉCTRICA
POR: CLAUDIA NAIOMI SOTO BARRA
FECHA: 2017
PROF. GUÍA: SR. JAVIER RUIZ DEL SOLAR SAN MARTIN

RECONOCIMIENTO RÁPIDO DE OBJETOS USANDO OBJECT PROPOSALS Y DEEP LEARNING

El reconocimiento (o detección) de objetos es un área activa y en continua mejora de la visión computacional. Recientemente se han introducido distintas estrategias para mejorar el desempeño y disminuir los costos y el tiempo de detección. Entre estas, se encuentran la generación de *Object Proposals* (regiones en la imágen donde hay alta probabilidad de encontrar un objeto) para acelerar la etapa de localización, como respuesta al paradigma de ventana deslizante; el cada vez más popular uso de redes *Deep Learning* y, en particular, para la clasificación y detección de imágenes, las redes convolucionales (CNN).
Si bien existen diversos trabajos que utilizan ambas técnicas, todos ellos se centran en tener una buena *performance* en conocidas bases de datos y competencias en lugar de estudiar su comportamiento en problemas reales y el efecto que tiene la modificación de arquitecturas de redes convencionales y la elección adecuada de un sistema de generación de *proposals*.

En este trabajo de título, entonces, se tiene como objetivo principal el caracterizar métodos de generación de *proposals* para su uso en el reconocimiento de objetos con redes CNN, comparando el desempeño tanto de los *proposals* generados como del sistema completo en bases de datos fabricadas manualmente. Para estudiar el sistema completo, se comparan dos estructuras conocidas, llamadas R-CNN y Fast R-CNN, que utilizan de distintas formas ambas técnicas (generación de *proposals* y detección) y donde se considera en el estado del arte *mejor* Fast R-CNN. Se propone en este trabajo que esta hipótesis no es del todo cierta en el caso de que se trabaje con un número suficientemente bajo de *proposals* (donde las bases de datos acá construidas se enfocan en precisamente asegurar una cantidad baja de objetos de tamaños similares presentes en cada una: objetos sobre superficies y objetos de una sala de estar) y se acelere el proceso de clasificación alterando el tamaño de entrada de la red convolucional utilizada.

Se eligieron tres métodos de generación de *Proposals* de la literatura a partir de su desempeño reportado, y fueron comparados en distintos escenarios sus tiempos de procesamiento, calidad de *proposals* generados (mediante análisis visual y numérico) en función del número generados de estos. El método llamado BING presenta una ventaja sustancial en términos del tiempo de procesamiento y tiene un desempeño competitivo medido con el recall (fracción de los objetos del *ground truth* correctamente detectados) para las aplicaciones escogidas. Para implementar R-CNN se entrenan dos redes del tipo SqueezeNet pero con entradas reducidas y seleccionando los 50 mejores *proposals* generados por BING se encuentra que para una red de entrada 64x64 se alcanza casi el mismo recall ($\sim 40\,\%$) que se obtiene con el Fast R-CNN original y con una mejor precisión, aunque es 5 veces más lento (0.75s versus 0.14s).

El sistema R-CNN implementado en este trabajo, entonces, no sólo acelera entre 10 y 20 veces la etapa de generación de *proposals* en comparación a su implementación original, si no que el efecto de reducir la entrada de la red utilizada logra disminuir el tiempo de detección a uno que es sólo 5 veces más lento que Fast R-CNN cuando antes era hasta 100 veces más lento y con un desempeño equivalente.

A mí, por sobrevivir.

Agradecimientos

Agradezco en primer lugar a mi familia. A mis padres, porque nunca me faltó nada, porque nunca me cuestionaron, me apoyaron y me motivaron a ser lo que he llegado a ser. A mi hermano, por estar siempre conmigo, por la complicidad de tantos años, y por todo el orgullo mutuo que hemos construido.

A Alex, compañero de los últimos dos años que ha estado conmigo en estos procesos de cambio, suyos y míos, haciendo los días más fáciles y hermosos.

A mis amigos: Álvaro por ser mi compañero constante los primeros 4 años de carrera, y presente hasta el último día, siempre te tendré en mi corazón. A Hormi y Bastián, que han sido parte de este crecer desde el primer día. A los *champis*: Eileen, Víctor, Nicolás, Matías, Edo, Vermeersch, Nacha, Dani y Raúl; mis primeros amigos en eléctrica que no sólo hicieron de la carrera un camino más interesante y entretenido, sino también de la vida. A Pelao (José), Iván, Vikingo (Ignacio), Pelo (Pablo), Caro Mayol, Gonzalo y Javier Rojas; que también marcaron mi corazón con su amistad. Un abrazo a todos.

Tabla de Contenido

Índice de Tablas

Índice de Ilustraciones

Capítulo 1

Introducción

Motivación

El reconocimiento de objetos es un área de la visión computacional y procesamiento de imágenes que, como su nombre indica, tiene el fin de reconocer si están presentes determinados objetos en imágenes, y en qué partes de éstas se encuentran.

Esto, que es algo que los humanos hacen fácilmente, no es trivial computacionalmente, y las técnicas clásicamente utilizadas requieren hacer uso de ventanas deslizantes: buscar exhaustivamente el objeto (a partir de la extracción de características o algún tipo de *matching*) comparando el objeto buscado con "ventanas"de la imagen, como se ve en la figura 1.1. Estas ventanas deben recorrer la imagen completa y, si se quisiera reconocimiento a diferentes escalas, iterar con ventanas a distintas escalas.

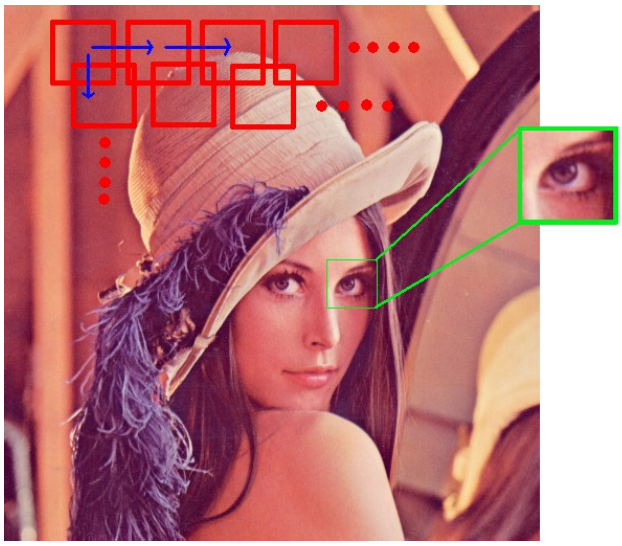


Figura 1.1: Ejemplo de Ventana Deslizante.
Fuente: https://rvlab.icg.tugraz.at/project_page/project_efficient_ncc/project_efficient_ncc.htm

Para acelerar el proceso de detección de objetos y disminuir los costos asociados, recientemente se ha introducido la técnica de generar *Object Proposals* como paso previo al reconocimiento en sí mismo. Consiste en, dada una imagen de entrada, entregar como salida un número de regiones en ella que tengan una alta probabilidad de contener objetos. Por ejemplo, en la figura 1.2 se puede apreciar lo que sería una generación exitosa de *Proposals* en objetos tipo Auto.

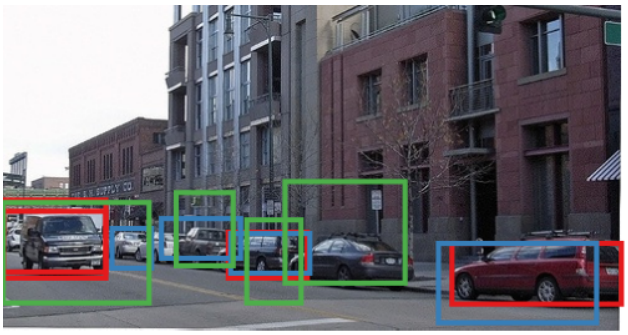


Figura 1.2: Ejemplo de Generación de Proposals.
Fuente: Hosang et al.[1]

Por otro lado, en los últimos años ha habido un crecimiento importante del uso de redes *Deep Learning* para resolver distintos tipos de problemas de reconocimiento de patrones, entre ellos la localización y detección de objetos donde se usa un tipo particular de red llamadas redes convolucionales (o CNN como su nombre en inglés, *Convolutional Neural Networks*).

El utilizar una mezcla de ambas técnicas para el reconocimiento de objetos no es algo nuevo, y hay estructuras definidas de como implementarlo. Sin embargo, no hay claridad conceptual ni práctica consensuada acerca de lo que caracteriza una buena generación de *Object Proposals* para cualquier escenario, y por ende, qué estructura de sistema resultaría más conveniente para cualquier escenario. Los trabajos en este tema usualmente están orientados a rendir bien sobre una o más bases de datos sin considerar qué sesgos trae esto. En particular, se encuentran dos arquitecturas en la literatura con alto desempeño sobre bases de datos reconocidas, R-CNN y Fast R-CNN (cuyo detalle se presentará en el siguiente capítulo), que abordan el problema con estas técnicas pero no toman en cuenta la influencia de la generación de *Proposals* (particularmente, la cantidad generada en función de la aplicación) al momento de afirmar que uno es mejor que el otro. Hay una tercera arquitectura (Faster R-CNN) que si bien muestra mejores resultados, se escapa del alcance del análisis que se desea realizar en este trabajo, pues en esta se realiza la generación de *Proposals* como tarea de la red convolucional, con lo cual no es posible estudiar la influencia del número de *Proposals* en el desempeño.

Objetivo General

Como se mencionó anteriormente, los métodos actuales que hacen uso simultáneo de las técnicas de generación de *Proposals* y redes CNN para la detección de objetos, no toman en cuenta ni la cantidad de *Proposals* generados ni la arquitectura de las redes, y cómo se relacionan estos factores con el problema que se quiere resolver (pues estos trabajos están orientados a rendir bien en bases de datos masivas y no en aplicaciones particulares). Esto tiene una implicancia directa en como se ve afectada la rapidez de los métodos.

En función de esto, el objetivo general de este trabajo es analizar el efecto de los factores mencionados en la rapidez de detección, comparando el desempeño de dos de las arquitecturas con mejores desempeños reportados y cuyo diseño puede verse directamente afectado por estos, R-CNN y Fast R-CNN, adaptando el sistema considerado *peor* (R-CNN) para obtener un resultado igual o mejor al reportado por Fast R-CNN (en tiempo de ejecución y desempeño de detección).

Objetivos Específicos

- Caracterizar aplicaciones en las que se pueda asegurar que con pocos *Proposals* se obtenga un buen desempeño. Esto es, básicamente, pocos objetos a detectar por imagen.
- Elegir un método existente de generación de Object Proposals desde el estado del arte para la evaluación de desempeño final. Para esto se necesita:
  - Realizar un estudio previo de métodos existentes y comparar resultados reportados.
  - Seleccionar métodos con mejor desempeño reportado y caracterizar la generación de *Proposals* en función del tiempo de ejecución y recall por imagen tanto en una base de datos conocida (VOC 2007) como en las aplicaciones determinadas.
  - A partir de estos resultados realizar la elección de un método de generación de *Proposals* adecuado para la implementación de las arquitecturas a comparar.
- Implementar los sistemas R-CNN y Fast R-CNN para comparar su desempeño. Para esto se debe:
  - Seleccionar las redes convolucionales a utilizar en cada sistema.
  - Identificar un límite teórico que iguale el desempeño en tiempo de ejecución entre ambos sistemas.
  - Adaptar R-CNN para cumplir el límite teórico.
- Comparar desempeño de detección entre ambos sistemas sobre las aplicaciones determinadas.

Alcances

Para el desarrollo de este sistema se hará uso de métodos existentes, disponibles al público y serán adaptados según los objetivos planteados. Las imágenes sobre las cuales se trabajará para entrenar en caso de ser necesario, validar y caracterizar el sistema serán RGB, y no se considerarán imágenes de profundidad u otro tipo. El desempeño será medido en términos de precisión y tiempo de procesamiento, pero no necesariamente en uso de recursos computacionales.
Todos los experimentos realizados fueron en un computador ASUS X556U con Intel Core i5-6200U CPU, 2.30GHz x 4 y GeForce 940MX para el uso de GPU.
Los generadores de *proposals* fueron adaptados de sus códigos originales disponibles en Github. Para la red convolucional de R-CNN se utiliza el framework *darknet* [5] (C) mientras que para la red de Fast R-CNN se utiliza el framework caffe [6] y su código original [7] (Python)

Capítulo 2

Marco Teórico

Para entender este trabajo de título se deben definir y explicar los conceptos y métodos relevantes para el mismo. Estos pueden dividirse en dos secciones, haciendo referencia al título de este trabajo: la generación de *Objecto Proposals* y Deep Learning.

2.1. Generación *Object Proposals*

La generación de *Object Proposals* es la técnica que se ha propuesto como alternativa a la clásica ventana deslizante para obtener regiones en las que haya alta probabilidad de encontrar un objeto. Pero, ¿qué es un objeto? Esta es la primera definición que debemos fijar para poder entender el problema. A continuación, se presentan distintos enfoques para la generación de *Proposals*, ejemplos correspondientes y un análisis comparativo. Finalmente, se describirán los tres métodos escogidos como alternativas a ser utilizados en el sistema final.

