UNIVERSIDAD
DE GRANADA

Facultad de Ciencias
Escuela Técnica Superior de Ingenierías Informática y de Telecomunicaciones

Grado en Ingeniería Informática y Matemáticas

TRABAJO DE FIN DE GRADO

Localización de Regiones de Interés Utilizando Aprendizaje Profundo

Presentado por:
Laura Gómez Garrido

Tutor:
Jesús Chamorro Martínez
Ciencias de la Computación e Inteligencia Artificial

Curso académico 2019-2020

Localización de Regiones de Interés Utilizando Aprendizaje Profundo

Laura Gómez Garrido

Laura Gómez Garrido *Localización de Regiones de Interés Utilizando Aprendizaje Profundo*.
Trabajo de fin de Grado. Curso académico 2019-2020.

Responsable de tutorización	Jesús Chamorro Martínez *Ciencias de la Computación e Inteligencia Artificial*	Grado en Ingeniería Informática y Matemáticas Facultad de Ciencias Escuela Técnica Superior de Ingenierías Informática y de Telecomunicaciones Universidad de Granada

Declaración de originalidad

D./Dña. Laura Gómez Garrido

Declaro explícitamente que el trabajo presentado como Trabajo de Fin de Grado (TFG), correspondiente al curso académico 2019-2020, es original, entendida esta, en el sentido de que no ha utilizado para la elaboración del trabajo fuentes sin citarlas debidamente.

En Granada a 13 de agosto de 2020

Fdo: Laura Gómez Garrido

Dedicatoria (opcional)
Ver archivo `preliminares/dedicatoria.tex`

Índice general

Índice de figuras IX

Índice de tablas XI

Agradecimientos XIII

Summary XV

Introducción XVII

I. ¿Estado del arte? / ¿Introducción? 1

1. Red Neuronal 3
1.1. El problema de clasificar una imagen. 3
1.2. Clasificadores Lineales 4
1.3. El modelo de una neurona 6
1.4. La red completa 7

2. Redes Neuronales Convolucionadas (CNN) 9
2.1. Capas Convolucionadas o Convolutionals 9
2.2. Capas de Agrupación o Pooling 10
2.3. Ejemplos 10

3. Paradigmas de detección 11
3.1. Dos pasos o basados en RCNN 12
3.2. Un paso o basado en YOLO 13
3.3. Feature Pyramid Networks 13
3.4. Modelo elegido 13

A. Primer apéndice 15

Glosario 17

Bibliografía 19

Índice de figuras

1.1. Cómo un ordenador ve una imagen 3
1.2. Comparación entre una neurona biológica (izquierda) y el modelo matemático (derecha) 7
1.3. Ejemplos de redes totalmente conectadas (*fully-connected*) 8

2.1. Capa de Convolución. La zona grisácea corresponde a un filtro. 10

3.1. Detección, clasificación y segmentación. [WSH19] 11
3.2. Comparativa de algunos modelos de la familia R-CNN. [WSH19] 12

Índice de tablas

Agradecimientos

Agradecimientos del libro (opcional, ver archivo `preliminares/agradecimiento.tex`).

Summary

An english summary of the project (around 800 and 1500 words are recommended).
File: `preliminares/summary.tex`

Introducción

De acuerdo con la comisión de grado, el TFG debe incluir una introducción en la que se describan claramente los objetivos previstos inicialmente en la propuesta de TFG, indicando si han sido o no alcanzados, los antecedentes importantes para el desarrollo, los resultados obtenidos, en su caso y las principales fuentes consultadas.

Ver archivo `preliminares/introduccion.tex`

Parte I.

¿Estado del arte? / ¿Introducción?

A continuación, explicaremos de forma concisa lo que es una *Red Neuronal* y una *Red Neuronal Convolucionada* para, seguidamente, hablar sobre los últimos avances en la localización de regiones de interés en imágenes.

1. Red Neuronal

1.1. El problema de clasificar una imagen.

Cuando observamos una imagen, podemos localizar varios elementos a partir de los cuáles esta se encuentra compuesta con tan sólo un vistazo. Sin embargo, para un ordenador no se trata de algo tan sencillo puesto que sólo es capaz de ver un gran conjunto de números que no tienen por qué tener relación alguna entre sí.

