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 11 de noviembre de 2020

Fdo: Laura Gómez Garrido

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

Índice general

Índice de figuras

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.

Conceptos previos

En esta parte, introduciremos una serie de conceptos previos necesarios para poder hablar con propiedad en adelante. Explicaremos los conceptos de red neuronal y red neuronal convolucionada. Veremos que se las redes neuronales son aproximadores universales y, además, conoceremos el concepto operación no local así como ver su consistencia.

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.

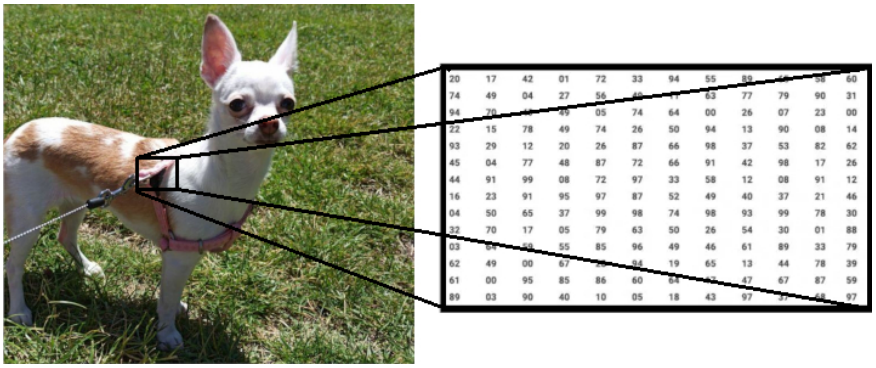

Figura 1.1.: Cómo un ordenador ve una imagen

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 pocos 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 secciones futuras, 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, para cada etiqueta, determinar las *k* imágenes cuya distancia, siendo Manhattan y Euclídea las más comunes, entre los píxeles de nuestra imagen sea menor. La etiqueta idónea será aquella cuya dicha distancia sea menor considerando las *k* imágenes.

De esta sencilla propuesta surgen diversos problemas. Las métricas más comunes de este clasificador suelen darle mayor importancia al valor concreto de los píxeles, en lugar de a las formas que de las que está compuesta la figura, provocando 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

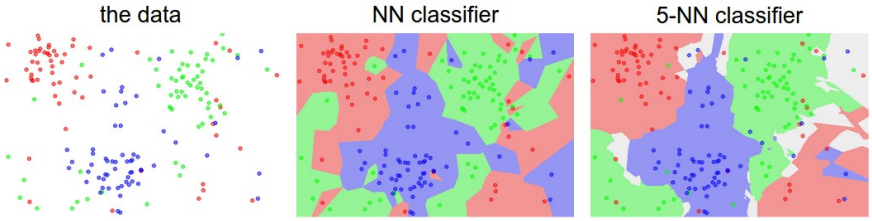

Figura 1.2.: Un ejemplo de la diferencia entre un *Nearest Neighbor classifier* y un *5-Nearest Neighbor classifier* con puntos bidimensionales y tres clases.[aut]

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 imágenes a comparar, ya sea por aumentar el número de etiquetas o nutrir de más datos las ya existentes.

Debido a la falta de escalabilidad de este método de clasificación no paramétrico, existe un gran problema ante el incremento de los datos de entrenamiento. Un primer ejemplo de la búsqueda de aumentar la escalabilidad del algoritmo sin un aumento de los tiempos de clasificación, podrían ser los clasificadores lineales.

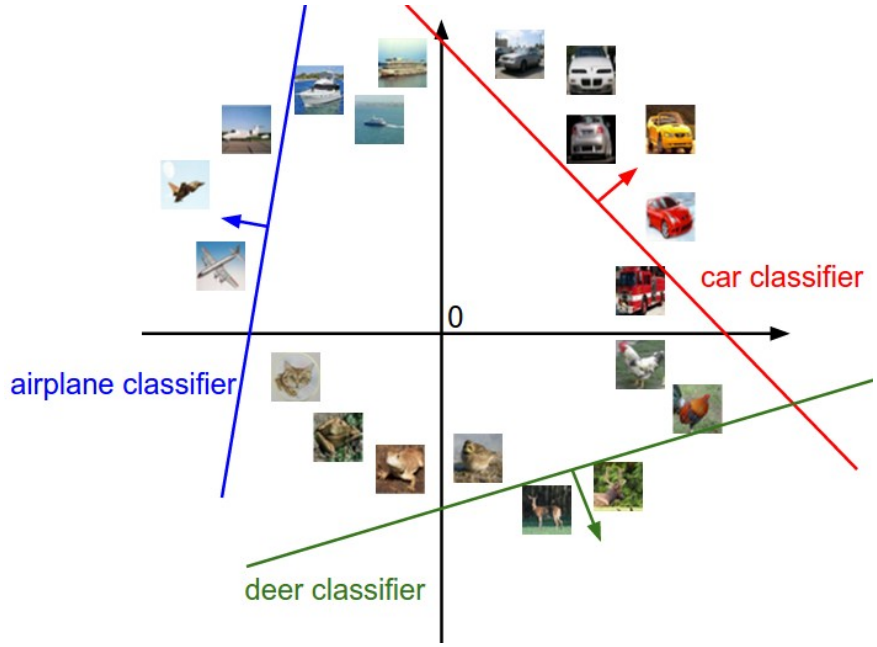

Figura 1.3.: Ejemplo de representación de un clasificador lineal con tres etiquetas.[aut]

Consideremos $D \in \mathbb{N}$ la dimensión de nuestras imágenes o datos de entrada 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. Utilizando estos datos, podremos entrenar el clasificador lineal que será una función del tipo

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

con la cual estaremos dividiendo el espacio de resultados utilizando hiperplanos. Existen múltiples tipos de clasificadores lineales, como por ejemplo una *máquina de vectores de soporte multiclase (Multiclass SVM)* o un clasificador SoftMax que tienen como principal diferencia la función de pérdida que utilizan para penalizar las etiquetas incorrectas.

1.2. El modelo de una neurona

Cuando comenzamos planteando nuestro problema buscábamos conseguir clasificar una imagen con una probabilidad de acertar similar o superior a la humana. 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 consideraremos los modelos de redes neuronales, que buscan ser capaces de simular cómo funciona un cerebro humano para aprender cómo reconocer distintos elementos de la misma forma que nosotros lo hemos ido haciendo a lo largo de nuestra vida.

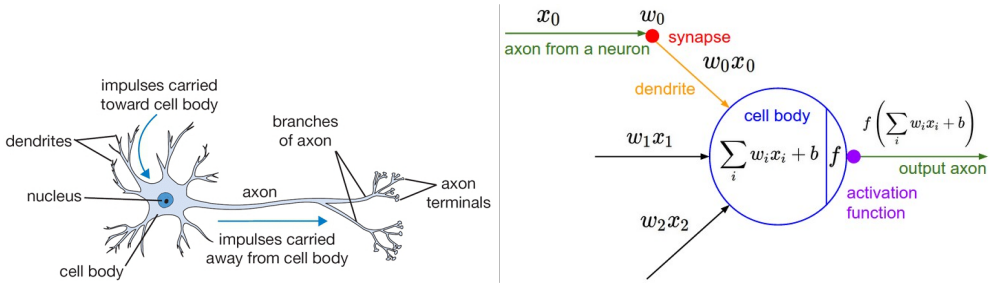

Figura 1.4.: Comparación entre una neurona biológica (izquierda) y el modelo matemático (derecha) [aut]

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 1.4 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.

Si consideramos que tenemos D dentritas y que por la i-ésima dentrita recibimos la información x_i con un peso o fuerza de sinapsis w_i, tendríamos que nuestro núcleo trabaja con el vector de información $(w_1 x_1, ..., w_D x_D)$ que podemos estimar como $\sum_i w_i x_i$ y sumarle una determinada constante b propia de la neurona. El umbral de activación y la señal enviada al resto de neuronas será la evaluación a través de una *función de activación* Def. 1.1. Cabe destacar la similitud existente entre el modelo de una neurona y un clasificador lineal.

Definición 1.1. Una *función de activación* es una función $f : \mathbb{R}^n \to \mathbb{R}^n$ siendo $n \in \mathbb{N}$ el número de neuronas de la capa a evaluar.

A continuación, mencionaremos los ejemplos más representativos utilizados como funciones de activación:

FIXME: Sugerencia: Añadir una gráfica para cada función de activación

- *Función sigmoide* o *logística* $\sigma : \mathbb{R}^n \to [0,1]^n$ definida como $\sigma_i(x_i) = \frac{1}{1+e^{-x_i}}$. Se suele utilizar en la última capa de nuestra red, cuando nuestras imágenes pueden pertenecer a varias clases o etiquetas al mismo tiempo al ser el resultado para cada componente independiente del resto.

- *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 logística que normaliza la salida para obtener la probabilidad de pertenecer a cada clase obteniendo así que la suma de todas las salidas de esta capa sería 1. Aquí, la i-ésima neurona tendría función de activación $\sigma_i(x) = \frac{e^{x_i}}{\sum_{j=1}^{K} e^{x_j}}$ y utilizaría $L(x) = \frac{1}{N}\sum_{i=1}^{N} -\ln \frac{e^{\sigma_{y_i}(x)}}{\sum_{j=1}^{D} e^{\sigma_j(x)}} = \frac{1}{N}\sum_{i=1}^{N}(-\sigma_{y_i}(x) + \ln \sum_{j=1}^{D} e^{\sigma_j(x)})$ como función de pérdida para el entrenamiento.

- *Función ReLU*, cuyo nombre completo sería *Rectified Linear Unit*. Se suele utilizar en las capas intermedias de nuestra red y es de la forma $f(x) = \max(0, x)$.

- *Función tanh* se trata de de una centralización de la función sigmoide. $Tanh(x) = 2\sigma(x) - 1$.

1.3. 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 dirigido.

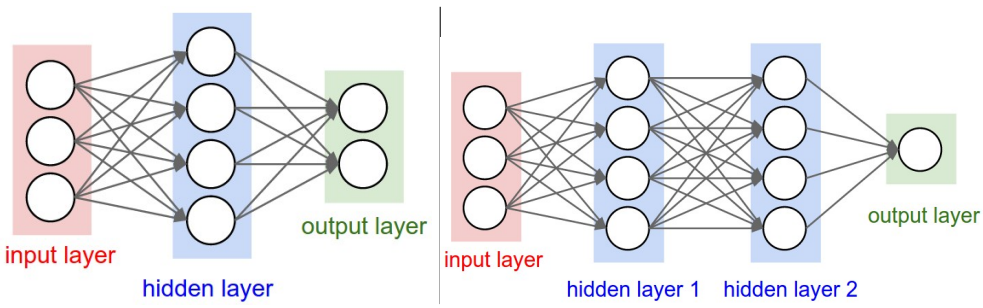

Figura 1.5.: Ejemplos de redes totalmente conectadas (*fully-connected*)

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 de las *capas ocultas* cuya distribución y conexiones dependen del modelo en concreto

que queramos desarrollar, quedando a nuestro criterio.

Así, podemos afirmar que una red neuronal es la aplicación sucesiva de funciones de la forma

$$F(x) = \sum_{j=1}^{n} \alpha_j \sigma(w_j^T x + b_j) \quad \forall x \in \mathbb{R}^n,$$

donde $w_j \in \mathbb{R}^n$, $\alpha_j, b_j \in \mathbb{R}$ serán fijas una vez entrenada la red y $\sigma : \mathbb{R} \to \mathbb{R}$ la función de activación elegida en cada capa.

1.3.1. Función de pérdida y métricas

La función de pérdida, o *loss function*, se trata de la función objetivo que querremos minimizar en el problema de optimización planteado durante el entrenamiento de una red neuronal. Existen muchas funciones de pérdida, que se pueden englobar en tres grandes categorías:

- Funciones de pérdida probabilísticas. Son aquellas cuyo resultado es interpretable como una medida de probabilidad. Un ejemplo de esto es la entropía cruzada, o *cross entropy*, que mide la media de bits necesarios para identificar un evento.
- Funciones de pérdida de regresión. Son aquellas cuyos valores son reales o continuos. Como ejemplos se tiene el error cuadrático medio o la similitud del coseno.
- Funciones de pérdida para clasificaciones "máximo-margen". Un ejemplo de este tipo es la función de pérdida de bisagra, o *hinge loss function*.

Existen varias formas de medir el funcionamiento de una red neuronal sin que estas medidas lleguen a afectar al entrenamiento. Hay múltiples métricas de evaluación que se pueden utilizar en diferentes contextos, siendo la más común la precisión, exactitud o *accuracy*.

La métrica de *accuracy* corresponde al número de datos correctamente predichos con respecto a todos los puntos. Formalmente está definida como el número de verdaderos positivos y negativos divido entre los falsos positivos, verdaderos positivos, falsos negativos y verdaderos negativos.

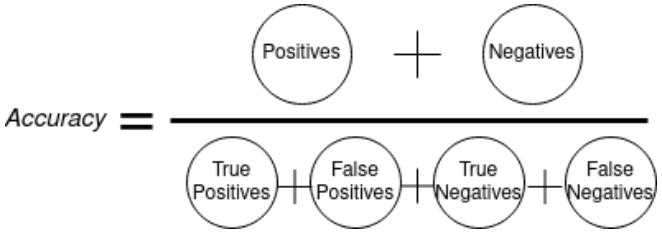

Figura 1.6.: Exactitud o *accuracy*

Por otro lado, se mencionará también la intersección sobre la unión, mayormente conocida como *intersection over union (IoU)*, que contabiliza el área de intersección y de unión entre las predicciones y los verdaderos valores para después dividir el primero entre el segundo.

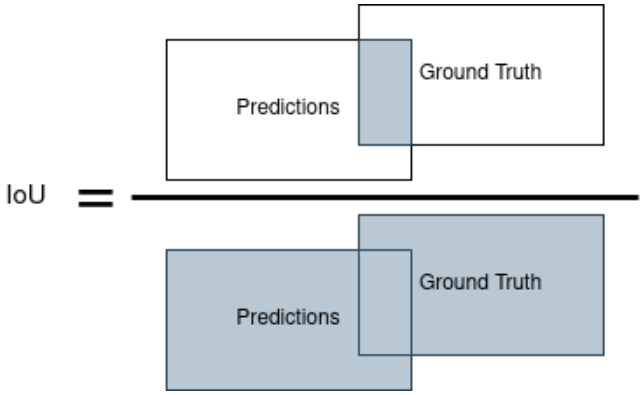

Figura 1.7.: intersección sobre la unión o *intersection over union (IoU)*

1.3.2. Propagación y Optimización

Con el fin de simplificar la explicación de los algoritmos que se verán en esta subsección, nos centraremos en el caso de las redes neuronales prealimentadas, o *feed forward neural networks*, que son aquellas donde las conexiones entre las distintas neuronas no forman un ciclo. También son conocidas como redes neuronales hacia adelante puesto que la información únicamente se mueve hacia adelante, diferenciándose de las redes neuronales recurrentes.

El algoritmo que computa el avance de la información a través de estas redes recibe el nombre de propagación hacia adelante, o *forward propagation*, [GBC16]. Durante el entrenamiento, la información de entrada continua hasta producir un coste escalar $J(\theta)$ y el algoritmo de propagación hacia atrás, o *back propagation*, permite que ese coste fluya hacia atrás en la red para calcular el gradiente.

Algorithm 1 Propagación hacia adelante mediante una red neuronal típica, totalmente conectada, y el cálculo de la función de coste. La función de pérdida $L(\hat{y}, y)$ depende de la salida de la red $\hat{y}$ y del objetivo y. Para obtener el coste total J, a la pérdida se le debería de añadir un regularizador $\Omega(\theta)$, donde θ contiene todos los parámetros (pesos y sesgos). [GBC16]

Require: Profundidad de la red l
Require: $W^{(i)}, i \in \{1, ..., l\}$, las matrices de pesos del modelo
Require: $b^{(i)}, i \in \{1, ..., l\}$, los parámetros de sesgos del modelo
Require: x, la entrada al proceso
Require: y, la salida objetivo
  $h^{(0)} = x$
  for $k = 1, ..., l$ **do**
    $a^{(k)} = b^{(k)} + W^{(k)} h^{(k-1)}$
    $h^{(k)} = \sigma(a^{(k)})$
  end for
  $\hat{y} = h^{(l)}$
  $J = L(\hat{y}, y) + \lambda \Omega(\theta)$

A menudo, se piensa que con la propagación hacia atrás se está realizado todo el proceso

de aprendizaje de la red, cuando este algoritmo únicamente calcula el valor del gradiente y serás otro algoritmo, como, por ejemplo, el descenso del gradiente estocástico (*stochastic gradient descent*), es el utilizado para realizar el aprendizaje por la red. Los algoritmos utilizados para calcular este aprendizaje reciben el nombre de optimizadores.