2.1.1. ¿Qué es un Objeto?

Alexe et al. ([8], [9]) proponen una definición de objetos que es referenciada frecuentemente en trabajos posteriores al mismo: *"Los objetos son cosas autónomas con un borde y centro bien definidos,..., en oposición a cosas amorfas de fondo"*, adicionalmente, señalan que un objeto debe tener *al menos* una de las siguientes características:

- Un borde cerrado bien definido en el espacio.
- Una apariencia diferente a lo que lo rodea.
- Es único dentro de la imagen y resalta como saliente.

[10] y [11] señalan que una buena medida de *Objectness* (o un buen método de generación de *proposals* en general) debe cumplir las siguientes características:

1. Alta tasa de detección de objetos.

2. Pequeño número de *Proposals*.
3. Gran eficiencia computacional.
4. Buena habilidad de generalización.

Otro enfoque que define un objeto es descrito en [12], que plantea que las imágenes son intrínsicamente jerárquicas. Por ejemplo, en una escena de un comedor, un tenedor está sobre un plato, el plato se encuentra sobre una mesa; para distintos contextos, entonces, *mesa* puede referirse a la mesa por sí misma o todo lo que contiene.

Hosang et al. [1] realiza un extensivo estudio de evaluación de métodos de generación de *proposals*, basándose en que hasta la fecha de su publicación (2015), tanto en Pascal VOC como en ImageNet, los mejores resultados obtenidos fueron con el uso de Object Proposals. Los códigos de evaluación están disponibles en github [1]. Se evalúan *class-agnostic proposals* (detección independiente de clase) para la detección de *Bounding Boxes* (literalmente *cajas envolventes*, es decir, cuadros en la imagen) en imágenes, y donde los resultados de segmentación se convierten a *Bounding Boxes*. Los métodos de generación de Object Proposals los clasifica en dos grupos, de agrupación (generalmente basados en la segunda noción de objeto descrita anteriormente) y en métodos de puntuación de ventanas (generalmente basados en la primera noción de objeto descrita anteriormente).
Para la evaluación, en métodos donde no se tiene control directo sobre el número de *proposals*, se utiliza el top k de ventanas generadas, se controla k indirectamente desde los parámetros, o se toma un k aleatorio (en caso de no estar ordenados según puntuación), además, se eliminan las ventanas duplicadas en caso de que existan.

A continuación, se describen los distintos métodos de generación de proposals basados en agrupación y en puntuación de ventanas estudiados en [1].

2.1.2. Métodos de Agrupación

La característica de este tipo de métodos es generar múltiples, y posiblemente superpuestos, segmentos que podrían pertenecer a objetos, y se les clasifica en tres tipos según cómo generan los *proposals*: Métodos que generan *proposals* a partir de agrupar superpixels (SP), usualmente usando [13]; métodos que resuelven múltiples problemas de *graph cut* (GC) con diversos seeds; y métodos que generan *proposals* directamente a partir de los bordes de la imagen (EC), como por ejemplo de [14] y [15]. Los métodos citados en [1] son los siguientes:

- **Selective Search** (SP) [16], [12]: Como se dijo anteriormente, este método asume una jerarquía en las imágenes. Utiliza una diversificación de estrategias (distintos criterios de agrupamiento y espacios de color) para unir superpixels (obtenidos de [13]).
- **Randomized Prim's** (SP) [17]: Algoritmo para generar *proposals* basado en el algoritmo de Prim, llamado Randomized Prim's, en donde se unen los superpixels obtenidos de [13] con pesos en los bordes que representan la verosimilitud de un objeto.
- **Rantalankila** (SP) [18]: Similar a Selective Search pero utiliza distintos *features* (características), como SIFT.

[1] https://github.com/hosang/detection-proposals

- **Chang** (SP)[19]: Combina saliencia y el concepto de Objectness [8] con un modelo gráfico para unir superpíxels.
- **CPMC** (GC)[20], [21]: Evita la segmentación inicial y calcula *graph cuts*. Explora el espacio de regiones que pueden inferirse de medidas locales y luego, en el espacio de regiones generadas utiliza una combinación de *features* avanzados y de bajo nivel para inducir un ranking más preciso de cada región en términos de probabilidad de mostrar regularidades de tipo objeto.
- **Endres** (GC)[22], [23]: Segmentación jerárquica a partir de bordes de oclusión. Propone que la geometría estimada y los bordes (más el color y textura) pueden ser usados para recuperar los bordes de oclusión de objetos. Proponen este enfoque a partir de estudios que señalan que *un sistema visual puede tener un sistema de localización de objetos funcional sin tener un sistema de identificación de objetos.* (Supuesto asumido por la técnica de uso de *proposals* en general)
- **Rigor** (GC)[24]: Variante de CPMC que acelera el tiempo de computación usando un detector de bordes más rápido ([15]).
- **Geodesic** (EC)[25]: Sobresegmentación de la imagen basada en [15]. Se usan clasificadores que plantan *seeds* para una transformada geodésica. Conjuntos de nivel de cada transformada de distancia definen las segmentaciones de *proposals*.
- **MCG** (EC)[26]: Segmentación jerárquica a partir de [15]. Los segmentos se unen según la intensidad de los bordes y las ventanas resultantes se puntúan de acuerdo a características como la forma, tamaño, locación, intensidad de bordes.

2.1.3. Métodos de Puntuación de Ventanas

Estos métodos puntúan cada ventana candidata según cuán probable es que contengan un objeto. Sólo retornan *Bounding Boxes* y tienden a ser más rápidos.

- **Objectness** [8], [9]: Uno de los primeros y más conocidos métodos de generación de *proposals*. Se selecciona un conjunto inicial de *proposals* de acuerdo a lugares salientes (a partir de la FFT (*Fast Fourier Transform*) espectral residual), los que luego se puntúan de acuerdo a características como color, bordes, ubicación, tamaño, y *superpixel straddling* (lo que definen como *Objectness*).
- **Rahtu** [27]: Utilizando un enfoque similar a Objectness, inicia con regiones de *proposals* generadas a partir de superpixels, pares, y tripletas de superpixels, las cuales puntúa según su *Objectness* pero con características adicionales: integral de borde de superpixel, distribución de borde de frontera y simetría de ventana.
- **BING** [10]: Utiliza un clasificador lineal sobre *features* de bordes, el cual se aplica con el paradigma de ventana deslizante.
- **EdgeBoxes** [11]: Trabaja sobre el supuesto de que el número de contornos totalmente contenidos por un *Bounding Box* es indicativo de la probabilidad de que dicho *Bounding Box* contenga un objeto.
- **Feng** [28]: Define la detección de objetos salientes como la detección de objetos que son "distintos en cierta medida", pero no aquellos camuflados, muy pequeños o oclusos.

Genera *proposals* a partir de objetos salientes y los puntúa según medidas de saliencia.

- **Zhang** [29], [30]: Ranking SVMs sobre características de gradiente.

- **Randomized Seeds** [31]: Supone que un *Bounding Box* tiene más probabilidad de contener un objeto cuando hay una línea cerrada en O (imagen que indica cuáles píxeles de Randomized Seeds están de acuerdo en que hay un borde de superpixel) que se ajusta al *Bounding Box*.

2.1.4. Métodos de generación de *proposals* seleccionados

La mayoría de los métodos fueron entrenados y evaluados con la base de datos Pascal VOC (base de datos para tareas de clasificación y detección, cuenta con cerca de 10000 imágenes para 20 clases de objetos), lo que genera la duda de qué tan generalizable resulta un método. Esta duda ha sido discutida en [1] y [32], particularmente este último plantea que usar el protocolo de evaluación típico (usar una base de datos parcialmente anotada como Pascal VOC para calcular el desempeño en términos del recall) es un buen protocolo cuando se trata de usar un método como parte de un pipeline de detección (como es este caso) pero no para tareas que sean independiente de las categorías, mientras que [1] verifica esta noción evaluando cada método en las bases de datos ImageNet (más de 10 millones de imágenes con 1000 clases de objetos) y MSCOCO (2.5 millones de imágenes con 91 clases de objetos), además de evaluar la repetilibidad de cada uno,

Un resumen de los resultados de [1] se ve en la tabla 2.1, para cuyo análisis se definen los siguientes conceptos:

- **Recall** Medida estadística que representa la razón entre verdaderos positivos y la suma de verdaderos positivos y falsos negativos. En otras palabras, es el porcentaje de muestras correctamente clasificadas con respecto a lo real, sin considerar el impacto de generar falsos positivos.

- **Repetibilidad** [1] lo define como la capacidad de ser consistente en los resultados para imágenes levemente alteradas (resolución y ruido).

- **Precisión de Detección** Medida estadística que representa la razón entre verdaderos positivos y la suma de verdaderos positivos y falsos positivos.

Tabla 2.1: Tabla de comparación de los distintos métodos evaluados en [1] según los experimentos realizados en él, donde '-' indica sin información, y '*','**','***' indican de mal a mejor los resultados
Fuente: Hosang et al. [1]

Método	**Enfoque**	**Tiempo [s]**	**Repetibilidad**	**Recall**	**Detección**
BING	**Puntuación**	**0.2**	***	*	-
CPMC	Agrupación	250	-	**	*
EdgeBoxes	**Puntuación**	**0.3**	**	***	***
Endres	Agrupación	100	-	***	**
Geodesic	Agrupación	1	*	***	**
MCG	Agrupación	30	*	***	***
Objectness	Puntuación	3	-	*	-
Rahtu	Puntuación	3	-	-	*
RandomizedPrim's	Agrupación	1	*	*	**
Rantalankila	Agrupación	10	**	-	**
Rigor	Agrupación	10	*	**	**
SelectiveSearch	**Agrupación**	**10**	**	***	***

De la tabla 2.1 y de las conclusiones extraídas en [1] se destacan tres métodos: EdgeBoxes y SelectiveSearch, pues ambos presentan un alto recall y precisión y una aceptable repetibilidad; y BING, por otro lado, es el método más rápido y con la más alta repetibilidad.

Se realizan experimentos preliminares (tabla 2.2) sobre la implementación de estos métodos en la base de datos VOC 2007 [33] y los resultados coinciden particularmente en el buen desempeño de SelectiveSearch y EdgeBoxes. Por otro lado se ve que en el resultado preliminar el tiempo de ejecución requerido por SelectiveSearch es considerablemente menor que los que se ven en la tabla 2.1 y que a priori BING también presenta alto recall.

Tabla 2.2: Tabla de Comparación

Método	**s/imagen**	**Recall 200 Boxes**	**Recall 400 Boxes**	**Recall 800 Boxes**	**Recall 1400 Boxes**
BING [10]	0.0128	0.851	0.905	0.949	0.968
EdgeBoxes [11]	0.4358	0.860	0.920	0.960	0.970
SelectiveSearch [16] [12]	3.0414	-	-	-	0.967

A continuación se entregará una descripción más acaba de estos métodos, tanto como en su metodología como en los resultados reportados por sus autores.

,

2.1.5. BING

Se utiliza una versión binarizada de normas de gradientes (*binarized normed gradients*, BING) como *features* de 8x8 sobre distintas ventanas de la imagen de entrada para aprender un modelo $\mathbf{w} \in \Re^{64}$ que permite darle un puntaje a cada ventana [10].

Se escanea la imagen sobre tamaños de ventana i predefinidos. Cada ventana es puntuada con un modelo lineal $\mathbf{w}$,

$$s_l = \langle \mathbf{w}, \mathbf{g}_l \rangle \tag{2.1}$$
$$l = (\mathrm{i}, x, y) \tag{2.2}$$

donde s_l, g_l, l, i y (x, y) son el puntaje del filtro, la norma de gradiente, localización, tamaño y posición de la ventana, respectivamente. Usando *non-maximal suppression (NMS)*, se selecciona un conjunto de *proposals* para cada tamaño i. Notando que algunos tamaños tienen mayor probabilidad de presentar objetos, se define el puntaje calibrado como

$$o_l = v_\mathrm{i} \cdot s_l + t_\mathrm{i} \tag{2.3}$$

donde v_i, $t_\mathrm{i} \in \Re$ son un coeficiente y un *bias* independientemente aprendidos para cada tamaño i.

El proceso de aprendizaje sigue la idea general de dos SVM en cascada:

1. Se aprende el modelo $\mathbf{w}$ de la ecuación (2.1) usando un SVM lineal. Se utilizan los *features* de normas de gradientes de las ventanas correspondientes a objetos del *ground truth* y de ventanas aleatorias correspondientes a fondo (*background*) como muestras positivas y negativas, respectivamente.
2. Para aprender v_i y t_i de la ecuación (2.3) usando un SVM lineal, se evalúa la ecuación (2.1) a un tamaño i para imagenes de entrenamiento y se usan los *proposals* seleccionados con NMS como muestras de entrenamiento con sus puntajes de filtro s_l como *features*.