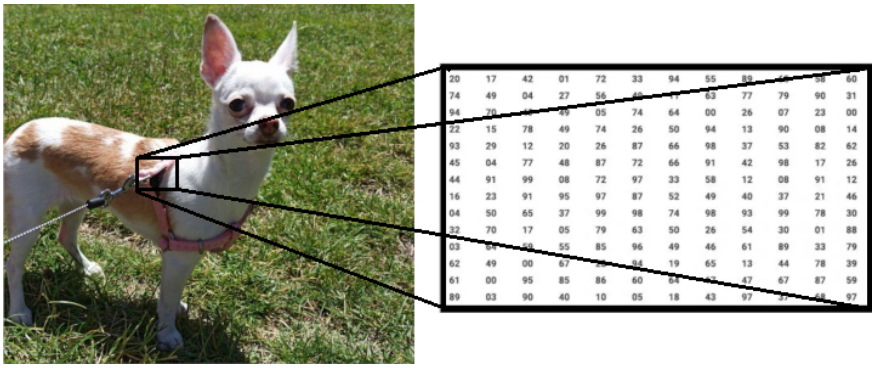


Figura 1.1.: Cómo un ordenador ve una imagen

Idealmente, en esta imagen desearíamos que fuera capaz de identificar que se trata de un perro, en concreto de un chiguagua, de pelaje blanco y manchas color café que se encuentra sobre un césped. Estos poquitos datos que para nosotros parecen tan triviales necesitan de horas y horas de computación para ser obtenidos a partir de una imagen cualquiera.

Dejamos todos estos detalles, a los cuales esperamos poder llegar en un futuro no muy lejano, y simplificamos el problema a tener un conjunto de etiquetas y buscar con cuál de todas ellas tiene mayor relación nuestra imagen.

Una primera idea, sería utilizar utilizar *k-Nearest Neighbor Classifier*, en adelante *k*-NN. Este clasificador consiste en que para cada etiqueta se correspondan *k* imágenes y consideraremos como etiqueta idónea para nuestra clasificación aquella cuya distancia, siendo Manhattan y Euclídea las más comunes, entre los píxeles con nuestra imagen sea menor. Nos encontramos con que el tiempo de entrenamiento de nuestro clasificador sería ínfimo en comparación con el tiempo de clasificación, si bien se pueden hacer diversas mejoras que cambien estos hechos.

De esta sencilla propuesta, surgen diversos problemas. Al darle mayor importancia al valor concreto de los píxeles, en lugar de a las formas que de las que está compuesta la figura, nos encontramos con que valores como los colores de fondo pueden influir más en la clasificación que los propios píxeles de la figura que queremos clasificar. En el ejemplo de la

imagen del chiguagua, si considerásemos las etiquetas verde y perro, tendríamos que por lo general como respuesta el color verde, pese a que estamos más interesados por la mascota en sí. Otro gran problema, sería la baja escalabilidad que nos proporciona esta solución, al incrementarse enormemente el costo computacional de clasificación conforme aumenta el número de etiquetas.

El siguiente paso, es buscar una forma de "memorizar" los datos de entrenamiento de forma que no tengamos que estar comparándolos con todos ellos cuando queramos clasificar una imagen. Lo que buscamos es poder valorar de alguna forma la imagen completa y a partir de esta "puntuación" conocer qué etiqueta le corresponde mejor a nuestra imagen. De esta forma, veríamos drásticamente reducido el tiempo de clasificación, sacrificando para ello el tiempo de entrenamiento, y podríamos ampliar en varias unidades la cantidad de datos de entrenamiento utilizados, aumentando así la precisión de nuestro clasificador.

1.2. Clasificadores Lineales

Antes de nada, vamos a comenzar contextualizando matemáticamente el entorno en el que nos encontramos. Una vez realizado esto podremos hablar correctamente de los clasificadores lineales y así poder extenderlos naturalmente a los conceptos de Red Neuronal y de Red Neuronal Convolucionada.

Sea $D \in \mathbb{N}$ la dimensión de nuestras imágenes, por lo general el número de píxeles que estas poseen, y $K \in \mathbb{N}$ la cantidad de etiquetas o categorías bajo las cuales pueden ser clasificadas. Siendo $N \in \mathbb{N}$ el número de ejemplos que utilizaremos para entrenar nuestro clasificador, tendremos que para cada dato de entrenamiento $x_i \in \mathbb{R}^D$ $i = 1, ..., N$ le corresponde una etiqueta $y_i \in 1, ..., K$ tal que juntos conforman el par (x_i, y_i) de imagen y categoría a la que pertenece.

Para que no sea más fácil de entender este contexto, tomaremos como ejemplo el conjunto de datos CIFAR-10 que consiste en 60000 imágenes RGB de dimensión 32x32 y 10 categorías. De esta forma, tendríamos que $D = 32 \cdot 32 \cdot 3 = 3072$, $K = 10$ y $N = 60000$.