La regla de la cadena del cálculo es utilizada para calcular las derivas de funciones compuestas por otras funciones cuyas derivadas son conocidas. De esta forma, el algoritmo de propagación hacia atrás realiza el cálculo de esta regla de la cadena, en un orden específico de operaciones que es muy eficiente.

Algorithm 2 Computación hacia atrás para la red neuronal profunda del algoritmo Subsección 1.3.2, que usa, en adición a la entrada x, un objetivo y. [GBC16]

Después del cálculo hacia adelante, calculamos el gradiente de la capa de salida:
$g \leftarrow \nabla_{\hat{y}} J = \nabla_{\hat{y}} L(\hat{y}, y)$
for $k = l, l-1, ..., 1$ **do**
  Se convierte el gradiente de la salida de las capas en un gradiente en la función de activación:
  $g \leftarrow \nabla_{a^{(k)}} J = g \odot \sigma'(a^{(k)})$
  Se calcula el gradiente el los pesos y sesgos, incluidos los términos de regularización cuando estos sean necesarios:
  $\nabla_{b^{(k)}} J = g + \lambda \nabla_b^(k) \Omega(\theta)$
  $\nabla_{W^{(k)}} J = g h^{(k-1)} + \lambda \nabla_W^{(k)} \Omega(\theta)$
  Se propaga el gradiente.
  $g \leftarrow \nabla_{h^{(k-1)}} J = W^{(k)T} g$
end for

Todo esto es para poder plantear el objetivo más importante del entrenamiento de una red neuronal, el resolver un problema de optimización planteado al querer minimizar el valor de la función de pérdida.

Para ello, se describirá el algoritmo del descenso del gradiente estocástico, o *stochastic gradient descent (SGD)*, que, junto a sus variantes, son los algoritmos de optimización más utilizados en el aprendizaje profundo.

Algorithm 3 Descenso del gradiente estocástico con cantidad de movimiento, o *momentum*. [GBC16]

Require: Ratio de aprendizaje ϵ y *momentum* α.
Require: Parámetro inicial θ y velocidad inicial v.
while no se cumpla el criterio de parada **do**
  Sea $\{x^{(1)}, ..., x^{(m)}\}$ un subconjunto del conjunto de entrenamiento con sus correspondientes etiquetas $y^{(i)}$.
  Calcular la estimación del gradiente: $g \leftarrow \frac{1}{m} \nabla_\theta \sum_i L(f(x^{(i)}; \theta), y^{(i)})$.
  Calcular la actualización de la velocidad: $v \leftarrow \alpha v - \epsilon g$.
  Aplicar actualización: $\theta \leftarrow \theta + v$.
end while

Finalmente, describiremos la variante *Adam* del *SGD* puesto que será la que utilicemos como optimizador más adelante.

Algorithm 4 El algoritmo *Adam*. [GBC16]

Require: Tamaño de paso ϵ. (Sugerido por defecto: 0.0001)
Require: Ratio exponencial de caída para las estimaciones de momentos, ρ_1 y ρ_2 en $[0, 1)]$. (Sugerido por defecto 0.9 y 0.999 respectivamente)
Require: Una pequeña constante δ utilizada para una estabilización numérica. (Sugerida: 10^{-8})
Require: Un parámetro inicial θo
  Incializar las variables de momentos $s = 0$ y $r = 0$.
  Inicializar el paso de tiempo $t = 0$.
  while No se cumple el criterio de parada **do**
    Sea $\{x^{(1)}, ..., x^{(m)}\}$ un subconjunto del conjunto de entrenamiento con sus correspondientes etiquetas $y^{(i)}$.
    Calcular la estimación del gradiente: $g \leftarrow \frac{1}{m} \nabla_\theta \sum_i L(f(x^{(i)}; \theta), y^{(i)})$.
    Actualizar la estimación del primer momento sesgado: $s \leftarrow \rho_1 s + (1 - \rho_1) g$.
    Actualizar la estimación del segundo momento sesgado: $r \leftarrow \rho_2 r + (1 - \rho_2) \odot g$.
    Corregir el sesgo del primer momento: $\hat{s} \leftarrow \frac{s}{1-\rho_1^t}$.
    Corregir el sesgo del segundo momento: $\hat{r} \leftarrow \frac{r}{1-\rho_2^t}$.
    Calcular la actualización: $\Delta\theta = -\epsilon \frac{\hat{s}}{\sqrt{\hat{r}}+\delta}$ (Operaciones aplicadas elemento a elemento)
    Aplicar la actualización: $\theta \leftarrow \theta + \Delta\theta$
  end while

Para más detalles sobre estos algoritmos, se recomienda la consulta de los libros *Deep Learning* [GBC16] y *Learning from data* [AMMIL12].

2. Teoremas de Aproximación Universal

Hasta ahora se ha definido una estructura con una serie de parámetros con la esperanza de poder llegar a resolver el problema planteado. Sin embargo, en ningún momento hemos utilizado ningún resultado que nos asegure que dicha estructura puede llegar a resolver el problema que deseamos. A continuación, mostraremos diversos resultados, con diversos niveles de generalidad, que muestran cómo esta estructura es un aproximador universal y por ello resuelve nuestro problema. Estos resultados se dividen en dos grandes categorías *anchura indeterminada* y *profundidad indeterminada*.

2.1. Anchura indeterminada

2.1.1. George Cybenko, 1989

Cybenko, en [Cyb89], demostró la capacidad de aproximación en el caso de anchura indeterminada y profundidad fijada para funciones de activación sigmoidales Def. 2.1. A partir de esta versión clásica, se logró considerar otras funciones de activación, e incluso, demostrar que era gracias a la arquitectura en sí misma, y no a la función de activación, que las redes neuronales eran aproximadores universales [KH91]. Aquí estudiaremos la versión del teorema para funciones continuas, pero debemos destacar que en el mismo artículo se mencionan también versiones para otros espacios de funciones.

Definición 2.1. Una función $\sigma : \mathbb{R} \to \mathbb{R}$ es *sigmoidal* si

$$\lim_{t \to +\infty} \sigma(t) = 1 \quad y \quad \lim_{t \to -\infty} \sigma(t) = 0.$$

Definición 2.2. Dada una σ-álgebra M, una *medida con signo* μ sobre M es una función del conjunto $\mu : M \to [-\infty, +\infty]\sigma$-aditiva.

Definición 2.3. Una medida se dice *de Borel* si está definida sobre una σ-álgebra de Borel, es decir, la engendrada por los abiertos del espacio.

Definición 2.4. Una *medida con signo regular de Borel* sobre una σ-álgebra M es una medida con signo que cumple

$$\mu(E) = \inf\{\mu(V) : E \subset V, V \text{ abierto}\} = \sup\{\mu(C) : C \subset E, C \text{ cerrado}\}$$

para todo conjunto de Borel $E \in M$

En adelante, utilizaremos I_n para referirnos al cubo unidad n-dimensional, $[0,1]^n$ y para el espacio de las medidas finitas con signo regulares de Borel sobre I_n utilizaremos $M(I_n)$. Además, $C(I_n)$ denotará el espacio de funciones continuas en I_n.

Definición 2.5. Una función $\sigma : \mathbb{R} \to \mathbb{R}$ es *discriminatoria* si, para una medida $\mu \in M(I_n)$ con

$$\int_{I_n} \sigma(w^T x + b) d\mu(x) = 0$$

para todo $w \in R^n$ y $b \in \mathbb{R}$, implica que $\mu = 0$.

Lema 2.1. *Cualquier función sigmoidal medible y acotada es discriminatoria. En particular, las funciones sigmoidales continuas son discriminatorias.*

Demostración. □

Teorema 2.1. *Sea σ una función discriminatoria continua. Entonces, las sumas finitas de la forma*

$$G(x) = \sum_{i=1}^{n} \alpha_i \sigma(w_i^T x + b)$$

son densas en $C(I_n)$. En otra palabras, dados $f \in C(I_n)$ y $\varepsilon > 0$, existe una suma, $G(x)$, de la forma anterior, para la cual

$$|G(x) - f(x)| < \varepsilon \quad \forall x \in I_n.$$

Demostración. □

Teorema 2.2. *Sea σ una función sigmoidal continua. Entonces, las sumas finitas de la forma*

$$G(x) = \sum_{i=1}^{n} \alpha_i \sigma(w_i^T x + b)$$

son densas en $C(I_n)$. En otra palabras, dados $f \in C(I_n)$ y $\varepsilon > 0$, existe una suma, $G(x)$, de la forma anterior, para la cual

$$|G(x) - f(x)| < \varepsilon \quad \forall x \in I_n.$$

Demostración. Para esta demostración, se combina Lema 2.1 y Teorema 2.1, notando que las funciones sigmoidales continuas satisfacen las condiciones de este lema. □

2.1.2. Kurt Hornik, 1989 y 1991

En 1898 [HSW89], Hornik demostró que las redes neuronales con una capa oculta que utiliza funciones de aplastamiento (sigmoidal Def. 2.1 y no decreciente según la *definición 2.3* del mismo artículo) arbitrarias son capaces de aproximar cualquier función Borel-medible de un espacio finito unidimensional con cualquier grado deseado de precisión.

Más tarde, en 1991 [KH91], extendió los teoremas de Cybenko mostrando que no necesariamente tenía que utilizar funciones de activación sigmoidales para los espacios de funciones considerados. A continuación enunciaremos el teorema para el espacio de funciones continuas.

Teorema 2.3. *Sea $\sigma : \mathbb{R} \to \mathbb{R}$ una función continua, acotada y no constante. Entonces, las sumas finitas de la forma*

$$G(x) = \sum_{i=1}^{n} \alpha_i \sigma(w_i^T x - b)$$

son densas en $C(X)$ para todos los subconjuntos compactos X de $\mathbb{R}^m$.

2.2. Profundidad indeterminada

2.2.1. Zhou Lu, Hongmin Pu, Feicheng Wang, Zhiquang Hu y Liwei Wang, 2017

Estos autores demostraron [LPW+17] el caso de profundidad indeterminada para funciones Lebesgue-integrables y una función de activación ReLU. Este teorema fue presentado como una versión dual de las demostraciones para anchura indeterminada y abre el camino para nuevas demostraciones en el caso de anchura indeterminada.

Teorema 2.4. *Para cualquier función Lebesgue-integrable $f : \mathbb{R}^n \to R$ y cualquier $\varepsilon > 0$, existe A una red totalmente conectada con función de activación ReLU y una anchura $d_m \leq n + 4$, tal que la función F_A representada por esta red satisface*

$$\int_{\mathbb{R}^n} |f(x) - F_A(x)| dx < \varepsilon.$$

2.2.2. Patrick Kidger y Terry Lyons, 2019

Una de las variantes más recientes [KL19], fue presentada para el caso de una función de activación no afín, que sea continuamente diferenciable y con derivada no nula en al menos un punto. Destaca porque con sus consideraciones abarca las funciones de activación utilizadas en la práctica, incluyendo las funciones de activación polinómicas. En el artículo, se consideran otras extensiones o variaciones al teorema que enunciaremos.

Definición 2.6. Sea $\sigma : \mathbb{R} \to \mathbb{R}$ y $n, m, k \in \mathbb{N}$. Entonces $NN^{\sigma}_{n,m,k}$ representa la clase de funciones $R^n \to \mathbb{R}^m$ descritas por una red neuronal hacia adelante con n neuronas en la capa de entrada, m neuronas en la capa de salida, y un numero arbitrario de capas ocultas, para las cuales k neuronas tienen como función de activación la función σ. Cada neurona de la capa de salida tiene la función de activación identidad.

Teorema 2.5. *Sea $\sigma : \mathbb{R} \to \mathbb{R}$ una función continua y no polinómica que es continuamente diferenciable en al menos un punto, con derivada no nula en dicho punto. Sea $K \subset \mathbb{R}^n$ un compacto. Entonces $NN^{\sigma}_{n,m,n+m+2}$ es denso en $C(K; \mathbb{R}^m)$ con respecto a la norma del supremo.*

Teorema 2.6. *Sea $\sigma : \mathbb{R} \to \mathbb{R}$ una función polinómica y no afín. Sea $K \subset \mathbb{R}^n$ un compacto. Entonces $NN^{\sigma}_{n,m,n+m+2}$ es denso en $C(K; \mathbb{R}^m)$ con respecto a la norma del supremo.*

3. 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 a presentar.

Para la edición y manipulación de imágenes, es común utilizar la operación de convolución como filtro. Las redes neuronales son aquellas que utilizan estos filtros, de ahí su nombre, para analizar las distintas características de las imágenes. Si bien, podemos fijar los pesos manualmente para utilizar algunos de los filtros más comunes en la edición de imágenes, se ha comprobado experimentalmente que por lo general se obtienen mejores resultados si es la propia red quien fija estos pesos a través de un proceso de aprendizaje.

En adelante, dejaremos de considerar que nuestros datos de entrada tienen una estructura de vector y consideraremos que estamos trabajando con matrices tridimensionales, es decir, nuestros datos estarán organizados como si fueran ortoedros. Esto lo representaremos como $M_{a,h,c}(\mathbb{R})$ con $a, h, c \in \mathbb{N}$ donde a representará la anchura, h la altura y c el número de canales. Para simplificarlo, cada entrada $X \in M_{a,h,c}(\mathbb{R})$ representará c imágenes en escala de grises de dimensiones $a \times h$. Como detalle, una imagen RGB está representada como tres imágenes en escala de grises($c = 3$) donde cada canal representa un color primario y con la combinación de los tres obtenemos una imagen a color.

3.1. Capas Convolucionadas o Convolutionals

Internamente, cada neurona de una capa convolucional posee un *kernel* o *filtro* $W \in M_{r,s,c}(\mathbb{R})$ donde r, s y c son parámetros prefijados y una variable $b \in \mathbb{R}$ bias. Para cada entrada $X \in M_{a,h,c}(\mathbb{R})$ 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.

Figura 3.1.: Capa de Convolución. La zona grisácea corresponde al resultado de un filtro.

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 $x \in M_{7,7,3}$. Nótese, que el filtro no es aplicado en todas las submatrices sino que 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.

3.2. Capas de Agrupación o Pooling

FIXME: Same convoluciones

3.3. 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

4. Operador no local

En la sección anterior se ha visto que una operación de convolución posee un kernel que limita la cantidad de posiciones que son observadas de forma que la cantidad de parámetros de la capa de convolución es dependiente al tamaño de este núcleo. Esto tiene como desventaja que, para cada posición de nuestra entrada de datos, las posiciones que influyen al resultado de la convolución están restringidas a un subconjunto de tamaño fijo de los datos de entrada por lo que se suele decir que es una operación *local*. Si quisiéramos utilizar todos los datos de entrada para cada posición de la imagen, tendríamos un aumento considerable de parámetros a utilizar, con los consecuentes problemas de memoria y entrenamiento.

El objetivo de esta sección es definir un tipo de operación que sea capaz de extraer características de nuestros datos de entrada utilizando toda su dimensión y sin tener un aumento drástico del número de parámetros a entrenar, será a lo que llamaremos un operador no local. La necesidad de este tipo de operación nace del hecho de que, por lo general, las imágenes a estudiar no suelen ser un conjunto de *collage* sin relación entre sí, sino que el entorno de nuestra imagen puede llegar a ser de utilidad a la hora de distinguir qué objetos estamos clasificando.

Para simplificar la definición, nos restringiremos al contexto de espacios vectoriales de dimensión finita sobre el cuerpo de los números reales.

Definición 4.1. Sea $x = (x_1, ..., x_D) \in \mathbb{R}^D$ una señal de entrada, $u : \mathbb{R} \times \mathbb{R} \to \mathbb{R}$ un producto escalar, $v : \mathbb{R} \to \mathbb{R}$ función real evaluada y $C : \mathbb{R}^D \to \mathbb{R}$. Se define una operación no local genérica o *generic non-local operation* $f : \mathbb{R}^D \to \mathbb{R}^D$ con $f(x) = (f_1(x_1), ..., f_D(x_D))$ aquella donde:

$$f_i(x_i) = \frac{1}{C(x)} \sum_{j=1}^{D} u(x_i, x_j) v(x_j) \quad \forall i = 1, ..., D$$

En este contexto, $v(x_j)$ será una representación de la señal de x entrada en la posición j, u representará una relación, por ejemplo la afinidad, entre las posiciones i y j.