La figura 2.1d) muestra como se ve $\mathbf{w}$ y es posible concluir que los pesos más grandes del borde favorecen la detección de bordes de objetos que delimitan estos con el fondo.

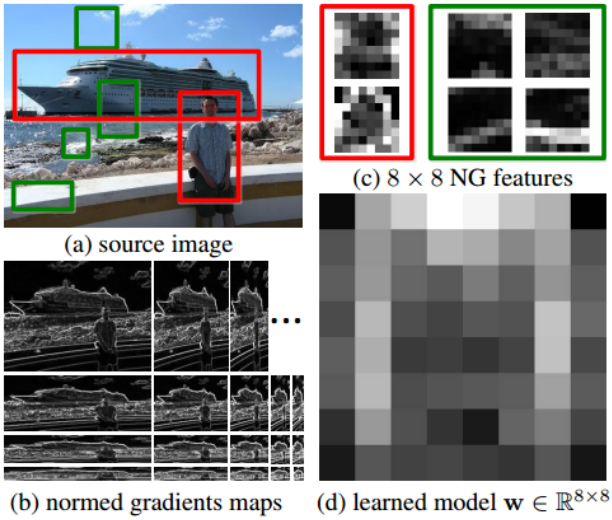


Figura 2.1: En a) se ve la imagen de entrada. b) muestra los mapas de normas de gradientes para la imagen en distintas escalas y *aspect ratios*. c) es un ejemplo de lo que sería un *feature* de 8x8. d) es un ejemplo del mapa del modelo lineal aprendido.
Fuente: Cheng et al.[10]

Se justifica el uso de normas de gradientes ya que, como se ve en la figura 2.1b), incluso a distintos tamaños de ventanas, los objetos presentes en la imagen vistos como bordes cerrados presentan poca variación en esta forma de representación, pues los *features* de normas de gradientes son insensibles a la traslación, escala y razón de aspecto.

Entrenado y evaluado con VOC 2007 reporta resultados en los que el sistema es capaz de obtener un *Detection Rate* de 99.5 % con 5000 *proposals* en 0.003 segundos.

2.1.6. EdgeBoxes

Es un método basado en la hipótesis de que un buen indicador de la presencia de un objeto en una ventana es el número de bordes totalmente contenidos dentro de ella. Se dice que un contorno está completamente contenido en la ventana si todos los pixeles en el borde pertenecientes al contorno yacen en el interior ventana.

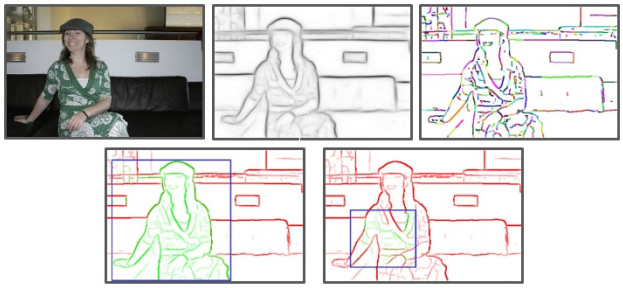


Figura 2.2: Ejemplo de como una imagen es procesada por EdgeBoxes. En la primera fila se ve la imagen original, la detección de bordes y la posterior agrupación de estos. En la segunda fila se ve lo que sería un *bounding box* correcto (izquierda) y lo que sería uno incorrecto (derecha). Los bordes en verde se predicen como parte de un objeto mientras que los bordes en rojo no.
Fuente: Zitnick&Dollár[11]

La metodología consiste en:

- Utilizar *Structured Edge Detector* [34] como primer detector de bordes.
- Clusterizar bordes colindantes similares según orientación y posición relativa generando *bounding boxes* (mediante una búsqueda con ventana deslizante). Dado un set de bordes $s_i \in S$. Para un par de grupos s_i y s_j, la afinidad entre ambas está sujeta a su posición media x_i y x_j y su orientación media θ_i y θ_j:

$$a(s_i, s_j) = |\cos(\theta_i - \theta_{ij})cos(\theta_j - \theta_{ij})|\gamma \tag{2.4}$$

  Donde θ_{ij} es el ángulo entre x_i y x_j. El valor de γ puede ser variado para la ajustar la afinidad dada por la orientación, usándose $\gamma = 2$ en la mayoría de los casos. Si los grupos de bordes están separados por más de dos pixeles la afinidad se fija en el cero.
- Calcular un puntaje a un bounding box. Para cada s_i se calcula un valor continuo $w_b(s_i) \in [0, 1]$ tal que si s_i está completamente contenido en b, $w_b(s_i) = 1$, por el contrario, si $s_i = 0$ no está contenido entonces $w_b(s_i) = 0$. Sea S_b el set de grupos de bordes que se superponen en la ventana b-ésima, S_b se calcula con una estructura de datos eficiente. Para cada $s_i \in S_b$, $w_b(s_i)$ es puesto en cero. Similarmente $w_b(s_i) = 0$ para cada s_i tal que $\overline{x}_i \notin b$ dado que todos esos pixels están afuera de b o $s_i \in S_b$. El resto de los grupos de bordes cuales $\overline{x}_i \in b$ y $s_i \notin S_b$ se calcula $w_b(s_i)$ como sigue:

$$w_b(s_i) = 1_m ax_T \prod_{j}^{|T|-1} a(t_j, t_{j+1}) \tag{2.5}$$

Donde T es un camino ordenado de bordes de grupos con largo $|T|$ que comienza con $t_1 \in S_b$ y termina $t_{|T|} = s_\mathrm{i}$. Si no existe tal camino, se define $w_b(s_\mathrm{i}) = 1$. Esta ecuación define la mayor afinidad entre los bordes de grupo s_i y un grupo superpuesto de bordes. Dado que la mayoría de las afinidades son cero, esto se puede hacer eficientemente. Usando los cálculos de valores de w_b se puede definir el *score o puntaje como:*

$$h_b = \frac{\sum_\mathrm{i} w_b(s_\mathrm{i}) m_\mathrm{i}}{2(b_w + b_h)\kappa} \tag{2.6}$$

Donde b_w y b_h es el largo y ancho de la ventana.

- *Se ocupa* non-maximal suppression *(NMS) para filtrar* bounding boxes *una vez están puntuados y ordenados.*

Para Pascal VOC 2007 reporta resultados de un recall del 96 % para un IoU (*Insersection over Union*) de 0.5 y un recall de sobre 75 % para un IoU de 0.7 con 1000 *proposals*. [11]

El método, además, presenta algunas variantes en las cuales se ajustan unos parámetros, α y β (que controlan el tamaño de las ventanas deslizantes y el umbral de *NMS*, respectivamente) que garantizan mejores resultados para distintos IoU y estos se encuentran detallados en la figura 2.3

	IoU = 0.5		IoU = 0.7		IoU = 0.9				
	AUC	Recall	AUC	Recall	AUC	Recall	Runtime	α	β
Edge boxes 50	**.64**	**96%**	.36	55%	.04	5%	**.25s**	.65	.55
Edge boxes 70	.58	89%	**.45**	**76%**	.06	9%	**.25s**	.65	.75
Edge boxes 90	.38	59%	.28	46%	**.15**	**28%**	2.5s	.85	.95

Figura 2.3: Resultados para las distintas variantes de Edge Boxes
Fuente: Zitnick&Dollár[11]

2.1.7. SelectiveSearch

Combinación de métodos de segmentación y búsqueda exhaustiva con el fin de explotar la estructura de la imagen para generar posiciones de posibles objetos y obtener todas las posibles posiciones. Utiliza estrategias de agrupamiento para ir fusionando superpixels con lo cual se generan *proposals*.

Primero se utiliza [13] para obtener un conjunto de regiones iniciales **R**. A continuación se agrupan las regiones: se calculan las similaridades entre todas las regiones colindantes y se agrupan las dos más similares. Se repite el proceso de calcular la similaridad entre todas las regiones hasta que toda la imagen sea una sola región. En cada iteración, la región obtenida de la unión de dos regiones es añadida al conjunto **R** el cual es finalmente el conjunto de ventanas propuestas.

Para el diseño del algoritmo además se utilizan tres estrategias para diversificar este proceso: se usan distintos espacios de color con diferentes propiedades de invarianza; se usan distintas medidas de similaridad (como color, textura, tamaño); y finalmente, variar las regiones iniciales.

Se entregan como *proposals* finales la unión de las ventanas obtenidas con múltiples combinaciones de estrategias de agrupación, ordenadas según la iteración del algoritmo en las que fueron generadas (favoreciendo las regiones de mayor tamaño) y, finalmente, eliminando las regiones duplicadas.

Figura 2.4: Ejemplo del algoritmo de agrupamiento jerárquico.
Fuente: Uijlings et al. [12]

De [12] se tiene que para Pascal VOC 2007 alcanzan un recall de 96.7 % con 1536 ventanas.

2.2. Deep Learning

Los métodos denominados *Deep Learning* aprenden a representar objetos desde un bajo nivel (una entrada sin procesar) hasta uno alto y más abstracto [35]. Esto se construye mediante varias *capas* que representan los distintos niveles de representación; de ahí el nombre *Deep Learning*: Aprendizaje profundo.[36].

El ejemplo típico, y el esencial, de un método del tipo *Deep Learning* son los perceptrones multi-capa, que es básicamente una función matemática que mapea una entrada a una salida; ésta es una composición (o red) de múltiples funciones no-lineales más simples [36] llamadas neuronas o perceptrones (como el de la figura 2.5), donde los pesos asociados a cada una en la composición se entrenan (típicamente mediante *Backpropagation*) para aprender la función que relaciona las entradas con las salidas de un problema de aprendizaje supervisado.

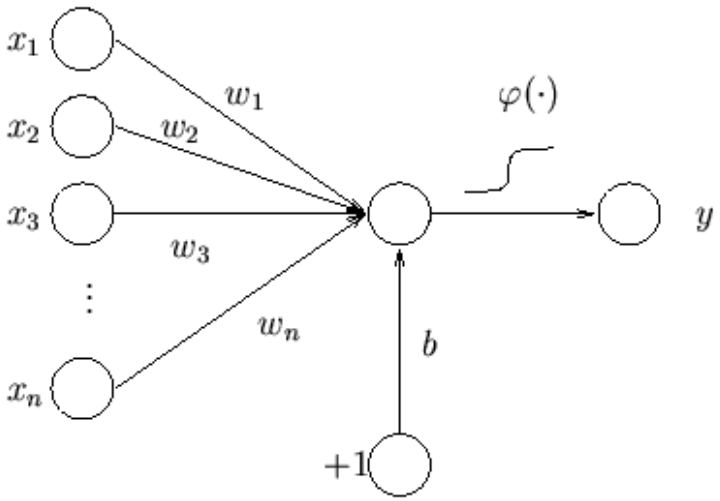


Figura 2.5: Modelo de un perceptrón donde $\{x_1, ..., x_n\}$ es el vector de entrada, $\{w_1, ..., w_n\}$ el vector de pesos, b un sesgo y φ la función no-lineal de activación.
Fuente: https://www.hiit.fi/u/ahonkela/dippa/node41.html

En la figura 2.6 se aprecia un ejemplo de cómo se construye un modelo *Deep Learning* y cómo aumenta la abstracción de las capas para la representación de, en este caso, una imagen: se reciben los píxeles sin procesar, luego se detectan los bordes de la imagen y patrones espacio-frecuenciales, a continuación, esquinas y contornos, finalmente se pueden distinguir partes de objetos con lo cual es posible clasificar qué es lo que hay presente en la imagen.

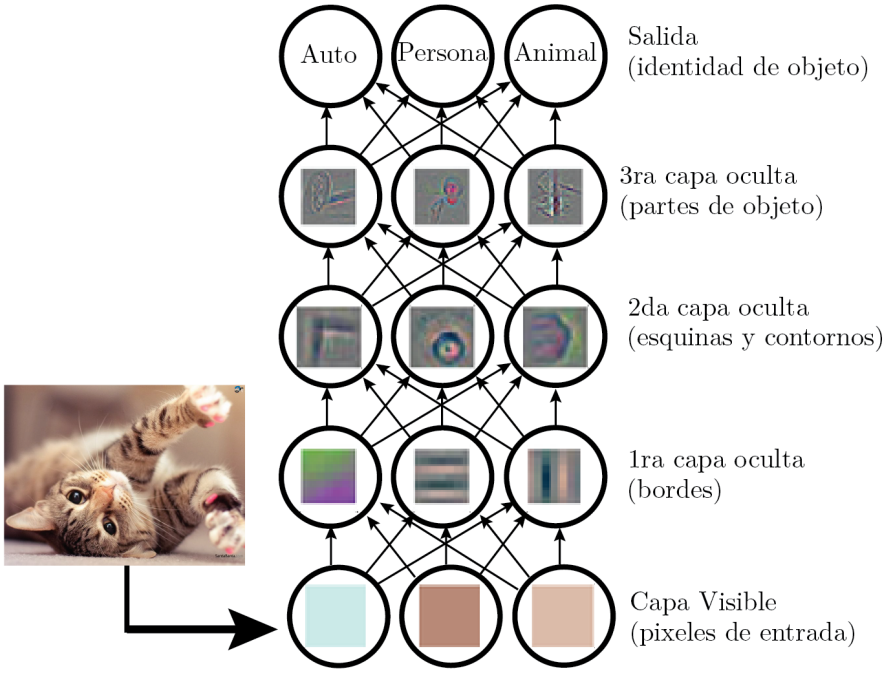


Figura 2.6: Ejemplo de un modelo *Deep Learning*.
Fuente: Goodfellow et al. [36]

Es importante destacar que, en este tipo de modelos, las características que se usan para las tareas de aprendizaje no son diseñadas a mano, sino que se aprenden a partir de los datos.