Definición 1.1. Definiremos *función de puntuación* o *score function* como una función $f : \mathbb{R}^D \to \mathbb{R}^K$ que asigna los píxeles de la imagen sin procesar una serie de puntuaciones para cada etiqueta.

Una función de puntuación nos revela la probabilidad que tiene cada imagen de pertenecer a cada una de la distintas etiquetas de las que disponemos. Podemos definir un *clasificador lineal* o *linear classifier* como aquel que utiliza una función de puntuación lineal, es decir, de la forma:

$$f(x) = W \cdot x + b \quad W \in M_{K,D}(\mathbb{R}) \;\; b \in \mathbb{R}^K \;\; \forall x \in \mathbb{R}^D$$

Dicho esto, existen diferentes tipos de clasificadores lineales y la principal diferencia entre ellos reside en la función de pérdida que utilicen a la hora de entrenar la red.

Definición 1.2. Una *función de pérdida* o *loss function* es aquella que durante el entrenamiento de un clasificador se encarga de penalizar las etiquetas incorrectas.

FIXME: ¿Hablar de SVM, SoftMax y Cross-Entropy?

Llegados a este punto, debemos de explicar cómo funcionan los clasificadores lineales, es decir, cómo estos son entrenados y para qué se emplean las funciones de puntuación y de pérdida. Dado poseemos un conjunto de ejemplos con su correspondiente etiqueta pero desconocemos el valor de los parámetros W y b, nuestro objetivo es utilizar dichos datos de entrenamiento para estimar estos valores.

Para ello, comenzamos haciendo una conjetura sobre un posible valor para nuestros parámetros, evaluamos nuestra función de puntuación con todos nuestros datos de entrenamiento y seguidamente utilizamos la función de pérdida para estimar cómo de bien funciona nuestra conjetura. Analizamos los resultados y hacemos una nueva conjetura que los mejore, repitiendo el proceso un número lo suficientemente grande de veces.

Seguidamente, debemos preguntarnos cómo realizamos la nueva conjetura de forma que nos aseguremos tener unos resultados mejores. Hacerlo de forma totalmente aleatoria, repitiéndolo hasta obtener una pérdida lo suficientemente pequeña, no parece una buena idea puesto que es difícil saber cuándo encontraremos una buena respuesta. Entre las distintas técnicas, nos encontramos con las basadas en la *búsqueda aleatoria local* y las que se basan en el *Gradiente Descendiente*, entre otras muchas. Como ejemplo, tomaremos el algoritmo del gradiente descendiente y minimizaremos la función de pérdida para llegar a aquellos pesos que menor error nos den.

FIXME: Enlazar pdfs que expliquen bien cómo funciona el gradiente descendiente y la búsqueda aleatoria local

```
while condicion_de_parada :
  weight_grad=evaluate_grad(loss_fun,x,y)
  weight = -step_size*weight_grad
```

El código mostrado más arriba se trata de una versión muy simplificada de cómo funcionaría de cómo podríamos implementar el algoritmo, teniendo en cuenta que nosotros mismos podremos modificar la condición de parada de acuerdo a nuestras necesidades y que el valor *step_size* es algo que podremos fijar de la misma forma, sabiendo que este nos indica cuánto queremos avanzar en la dirección que nos indica el gradiente. FIXME: Enlazar pdfs que expliquen la importancia de estos valores y cómo fijarlos.

FIXME: Añadir implementación en Tensorflow de un clasificador lineal.

Ejemplo 1.1. Fijemos una etiqueta, $y = 0$ por lo que $K = 1$. Tendríamos que nuestra función de puntuación sería $f : \mathbb{R}^D \to \mathbb{R}$ donde $W = (w_1, ...w_D) \in \mathbb{R}^D$ y $b \in \mathbb{R}$ luego

$$f(x) = W \cdot x + b = \sum_{j=1}^{D} w_j x_j + b.$$

Así mismo, tomamos como función de pérdida

$$L(x, f) = -\ln \frac{1}{\sum_{j=1}^{D} e^{f_j}} = -\ln \frac{1}{\sum_{j=1}^{D} e^{w_j x_j + b}} = \ln \sum_{j=1}^{D} e^{w_j x_j + b}$$

que es un caso particular de *Entropía Cruzada* o *Cross-entropy*.