Proseguiremos enunciando varios ejemplos de funciones que pertenezcan a dicha definición para mostrar la diversidad que podemos elegir a la hora de implementar un modelo que siga esta estructura. Por simplicidad, consideraremos que $v(x_j) = W_v x_j$ donde W_v es una matriz de pesos que tiene que ser aprendida, siendo esta fácilmente implementable como una convolución de núcleo de tamaño 1.

1. *Gaussiana* $u(x_i, x_j) = e^{x_i^T x_j}$ y $C(x) = \sum_{j=1}^{D} u(x_i, x_j)$.
2. *Embedded Gaussian* $u(x_i, x_j) = e^{\phi_i(x_i)^T, \theta_j(x_j)}$ donde $\phi_i(x_i) = W_{\phi_i} x_i$ y $\theta_j(x_j) = W_{\theta_j} x_j$ y $C(x) = \sum_{j=1}^{D} u(x_i, x_j)$. Nótese que el módulo *self-attention* [VSP+17] es un caso particular de este ejemplo.

3. *Dot-product similarity* $u(x_i, x_j) = \phi_i(x_i)^T \theta_j(x_j)$ donde $\phi_i(x_i) = W_{\phi_i} x_i$, $\theta_j(x_j) = W_{\theta_j} x_j$ y $C(x) = D$.

4. *Non-local means* [BCM05]

Para poder afirmar que un operador no local cumple con lo que buscamos, primero se debe demostrar que las operaciones de este tipo nos dan el resultado deseado. Para ello, primero deberemos de enunciar una serie de conceptos.

4.1. Conceptos previos

En adelante, el conjunto Ω será un espacio muestral y $\mathcal{A}$ será una *σ-álgebra* sobre dicho Ω. Además, P será una probabilidad aplicada a los elementos de la σ-álgebra $\mathcal{A}$. Esto será representado como $(\Omega, \mathcal{A}, P)$ siendo un espacio probabilístico y será donde trabajemos.

Definición 4.2 (Proceso estocástico)**.** Un *proceso estocástico* es una familia de variables aleatorias $\{X_t\}_{t \in T}$ en T, donde T es un conjunto ordenado arbitrario y cada variable aleatoria está definida sobre un espacio de probabilidad $(\Omega, \mathcal{A}, P)$.

A partir de ahora, siempre que nos refiramos a un proceso nos estaremos refiriendo a un proceso estocástico bajo la definición anterior.

Definición 4.3 (Proceso con incrementos independientes)**.** Un proceso estocástico tiene *incrementos independientes* si $\forall n > 1, \forall t_1 < \ldots < t_n \in T$

$$X_{t_1}, X_{t_2} - X_{t_1}, \ldots, X_{t_n} - X_{t_{n-1}} \text{son variables aleatorias independientes.}$$

Definición 4.4 (Proceso con incrementos estacionarios)**.** Un proceso estocástico tiene *incrementos estacionarios* si $\forall s < t \in T, \forall h \in T$

$$(X_t - X_s) \sim (X_{t+h} - X_{s+h}).$$

$X \sim Y$ indica que X sigue la misma distribución que Y.

Definición 4.5 (Proceso estacionario)**.** Un proceso estocástico es *estrictamente estacionario* si $\forall n, \forall t_1 < \ldots < t_n \in T, \forall h \in T$

$$(X_{t_1}, \ldots, X_{t_n}) \sim (X_{t_1+h}, \ldots, X_{t_n+h}).$$

A continuación, definiremos el concepto proceso mezclado. En la literatura, los cuatro tipos de mezcla normalmente son referidos como ψ-mezclado, Φ-mezclado (o uniformemente mezclado), ρ-mezclado (o mezcla basada en la correlación maximal) y α-mezclado (o fuertemente mezclado). Cambiaremos la notación a Φ_i-mezcla $i = 1, ..., 4$ respectivamente, de forma que una Φ_i-mezcla implique una Φ_{i+1}-mezcla con $i = 1, ..., 3$.

Por notación, diremos también que $\mathcal{A}_m^n$ será la σ-álgebra inducida por las variables aleatorias Z_j con $m \leq j \leq n$ en el espacio muestral Ω. Entonces:

Definición 4.6 (Proceso mezclado)**.** La secuencia $\{Z_j\}_{j\geq 1}$ es conocida Φ_i-mezclada con $i = 1, ..., 4$ si, para cada $A \in \mathcal{A}_1^k$ y para cada $B \in \mathcal{A}_{k+n}^{\infty}$, las siguientes desigualdades son satisfechas:

- Para Φ_1-mezclada:

$$|P(A \cap B) - P(A)P(B)| \leq \Phi_1(n)P(A)P(B) \text{ con } \Phi_1(n) \downarrow 0 \text{ cuando } n \to \infty$$

- Para Φ_2-mezclada:

$$|P(A \cap B) - P(A)P(B)| \leq \Phi_2(n)P(A) \text{ con } \Phi_2(n) \downarrow 0 \text{ cuando } n \to \infty$$

- Para Φ_3-mezclada:

$$|P(A \cap B) - P(A)P(B)| \leq \Phi_3(n)[P(A)P(B)]^{\frac{1}{2}} \text{ con } \Phi_3(n) \downarrow 0 \text{ cuando } n \to \infty$$

- Para Φ_4-mezclada:

$$|P(A \cap B) - P(A)P(B)| \leq \Phi_4(n) \text{ con } \Phi_4(n) \downarrow 0 \text{ cuando } n \to \infty$$

4.2. Consistencia de un operador no local

La idea es que, bajo supuestos estacionarios, para cada píxel i una operación no local converge a la expectativa condicional de un píxel i observado en un vecindario del píxel. En este caso las condiciones de estacionariedad equivalen a decir que, a medida que crece el tamaño de la imagen, podemos encontrar muchas regiones similares para todos los detalles de la imagen. Es decir, conforme aumenta el tamaño de la imagen es más probable que, si encontramos un objeto perteneciente a determinada categoría, encontremos más objetos pertenecientes a dicha categoría.

Sea V un campo aleatorio (random field) y los datos de entrada v una realización de V. Sea Z una secuencia de variables aleatorias con $Z_i = \{X_i, Y_i\}$ con $Y_i = V(i)$ es real valuada y $X_i = V(I\backslash\{i\})\mathbb{R}^p$ valuada donde I representa el conjunto de posiciones. Una operación no local genérica será un estimador de la esperanza condicionada $E[Y_i|X_i = v(I\backslash i)]$.

Teorema 4.1. *Sea $Z = \{V(N_i\backslash i), V(i)\}_{i\geq 1}$ un proceso estocástico estrictamente estacionario y mezclado. Sea f^n una operación no local genérica aplicada a la secuencia $Z_n = \{V(N_i\backslash\{i\}), V(i)\}_{i\geq 1}^n$. Entonces,*

$$|f^n(j) - [Y_j|X_j = v(I\backslash\{j\})]| \to 0 \; a.s \; \forall j \in 1, ..., n$$

.

En un contexto más general, la demostración se puede encontrar en [BCM05] bajo la suposición de cualquiera de cuatro tipos de procesos mezclados enunciados en Def. 4.6

Parte II.

Segmentación semántica

Descripción :D FIXME

5. Introducción

Dependiendo del contexto de trabajo, se suelen utilizar diferentes terminologías para referirse a los mismos conceptos. Por ello, antes que nada, se mencionará los términos utilizados para referirnos a diferentes objetivos o problemas basándonos en [WSH19]. Figura 5.1

Diremos que estamos en un problema de clasificación de imágenes (global), o *(global) image classification*, cuando se pretenda asignar una única etiqueta a una imagen, de forma que esta categoría represente a toda la imagen o sea la que más se acerque a ella.

El siguiente paso, es la detección de objetos, o *object detection*, que será cuando la imagen pueda ser dividida en múltiples cuadros, o *frames*, que serán clasificados con una etiqueta cada uno, obteniendo así múltiples etiquetas para la misma imagen y conociendo la localización correspondiente a cada categoría.

Cuando se busca conocer los píxeles concretos pertenecientes a determinada categoría, estaremos ante un problema de segmentación semántica, o *semantic segmentation*, en el que estaremos distinguiendo las regiones de la imagen por el significado que estas pueden conllevar consigo.

Finalmente, cuando no sólo se distingue entre qué píxeles pertenecen a determinada categoría sino que también distinguimos cuáles conforman distintos objetos, estaremos ante un problema de instanciación semántica de objetos, o *instance segmentation*.

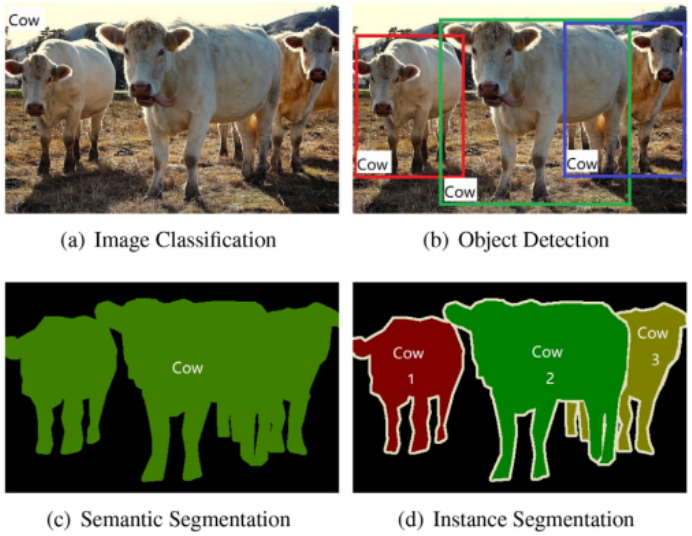

Figura 5.1.: Detección, clasificación y segmentación. [WSH19]

Para indicar las posiciones de los elementos, por lo general se utilizarán *bounding-box* o *bbox* que serán *frames* 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.

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. En concreto, estaremos ante un problema de segmentación semántica.

El objetivo no es utilizar directamente la mejor de las opciones sino explorar otras opciones, ya existentes, que puedan llegar a ser de utilidad en las investigaciones venideras. En particular, nos centraremos en los *non-local block* Capítulo 4 Capítulo 7 al crear ejemplos de redes neuronales y comprobar el comportamiento ante la eliminación o sustitución de este bloque.

Como tampoco debemos olvidarnos de lo que realmente se utiliza, en la siguiente sección hablaremos de del estado del arte para la detección de objetos, segmentación semántica e instanciación semántica en conjunto, al ser problemas íntimamente relacionados. Seguidamente mostraremos las estructuras analizadas y algunas de las gráficas obtenidas durante el entrenamiento de estas.

6. Estado del arte

La gran mayoría de los paradigmas de detección utilizan una red neuronal pre-entrenada como clasificador dentro de sus estructuras. Recibiendo 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.

En particular, se mencionará dos de las familias más fuertes para la detección en imágenes, la familia RCNN y la famila YOLO.

6.1. 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 que introduce *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. Además, las propuestas dejan de ser escaladas para coincidir con una determinada dimensión y se introduce en una capa especial llamada *Region of Interest Pooling Layer (RoI pooling)* que extrae un vector de longitud fija que será la entrada de las siguientes ramas. A continuación, el modelo se divide en dos ramas: un clasificador SoftMax y un modelo de regresión que calcula las *bbox*.

En *Faster R-CNN* [RHGS15] se sustituye Selective Search por una red neuronal de generación propuestas, conocida como *anchor boxes*, 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-entrenada antes de ser añadido al modelo sino que aprende de forma conjunta con toda la red.

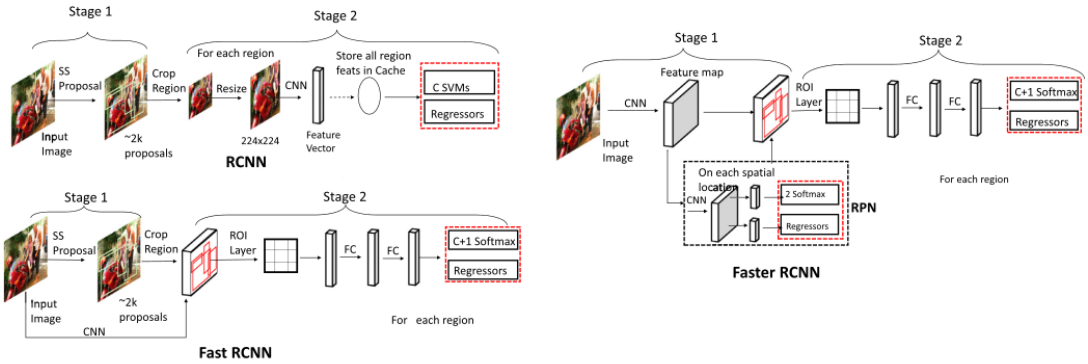

Figura 6.1.: Comparativa de algunos modelos de la familia R-CNN. [WSH19]

La última mejora de esta familia viene representada por *Mask R-CNN* [HGDG17] que sustituye la capa RoI pooling introducida en el modelo Fast R-CNN por una RoI alignment que retiene más información de las características obtenidas por la CNN compartiendo la misma estructura de entrada y salida de datos con la RoI pooling. 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. Así, en el mismo punto que sus predecesores eran divididos en tres ramas independientes, Mask R-CNN se divide en tres ramas: un clasificador SoftMax, un regresor de *bbox* y un otro regresor de pixel level.

Mencionar Mesh R-CNN https://arxiv.org/abs/1906.02739
Mencionar R-FCN https://arxiv.org/pdf/1605.06409.pdf
Libra R-CNN https://arxiv.org/pdf/1904.02701.pdf

6.2. Un paso o basados en YOLO

Estos modelos destacan por no hacer una separación directa de la generación de propuestas y la predicción de estas. Destaca *YOLO* y sus sucesivas mejoras, así como los múltiples algoritmos basados en ellas, pero existen muchos más modelos que pueden ser identificados con esta estructura.

En junio de 2015 aparece *You Only Look Once (YOLO)* [RDGF15], un nuevo algoritmo que pretende realizar al mismo tiempo la generación de propuestas y la clasificación de estas, buscando así una mayor eficiencia computacional. Para ello, tras redimensionar la imagen en 448x448 píxeles, el algoritmo divide la imagen en una cuadrícula cuyas celdas se encargan de realizar un número fijo de propuestas indicando en cada una de ellas la probabilidad que tiene de ser un objeto, la *bbox* en la que se encuentra y la probabilidad de pertenecer a cada una de las clases de objetos de las que disponemos. Finalmente, descarta aquellas propuestas con baja probabilidad de ser un objeto y utiliza un algoritmo de *non-max supression* [BSCD17] que unifica y combina las *bbox* con las máximas áreas compartidas.

Buscando una mejora de la predicción pero sin perder el enfoque de la velocidad, surge

YOLO9000 o *YOLOv2* [RF16] que sigue la misma estructura que su predecesor pero sustituyendo algunos fragmentos por otros con mayor rendimiento. YOLOv2 redimensiona la imagen original a 416x416 píxeles y sustituye las capas totalmente conectadas que utilizaba para la generación de propuestas por un modelo de *anchor boxes* [RHGS15] junto con otros cambios menores.

La siguiente versión *YOLOv3* [RF18] decide no darle tanta importancia a la velocidad y centrarse en solucionar los problemas de detección presentados por sus antecesores. Mientras que YOLOv2 utiliza una arquitectura con 30 capas, YOLOv3 utiliza 106 capas totalmente convolucionadas e introduce la utilización de *residual blocks*, *skip connections* y *upsampling*. Esta nueva arquitectura, realiza la detección de objetos en tres escalas distintas en diferentes profundidades de la red Darknet-53 y utilizando una estructura piramidal [LDG+16] para comunicar la detección entre las diferentes escalas.

Mencionar YOLOv4 https://arxiv.org/pdf/2004.10934.pdf

7. Non-local neural networks

En [WGGH17] se introduce el concepto de *non-local neural networks* que utiliza una arquitectura ascendente y descendente, *up-down architecture* [PLCD16], para extraer las características de un clasificador a distintos niveles de profundidad y utilizarlas para segmentar semánticamente la imagen Figura 5.1.