2.2.1. Redes Convolucionales

Para el procesamiento de imágenes, si se quisiera usar un enfoque clásico como el perceptrón multi-capa, surgen diversos problemas: estos modelos tienen un número altísimo de pesos, por lo que el error de generalización que puede ocurrir si los datos no son suficientes es potencialmente alto; además, no se dispone de invarianza con respecto a traslaciones o cualquier tipo de distorsión en las imágenes de entrada [37].

Una red convolucional contiene tres elementos que garantizan que exista un nivel de invarianza: campos receptores locales (*local receptive fields*), pesos compartidos, y, a veces, submuestreo temporal o espacial. Los campos receptores se refiere a que cada 'neurona' sólo procesa una parte de la entrada (el campo receptor), como una operación de convolución. Además, esto reduce el número de pesos a actualizar durante el entrenamiento, pues se utilizan las mismas máscaras en cada capa convolucional. Usualmente después de una capa convolucional se realiza una función de sub-muestreo o promedio (*pooling*) [37]. Para tareas de clasificación, además, suelen agregarse capas llamadas 'totalmente conectadas' (*Fully Connected*, FC) que corresponden a capas clásicas de tipo perceptrón, y/o se aplica la función

softmax para retornar un vector de probabilidades.

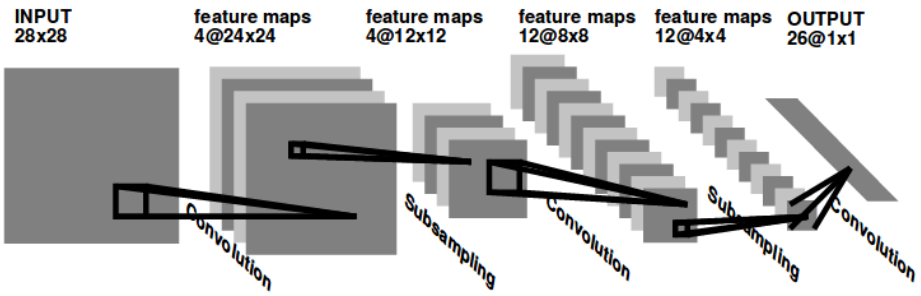


Figura 2.7: Le-Net-5: Ejemplo de red convolucional, utilizada para la clasificación de caracteres de escritura.
Fuente: Lecun&Bengio[37]

En la figura 2.7 se muestra la red convolucional utilizada por LeCun et al. [38] para reconocer caracteres escritos.

Este modelo fue propuesto el año 1990, pero debido a las limitaciones de hardware no fue popular sino hasta el año 2012 donde en la competencia ILSVRC [39], Krizhevsky et al. [40] desarrollaron una red convolucional (AlexNet) que logró clasificar el conjunto de test de ImageNet con una tasa de errores menor en la mitad de los métodos ganadores hasta ese momento.

2.2.2. Redes Utilizadas

En este trabajo de título se utilizarán dos redes convolucionales para la implementación de los sistemas finales. A continuación se detallan las arquitecturas de ambas.

AlexNet

Como muestra la figura 2.8 consta de 5 capas convolucionales y 3 totalmente conectadas. Entre las novedades que introdujo esta red se incluyen el uso de la función de activación ReLU (*Rectified Linear Units*, definida como $\varphi(x) = max(x, 0)$) justificada por la similitud entre esta y las funciones de activación de las neuronas cerebrales y, además, la aceleración que añaden al proceso de entrenamiento. Esta función es aplicada en la salida de cada capa convolucional y totalmente conectada.

Otra novedad es un entrenamiento más eficaz con el uso de múltiples GPUs.
La figura 2.8 muestra la estructura de la red y cómo esta se divide para entrenar con 2 GPUs: Los kernels (ventanas de convolución) de la segunda, cuarta y quinta capa convolucional están conectados solamente a los mapas convolucionales resultantes de la capa anterior que residen en la misma GPU. Los kernels de la tercera capa convolucional están conectados a todos los

mapas de la segunda capa. Las neuronas de las capas totalmente conectadas están conectadas a todas las neuronas de la capa anteior.
La salida de la última capa totalmente conectada entra a una función softmax que entrega una distribución sobre las 1000 clases de ImageNet.

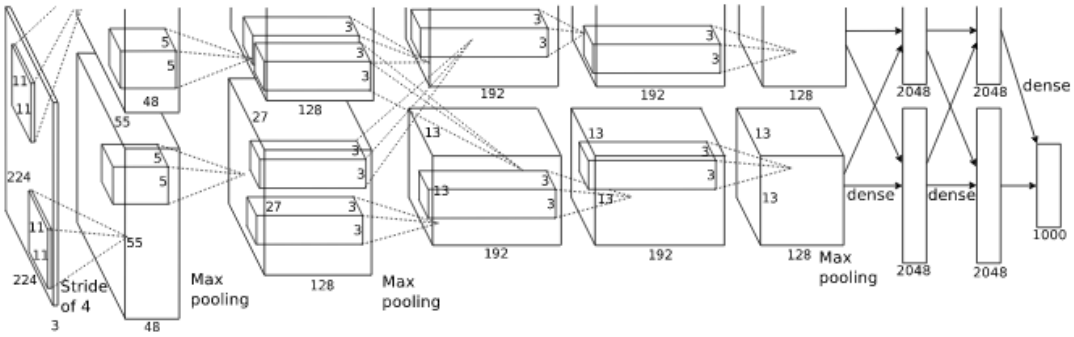


Figura 2.8: Estructura de AlexNet
Fuente: Krizhevsky et al.[40]

El diseño de esta red, además, incluyó las técnicas de *dropout* y *data augmentation* durante el entrenamiento.

Data augmentation, o aumento de datos, corresponde a la técnica de transformar las imágenes de entrenamiento para aumentar el tamaño de la base de datos y prevenir el sobreajuste. En este caso se utilizan dos formas de *Data augmentation*: una es tomar *parches* aleatorios de 224x224 de las imágenes de entrenamiento de 256x256 y utilizando estos como conjunto de entrenamiento; la segunda forma es alterar las intensidades de los canales RGB en las imágenes de entrenamiento.

Dropout consiste en fijar en 0 la salida de cada neurona con una probabilidad de 0.5. Las neuronas a las que le ocurre esto no contribuyen al paso hacia adelante por la red ni al *backpropagation*. Esto resulta en que para cada nueva muestra de entrenamiento, la red presenta una arquitectura distinta, pero cada arquitectura contiene los mismos pesos. Esta técnica reduce la co-dependencia entre neuronas, facilitando el aprendizaje de características más robustas.

Los resultados obtenidos en la competencia ILSVRC-2012 sentaron el precedente para el uso de redes convolucionales en aplicaciones de visión computacional.

SqueezeNet

SqueezeNet [41] contiene 50 veces menos parámetros que AlexNet con resultados iguales o superiores. Esta se inspira en el uso de módulos *inception* propuestos en la red GoogLeNet [42] que consisten en convoluciones de tamaño 1x1; estos se alternan con filtros de 3x3 y nombran esta combinación *fire modules* (ver figura 2.9).

La arquitectura consiste en una primera capa convolucional (conv1), seguida de 8 *fire modules* (fire2-9), cada uno de y finalizando con una capa convolucional final (conv10).

ReLU es utilizada como función de activación en las capas de los *fire modules*.

Se aplica *Dropout* a partir del módulo fire9.

Se aprovechará la capa final (que mediante un *Average Pooling* y *Softmax* entregan la probabilidad de pertenencia a cada clase) de la red para realizar la clasificación de cada *proposal*.

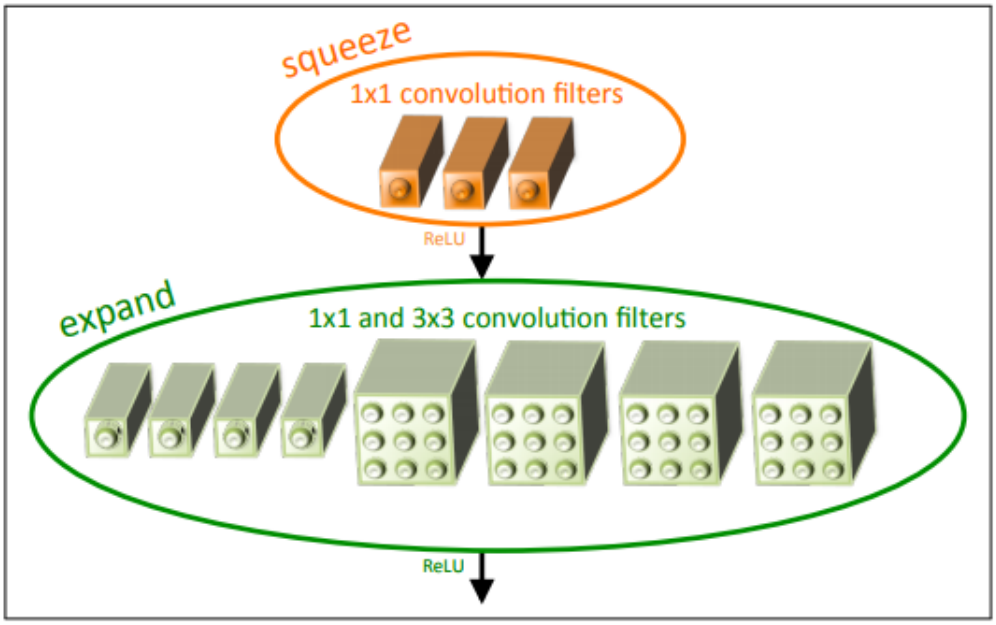


Figura 2.9: Organización de los filtros de convolución en un *fire module*
Fuente: Iandola et al.[41]

El que el número de parámetros sea tan bajo da lugar a la ventaja de poder entrenarla durante muchas iteraciones generando varios respaldos de los pesos generados, sin que esto sea un gasto computacional enorme en términos del uso de memoria.

2.2.3. R-CNN y Fast R-CNN

En este trabajo se quiere comparar los desempeños de determinados sistemas de detección de objetos que contienen generación de *Proposals* y redes CNN en su arquitectura, con tal de demostrar cómo influye la generación de *proposals* en estas como parte de su arquitectura y el tipo de problema a resolver. En particular, hay tres arquitecturas que hacen uso de generadores de *proposals* y redes CNN, y que han evolucionado desde un mismo origen: R-CNN [2], Fast R-CNN [3] y Faster R-CNN [4].
Como sus nombres sugieren, cada una se implementa con el objetivo de ser más rápida que la anterior y cada una ha destacado en los primeros lugares con su desempeño en conocidas bases de datos en los años que fueron publicadas..

En la figura 2.10 se ve a grandes rasgos el esquema del sistema R-CNN. A partir de una imagen de entrada se generan *proposals*, los cuales entran secuencialmente a una red convolucional que extrae caractrísticas de cada uno y los clasifica como un objeto con cierta probabilidad.

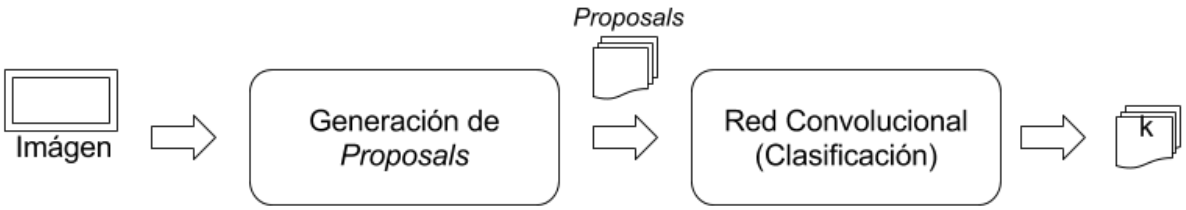


Figura 2.10: Arquitectura R-CNN [2]

En la figura 2.11 se ve el esquema de Fast R-CNN. De la imagen de entrada ocurren dos procesos que pueden ser realizados de forma independiente: la extracción de características de la imagen completa y la generación de *proposals*. A continuación se realiza una proyección de los *proposals* sobre el mapa de características y por cada uno de ellos se realiza una clasificación.