Este clasificador recibe el nombre de *Binary Softmax classifier* o *Binary Logistic Regression classifier*. De esta forma, si consideramos la función probabilística *sigmoide* tendríamos que el minimizar la función de pérdida estaríamos aumentando la probabilidad clasificar correctamente nuestros datos puesto que:

$$P(y = 0|x; W) = \sigma(f(x)) = \frac{1}{1 + e^{-f(x)}}.$$

Observación 1.1. Estamos calculando la probabilidad de coincidir o no con una determinada etiqueta, recibe el nombre de binario porque también podríamos considerar que tenemos dos etiquetas y que si no perteneces a una forzosamente perteneces a la otra. Este modelo puede por tanto construirse también utilizando ambas etiquetas y con mejores resultados a la hora de clasificar puesto que durante el entrenamiento la función de pérdida es ligeramente modificada para tener en cuenta la otra etiqueta. En cualquier caso, ambas construcciones siguen la misma interpretación probabilística y, además, tenemos que $P(y = 1|x; W) = 1 - P(y = 0|x; W)$.

Ejemplo 1.2. Tomamos como función de pérdida $L(x, f) = \frac{1}{N}\sum_{i=1}^{N}\sum_{j\neq y_i}\max(0, f_j - f_{yi} + \Delta)$ donde $\Delta \in \mathbb{R}$ es un parámetro que se suele ajustar utilizando una técnica llamada *validación cruzada* o *cross validation*. Esta función suele recibir el nombre de *hinge loss* y el clasificador lineal es conocido como *Multiclass Suport Vector Machine (SVM)*. Este clasificador "quiere"que la etiqueta correcta para cada imagen tenga una puntuación mayor que las incorrectas por un margen fijo Δ.

1.3. El modelo de una neurona

Cuando comenzamos planteando nuestro problema buscábamos conseguir clasificar una imagen con una probabilidad de acertar similar o superior a cómo lo haríamos nosotros mismos. Teníamos el problema de que un ordenador no era capaz de pensar o razonar de la misma forma que un ser humano y buscábamos una forma de clasificar esta información a pesar de ello. Es aquí donde nacen los modelos de redes neuronales, en un intento por ser capaces de simular cómo funciona nuestro propio cerebro en nuestros clasificadores y que estos sean capaces de aprender cómo reconocer los distintos elementos de la misma forma que nosotros lo hemos ido haciendo a lo largo de nuestra vida.

Nuestro cerebro está formado por múltiples neuronas interconectadas entre sí que están constantemente transmitiéndose información y aprendiendo a través de todos los datos que reciben. Si nos fijamos en 1.2 podemos ver que una neurona real esta formada por dentritas que son quienes, a través del proceso de sinapsis, reciben la entrada de información, que es asimilada y transformada por su núcleo antes de ser transmitida a la siguientes neuronas a través de las divisiones de su axón en caso de que supere un cierto umbral y se active.

De esta forma, si consideramos que tenemos D dentritas y que por la i-dentrita recibimos el la información x_i con un peso o fuerza de sinapsis w_i, tendríamos que nuestro núcleo trabaja con el vector de información $(w_1x_1, ..., w_Dx_D)$ que podemos condensar utilizando la norma $||\cdot||_1$ como $\sum_i w_ix_i$ y sumarle una determinada constante b propia de la neurona.

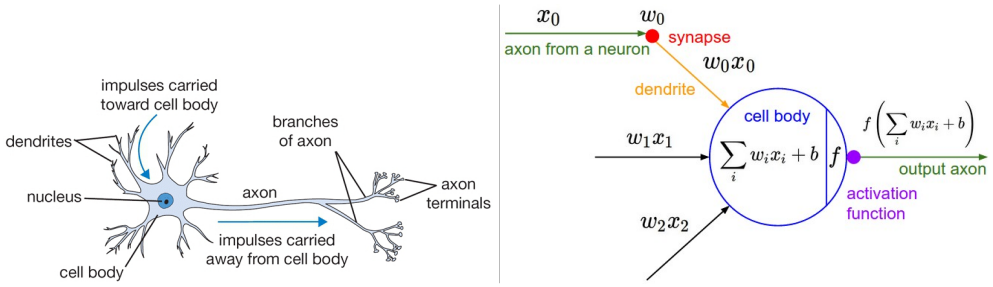


Figura 1.2.: Comparación entre una neurona biológica (izquierda) y el modelo matemático (derecha)

El umbral de activación y la señal enviada al resto de neuronas la representaremos con la *función de activación* que será una función $f : \mathbb{R} \to \mathbb{R}$.