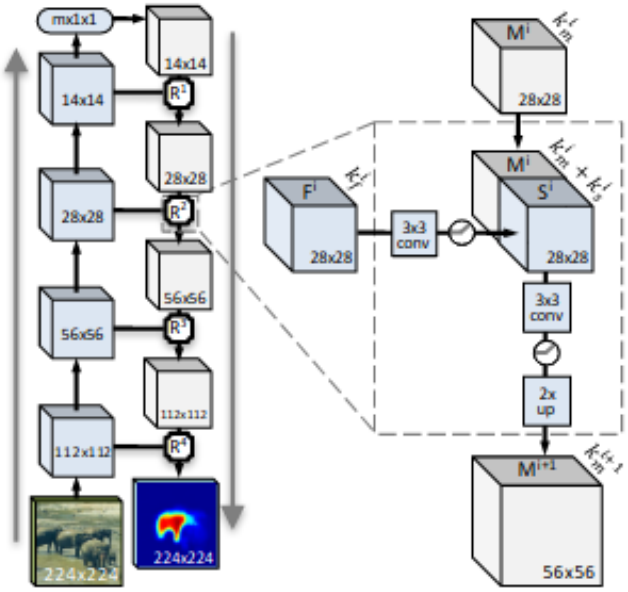

Figura 7.1.: Ejemplo de una arquitectura ascendente y descentente con saltos de conexiones[PLCD16].

En la parte ascendente de este tipo de arquitecturas nos encontraremos con un clasificador de imágenes, del cual extraeremos información a distintos niveles de profundidad que será unificada y analizada en la parte descendente de la arquitectura para poder predecir los distintos segmentos semánticos.

Las *non-local neural networks* se distinguen por la utilización de *non-local block*, o bloques no locales, como extractores de características en los distintos niveles de profundidad. Estos bloques serán de la forma:

$$z_i = W f_i(x_i) + x_i,$$

donde i representa el índice de una posición de salida, x_i el i-ésimo elemento de la entrada x, f_i la i-ésima coordenada de una operación no local f (Capítulo 4) y W un peso que será entrenado por la red.

8. Diseño

Para el desarrollo de la red neuronal que se ha implementado, se ha utilizado el artículo *Real-time Semantic Segmentation with Fast Attention* [HPC+20] publicado el 9 de julio de 2020. En este artículo se busca implementar una red neuronal capaz de realizar segmentación semántica en tiempo real, sin una gran pérdida de precisión y pudiendo ser ejecutada en dispositivos con pocos recursos.

Siguiendo sus ideas, se ha utilizado como clasificador un ResNet-18 [HZRS15] para la región ascendente de las *non local neural network* a desarrollar. Como bloque no local Figura 8.1 se ha utilizado el, que los autores denominan, *fast attention module*, que implementa como operación no local Capítulo 4 una *dot-product similarity* con $v(x_j) = W_{v_j} x_j$ que tendrá la función de extraer y analizar las características del clasificador.

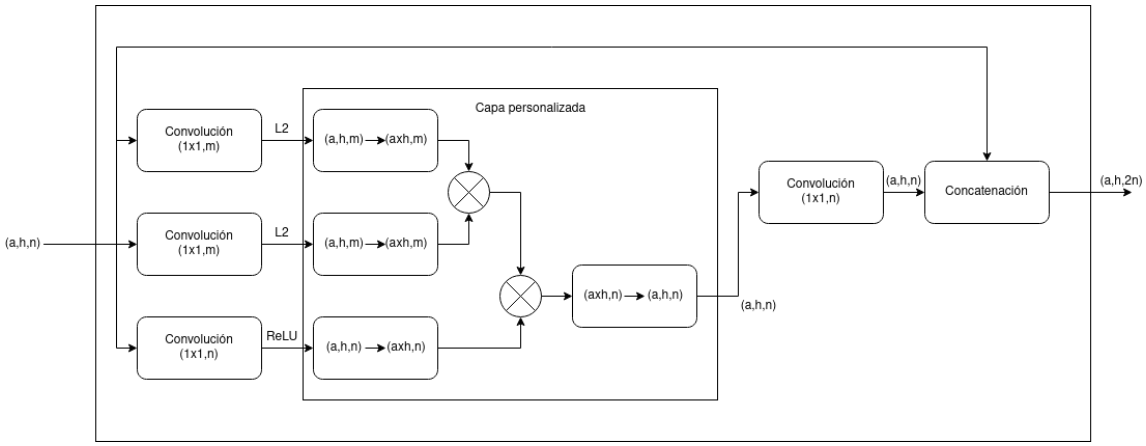

Figura 8.1.: Módulo *fast attention*. Diseño implementado del bloque no local Capítulo 7 que utiliza la operación no local Capítulo 4 siguiendo [HPC+20].

Este módulo permite la fijación del parámetro m, correspondiente al número de canales que tendrán las convoluciones que poseen regulación L2 y que serán mantenidas por el resto de operaciones hasta desaparecer esta cantidad por el producto matricial implementado. Por otro lado, el parámetro n corresponderá a los canales de los datos de entrada.

A nivel comparativo, como extractor de características, se utilizará también una convolución Capítulo 3 de núcleo de dimensión 2. El objetivo será comprobar empíricamente que el número de variables utilizadas no es lo que determinada la efectividad de un modelo, sino el cómo estas son utilizadas. Por ello, se ha decidido utilizar esta capa que es comúnmente utilizada para los problemas que estamos mencionando y que utiliza, únicamente, 4 variables por cada 3 utilizadas por el módulo *fast attention* Figura 8.1 implementado.

Para reconstruir las formas a través de las características extraídas nos hemos basado en el modelo U-Net [RFB15], en particular en la implementación realizada por el tutorial de segmentación de imágenes de Tensorflow [Ten] Figura 8.2.

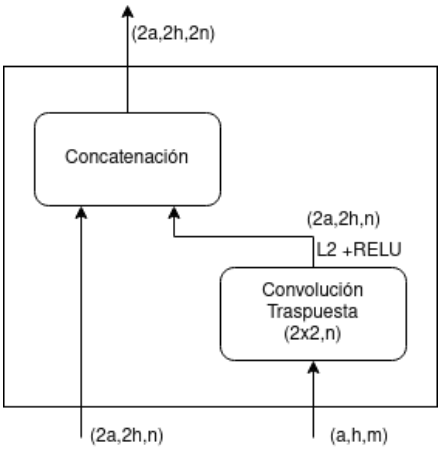

Figura 8.2.: Diseño implementado del bloque de fusión de capas utilizado para la reconstrucción en la sección descendente de la arquitectura.

Así, como estructura base tendremos:

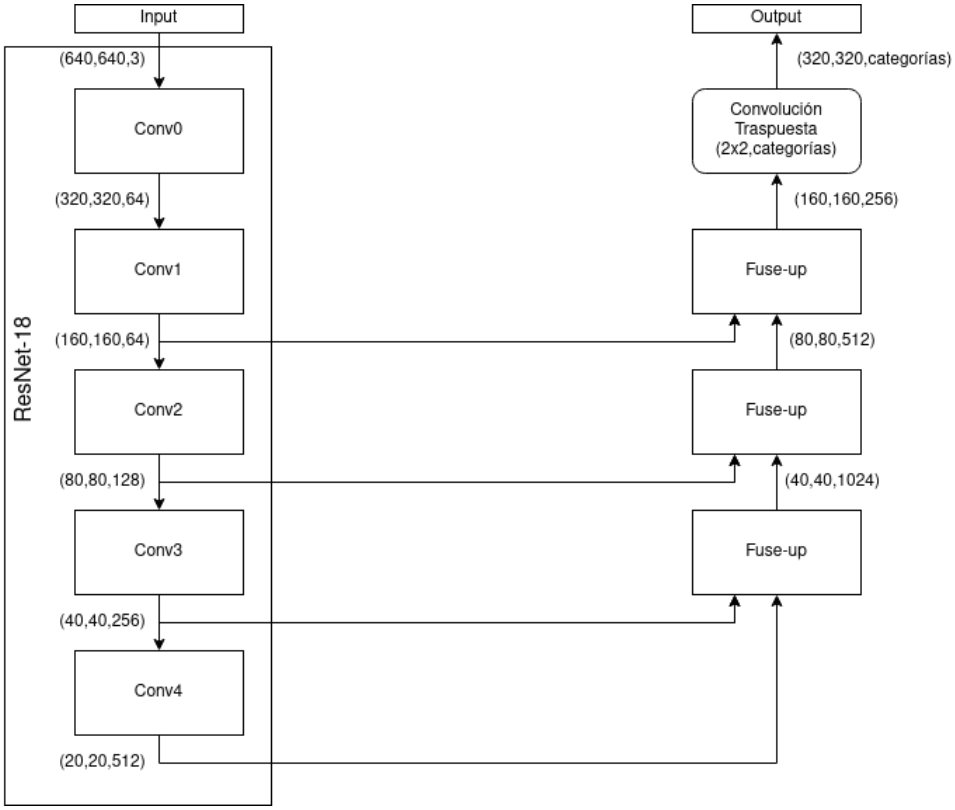

Figura 8.3.: Diseño base basado en U-net [RFB15] que utiliza como *backbone* una ResNet-18 [HZRS15].

Que al añadir el bloque no local denominado como módulo *fast attention* queda como:

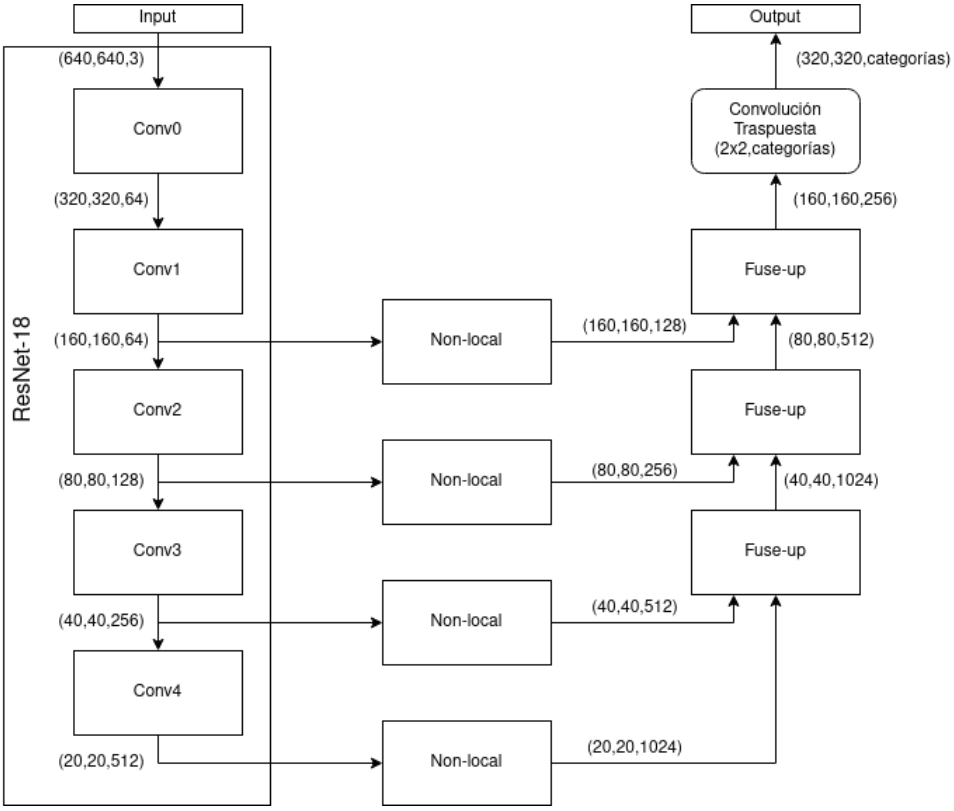

Figura 8.4.: Diseño base basado en U-net [RFB15] que utiliza como *backbone* una ResNet-18 [HZRS15]. Utiliza el módulo Figura 8.1 como extractor de características.

Sin embargo, en la siguiente sección veremos que al aumentar el número de categorías los resultados decaen. Por ello, se añade una nueva convolución al final de la reconstrucción y se modifica la convolución traspuesta original, con el fin de analizar las características obtenidas en los diferentes niveles de profundidad, quedando la estructura base como:

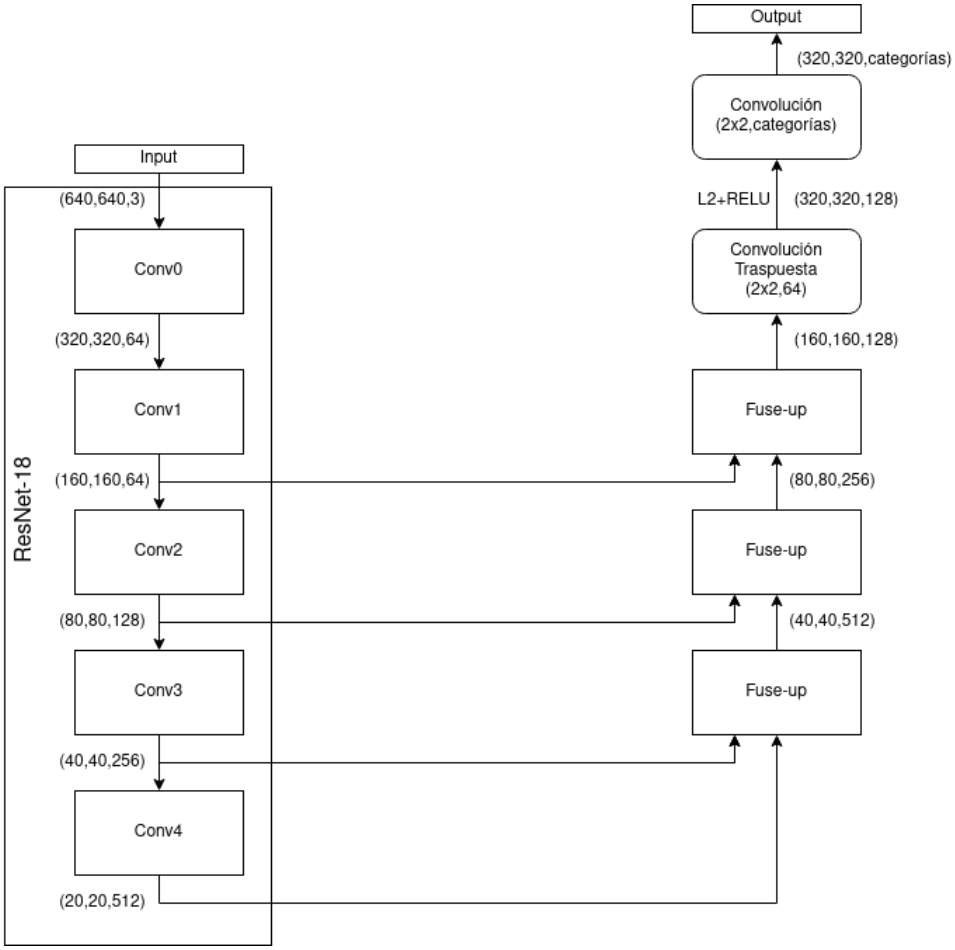

Figura 8.5.: Diseño base basado en U-net [RFB15] que utiliza como *backbone* una ResNet-18 [HZRS15]. Utiliza una convolución extra al final para analizar la información obtenida y distinguir a través de esta.