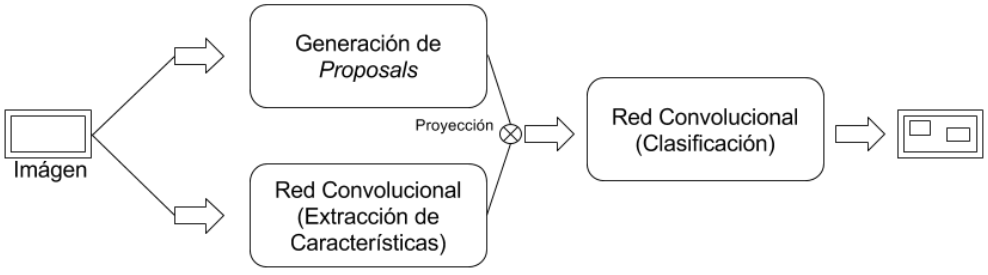


Figura 2.11: Arquitectura Fast R-CNN [3]

Finalmente, en la figura 2.12 se tiene el esquema de Faster-RCNN. Este funciona con la misma lógica que Fast R-CNN, con la diferencia de que en lugar de utilizar un método

externo de generación de *Proposals*, estos se generan a partir de las características extraídas de las capas convolucionales de la red utilizada.

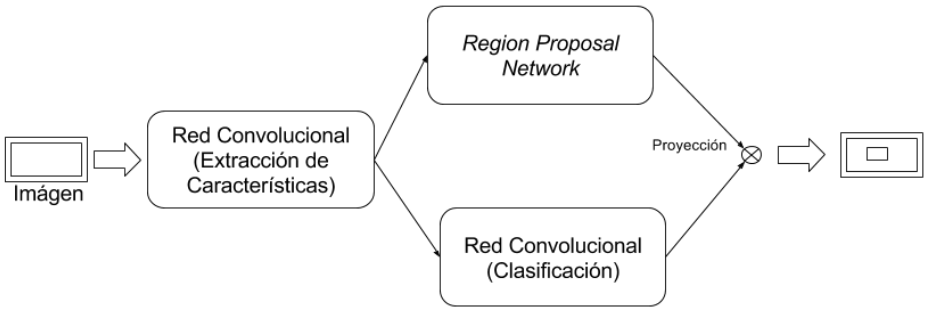


Figura 2.12: Arquitectura Fast R-CNN [3]

Se recuerda entonces que, como en este caso se realiza la generación de *Proposals* como tarea de la red convolucional, se escapa del análisis a realizar en este trabajo, pues es posible estudiar la influencia del número de *Proposals* en el desempeño y su relación con la arquitectura de red utilizada.

La comparación se realizará entre R-CNN y Fast R-CNN, sistemas cuyas arquitecturas originales se explican con mayor detalle a continuación.

R-CNN

Como se ve en la figura 2.13, este sistema consta de tres etapas:

1. **Extracción de *Proposals*:** Se generan alrededor de 2000 *proposals* utilizando Selective Search.
2. **Extracción de características:** Se utiliza una red CNN para extraer características de cada *proposal* generado. La red en el trabajo original es una AlexNet sin la última capa.
3. **Clasificación:** Se clasifican los *proposals* a partir de las características extraídas con SVM (*Support Vector Machine*).

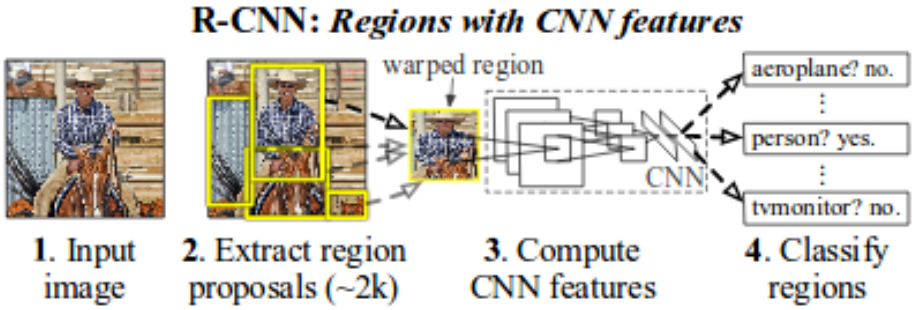


Figura 2.13: Sistema de detección R-CNN
Fuente: Girschick et al.[2]

Este sistema reporta un buen desempeño en las bases de datos en las cuales fue probado pero no es eficiente en el tiempo. Particularmente, por el uso de Selective Search, el cual ya se vio que puede ser muy lento en comparación a otros métodos, y porque se debe hacer un paso por la red para cada *proposal* generado, donde estos adicionalmente deben ser redimensionados al tamaño de entrada de la red (227x227).

Fast R-CNN

Como se ve en la figura 2.14 la arquitectura es más compleja y consta de las siguientes etapas:

1. Se obtienen *proposals* para una imagen, también con Selective Search.
2. Estos *proposals*, también llamados regiones de interés (*regions of interes*, ROI) entran a una red convolucional como proyecciones de la imagen.
3. Cada región pasa por una capa de *pooling* y es mapeada a un vector de características mediante capas totalmente conectadas.
4. Cada región entrega finalmente dos salidas: la probabilidad de pertenencia a las clases y una regresión de *bounding boxes* por clase.

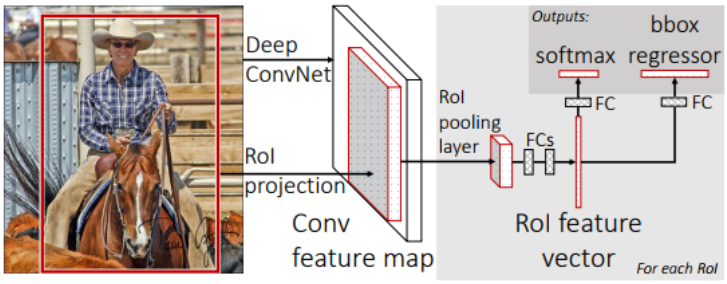


Figura 2.14: Sistema de detección Fast R-CNN
Fuente: Girshick et al.[3]

Este sistema fue implementado con tres redes: AlexNet, VGG-M y VGG16 (en donde la última capa totalmente conectada es reemplazada por lo indicado en la etapa 4), siendo el más rápido el de AlexNet, evidentemente dado el menor tamaño de la red.
Este sistema reporta un desempeño similar al de R-CNN pero siendo mucho más rápido, particularmente porque, como fue dicho, los *proposals* entran a la red simultáneamente como proyecciones de la imagen.

Capítulo 3

Metodología y análisis previos

En este trabajo se habla de la detección rápida de objetos utilizando *Object Proposals* y Deep Learning. Como se vio en el capítulo anterior, en la literatura se han destacado una serie de métodos de este estilo, y de estos se propone que la hipótesis de que Fast R-CNN es siempre mejor (en velocidad y desempeño) que R-CNN no está bien analizada.

Los problemas analizados en ambos trabajos originales [2][3] estaban orientados a rendir sobre bases de datos en las cuales el número de clases es alto (por ejemplo, las 1000 clases de ImageNet) o de muy distinta índole (como las 20 clases de Pascal VOC). Esto implica que para obtener un buen desempeño en la generación de *proposals* es necesario un número muy alto de estos (generalmente del orden de 10^3) si se busca detectar todos los objetos presentes en la imagen.
En este trabajo de título se propone presentar situaciones en las que el número de clases sea bajo, el número de objetos por imágen también, y además que sean de escenas similares. Esto nos permite plantear la hipótesis de que con un número mucho menor de *proposals* será posible localizar todos o la mayoría de los objetos.

Por otro lado, no se ha tomado en cuenta de que al ser los *proposals* procesados secuencialmente en el sistema R-CNN, el tamaño de entrada de las redes utilizadas en el trabajo original son ineficientemente grandes (227x227, típicamente), cuando el tamaño de los *proposals* pueden ser significativamente menor. Reducir el tamaño de la entrada de la red utilizada disminuiría considerablemente el tiempo de ejecución en la primera capa.

Esta propuesta la podemos extraer del anális de las ecuaciones (10,11,13 y 14) expuestas en [46]:
Sea t_{R-CNN} el tiempo que toma el pasar una imagen por el sistema:

$$t_{R-CNN} \simeq t_{rp} + t_{CONV} + t_{FC} \tag{3.1}$$

$$t_{CONV} \propto N_r \cdot S_x \cdot S_y \tag{3.2}$$

con t_{rp} el tiempo que toma generar los *proposals*, t_{CONV} el tiempo asociado al paso por la parte convolucional red, t_{FC} el tiempo al paso por las capas totalmente conectadas, $Sx \cdot S_y$ el tamaño de la entrada y N_r el número de *proposals*.

Análogamente para Fast R-CNN se tiene:

$$t_{FastR-CNN} \simeq t_{rp} + t_{CONV'} + t_{FC'} \tag{3.3}$$

$$t_{CONV'} \propto \cdot S_x \cdot S_y \tag{3.4}$$

De estas ecuaciones se ve una relación directa entre el tamaño de la entrada de la red y el tiempo de ejecución.

En base a estas hipótesis y los objetivos propuestos, este trabajo de título se puede subdividir en las etapas descritas a continuación.

3.1. Metodología experimental

Como fue dicho, es necesario determinar aplicaciones que cumplan, en lo posible, las siguientes condiciones:

- Pocas clases de objetos.
- Pocos objetos por imagen.
- Objetos de escenas similares.

La primera aplicación escogida corresponde detectar objetos sobre mesas o estantes. Esta cumple con la condición de tener pocos objetos de interés en ella y además siguiendo un patrón definido en su ubicación (lo que facilita la localización con pocos *proposals*).

La segunda aplicación escogida corresponde a detectar objetos dentro de una escena de interiores (sala de estar).Esta cumple, también, con pocos objetos por imagen, y, adicionalmente, un número bajo de clases.

Para evaluar el desempeño de ambas aplicaciones, se confecciona una primera base de datos (A) a partir de imágenes seleccionadas de VOC2007, de búsqueda en repositorios online y de la RoboCup2016, teniéndose un total de 73 imágenes; y una segunda base de datos (B), consistentes de imágenes de Instagram [43] correspondientes a escenas de salas de estar, con 147 imágenes. Notar que como conjuntos de prueba el número de imágenes en cada una de las base de datos construidas es bastante bajo, por lo cual es importante señalar que los resultados presentados serán más bien una demostración preliminar que totalmente representativos.

Estas base de datos deben etiquetarse manualmente con la posición de los objetos presentes y el nombre de los mismos, para ello se usa un toolbox de etiquetado [44] disponible en github con un *branch* especializado para etiquetar con múltiples clases [45].

Las anotaciones en la base A corresponden a todos los objetos presentes en la imagen, lo que implica un alto número de tipos (o clases) de objetos, pero como se explica más adelante, esto no será de importancia. Por otro lado, para la base B se etiquetan los objetos correspondientes a las siguientes clases:

- Botella (*Bottle*)

- Silla (*Chair*)
- Plantas (*Potted Plant*)
- Sofá
- Mesa (*Table*)
- TV/Monitor

A partir de este etiquetado evaluar la calidad de la generación de *proposals* (mediante el recall) eligiendo las regiones propuestas que cuenten con una IoU (explicada a continuación) mayor a 0.5.

Se define IoU (*Intersection over Union*, o intersección sobre unión) como la razón entre la intersección de dos áreas (en este caso, la real y la generada) y la unión de las mismas, como muestra la figura 3.1:

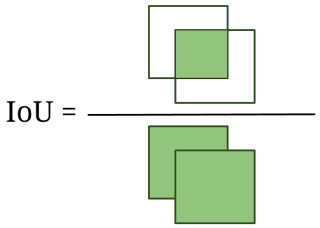


Figura 3.1: Intersección sobre unión (IoU)

El desempeño obtenido debe medirse en función de los parámetros del método y el número de *proposals*, con el fin de luego ajustar el sistema completo para asegurar un mejor rendimiento.

3.2. Caracterización y elección de métodos de generación de *proposals*

Como ha sido mencionado anteriormente, el objetivo es poder generar *proposals* de forma tal que con un bajo número de estos se obtenga un alto recall.
En la sección anterior se seleccionaron tres métodos de generación de *proposals* (BING, EdgeBoxes y SelectiveSearch) en función de su alto desempeño preliminar en términos de recall, precisión, repetibilidad y tiempo de ejecución (tabla 2.1 y tabla 2.2). Para poder escoger uno de estos para que forme parte del sistema final se deben caracterizar en base a las siguientes métricas seleccionadas:

1. Tiempo de ejecución vs. Resolución, para lo cual se toma la misma imagen en distintos tamaños y se grafican los resultados.
2. Recall en VOC2007 vs Número de regiones generadas.

El punto 2 ya fue analizado en la tabla 2.2.
Estas caracterizaciones permitirán determinar qué método es más eficiente para cualquier tipo de imagen tanto en el tiempo (pues se busca que sea rápido) y una primera visión del desempeño según el número de ventanas generadas.

De estos resultados se debe entonces escoger un método y luego aplicar estas métricas (tiempo de ejecución y recall/regiones generadas) sobre las aplicaciones escogidas de forma tal de verficar y garantizar un buen desempeño en el sistema completo.