A continuación, mencionaremos los ejemplos más representativos utilizados como funciones de activación, enlazándolos con lo ya visto anteriormente:

- *Función sigmoide* $\sigma : \mathbb{R} \to [0,1]$ definida como $\sigma(x) = \frac{1}{1+e^{-x}}$. En este caso, nos encontramos con el ejemplo 1.1 transformado en una neurona cuya función de pérdida utilizada durante el entrenamiento suele ser la misma que en el ejemplo mencionado. Se suele utilizar en la última capa de nuestra red, cuando nuestras imágenes pueden pertenecer a varias clases o etiquetas al mismo tiempo.
- *Función Softmax* utilizada normalmente en la última capa donde hay tantas neuronas como etiquetas y en el problema de clasificación donde una imagen puede pertenecer únicamente a una sola etiqueta o caso. Se trata de una modificación de función la sigmoide que regulariza la salida para obtener la probabilidad de pertenecer a cada clase como sucesos independientes obteniendo así que la suma de todas las salidas de esta capa sería 1. Aquí, la i-neurona tendría función de activación $\sigma_i(x) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}$ y utilizaría $L(x) = \frac{1}{N}\sum_{i=1}^{N} -\ln \frac{e^{fy_i}}{\sum_{j=1}^{D} e^{f_j}} = \frac{1}{N}\sum_{i=1}^{N}(-f_{yi} + \ln \sum_{j=1}^{D} e_j^f)$ como función de pérdida para el entrenamiento.
- *Función ReLU*, cuyo nombre completo sería *Rectified Lineal Unit*. Se suele utilizar en las capas intermedias de nuestra red y utiliza como función de activación $f(x) = \max(0, x)$ que nos recuerda al ejemplo 1.2.
- *Función Tanh* se trata de de una centralización de la función sigmoide. $Tanh(x) = 2\sigma(x) - 1$.

1.4. La red completa

Una red red neuronal es un conjunto de neuronas dividas en varias *capas* de forma que las neuronas de una capa pueden estar unidas o no con las neuronas de la capa anterior y de la siguiente formando así un grafo acíclico.

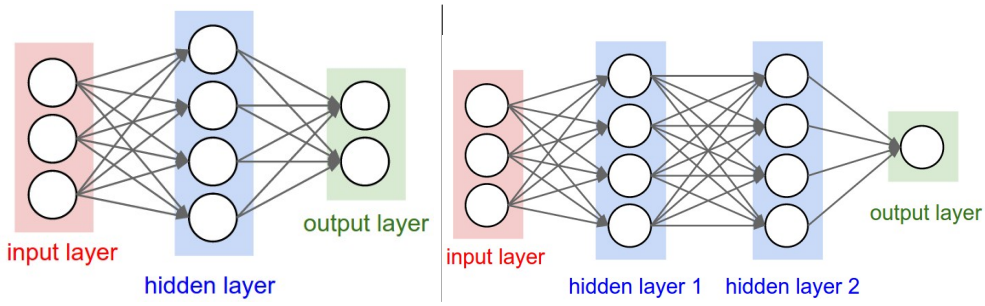


Figura 1.3.: Ejemplos de redes totalmente conectadas (*fully-connected*)

La primera capa recibe el nombre de *capa de entrada* que posee D neuronas, es decir, tantas como la dimensión de nuestros datos. Por otro lado tenemos la *capa de salida* con K neuronas, tantas como etiquetas o clases de clasificación tengamos. El resto de neuronas se distribuyen dentro las *capas ocultas* cuya distribución y conexiones dependen del modelo en concreto que queramos desarrollar, quedando a nuestro criterio.

A continuación toca hablar del entrenamiento de la red completa, a los conceptos que ya conocíamos del clasificador lineal 1.2 se añade el concepto de *Backpropagation* o *Propagación hacia atrás de errores* que hace uso de la *regla de la cadena* para, a través de los cálculos locales realizados por cada neurona, obtener la función completa de clasificación. Así, buscamos que las neuronas se auto organicen para ser capaces de localizar patrones similares de forman que sepan cómo reaccionar ante la presencia de ruido o datos incompletos. Esto hace a través de la comunicación de sus valores locales y gradientes entre las distintas neuronas, comenzando por la capa de salida y avanzando hacia atrás. FIXME: Mencionar importancia del Subgradiente para el cálculo del gradiente, al no estar definido para cualquier función.

FIXME: Definir epoch y la validación cruzada

2. Redes Neuronales Convolucionadas (CNN)

Hasta ahora, todo lo que hemos hablado es válido para casi cualquier contexto puesto que no hemos hecho ninguna presunción en cuanto a la estructura de los datos o sobre las peculiaridades que estos presentan, para nosotros todos los datos eran un vector con D componentes a los que les correspondía una etiqueta y no le dábamos importancia a la estructura interna que estos pudieran llegar presentar. Esto cambia con el uso de las *Redes Neuronales Convolucionadas* al ser un tipo de red exclusivo para imágenes en las cuáles no sólo podremos utilizar todos lo válido en el caso general sino que también utilizaremos una serie de bondades conocidas por el simple hecho de estar tratando con imágenes.