Por lo que añadiendo el bloque no local tendremos:

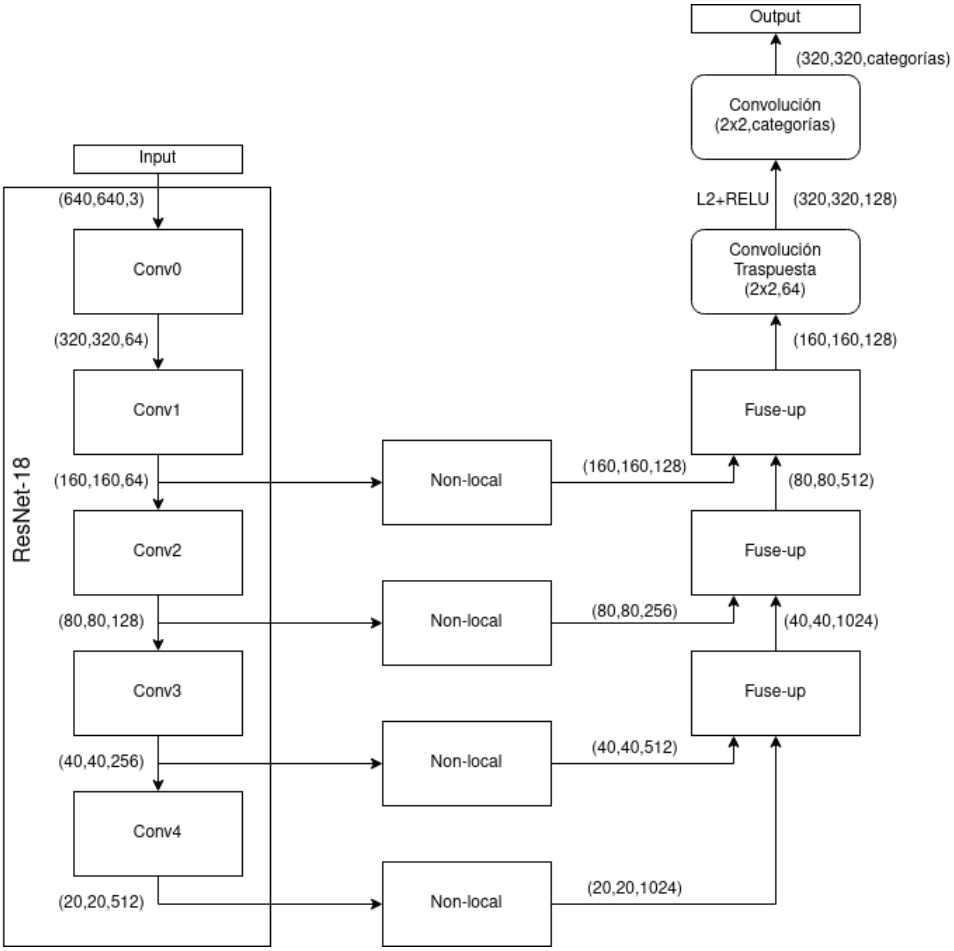

Figura 8.6.: Diseño base basado en U-net [RFB15] que utiliza como *backbone* una ResNet-18 [HZRS15]. Utiliza una convolución extra al final para analizar la información obtenida y distinguir a través de esta. Utiliza el módulo Figura 8.1 como extractor de características.

Y al sustituir el bloque no local como extractor de características por la convolución de núcleo de tamaño 2 se obtendrá:

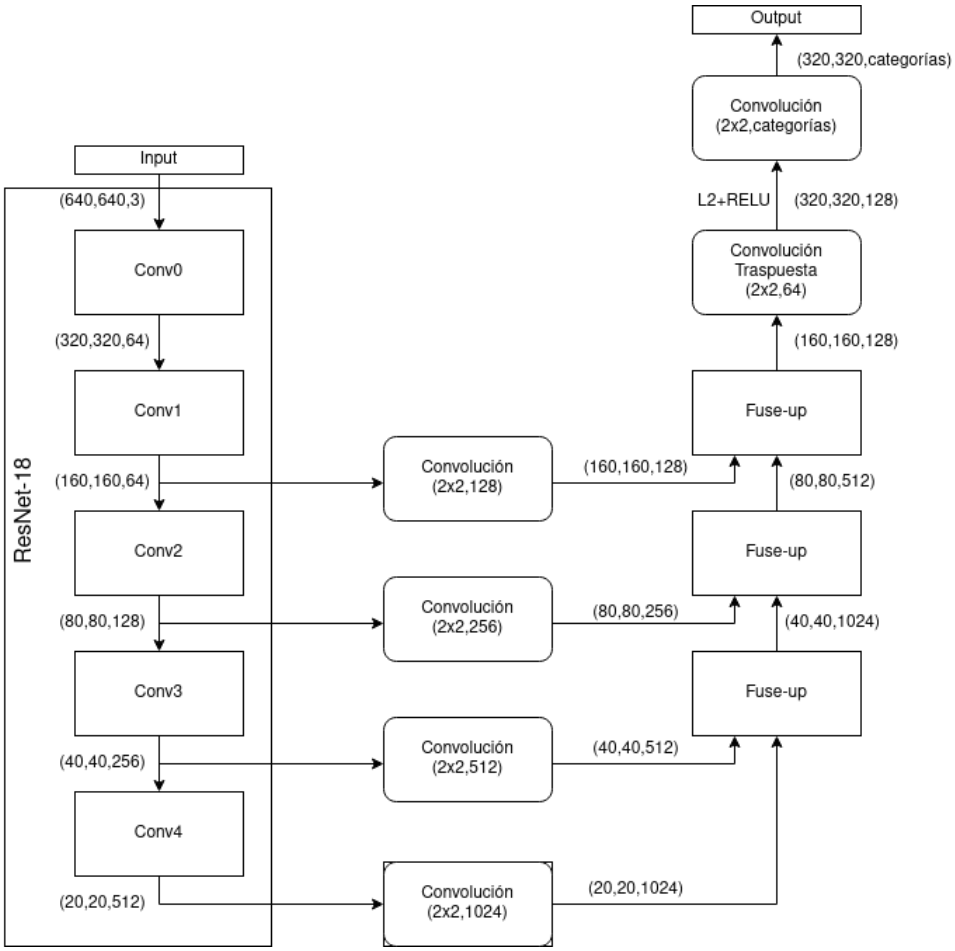

Figura 8.7.: Diseño base basado en U-net [RFB15] que utiliza como *backbone* una ResNet-18 [HZRS15]. Utiliza una convolución extra al final para analizar la información obtenida y distinguir a través de esta. Utiliza convoluciones de núcleo de tamaño 2 como extractor de características.

Al añadir otro bloque de *fuse up* surgió la posibilidad de que, añadiendo otro bloque no local, se consiguiera mejorar los resultados. Como se verá en la siguiente sección, no fue así.

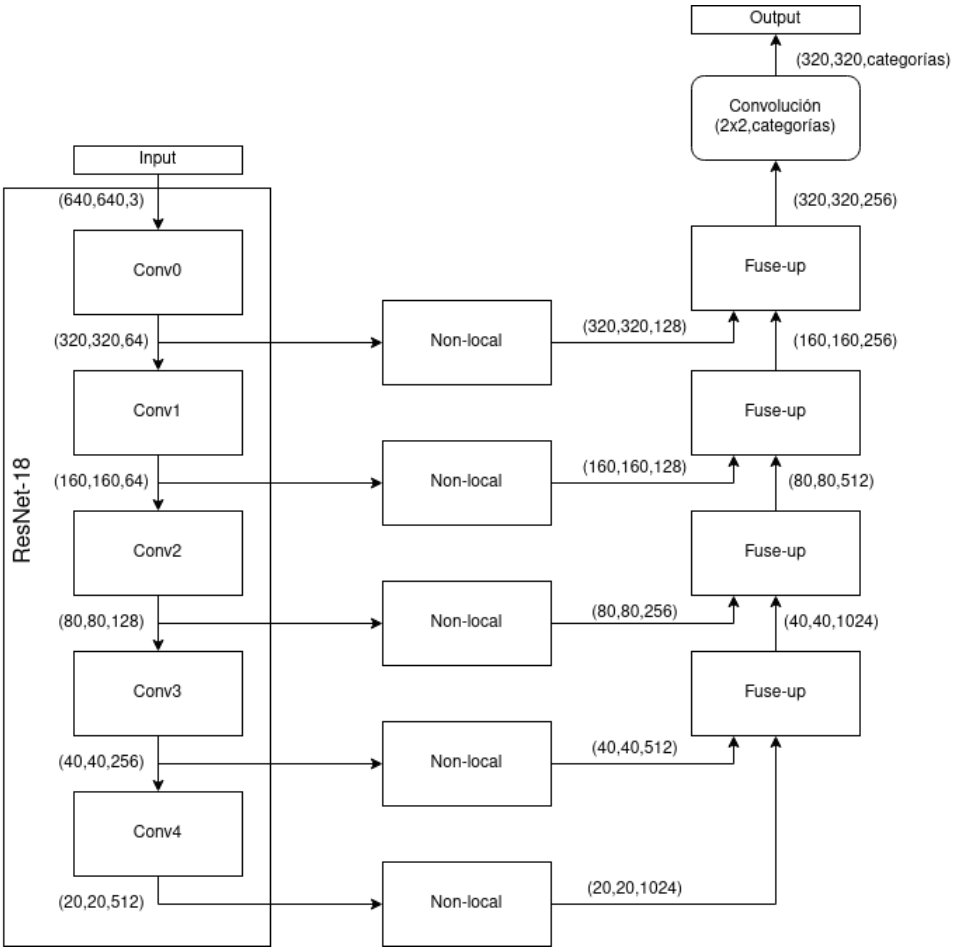

Figura 8.8.: Diseño base basado en U-net [RFB15] que utiliza como *backbone* una ResNet-18 [HZRS15]. Utiliza una convolución extra al final para analizar la información obtenida y distinguir a través de esta. Utiliza el módulo Figura 8.1 como extractor de características y posee un módulo extra.

Ante la posibilidad anterior, y su resultado, se diseñó otro planteamiento que veía reducido su número de bloques no locales, sin llegar a eliminarlos todos, para poder observar cómo esta reducción afectaba a la segmentación y si era necesario, o no, la utilización de todos los bloques.

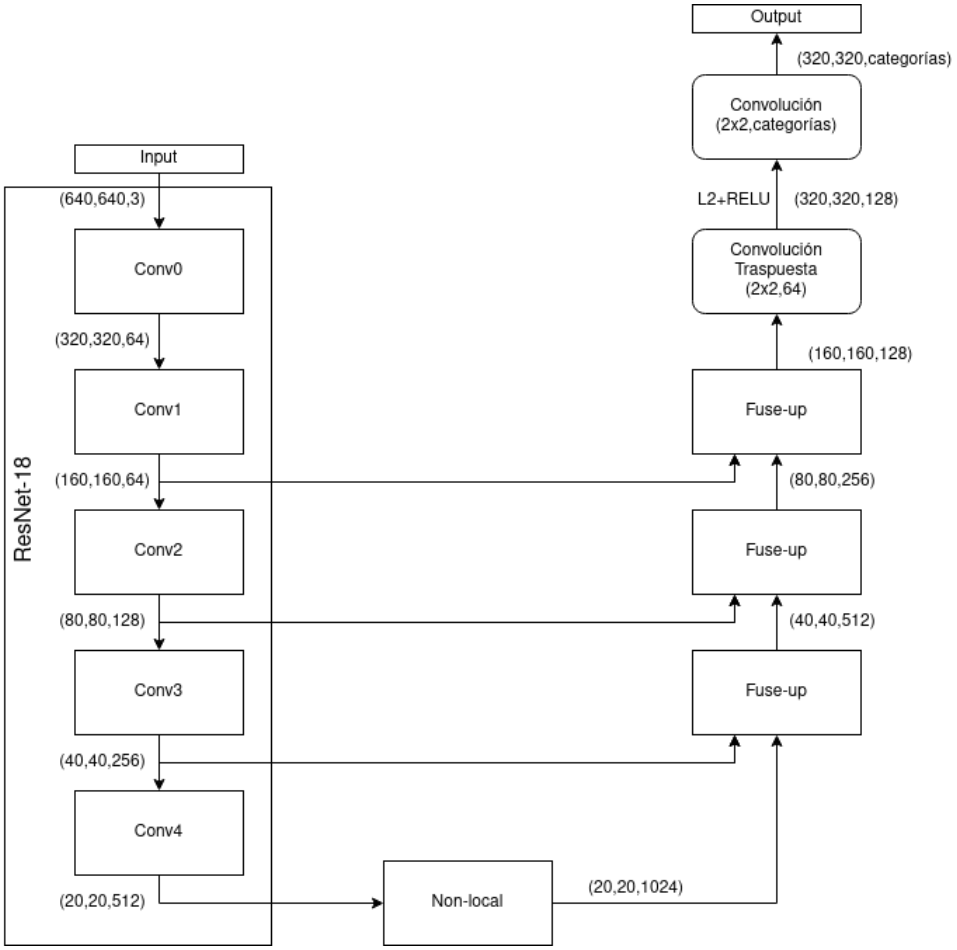

Figura 8.9.: Diseño base basado en U-net [RFB15] que utiliza como *backbone* una ResNet-18 [HZRS15]. Utiliza una convolución extra al final para analizar la información obtenida y distinguir a través de esta. Utiliza el módulo Figura 8.1 como extractor de características en la máxima profundidad.

Y comprobaremos también esta posibilidad utilizando la convolución de núcleo de tamaño 2:

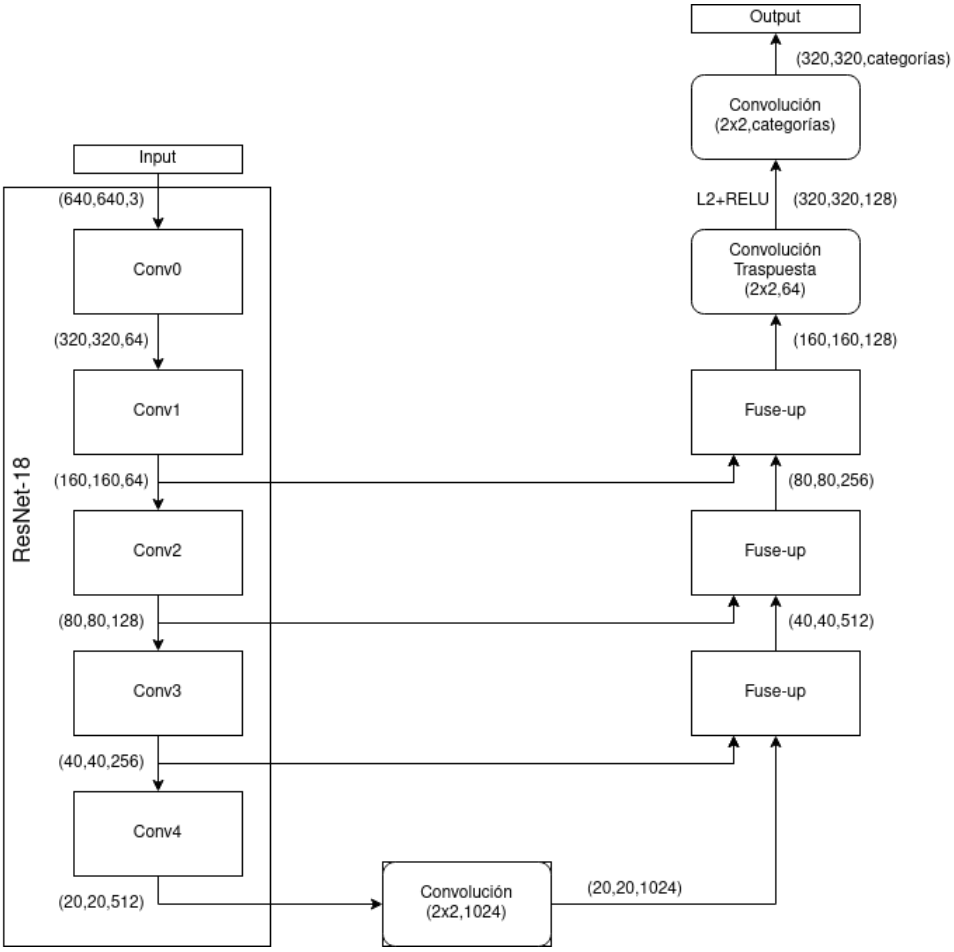

Figura 8.10.: Diseño base basado en U-net [RFB15] que utiliza como *backbone* una ResNet-18 [HZRS15]. Utiliza una convolución extra al final para analizar la información obtenida y distinguir a través de esta. Utiliza convoluciones de núcleo de tamaño 2 como extractor de características en la máxima profundidad.

9. Experimentos

9.1. Parámetros y configuración de entrenamiento

Como función de pérdida se ha utilizado la *Sparse Categorial Crossentropy* ofrecida por Keras [C+15]. Esta función aplica una función *SoftMax* a la salida de la red antes de calcular su entropía cruzada. Se diferencia de la *Categorical Crossentropy* por el formato de entrada de las etiquetas verdaderas, al requerir una matriz bidimensional en lugar de una tridimensional con el fin de reducir el cómputo.