3.3. Implementación de Sistemas Completos

3.3.1. Implementación RCNN modificado

Como el objetivo final de este trabajo de titulo comprende validar la hipótesis de que la configuración R-CNN podría ser más conveniente que Fast R-CNN, es necesario para ésta una red convolucional que reciba una entrada proporcional al tamaño de los *proposals* generados, tal que no haya un costo adicional de redimensionar la imagen a un mayor tamaño, y por consiguiente también cálculos más complejos por el mismo tamaño. De este modo, la red debe entonces modificarse para cumplir con esta condición.

Para evaluar el sistema completo de la configuración tipo R-CNN no sólo basta con seleccionar un método de generación de *proposals*, sino que también una red convolucional apropiada para el problema a resolver. El trabajo original utiliza AlexNet [2], pero cuenta con el problema de que esta red tiene una entrada de tamaño 227x227, mientras que el tamaño de los *proposals* generados es, la mayoría de las veces, mucho menor, y, por lo tanto, usar esta red no significaría una ventaja si el objetivo es una detección rápida, pues cada proposal debe pasar por la red secuencialmente.

Como se ha dicho anteriormente, en este trabajo se usará SqueezeNet, y se implementa usando el framework llamado *Darknet* [5], que está escrito en C y CUDA con soporte de GPU. Para enfrentar el problema señalado en el párrafo anterior, se modifica la red provista por el *framework*, cuya entrada es de 226x226, para recibir entradas de 32x32 y de 64x64, es decir, dos redes distintas, para evaluar el impacto que tiene el tamaño de la entrada en el resultado.

El modificar la arquitectura de la red implica no poder acceder a pesos pre-entrenados por lo que se debe entrenar cada red independientemente. Para esto se utilizan las imágenes de entrenamiento y validación de VOC 2007, donde para cada imagen se recortan los objetos presentes en cada imagen según las anotaciones disponibles y se utilizan, entonces, estos recortes como imágenes de entrenamiento. Para garantizar mejores resultados, adicionalmente se seleccionan sólo imagenes con los objetos pertenecientes a las 6 clases indicadas en 3.1, lo que da un total de 2694 imágenes.
Notar que el entrenamiento de la red para el sistema R-CNN debe ser con los *bounding boxes* pertenecientes a los objetos presentes en cada imagen en vez de la imagen completa.

Límite teórico de número de *proposals*

Se ha dicho que se pretende limitar el tamaño de la entrada de la red SqueezeNet. El impacto que genera este cambio se verá en el tiempo de ejecución de R-CNN, y como se muestra en las ecuaciones (3.1-4), este tiempo depende directamente además del número de *proposals* que entran a la red. Es entonces necesario encontrar el número teórico óptimo de estos tal que el tiempo de ejecución de R-CNN se equipare al de Fast R-CNN .

Asumiendo que el mayor peso del tiempo está en el paso por la red convolucional, si queremos estimar el número de *proposals* apropiado para asegurar una igualdad o mejora en el desempeño del sistema R-CNN se tiene entonces que debe darse, reemplazando los valores correspondientes en las ecuaciones (3.1-4):

$$t_{CONV} = t_{CONV'} \tag{3.5}$$

$$N_r \cdot S_{xRCNN} \cdot S_{yRCNN} = S_{xFAST} \cdot S_{yFAST} \tag{3.6}$$

$$N_r \cdot S_{xRCNN} \cdot S_{yRCNN} = N_r \cdot 227 \cdot 227 \tag{3.7}$$

Considerando el tamaño de la entrada de la red de Fast R-CNN, AlexNet, de 227x227. Se tiene entonces para cada caso (R-CNN con red de entrada 32x32 y de entrada 64x6):

$$N_{r32} = \frac{227 \cdot 227}{32 \cdot 32} = 50{,}321 \simeq 50 \tag{3.8}$$

$$N_{r64} = \frac{227 \cdot 227}{64 \cdot 64} = 12{,}580 \simeq 13 \tag{3.9}$$

Es decir, un número mayor a 50 *proposals* para una red de 32x32 en R-CNN haría a este sistema teóricamente más lento que Fast R-CNN; y lo mismo para la red de 64x64 con 13 *proposals*.

Este cálculo está hecho bajo el supuesto que ambas redes convolucionales utilizadas son la misma. Como este no es el caso, se compara (tabla 3.1) el tiempo para procesar la misma imagen a través de SquezeNet y AlexNet, (usando *Darknet*), notando que el tamaño de entrada de ambas redes es esencialmente el mismo (226x226 y 227x227 respectivamente):

Tabla 3.1: Tiempo de un paso por la red para la misma imagen

Red	**Tiempo [s/im]**
SqueezeNet	0.125948
AlexNet	0.141723

Se verifica entonces que ambas redes tienen un tiempo de ejecución similar.

3.3.2. Implementación Fast R-CNN

Se prueba Fast-RCNN usando el código original [7] adaptado para la base de datos utilizada y con los *proposals* generados con el método de generación seleccionado, y con un modelo

pre-entrenado (con VOC 2007) de CaffeNet[1] (pues es la configuración más rápida [3]).

Finalmente, sin pérdida de generalidad, se utilizará para la evaluación de ambos sistemas sólo la base de datos B, que se adecúa al entrenamiento con VOC2007 que tienen ambos sistemas. Esto pues las clases de objetos escogidas están presentes en VOC2007, a diferencia de la base de datos A, la cual presenta muchas clases de objetos de los cuales la mayoría no está presente en VOC2007.

3.4. Métricas de Desempeño

Cada configuración debe ser evaluada según los siguientes parámetros:

1. Tiempo de ejecución promedio.
2. Recall y Precisión de Clasificación de objetos localizados.

Con esta información, entonces, es posible realizar la comparación y, luego, concluir con respecto a los objetivos planteados.

[1] CaffeNet es una adaptación de AlexNet desarrollada para ser utilizada en el *Framework* Caffe [6], donde está desarrollado Fast R-CNN.

Capítulo 4

Resultados[1]

4.1. Caracterizaciones de Generación de *Proposals*

Se implementan los métodos de generación de *proposals* a partir de los códigos disponibles en Github: [47] [48] y [49]; los cuales se adaptan para las siguientes pruebas.
A menos que se indique lo contrario, los parámetros utilizados en cada prueba son los estándar provistos por cada código.

4.1.1. Resolución vs Tiempo

Como primera tarea se debe evaluar el desempeño de cada método seleccionado en términos de tiempo de ejecución. Es evidente que este factor depende del tamaño de entrada de la imagen. Las aplicaciones seleccionadas no garantizan a priori un tamaño fijo de estas por lo que es necesario ver el comportamiento de cada método y su varianza (pues se pueden presentar factores aleatorios, ya sea por la naturaleza del método como las condiciones de ejecución) a distintas resoluciones. Notar que en este caso no se considera el tipo de imagen, sólo el comportamiento a distintas resoluciones sobre la misma imagen.
Para medir resolución versus tiempo, se utiliza una imagen a distintas resoluciones como se ve en la figura 4.1).

[1]Se recuerda para esta sección que todos los experimentos realizados fueron en un computador ASUS X556U con Intel Core i5-6200U CPU, 2.30GHz x 4 y GeForce 940MX para el uso de GPU en el entrenamiento y prueba de redes convolucionales.

(a) 160x120

(b) 208x156

(c) 240x180

(d) 272x204

Figura 4.1: Algunas de las imágenes utilizadas en el análisis

El resto de las imágenes tienen los siguientes tamaños: 320x320, 368x276, 400x300, 432x324, 480x360, 528x396, 560x420, 592x444, 640x480, 688x516, 720x540 y 800x600

EdgeBoxes

Por cada imagen del conjunto mencionado, se realizan 500 iteraciones de generación de *proposals*, se calcula luego el promedio y varianza:

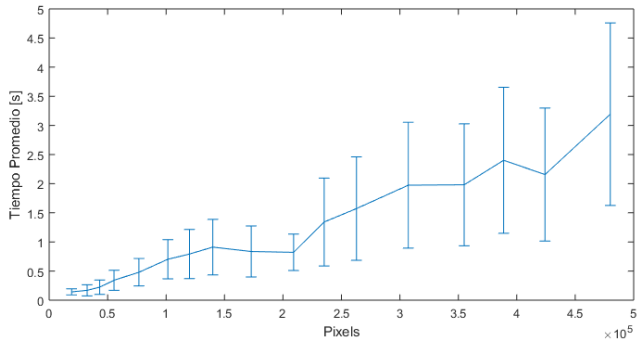


Figura 4.2: Gráfico Revolución vs Tiempo EdgeBoxes

SelectiveSearch

Por cada imagen del conjunto mencionado, se realizan 50 iteraciones de generación de *proposals*, se calcula luego el promedio y varianza:

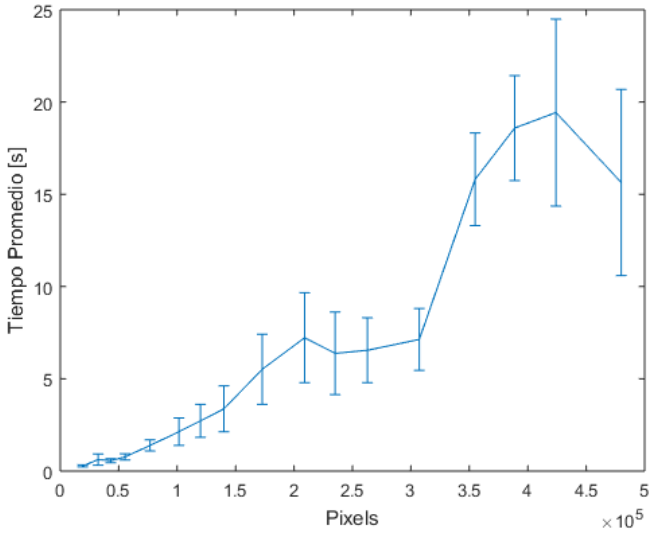


Figura 4.3: Gráfico Revolución vs Tiempo SelectiveSearch

BING

Para BING se realizan 500 iteraciones por cada imagen, se calcula el promedio y varianza (que en este caso resulta ser muy pequeña, pues es un método sin factores aleatorios):

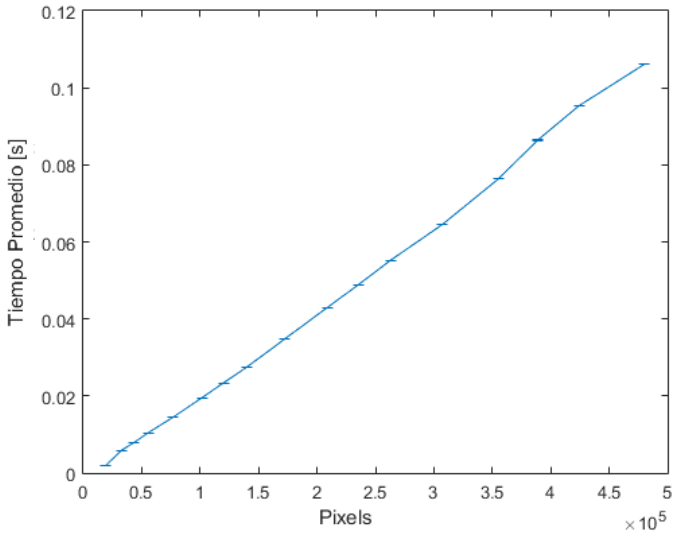


Figura 4.4: Gráfico Revolución vs Tiempo BING

Discusión

De los resultados preliminares podemos extraer las siguientes conclusiones iniciales:

- El tiempo versus resolución en EdgeBoxes crece aproximadamente linealmente, pero a medida que aumenta la resolución, la varianza aumenta progresivamente haciendo que el tiempo de procesamiento pueda ser incluso el doble.
- Para SelectiveSearch el tiempo versus resolución crece de forma aproximadamente exponencial, y con mucha más incerteza a mayor resolución.
- BING es mucho más fiable en términos de tiempo, pues se comporta completamente lineal y con mínima varianza. Por otro lado es el método más rápido por varios órdenes de magnitud sobre todo cuando se trata de imágenes de alta resolución.

4.2. Método seleccionado: BING

Del análisis anterior y el visto en la tabla 2.2 se extrae que BING es el mejor candidato, tanto por su considerablemente superior ventaja en tiempos de ejecución como en la promesa de buen recall a pocos *proposals* que se obtiene.
Como fue dicho en la sección 3.2. es entonces necesario a continuación verficar esta elección sobre las aplicaciones seleccionadas.

4.2.1. Recall en Bases de Datos

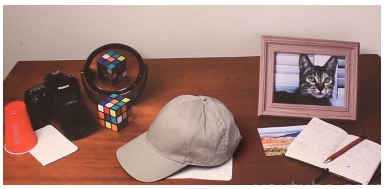
A continuación se presenta una evaluación en términos del recall para las bases de datos mencionadas en 3.1.

Base de Datos A

Se evalúa con distintos valores del parámetro *numPerSz* que indica un número máximo de *proposals* generados para cada resolución. Se toman entonces los *proposals* generados con un IoU mayor a 0.5 y se ilustran los resultados para algunas imágenes de esta base de datos (las que se muestran en la figura 4.5) en las figuras 4.6, 4.7 y 4.8. Por otro lado, los resultados del recall para cada caso se muestran en la tabla 4.1 tomando en cuenta, además, la selección de número de *proposals* máximos a tomar del total de los generados (es decir, si se fija el método para generar un máximo de 320 *proposals*, evalúo el recall del seleccionar el top 20, top 50 y top 100 de estos) donde el total está dado por numPerSz y sus valores fueron escogidos con tal de tomar distintos órdenes de magnitud.