Anteriormente, comentábamos el ejemplo del conjunto de datos CIFAR-10 donde teníamos que $D = 3072$ y mostrábamos cómo obteníamos ese valor. Para ello, utilizábamos la dimensión de la imagen de $ancho \times alto$ y la dimensión del *espacio de color* que estábamos utilizando. De esta forma, si en lugar de trasladar estos valores a $\mathbb{R}^D$ como vectores nos quedamos en $\mathbb{R}^{ancho} \times \mathbb{R}^{alto} \times \mathbb{R}^{\dim color}$ estaremos trabajando un espacio de matrices tridimensionales. De esta forma, siendo a el ancho, h la altura y c la dimensión del espacio de color, para cada matriz $m \in M_{a,h,c}(\mathbb{R})$ tendremos que dado un píxel conocemos de forma inmediata sus píxeles más próximos sin necesidad de hacer ningún cálculo complejo. Nos aprovecharemos de esto para definir las *capas convolucionadas* o *convolutionals* y las *capas de agrupación* o *pooling*.

2.1. Capas Convolucionadas o Convolutionals

Internamente, cada neurona de una capa convolucional posee un *kernel* o *filtro* $W \in \mathbb{R}^r \times \mathbb{R}^s \times \mathbb{R}^c$ donde r y s son parámetros prefijados y una variable $b \in \mathbb{R}$ bias. Para cada imagen $X \in \mathbb{R}^a \times \mathbb{R}^h \times \mathbb{R}^c$ tomamos una sección $x^{RS} \subset X$ donde $x^{RS} = (x^{RS}_{ijk})_{ijk}$ $i = R, ..., R + r,\ \ j = S, ..., S + s$ y $k = 1, ..., c$ con $R = 1, ..., a - r$ y $S = 1, ..., h - s$. Así, cada neurona realiza una *convolución matricial* y suma la variable b bias:

$$f(x^{RS}) = \sum_{k=1}^{c} \sum_{i=1}^{r} \sum_{j=1}^{s} w_{i,j,k} \cdot x^{RS}_{i+R,j+S,k} + b$$

Siendo esta su función de puntuación de las neuronas correspondientes a dicho kernel. A este filtro le corresponderán tantas neuronas como sean necesarias para cubrir todos los datos de entrada. Una capa de convolución, podrá tener tantos filtros como se quieran y cada uno de ellos tendrá tantas neuronas como sean necesarias para cubrir toda la imagen. Visualmente, se transforma un ortoedro en otro.

En esta demo del Curso de Stanford sobre Convolutional Neural Networks for Visual Recognition podemos ver el funcionamiento de dos filtros 3×3 ($r = 3, s = 3$) a una entrada $x \in \mathbb{R}^7 \times \mathbb{R}^7 \times \mathbb{R}^3$. Nótese, que el filtro no es aplicado en todas las submatrices sino que

Figura 2.1.: Capa de Convolución. La zona grisácea corresponde a un filtro.

avanza dos posiciones tanto vertical como horizontalmente, es decir, la capa posee un *paso* o *stride* de 2. En la formulación anterior, se ha supuesto que el paso es de tamaño 1. Además, en la demo se ha completado la matriz x con 0 hasta tener una dimensión de $9 \times 9 \times 3$ para asegurarnos de que recorremos todas las posiciones de x con el filtro. Tanto el paso como si la matriz es completada con algún otro número o no son parámetros que se prefijan al crear la capa, comunes a todos los filtros y controlan la dimensión de salida de la capa.

2.2. Capas de Agrupación o Pooling

FIXME: Same convoluciones

2.3. Ejemplos

FIXME: ¿Apéndices o distribuido en el pdf? Cosas a mencionar:

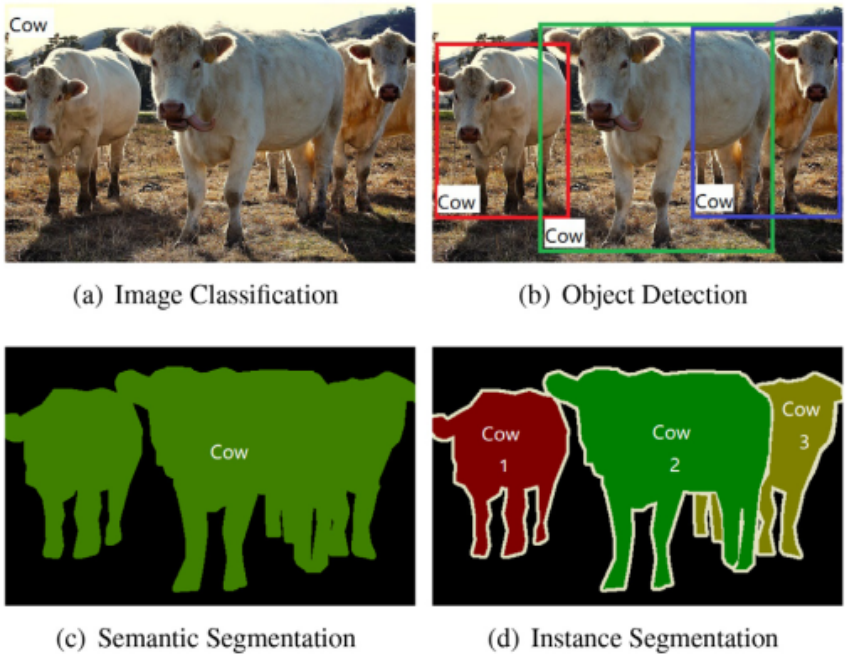
- Cómo se ven tras cada neurona y/o capa (cómo pooling y convolution modifican la imagen)
- Cómo los filtros modifican una imagen visualmente

3. Paradigmas de detección

Hasta este momento, hemos tratado un problema de clasificación global. Se suponía que la imagen tenía un único elemento que se quisiera clasificar e idealmente este estaría centrado con respecto el centro de la imagen. En adelante, querremos clasificar múltiples elementos pertenecientes a múltiples etiquetas distintas y querremos saber cuántos elementos hay, sus posiciones relativas a la imagen y la categoría a la que pertenecen cada una de ellas.

La gran mayoría de los paradigmas de detección utilizan una red neuronal pre-entrenada como clasificador dentro de sus estructuras. Suele recibir modificaciones en sus últimas capas ya sea sustituyéndolas, pasando por un proceso de *fine-tunning*, o eliminándolas antes de ser añadidas como un elemento invariante durante el entrenamiento del modelo. Estas redes suelen recibir el nombre de *backbone* y algunos modelos permiten la libre elección de estos clasificadores dependiendo del problema concreto que se desee abordar.

Para indicar las posiciones de los elementos, por lo general se utilizarán *bounding-box* o *bbox* que serán rectángulos que contendrán cada uno de los elementos o *pixel level* que colorea cada uno de los píxeles pertenecientes a determinada categoría, ambos con la mayor precisión posible.



(a) Image Classification (b) Object Detection

(c) Semantic Segmentation (d) Instance Segmentation

Figura 3.1.: Detección, clasificación y segmentación. [WSH19]

3.1. Dos pasos o basados en RCNN

El problema de detección se divide en dos etapas: una generación de propuestas y la realización de predicciones sobre estas propuestas. Como paradigmas de detección, destacan la familia *R-CNN* y los que se encuentran basados en esta.

Esta familia surge en noviembre de 2013 con *R-CNN* [GDDM13] que utiliza Selective Search [**?**] para generar 2000 bbox de propuestas que son redimensionadas para coincidir con las dimensiones de entrada una CNN pre-entrenada. Esta CNN debe volver a entrenarse, extrayéndole la última capa y añadiéndole una *máquina de vectores de soporte(SVM)* con las categorías originales más una nueva clase llamada "fondo"que engloba a todas las propuestas que no pertenecen a ninguna categoría. Finalmente, las propuestas clasificadas son combinadas a través de un modelo de regresión lineal para así obtener una bbox con mejor ajuste y precisión.

La principal diferencia con *Fast R-CNN* [Gir15] reside en que, en lugar de pasar por la CNN todas las propuestas de Selective Search, nuestra CNN recibe como entrada la imagen completa reduciendo el coste computacional al analizar una única vez las zonas de solapamiento entre propuestas. FIXME: Con lo apuntado en la pizarrita.

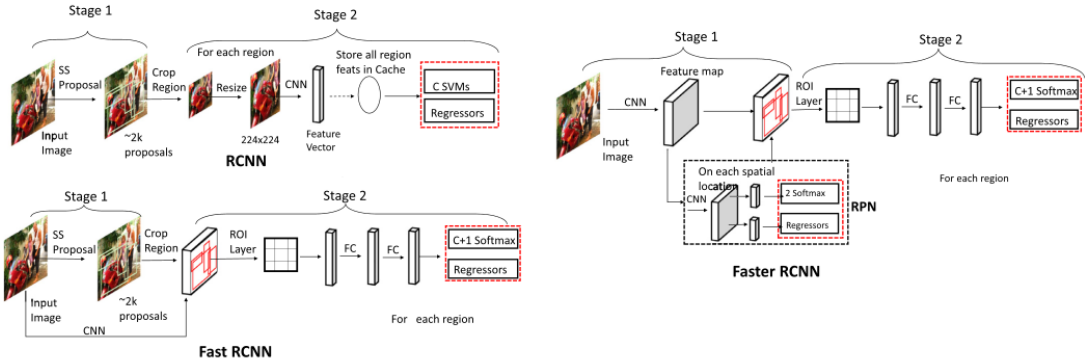


Figura 3.2.: Comparativa de algunos modelos de la familia R-CNN. [WSH19]