El optimizador empleado será emphAdam, con los parámetros por defecto, e iremos reduciendo el ratio de aprendizaje un factor de 0.1 cada 3 épocas sin reducir la pérdida en el conjunto de validación hasta un valor mínimo de 10^{-10}. Además, si ocurren 10 épocas y no mejora el valor de pérdida en el conjunto de validación, el entrenamiento se detendrá al interpretarse que ha llegado a su límite y no será capaz de aprender mucho más. Cada vez que obtengamos un mejor valor de pérdida en el conjunto de validación, se guardarán los pesos que posea la red neuronal en dicho momento.

Como métricas, se utilizará la clásica *accuracy*, o precisión, junto con la *Mean IoU*, o media de la intersección sobre la unión, siendo que será esta última la que realmente prediga cómo de bien se comporta la red. Esto es debido a que podríamos predecir que no existe ningún objeto en la imagen y obtener una precisión alta cuando realmente existen pequeños objetos que no han sido predichos. En [Tiu] se puede ver una explicación más detallada de este fenómeno y el por qué del uso de la intersección sobre la unión.

Como conjunto de datos de entrenamiento, se ha utilizado COCO 2017 [LMB+14] [COC] que posee un total de 91 categorías correspondientes a objetos del día a día. Estas categorías pueden ser agrupadas en hiper categorías, de forma que se realizarán entrenamientos con todo el conjunto de datos y también con aquellos pertenecientes a la hiper categorías de transportes. El objetivo es comparar el comportamiento en diferentes cantidades de categorías a aprender.

Para incrementar la variedad de imágenes sin aumentar la cantidad de estas, buscando obtener una mayor generalidad de la red sin aumentar el cómputo, se implementarán una serie de incrementos de datos, o *data augmentations*. Esto será la aplicación de un conjunto de transformaciones a las imágenes originales y a sus máscaras de etiquetas cada vez que estas sean consultadas para el entrenamiento. En particular, se aplicarán, en orden, entre una o dos transformaciones de las siguientes opciones [JWC+20]:

- Reflejo horizontal. En caso de salir esta opción, tendrá una probabilidad de 0.5 de ser aplicado.
- Escalado. La imagen será escalada de forma aleatoria, como mínimo será la mitad de

su escala original y como máximo se verá incrementada un 50 %.

- Traslación. La imagen será trasladada entre el −20 % y el +20 % de su tamaño, representando un valor negativo la traslación hacia la izquierda y un valor positivo la traslación hacia la derecha.

Se pueden llegar a aplicar muchas más operaciones para el aumento de datos, pero se ha decidido utilizar estas que son fácilmente interpretables y que, por el contexto actual de imágenes, no insertan información conflictiva tras la transformación. Un ejemplo de transformación a aplicar sería el reflejo vertical, que podría dar como resultado tener el cielo abajo y un bosque arriba de este. No es esperable recibir imágenes de este tipo, y por ello entrenar la red para tenerlas en cuenta podría emitir peores resultados.

9.2. Gráficas

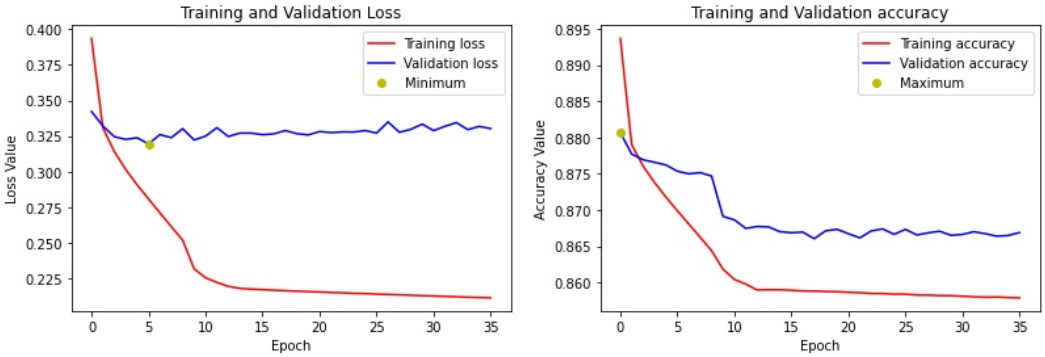

Figura 9.1.: Entrenamiento del modelo mostrado en Figura 8.3 sin aumento de datos y sin detención temprana. Subcategoría de Transportes.

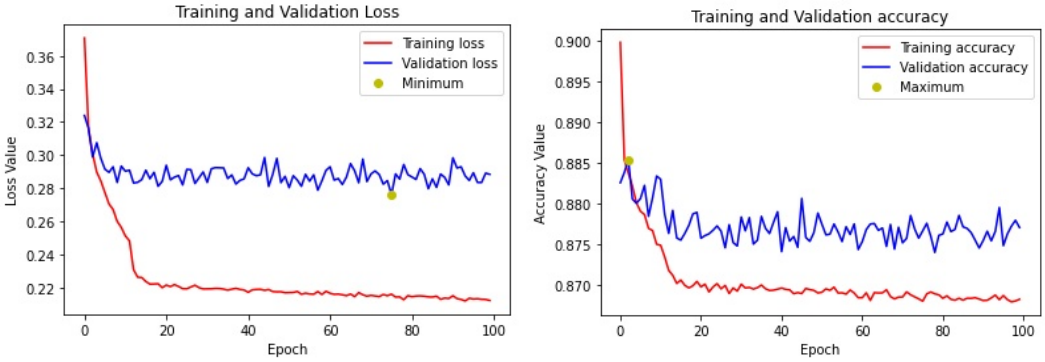

Figura 9.2.: Entrenamiento del modelo mostrado en Figura 8.3 con aumento de datos tanto en los datos de entrenamiento como en los de validación. Subcategoría de Transportes.

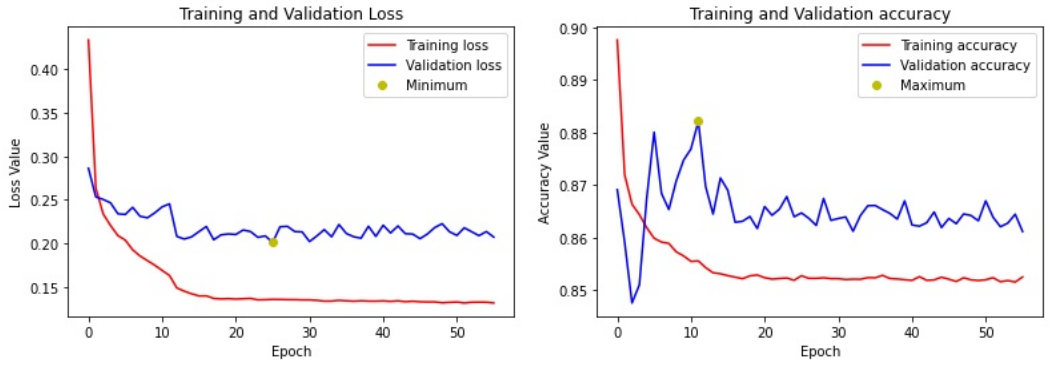

Figura 9.3.: Entrenamiento del modelo mostrado en Figura 8.4 con aumento de datos tanto en los datos de entrenamiento como en los de validación. 64 canales en el bloque de atención. Subcategoría de Transportes.

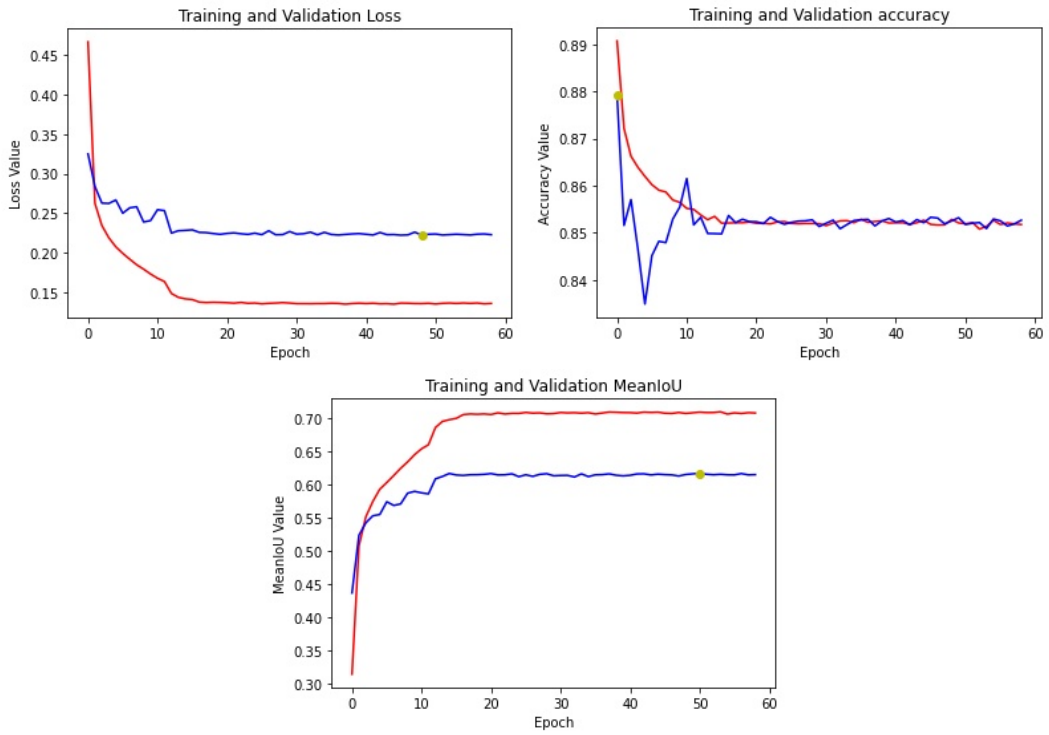

Figura 9.4.: Entrenamiento del modelo mostrado en Figura 8.4 con aumento de datos en los datos de entrenamiento. 64 canales en el bloque de atención. Subcategoría de Transportes.

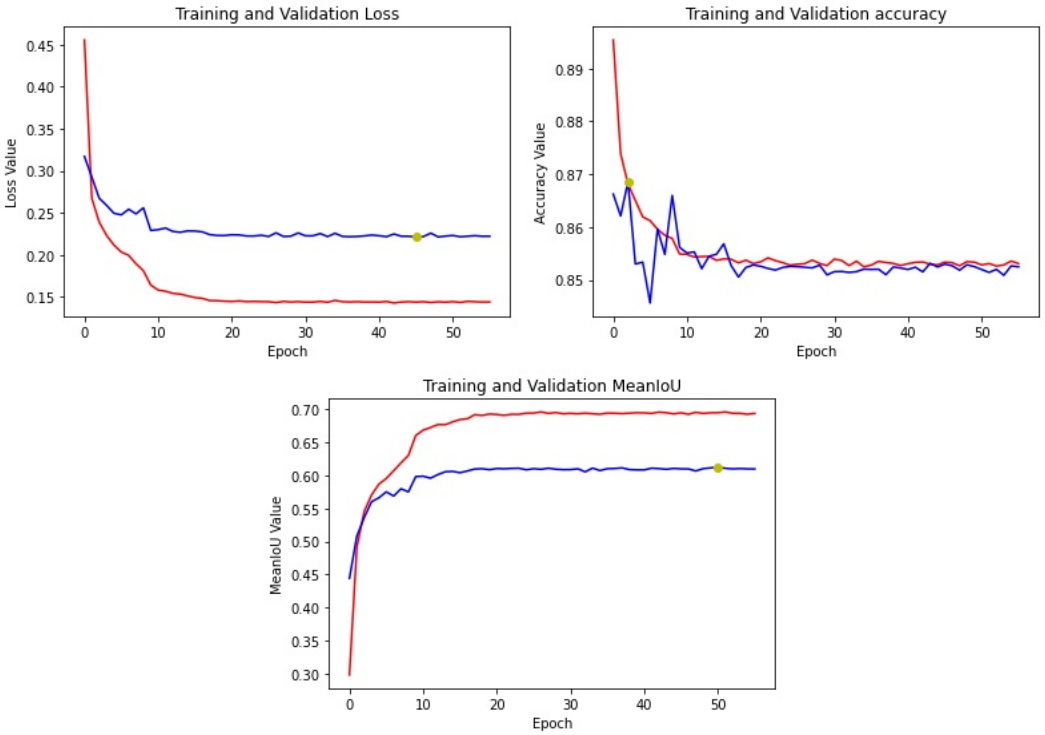

Figura 9.5.: Entrenamiento del modelo mostrado en Figura 8.4 con aumento de datos en los datos de entrenamiento. 128 canales en el bloque de atención. Subcategoría de Transportes.

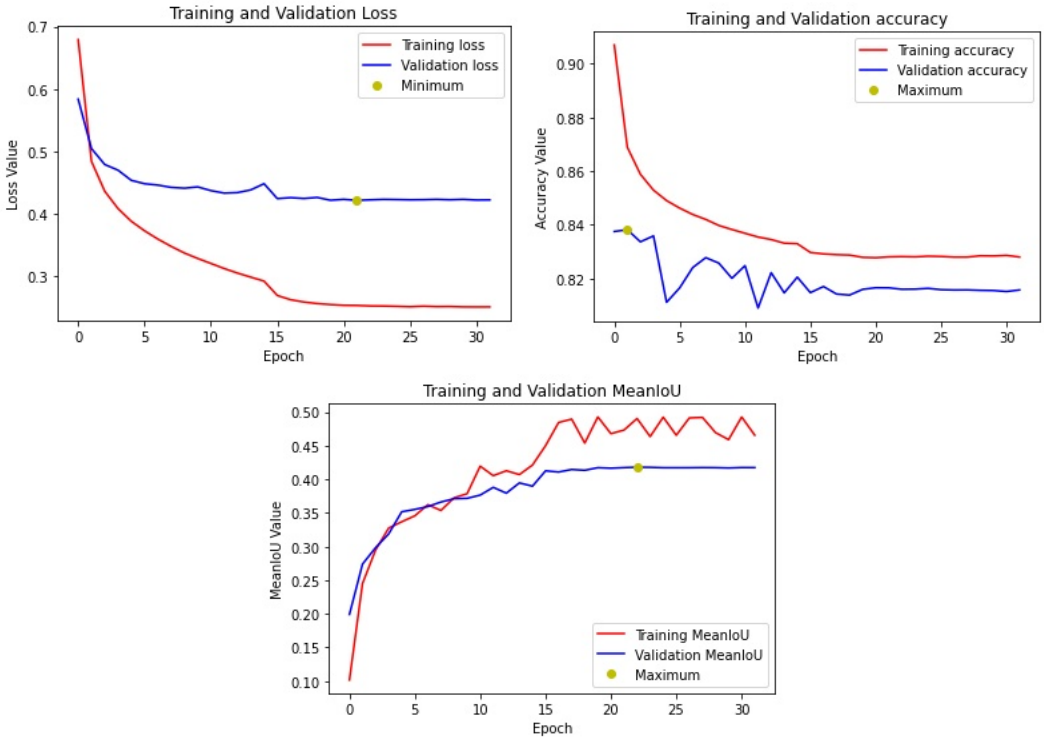

Figura 9.6.: Entrenamiento del modelo mostrado en Figura 8.4 con aumento de datos en los datos de entrenamiento. 64 canales en el bloque de atención. Conjunto de datos completo.