Tabla 4.1: Recall para distintos valores de numPerSz y número de *proposals* seleccionados

# **Proposals** | **numPerSz**	**2** (máx. 64 *props.*)	**10** (máx. 320 *props.*)	**130** (máx. 4160 *props.*)
20	0.213	0.138	0.136
50	0.292	0.304	0.205
100	-	0.603	0.401
1000	-	0.626	0.939
3000	-	-	0.942



(a) img1

(b) img10

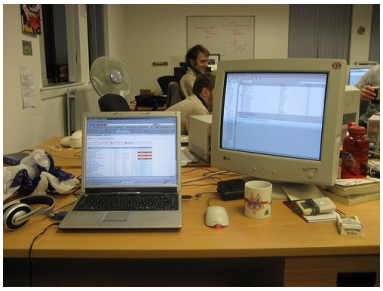
(c) img20

(d) img30

(e) img40

(f) img50

(g) img60

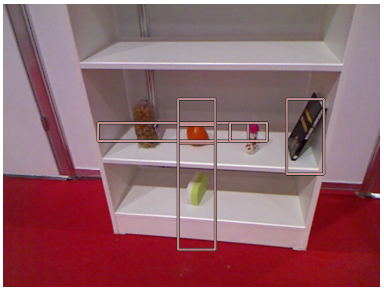


(h) img70

Figura 4.5: Imágenes de Prueba

(a) img1

(b) img10



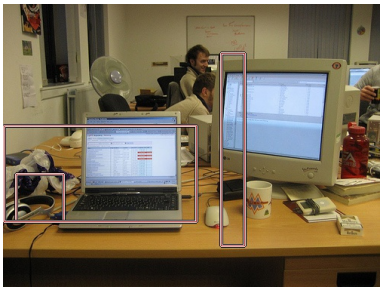
(c) img20

(d) img30

(e) img40

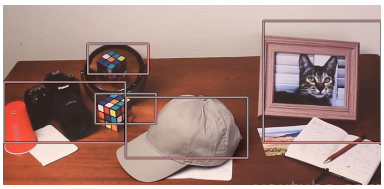
(f) img50

(g) img60



(h) img70

Figura 4.6: Resultados para numPerSz = 2



(a) img1

(b) img10

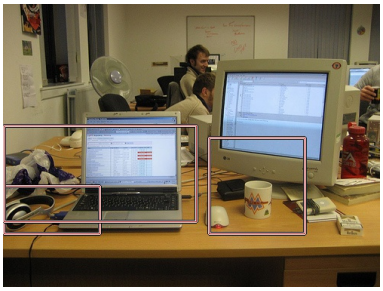
(c) img20

(d) img30

(e) img40

(f) img50

(g) img60



(h) img70

Figura 4.7: Resultados para numPerSz = 10

(a) img1

(b) img10

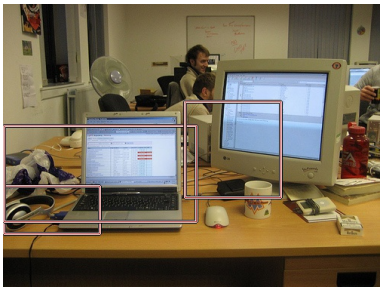
(c) img20

(d) img30

(e) img40

(f) img50

(g) img60



(h) img70

Figura 4.8: Resultados para numPerSz = 130

Base de Datos B

Una evaluación análoga donde se muestra el recall para cada caso se detalla en la tabla 4.2 y se ilustran los resultados de algunas imágenes seleccionadas (4.9) en las figuras 4.10, 4.11 y 4.12.

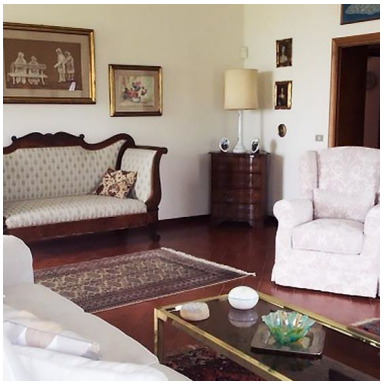
Tabla 4.2: Recall para distintos valores de numPerSz y número de *proposals* seleccionados

# Proposals | numPerSz	**2**	**10**	**130**
20	0.515	0.608	0.604
50	0.523	0.738	0.725
100	-	0.806	0.84
1000	-	0.819	0.958
3000	-	-	0.967

Es evidente que para esta base de datos la generación de *proposals* presenta un mejor recall y con un menor número generado de estos. Esto se explica fácilmente, pues, tal como se mencionó en 2.1.3., BING fue entrenado con VOC 2007 y las imágenes de la base de datos B contienen en su mayoría objetos de las clases presentes en VOC.

Es por esto que las configuraciones completas serán evaluadas con esta base de datos, donde vemos además, que el recall obtenido con 50 *proposals* es del rango del 70 %.

Considerando adicionalmente que seleccionar menos de 20 *proposals* equivale a un recall sumamente bajo, en lo que sigue se tomarán para ambas redes las selección de los 50 mejores *proposals*, asumiendo el costo en tiempo de ejecución que esto representa dados los resultados de las ecuaciones (3.8-9).



(a) img1

(b) img10

(c) img20

(d) img30

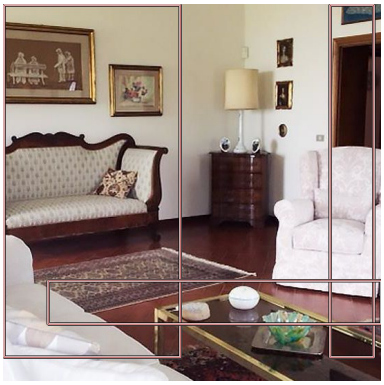
(e) img40

(f) img50

(g) img60

(h) img70

Figura 4.9: Imágenes de Prueba



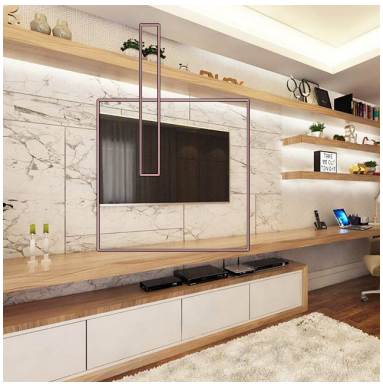
(a) img1

(b) img10

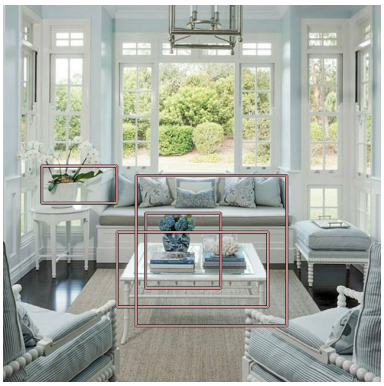
(c) img20

(d) img30

(e) img40



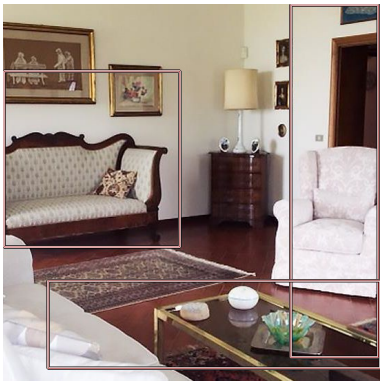
(f) img50



(g) img60

(h) img70

Figura 4.10: Resultados para numPerSz = 2



(a) img1

(b) img10

(c) img20

(d) img30

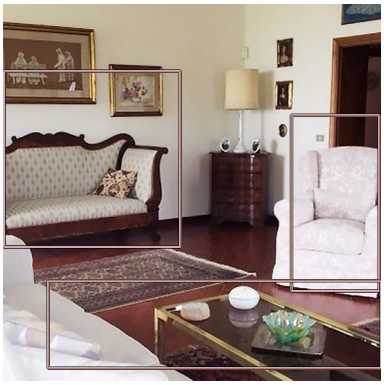
(e) img40

(f) img50

(g) img60

(h) img70

Figura 4.11: Resultados para numPerSz = 10



(a) img1

(b) img10

(c) img20

(d) img30

(e) img40

(f) img50

(g) img60

(h) img70

Figura 4.12: Resultados para numPerSz = 130

4.3. R-CNN Modificado

Como fue brevemente mencionado, las redes utilizadas en el sistema R-CNN fue entrenada mediante el framework *darknet*, utilizando como entrada los *bounding boxes* correspondientes a los objetos de interés en la base de datos VOC. Los pesos fueron guardados cada cierto número de iteraciones y de estos se realizó la evaluación de desempeño. Ambas redes fueron entrenadas con un *batch size* de 32 (por capacidad de GPU) y una tasa de aprendizaje de 0.1.

A continuación se presentan los resultados de evaluar en tiempo y desempeño la adaptación hecha de R-CNN.

4.3.1. Tiempo, Recall y Precisión

Tiempo

El tiempo promedio de calcular los *proposals* de la base de datos B con BING es de 0.0615452108844s. Y en la tabla 4.3 se indica el tiempo promedio del paso de las regiones a través de ambas redes implementadas.

Tabla 4.3: Tiempo Promedio de ejecución de redes SqueezeNet

Red	**Tiempo [s/im]**
32x32	0.414661202614
64x64	0.757623705882

Naturalmente, para la red de entrada 64x64 el tiempo es aproximadamente el doble que para la red de entrada 32x32, y el tiempo de BING se podría considerar despreciable en comparación.

Recall y Precisión

SqueezeNet entrega una probabilidad, o porcentaje de certeza, de la detección realizada, por lo que se toman para esta sección solamente las detecciones con un valor de esta certeza mayor al 50 %.
En la tabla 4.4 se muestran los resultados del recall promedio (porcentaje de objetos de ground truth detectados completamente) y de la precisión promedio (porcentaje de detecciones hechas que resultan ser correctas) en la base de datos B. La notación *it* indica una relación (aproximadamente; $it = 160 \cdot$ iteración) con el número de iteraciones realizadas durante el entrenamiento (es decir, representan pesos en distintos puntos del entrenamiento) y *nps* se refiere a la variable numPerSize de BING (es decir, se varía el número de *proposals* generados por sobre los cuales se eligen los mejores 50).

Tabla 4.4: Recall y precisión para red de entrada 32x32

Sistema	**Recall** %	**Precisión** %
4000 it - nps 2	0.0357	1.6221
4000 it - nps 10	2.2108	0.5782
4000 it - nps 130	2.2108	0.5455
4500 it - nps 2	0.1436	3.8091
4500 it - nps 10	13.1195	1.8855
4500 it - nps 130	10.9977	1.6662
5000 it - nps 2	0.1170	4.2228
5000 it - nps 10	8.5600	1.2500
5000 it - nps 130	7.7256	1.3355

Es evidente que para este caso el entrenamiento no dio buenos resultados, el recall es bajo en casi todos los casos. Por lo que a priori el *speed-up* que representa reducir la entrada no genera beneficios en términos del desempeño como detector.

En la tabla 4.5 se encuentra la evaluación análoga para el caso de la red con entrada de 64x64:

Tabla 4.5: Recall y precisión para red de entrada 64x64

Sistema	**Recall** %	**Precisión** %
1500 it - nps 2	19.5238	2.7061
1500 it - nps 10	23.6232	2.2653
1500 it - nps 130	23.5098	2.6070
2000 it - nps 2	22.4149	2.7274
2000 it - nps 10	24.8866	2.6659
2000 it - nps 130	24.9433	2.6259
2500 it - nps 2	21.5646	2.1314
2500 it - nps 10	27.9478	2.1666
2500 it - nps 130	28.2150	2.1811
3000 it - nps 2	27.2335	2.6141
3000 it - nps 10	33.1308	2.6302
3000 it - nps 130	33.3009	2.6340
3500 it - nps 2	28.0839	2.6915
3500 it - nps 10	**33.8273**	**2.6351**
3500 it - nps 130	33.7706	2.5833

En este caso el recall aumenta bastante, pero los valores no son concluyentes sin antes comparar con Fast R-CNN.
Por otro lado, es potencialmente posible mejorar estos resultados aumentando el tamaño del conjunto de entrenamiento (el cual, considerando el número de parámetros a entrenar, es bastante pequeño), el número de iteraciones o incluso la red utilizada (por ejemplo, usar AlexNet y reducir su entrada).

4.3.2. Comparación con Fast R-CNN

Para concluir el análisis, se repite el experimento esta vez con Fast-RCNN. Notar que en este caso se procesan todos los *proposals*, no sólo el top 50. Los resultados obtenidos se ven en la tabla 4.6 y la comparación entre el mejor caso de R-CNN adaptado y el Fast-RCNN con un número comparable de *proposals* (i.e. numPerSz = 2, que genera un máximo de 64) se ve en la tabla 4.7.