En *Faster R-CNN* [RHGS15] se sustituye Selective Search por una red neuronal de generación propuestas que no sólo reduce el tiempo de cómputo de la generación de propuestas sino que aporta un menor número de propuestas con mayor calidad y precisión. Un buen apunte de la red de generación de propuestas, es que esta no es pre-entrenado antes de ser añadido al modelo sino que aprende de forma conjunta con toda la red.

La última mejora de esta familia viene representada por *Mask R-CNN* [HGDG17] que sustituye la capa RoI pooling introducida en el modelo R-CNN por una RoI alignment que retiene más información de las características obtenidas por la CNN. Mask R-CNN aprovecha esto para realizar predicciones del tipo pixel level con mayor precisión a través de una interpolación bilineal que se ejecuta paralelamente con las capas totalmente conectadas de los modelos anteriores.

Mencionar Mesh R-CNN https://arxiv.org/abs/1906.02739

3.2. Un paso o basado en YOLO

3.3. Feature Pyramid Networks

https://arxiv.org/pdf/1612.03144.pdf

3.4. Modelo elegido

A. Primer apéndice

Los apéndices son opcionales.
Archivo: `apendices/apendice01.tex`

Glosario

La inclusión de un glosario es opcional.
Archivo: `glosario.tex`

$\mathbb{R}$ Conjunto de números reales.

$\mathbb{C}$ Conjunto de números complejos.

$\mathbb{Z}$ Conjunto de números enteros.

Bibliografía

Las referencias se listan por orden alfabético. Aquellas referencias con más de un autor están ordenadas de acuerdo con el primer autor.

[BM12] Joan Bruna and Stéphane Mallat. Invariant Scattering Convolution Networks. *arXiv e-prints*, page arXiv:1203.1513, March 2012. [No citado]

[GDDM13] Ross Girshick, Jeff Donahue, Trevor Darrell, and Jitendra Malik. Rich feature hierarchies for accurate object detection and semantic segmentation. *arXiv e-prints*, page arXiv:1311.2524, November 2013. [Citado en pág. 12]

[Gir15] Ross Girshick. Fast R-CNN. *arXiv e-prints*, page arXiv:1504.08083, April 2015. [Citado en pág. 12]

[HGDG17] Kaiming He, Georgia Gkioxari, Piotr Dollár, and Ross Girshick. Mask R-CNN. *arXiv e-prints*, page arXiv:1703.06870, March 2017. [Citado en pág. 12]

[HPC+20] Ping Hu, Federico Perazzi, Fabian Caba Heilbron, Oliver Wang, Zhe Lin, Kate Saenko, and Stan Sclaroff. Real-time Semantic Segmentation with Fast Attention. *arXiv e-prints*, page arXiv:2007.03815, July 2020. [No citado]

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. [No citado]

[HZXL19] Han Hu, Zheng Zhang, Zhenda Xie, and Stephen Lin. Local relation networks for image recognition. *CoRR*, abs/1904.11491, 2019. [No citado]

[LPM15] Minh-Thang Luong, Hieu Pham, and Christopher D. Manning. Effective approaches to attention-based neural machine translation. *CoRR*, abs/1508.04025, 2015. [No citado]

[OMS17] Chris Olah, Alexander Mordvintsev, and Ludwig Schubert. Feature visualization. *Distill*, 2017. https://distill.pub/2017/feature-visualization. [No citado]

[RHGS15] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. *arXiv e-prints*, page arXiv:1506.01497, June 2015. [Citado en pág. 12]

[VSP+17] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017. [No citado]

[WGGH17] Xiaolong Wang, Ross B. Girshick, Abhinav Gupta, and Kaiming He. Non-local neural networks. *CoRR*, abs/1711.07971, 2017. [No citado]

[WSH19] Xiongwei Wu, Doyen Sahoo, and Steven C. H. Hoi. Recent Advances in Deep Learning for Object Detection. *arXiv e-prints*, page arXiv:1908.03673, August 2019. [Citado en págs. IX, IX, 11, and 12]