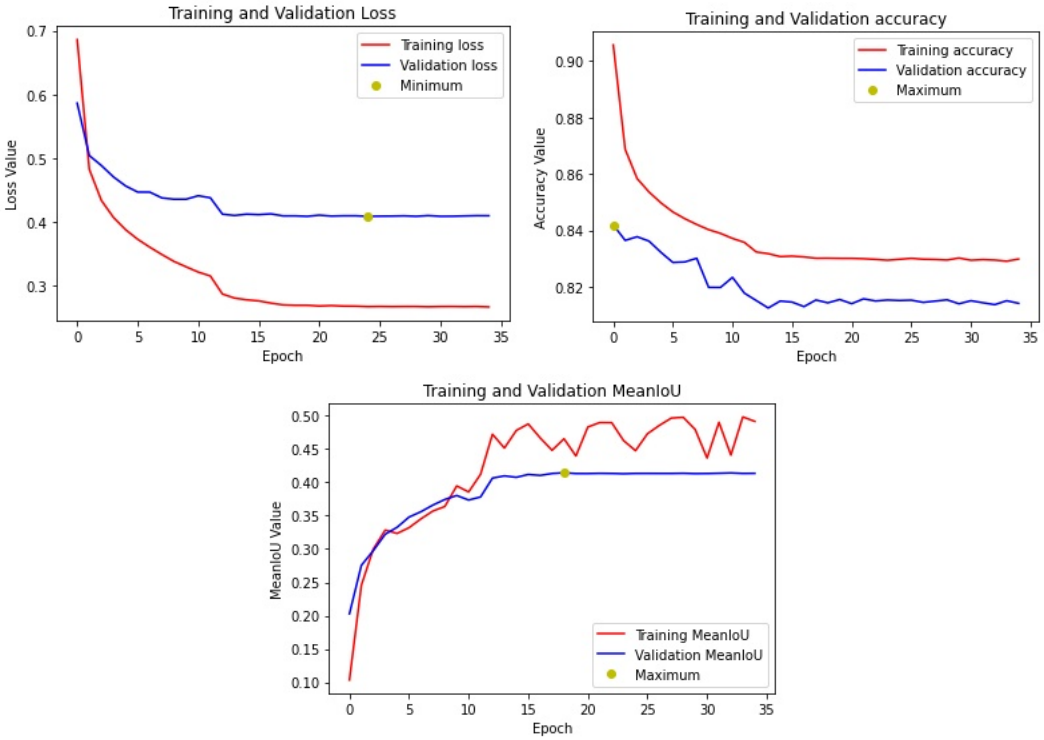

Figura 9.7.: Entrenamiento del modelo mostrado en Figura 8.4 con aumento de datos en los datos de entrenamiento. 128 canales en el bloque de atención. Conjunto de datos completo.

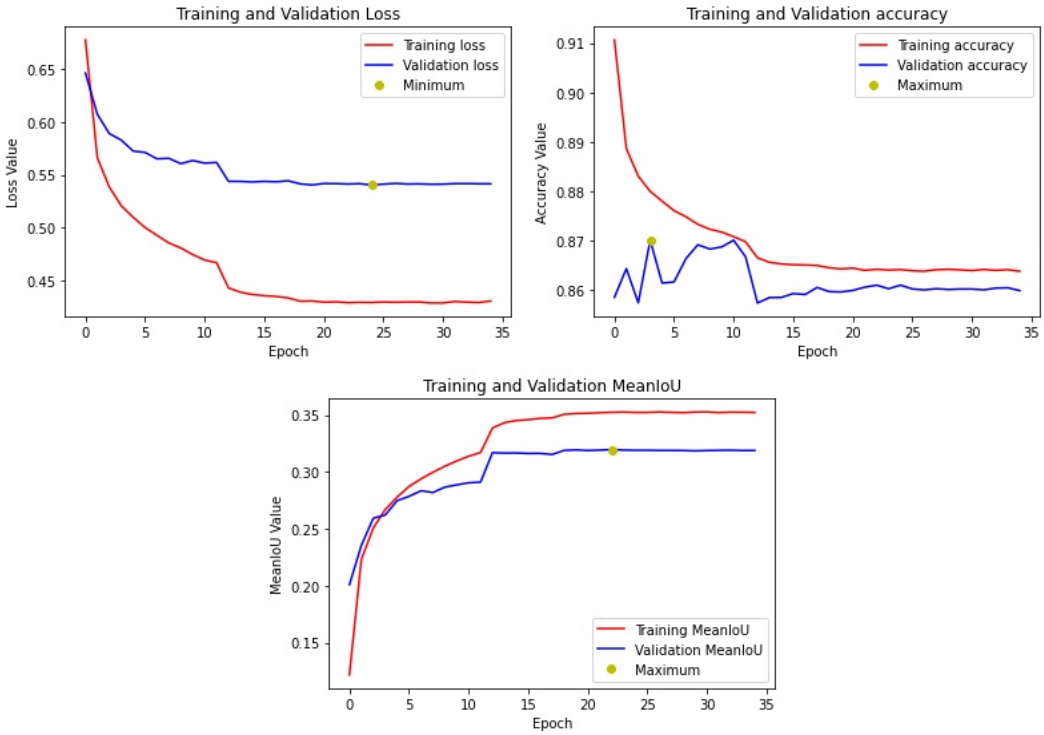

Figura 9.8.: Entrenamiento del modelo mostrado en Figura 8.5 con aumento de datos en los datos de entrenamiento. 64 canales en el bloque de atención. Conjunto de datos completo.

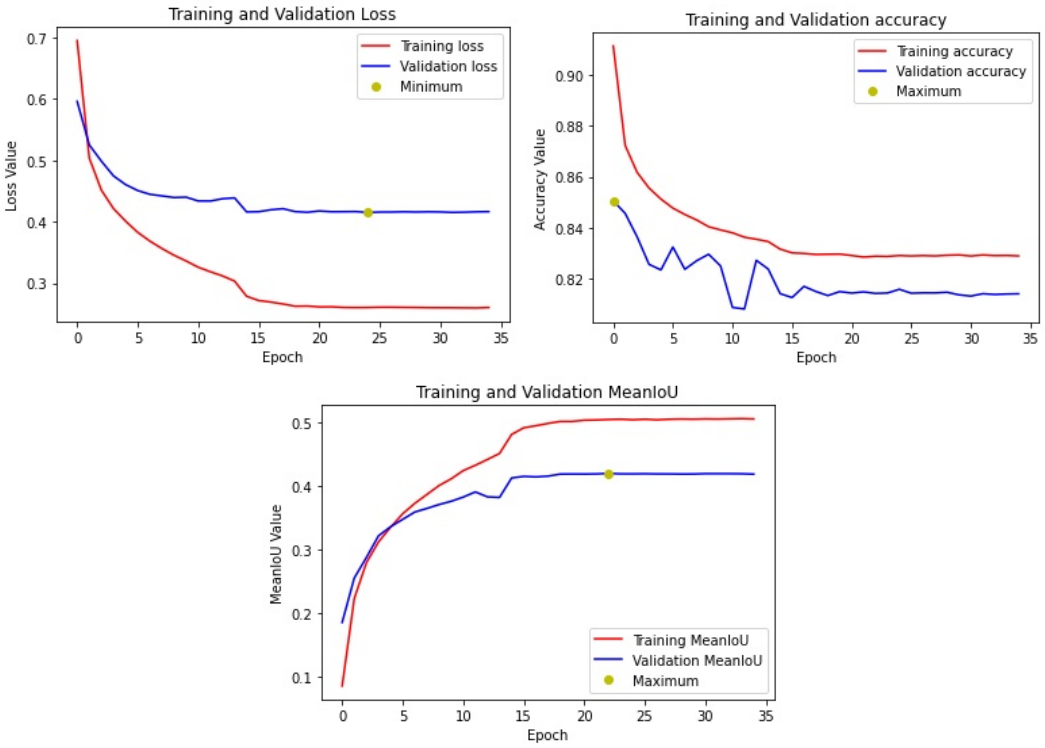

Figura 9.9.: Entrenamiento del modelo mostrado en Figura 8.6 con aumento de datos en los datos de entrenamiento. 64 canales en el bloque de atención. Conjunto de datos completo.

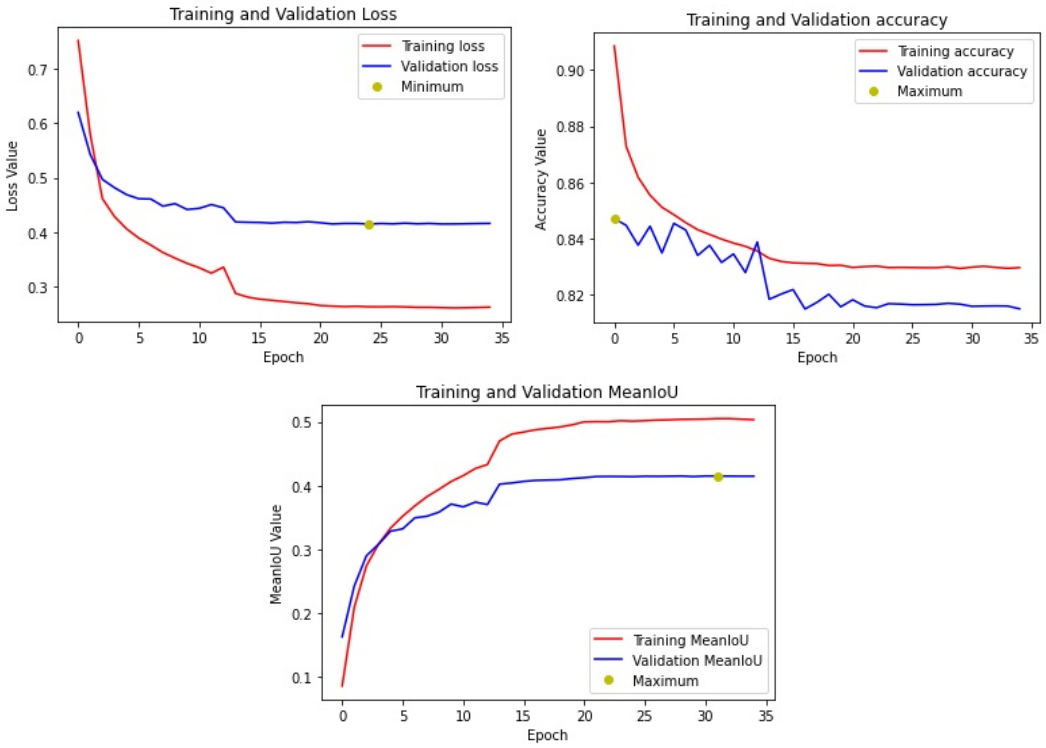

Figura 9.10.: Entrenamiento del modelo mostrado en Figura 8.8 con aumento de datos en los datos de entrenamiento. 64 canales en el bloque de atención. Conjunto de datos completo.

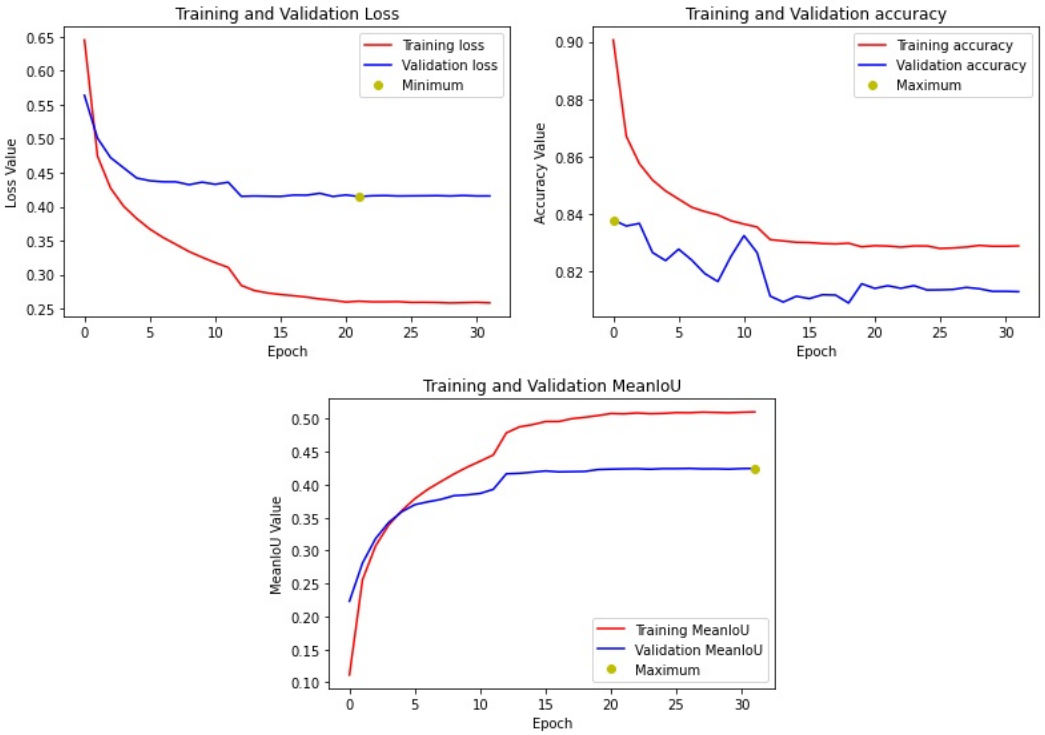

Figura 9.11.: Entrenamiento del modelo mostrado en Figura 8.9 con aumento de datos en los datos de entrenamiento. 64 canales en el bloque de atención. Conjunto de datos completo.

9.3. Análisis y conclusiones

Debido a la cantidad de gráficas, se irá seleccionando diferentes combinaciones a comparar, con el fin de resaltar de forma específica qué se quería analizar en cada momento.

9.3.1. Diferentes cantidades de bloques no locales.

Lo primero a comparar será los resultados obtenidos dependiendo de la cantidad de módulos no locales empleados en la red. En particular, se observarán las gráficas Figura 9.11, Figura 9.10 y Figura 9.9 que únicamente se diferencian en dicha cantidad.

Las gráficas correspondientes tienen una estructura similar, siendo que en las 3 se alcanza el mínimo valor de pérdida en el conjunto de validación entre las épocas 20 y 25. Además, en los tres modelos existe un aumento sustancial del valor medio de la intersección sobre la unión en el conjunto de validación entre las épocas 10 y 15, siendo que este coincide con la primera reducción del ratio de aprendizaje por llevar 3 épocas sin que viera reducido la pérdida en la validación.

A pesar de seguir realizándose reducciones del ratio de aprendizaje ante la falta de mejoría, no vuelven a producirse mejoras sustanciales durante el aprendizaje en ninguna de ellas.

Teniendo en cuenta esto, se comprueba empíricamente que no por añadir más cantidad de bloques no locales se podrán extraer más características de la imagen. Pero, además, debemos de tener en cuenta la gráfica Figura 9.8, correspondiente a la misma arquitectura pero sin bloques no locales, por lo que estos bloques sí establecen una mejoría ante la inexistencia de ningún otro.

9.3.2. Bloque no locales y convoluciones

9.3.3. Diferentes números de canales en los bloques de atención

Para esta comparativa, se podrán utilizar las gráficas Figura 9.4, Figura 9.5, Figura 9.6 y Figura 9.7. Siendo que la primera y tercera figura corresponderán a 64 canales mientras que las otras corresponderán a 128, todos ellos en el modelo Figura 8.6.

Aquí se puede apreciar que el incremento del número de canales, para este bloque y en estas circunstancias, permite acelerar la velocidad de aprendizaje pero no evita que el modelo alcance el mismo límite de reducción de pérdida y de aumento de la *mean IoU*.

9.3.4. Convolución extra tras la obtención de la dimensión deseada

A continuación, prestaremos atención a Figura 9.6 y a Figura 9.9, en la que destaca la reducción de oscilaciones al añadir la nueva convolución. A pesar de ello, la modificación no cumple con el objetivo de aumentar el valor medio de la intersección sobre la unión ni de reducir el valor de la pérdida.

Esta fue mantenida con el fin de visualizar mejor las modificaciones ofrecidas en las diferentes comparaciones.

9.3.5. Conclusiones

Pese a no obtener los mejores resultados, no pudiendo competir con los métodos presentes en el estado de arte actual para la segmentación semántica o la instanciación de objetos, se ha podido comprobar de forma empírica, en un ejemplo concreto, la utilidad de los bloques no locales para la extracción de características en imágenes.

Como se sigue comprobando día a día en las diferentes arquitecturas y modelos presentados, no se trata de añadir capas y capas que ofrezcan una mayor cantidad de variables para resolver los problemas planteados. En su lugar, es más adecuado el razonar qué capas o conjunto de capas será más útil en cada momento, buscando el conseguir los mejores resultados ocupando la menor cantidad posible de recursos.

En este caso, los bloques no locales han resultado ser una buena alternativa a la utilización de simples convoluciones como extractor de características, puesto que se obtenían mejores resultados con la utilización de una menor cantidad de memoria. También se ha podido comprobar que la adicción de estos bloques en exceso no supondrá ninguna mejora.

Se desea realizar la conjetura de que esta falta de mejora puede ser debida a que no se está extrayendo nuevas características en la arquitectura actual con el incremento de bloques, sino que se está recibiendo información con la relativa similitud que carece de la suficiente distinción como para aprender algo nuevo.

Por ello, y teniendo en cuenta Capítulo 4, se concluye que los bloques que implementan operaciones no locales pueden llegar a ser de utilidad, con un correcto uso, para aquellas redes neuronales en las que la correlación a lo largo del dato entrada es importante para realizar una predicción.