Tabla 4.6: Resultados Fast R-CNN

numPerSz	**Tiempo Promedio [s/im]**	**Recall** %	**Precisión** %
130	1.1775	48.3	1,7103
10	0.2692	46.3	1,33355
2	0.1400	34.1	0,7832

Tabla 4.7: Comparación entre el mejor caso de R-CNN adaptado y Fast-RCNN

Sistema	**Tiempo Promedio [s/im]**	**Recall** %	**Precisión** %
R-CNN modificado	0.76	33.8	2.63
Fast R-CNN	0.14	34.1	0.78

Se extrae a partir de estos resultados que Fast-RCNN sigue siendo más rápido (∼0.15s/im en comparación a los ∼0.75s/im de R-CNN). Sin embargo, se puede concluir que es posible obtener resultados comparables (ya que el recall es esencialmente el mismo para el mismo número de *proposals*) y, de los resultados del trabajo original [3], se reduce la diferencia entre los tiempos de ser entre 100 veces más lento a ser sólo 5 veces más lento. Adicionalmente, la precisión del R-CNN modificado es 3 veces mayor que la de Fast-RCNN, lo que implica un mejor desempeño según la métrica *precision/recall*.

La diferencia que existe entre el tiempo de la red de 32x32 (∼0.4s) y fast R-CNN, que deberían ser iguales de acuerdo al cálculo realizado en 3.3.1 se puede atribuir al no considerar el tiempo t_{FC} en la aproximación. Esto implica que para igualar el tiempo entre ambos sistemas se requerirían aún menos *proposals*, lo cual para este tipo de aplicaciones no es factible.

De todos modos, la estrategia de utilizar menos *proposals* resulta prometedora, considerando que para Fast-RCNN se procesan todos los generados (i.e. más de 4000) mientras que en el R-CNN adaptado sólo se utilizaron 50, y se obtuvieron resultados casi idénticos. Esto señala que procesar todos los *proposals* en los que con pocos se detectan la mayoría de objetos resulta sumamente ineficiente. He aquí además la importancia de elegir un buen método de generación de *proposals*.

El desempeño obtenido por R-CNN adaptado indica que es posible acercarse a Fast R-CNN tanto en tiempo como en la detección. Se propone para futuras mejoras el utilizar otras

redes convolucionales (de entrada pequeña) y distintas estrategias de entrenamiento (sea por ejemplo: cambiar el *batch size*, la tasa de aprendizaje, un mayor número de iteraciones o más muestras de entrenamiento).

Finalmente, se puede decir que con los resultados obtenidos es evidente concluir que efectivamente este problema no había sido analizado desde esta perspectiva en la literatura, en donde se ha encontrado la importante influencia de modelar un sistema para la aplicación de detección de objetos deseada, en oposición a modelarlo para rendir bien en bases de datos genéricas.

Conclusiones

La elección de un método de generación de *proposals* no es trivial; se requiere de un método que pueda generar un número bajo de estos, en poco tiempo, y que sean de buena calidad. En los sistemas comparados al final de este trabajo (R-CNN y Fast R-CNN) se había utilizado Selective Search, pero como acá fue analizado, este es magnitudes de veces más lento que otras opciones con desempeños similares. En particular, BING, al usar operaciones binarias en sus cálculos, acelera este proceso entre 10 y 20 veces sin sacrificar desempeño en la localización de objetos.

Se encuentra para el tiempo de ejecución, un límite teórico al número de *proposals* que igualarían el tiempo entre un sistema del tipo R-CNN y otro Fast R-CNN, el cuál sería de 50 *proposals* en caso de una red de entrada 32x32 (asumiendo la misma red). Los resultados experimentales indican que la red de entrada de 32x32 es 3 veces más lenta que Fast R-CNN para 50 *proposals*, esto en primer lugar se puede atribuir a que en las ecuaciones (3.5-3.9) se asumió despreciable el tiempo t_{FC} lo que agrega un factor de error al cálculo; por otro lado, otros factores que pueden influir en esta diferencia recaen en que se utilizaron distintos *Frameworks* y lenguajes de programación para cada sistema (Darknet y C para R-CNN, Caffe/OpenCV y Python/C++ para Fast R-CNN).
En lo que respecta a la red de 64x64, el tiempo de ejecución llega a ser de 5 veces mayor al de Fast-RCNN, sin embargo, esto es una indicación de mejora al considerar que originalmente la diferencia era de casi 100 veces y que se toma un número de *proposals* 4 veces mayor al límite teórico encontrado.

En términos de desempeño de detección, el recall de Fast R-CNN es sólo un 0.3 % superior al del R-CNN implementado, mientras que la precisión es 3 veces menor. Esto indica un desempeño de detección ligeramente superior por parte de R-CNN. Tomando en cuenta que el entrenamiento de R-CNN fue realizado con sólo 2694 imágenes y en un tiempo reducido, es posible asumir que mejoras de desempeño son factibles. Por ejemplo, agregando más imágenes de entrenamiento, entrenar con una GPU más avanzada o incluso utilizar otra arquitectura de red con entrada reducida. En el caso de que se obtuviera un desempeño que fuera suficientemente alto, se podría decir, entonces, que R-CNN es más útil que Fast-RCNN en casos donde los 0.6s de diferencia sean transables a favor de la detección en sí misma.

Como observación final, cabe decir que el utilizar R-CNN resulta más sencillo que Fast-RCNN para un usuario promedio. Esto pues *Darknet* sólo tiene como dependencia CUDA para el uso de GPU, mientras que Fast R-CNN utiliza Caffe y OpenCV, donde cada uno tiene un gran número de dependencias y dificultades para su instalación.

Bibliografía

[1] J. H. Hosang, R. Benenson, P. Dollár, and B. Schiele, "What makes for effective detection proposals?," *CoRR*, vol. abs/1502.05082, 2015.

[2] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Computer Vision and Pattern Recognition*, 2014.

[3] R. Girshick, "Fast r-cnn," in *International Conference on Computer Vision (ICCV)*, 2015.

[4] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards real-time object detection with region proposal networks," in *Advances in Neural Information Processing Systems (NIPS)*, 2015.

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

[6] 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.

[7] R. Girshick, "Fast r-cnn (github)," 2015. `https://github.com/rbgirshick/fast-rcnn`.

[8] B. Alexe, T. Deselaers, and V. Ferrari, "What is an object?," in *The Twenty-Third IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2010, San Francisco, CA, USA, 13-18 June 2010*, pp. 73–80, 2010.

[9] B. Alexe, T. Deselaers, and V. Ferrari, "Measuring the objectness of image windows," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 34, no. 11, pp. 2189–2202, 2012.

[10] M.-M. Cheng, Z. Zhang, W.-Y. Lin, and P. H. S. Torr, "BING: Binarized normed gradients for objectness estimation at 300fps," in *IEEE CVPR*, 2014.

[11] C. L. Zitnick and P. Dollár, "Edge boxes: Locating object proposals from edges," in *Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V*, pp. 391–405, 2014.

[12] J. Uijlings, K. van de Sande, T. Gevers, and A. Smeulders, "Selective search for object

recognition," *International Journal of Computer Vision*, 2013.

[13] P. F. Felzenszwalb and D. P. Huttenlocher, "Efficient graph-based image segmentation," *Int. J. Comput. Vision*, vol. 59, pp. 167–181, Sept. 2004.

[14] P. Arbelaez, M. Maire, C. Fowlkes, and J. Malik, "Contour detection and hierarchical image segmentation," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 33, pp. 898–916, May 2011.

[15] P. Dollár and C. L. Zitnick, "Fast edge detection using structured forests," *CoRR*, vol. abs/1406.5549, 2014.

[16] K. E. A. van de Sande, J. Uijlings, T. Gevers, and A. Smeulders, "Segmentation as Selective Search for Object Recognition," in *ICCV*, 2011.

[17] S. Manén, M. Guillaumin, and L. Van Gool, "Prime Object Proposals with Randomized Prim's Algorithm," in *International Conference on Computer Vision (ICCV)*, Dec. 2013.

[18] P. Rantalankila, J. Kannala, and E. Rahtu, "Generating object segmentation proposals using global and local search," in *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, OH, USA, June 23-28, 2014*, pp. 2417–2424, 2014.

[19] K.-Y. Chang, T.-L. Liu, H.-T. Chen, and S.-H. Lai, "Fusing generic objectness and visual saliency for salient object detection," in *IEEE International Conference on Computer Vision (ICCV)*, 2011.

[20] J. Carreira and et al., "Constrained parametric min-cuts for automatic object segmentation," 2010.

[21] J. Carreira and et al., "Cpmc: Automatic object segmentation using constrained parametric min-cuts," 2012.

[22] I. Endres and D. Hoiem, "Category independent object proposals," in *Proceedings of the 11th European Conference on Computer Vision: Part V*, ECCV'10, (Berlin, Heidelberg), pp. 575–588, Springer-Verlag, 2010.

[23] I. Endres and D. Hoiem, "Category-independent object proposals with diverse ranking," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 36, pp. 222–234, Feb. 2014.

[24] A. Humayun, F. Li, and J. M. Rehg, "RIGOR: reusing inference in graph cuts for generating object regions," in *2014 IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2014, Columbus, OH, USA, June 23-28, 2014*, pp. 336–343, 2014.

[25] P. Krähenbühl and V. Koltun, "Geodesic object proposals," in *Computer Vision - ECCV 2014 - 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part V*, pp. 725–739, 2014.

[26] P. Arbeláez, J. Pont-Tuset, J. Barron, F. Marques, and J. Malik, "Multiscale combina-

torial grouping," in *Computer Vision and Pattern Recognition*, 2014.

[27] E. Rahtu, J. Kannala, and M. B. Blaschko, "Learning a category independent object detection cascade," in *IEEE International Conference on Computer Vision, ICCV 2011, Barcelona, Spain, November 6-13, 2011*, pp. 1052–1059, 2011.

[28] J. Feng, Y. Wei, L. Tao, C. Zhang, and J. Sun, "Salient object detection by composition," in *Computer Vision (ICCV), 2011 IEEE International Conference on*, pp. 1028–1035, IEEE, 2011.

[29] Z. Zhang and P. H. S. Torr, "Object proposal generation using two-stage cascade svms," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 38, no. 1, pp. 102–115, 2016.

[30] Z. Zhang, J. Warrell, and P. H. S. Torr, "Proposal generation for object detection using cascaded ranking svms," in *The 24th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2011, Colorado Springs, CO, USA, 20-25 June 2011*, pp. 1497–1504, 2011.

[31] M. V. den Bergh, G. Roig, X. Boix, S. Manen, and L. J. V. Gool, "Online video SEEDS for temporal window objectness," in *IEEE International Conference on Computer Vision, ICCV 2013, Sydney, Australia, December 1-8, 2013*, pp. 377–384, 2013.

[32] N. Chavali, H. Agrawal, A. Mahendru, and D. Batra, "Object-proposal evaluation protocol is 'gameable'," *CoRR*, vol. abs/1505.05836, 2015.

[33] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL Visual Object Classes Challenge 2007 (VOC2007) Results." http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html.

[34] P. Dollár and C. L. Zitnick, "Structured forests for fast edge detection," in *Proceedings of the 2013 IEEE International Conference on Computer Vision*, ICCV '13, (Washington, DC, USA), pp. 1841–1848, IEEE Computer Society, 2013.

[35] Y. LeCun, Y. Bengio, and G. E. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[36] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[37] Y. LeCun, Y. Bengio, *et al.*, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, vol. 3361, no. 10, p. 1995, 1995.

[38] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Handwritten digit recognition with a back-propagation network," in *Advances in Neural Information Processing Systems (NIPS 1989)*, vol. 2, (Denver, CO), 1990.

[39] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," in *CVPR09*, 2009.

[40] 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.

[41] F. N. Iandola, M. W. Moskewicz, K. Ashraf, S. Han, W. J. Dally, and K. Keutzer, "Squeezenet: Alexnet-level accuracy with 50x fewer parameters and <1mb model size," *CoRR*, vol. abs/1602.07360, 2016.

[42] 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.

[43] "Instagram." `https://www.instagram.com`. Accessed: 2017-06-23.

[44] S. Qiu, "Bbox-label-tool." `https://github.com/puzzledqs/BBox-Label-Tool`, 2014.

[45] J. G. Shi Qiu, "Bbox-label-tool: multi-class branch." `https://github.com/puzzledqs/BBox-Label-Tool/tree/multi-class`, 2017.

[46] H. Mao, S. Yao, T. Tang, B. Li, J. Yao, and Y. Wang, "Towards real-time object detection on embedded systems," *IEEE Transactions on Emerging Topics in Computing*, vol. PP, no. 99, pp. 1–1, 2017.

[47] P. Dollar, "Structured edge detection toolbox." `https://github.com/pdollar/edges`, 2014.

[48] D. SUZUO, "Python implementation of the selective search." `https://github.com/belltailjp/selective_search_py`, 2015.

[49] S. Z. Ming-Ming Cheng, "Objectness proposal generator with bing."