Ejemplos de esto en distintos ámbitos, se puede encontrar a través de la tendencia creada por el artículo *Attention is all you need* [VSP+17] que, bajo la utilización de otra nomenclatura, utiliza y recomienda el uso de operaciones no locales para la traducción de texto.

Parte III.

Desarrollo de un sistema CBIR

Descripción :D FIXME

10. Recuperación de imágenes basadas en su contenido (CBIR)

En el contexto de las ciencias de información, en el Capítulo 1 de [MRS08] podemos encontrar una definición genérica para el contexto de las ciencias de computación del término recuperación de información"que dice que:

Definición 10.1. La recuperación de información, o *information retrieval (IR)*, consiste en encontrar material de una naturaleza no estructurada que satisface una necesidad de información dentro de grandes colecciones.

En particular, nos encontramos trabajando en la recuperación de imágenes que pertenece a la categoría de recuperación de información multimedia. Este, a su vez, se divide en dos tipos de recuperación, las basadas en texto, *text-based image retrieval (TBIR)*, o las basadas en su contenido, *content-based image retrieval (CBIR)*.

En [Tya17] se dice, en que los sistemas TBIR, los usuarios utilizan palabras clave o descripciones de las imágenes como consulta en una base de datos para recuperar las imágenes que sean relevantes a la palabra clave. Para ello, primeramente las imágenes deben de ser anotados con texto ya sea de forma manual o automática. Esto posee la desventaja de que los algoritmos de anotación automática de imágenes no son factibles para generar textos descriptivos para un amplio espectro de imágenes requiriendo la anotación manual que, a su vez, suele ser subjetiva y dependiente del contexto. Como ventaja, al realizar utilizar únicamente texto para describir la imagen, sus consultas son rápidas ya que la coincidencia de cadenas es un proceso que requiere menos tiempo de cómputo.

Por otro lado, la recuperación de imágenes basada en el contenido (CBIR), también conocida como consulta por el contenido de una imagen o *query by image content (QBIC)*, es una técnica automatizada que toma una imagen como consulta y devuelve un conjunto de imágenes similares a esta [Tya17]. La imagen consultada es convertida en la representación interna de un vector de características usando la misma rutina de extracción que fue utilizada para crear la base de datos. Se utiliza una medida de similitud para calcular las distancias entre los vectores de características de la imagen consultada y de las imágenes destino en la base de datos de características. Por último, la recuperación se realiza mediante un esquema de indexación que facilita la búsqueda eficiente de la base de datos de imágenes Figura 10.1.

En las páginas 7-8 y 22-26 de [JE99] se mencionan tres niveles de consultas en un CBIR siendo:

- Nivel 1: La recuperación de imágenes a través de características primitivas como pueden ser diferentes medidas de representación matemática de colores, texturas o formas. Un ejemplo sería un histograma de color que muestra la proporción de píxeles de cada color dentro de la imagen y la consulta sería encontrar imágenes con valores similares.

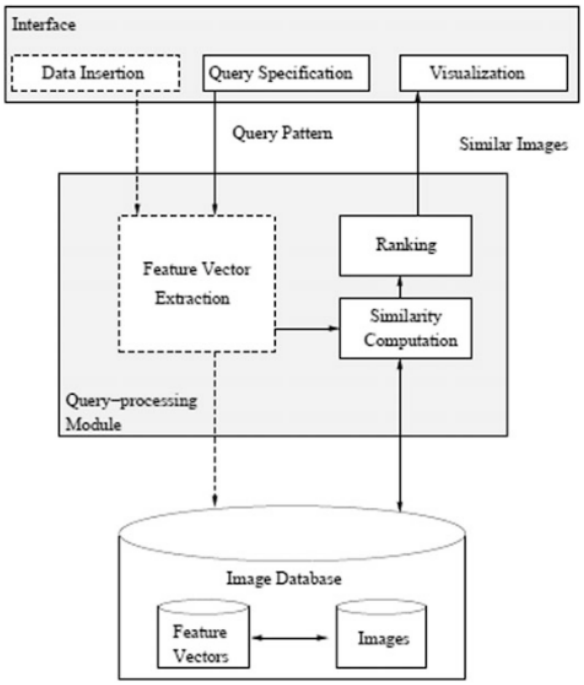

Figura 10.1.: Arquitectura de un sistema CBIR típico. [Tya17]

- Nivel 2: Comprende la recuperación por características derivadas, también conocidas como lógicas, incluyendo un cierto grado de inferencia lógica sobre la identidad de los objetos representados en la imagen. Además puede ser dividido en:
  - recuperación de objetos de un determinado tipo o categoría. Por ejemplo, "encuentra imágenes de un autobús de dos pisos".
  - recuperación de objetos individuales o personas. Por ejemplo, "encuentra una imagen de la torre Eiffle".
- Nivel 3: Se trata de la recuperación por atributos abstractos, incluyendo una cantidad significativa de razonamiento de alto nivel sobre el propósito de los objetos o escenas representadas. Por ejemplo, "encuentra imágenes de una multitud alegre".

El objetivo de este proyecto es desarrollar una aplicación en java que represente un sistema CBIR que siga una arquitectura como la mostrada en Figura 10.1. Este será capaz de extraer distintas características de una imagen y de comparar, o calcular la similitud, entre las imágenes de distintas formas para obtener distintas clasificaciones ordenadas que serán visualizadas en la aplicación.

Se podrán realizar consultas de nivel 1 a través de características de color y de nivel 2 utilizando segmentación semántica para la distinción de categorías con una red neuronal implementada en Python.

11. Planificación y presupuesto.

12. Requisitos

Antes de desarrollar la aplicación se debe comenzar con la exposición de los requisitos de datos, funcionales y no funcionales que esta requerirá para así poder tener en mente qué es lo que necesitamos, de qué es lo disponemos y cómo podremos trabajar de forma óptima para llegar al resultado deseado.

Teniendo en mente Figura 10.1, sabemos que debemos de tener los siguientes requisitos de datos:

1. Representación de una imagen
2. Vector de características, en adelante descriptor, por cada característica que se quiera extraer o analizar.
3. Base de datos capaz de almacenar los descriptores correspondientes a cada una de las imágenes que estarán almacenadas en dicha base o fácilmente accesibles a través de la información almacenada en ella.
4. Representación de un descriptor genérico almacenado en la base de datos.
5. Concepto de clasificador utilizado para extraer las características.
6. Concepto de comparador para calcular la similitud entre las distintas características.

Seguidamente, respecto a los requisitos funcionales debemos de considerar:

1. Abrir, guardar y cerrar una imagen.
2. Realización de consultas basadas en el contenido de una imagen.
3. Realización de consultas basadas en texto, siendo estas las categorías a las cuales pueden pertenecer nuestras imágenes.
4. Agrupación de categorías.
5. Extracción de descriptores de nivel 1.
6. Extracción de un descriptor de nivel 2.
7. Almacenaje de un conjunto de imágenes para poder ser comparadas.
8. Crear, modificar, guardar y abrir bases de datos.
9. Almacenaje de los descriptores correspondientes a dichas imágenes en una base de datos.
10. Enlace entre las imágenes y sus correspondientes descriptores para un fácil acceso.

11. Comparación entre descriptores.
12. Comparación utilizando múltiples descriptores de las propias imágenes.
13. Ordenación de imágenes resultantes de la consulta utilizando el resultado de la comparación entre descriptores.
14. Capacidad de visualizar las imágenes resultado de forma ordenada según la puntuación obtenida.
15. Capacidad de cambiar los comparadores utilizados sin que se vean afectados los descriptores utilizados.
16. Capacidad de crear nuevas bases de datos utilizando diferentes descriptores.
17. Capacidad de crear nuevas bases de datos utilizando múltiples descriptores.
18. Importar un clasificador a través de un fichero.

Respecto a los requisitos no funcionales nos referimos a las características de funcionamiento que nos servirán para mantener la calidad de nuestro programa. Estos serán, principalmente: rendimiento, durabilidad, estabilidad, seguridad, eficiencia, compatibilidad, garantía e integración de datos.

Finalmente, se debe comentar que todo el proyecto deberá de poder ejecutarse en ordenadores que no dispongan de muchos recursos en unos tiempos razonables, al ser el material del que se dispone y siendo esta la prueba del funcionamiento en uno de los peores casos posibles.

13. Análisis

Comenzaremos analizando los requisitos mencionados en el capítulo anterior para poder enfrentarnos correctamente al diseño de nuestro sistema.

Para los requisitos de datos se considera el encapsular apropiadamente cada uno de estos en una clase propia que sirva como una correcta representación de cada uno de los conceptos. En particular, si consideramos las especificaciones planteadas en los requisitos funcionales, se hace evidente la necesidad de la utilización de herencia para un correcto diseño del software al haber conceptos genéricos que más adelante se dividen teniendo diferentes especificaciones.

Además, es necesario desarrollar una aplicación de escritorio que haga mucho más intuitivo la utilización de cada uno de los elementos a implementar. Así, alguien sin conocimientos de programación, pero con interés o una cantidad relativa de estudios en la materia, podría utilizarla cómodamente.

Teniendo también en cuenta que sería ideal que la aplicación fuese multiplataforma y la existencia de la *Java Multimedia Retrieval (JMR)* [**?**], librería desarrollada por Jesús Chamorro Martínez que resuelve de forma genérica gran parte de las funcionalidades solicitadas, se ha decidido utilizar Java como lenguaje de programación.

Sin embargo, se debe de tener en cuenta que este proyecto ha sido planteado bajo la premisa de la utilización de aprendizaje profundo. Este será utilizado para la extracción de características de nivel 2, mediante la utilización de una red neuronal convolucionada Capítulo 3 que realizará una segmentación semántica para obtener diversas categorías así como los píxeles concretos de cada imagen que pertenecerán a cada categoría [**?**].

Si bien existen bibliotecas para el aprendizaje profundo en Java, estas no están tan desarrolladas como podrían estarlo en otros lenguajes de programación como Python. Por ello, se utilizará Python 3 para la creación, el entrenamiento y la carga de los modelos que se lleguen a probar a lo largo del proyecto.

Esto plantea la cuestión de cómo resolver la comunicación entre los distintos lenguajes, que será resulta mediante la utilización de una conexión TCP.

14. Diseño

Como se mencionó en el capítulo anterior, la aplicación será diseñada en dos partes conectadas por una conexión TCP. Comenzaremos mencionando las librerías utilizadas para cada una de las partes, prosiguiendo con mostrar una serie de diagramas que mostrarán la estructura interna utilizará nuestra aplicación haciendo uso del material disponible.

14.1. Librerías utilizadas

Comenzaremos mencionando las librerías utilizadas para la creación, entrenamiento y carga de la red neuronal. Principalmente se ha utilizado:

1. Tensorflow [AAB+15]: se trata de una plataforma de código abierto de extremo a extremo para el aprendizaje automático. Posee una API para diversos lenguajes de programación como son Javascript y Python además de estar disponible tanto para ordenadores como para dispositivos móviles, en este caso bajo su versión Tensorflow Lite. Como extra, permite utilizar varias CPUs o GPUs, con la consecuente posibilidad de aceleración GPU, y ofrece soporte experimental para *Cloud TPUs* en Keras y Google Colab. En particular se utilizará Tensorflow 2.0 que es compatible con Python 3.5 a 3.8.

2. Keras [C+15] : es una API construida sobre TensorFlow 2.0 que, según su propia página web, "está diseñada para seres humanos, no para máquinas". Está optimizada para GPU, CPU y TPU.Fue concebida para actuar como una interfaz en lugar de un framework de machine learning standalone por lo que ofrece un conjunto de abstracciones más intuitivas y de alto nivel, haciendo más sencillo el desarrollo de modelos de aprendizaje profundo.

3. Classification models Zoo - Keras [cla]: se trata de un conjunto de clasificadores entrenados en el conjunto de datos ImageNet [DDS+09]. De esta forma podremos obtener clasificadores que no están disponibles ni en Keras ni en TensorFlow, en particular ResNet18 [HZRS15].

4. Imgaug [JWC+20] es una biblioteca para el aumento de imágenes en los experimentos de aprendizaje automático. Soporta un gran rango de técnicas de aumento de datos, permite combinarlas fácilmente y ejecutarlas de forma aleatoria en múltiples núcleos CPU.

Para el tratamiento de los datos resultantes de la red neuronal con el fin de adaptarlos para utilizarlos en la aplicación, se ha utilizado el ecosistema basado en Python SciPy, principalmente las librerías:

1. SciPy [VGO+20] librería fundamental para el cálculo científico.

2. NumPy [HMvdW+20] paquete de vectores n-dimensionales.

3. Pandas [pdt20] [WM10] estructura de datos y análisis.

4. Matplotlib [Hun07] trazado completo en 2D.

Además, se ha utilizado scikit-image [vdWSN+14] (o skimage) que es una colección de algoritmos para procesamiento de imágenes y la visión por computador.

Como se mencionó en el capítulo anterior, para como esqueleto genérico del sistema CBIR se utilizará la biblioteca *Java Multimedia Retrieval (JMR)* [**?**] sobre la cual desarrollaremos en más detalle los elementos que utilizaremos. En particular, estaremos utilizando una versión actualizada por Miriam Mengíbar Rodríguez para su trabajo de fin de master [**?**].

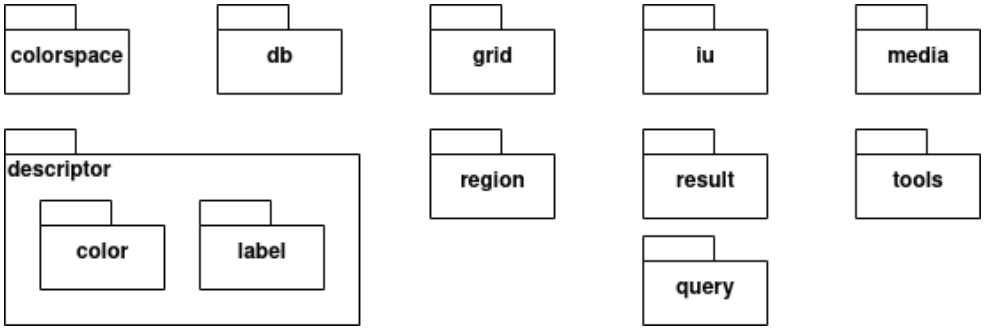

Figura 14.1.: Diagrama representante de algunos de los paquetes de la JMR. [**?**]

De los paquetes mostrados, concretamente se utilizarán:

- *db* : Paquete que simula una base de datos. Posee su propio elemento creado a partir de los datos de entrada y de los descriptores, compatibles con la comparación con las consultas y con una cómoda recuperación de los datos originales.
- *descriptor* : Paquete que engloba los elementos necesarios para una correcta representación de un descriptor o vector de características, como es el caso de la interfaz *Comparator* o la interfaz *MediaDescriptor*. Posee dos subpaquetes:
  - *descriptor.color* : Paquete que implementa descriptores de colores de acuerdo al estándar MPEG7 [**?**].
  - *descriptor.label* : Paquete que implementa un descriptor por etiquetas genérico así como definir las interfaces necesarias para la clasificación de las etiquetas.
- *region* : Paquete diseñado para la representación de una región contenida en una imagen. Si bien este paquete no se ha utilizado directamente, servirá como inspiración para la definición de la clase *MultiRegion* que se verá más adelante.

Además, también se utilizarán las clases *LabelGroup* y *TCPClient*, siendo esta última a través de la cual se consigue la conexión TCP con el módulo de python, ambas implementas por Miriam en su trabajo de fin de master [**?**].

14.2. Descriptores

A la hora de diseñar los descriptores, debemos de tener especial atención en no diseñar nada de lo que ya dispongamos. Por ello, no será necesario el diseño de descriptores de nivel 1 de tipo color, puesto que estos ya están diseñados e implementados a través de los paquetes *descriptor* y *descriptor.color* de la JMR [**?**].

El principal trabajo será el diseño e implementación de un descriptor de nivel 2 con su respectivo comparador. En particular, se ha diseñado el descriptor *RegionLabelDescriptor* que, dada una imagen, describirá cada una de las categorías pertenecientes a dicha imagen así como los píxeles concretos que pertenecerán a cada una de estas categorías. De esta forma, no sólo conoceremos qué etiquetas o categorías estarán presentes en la propia imagen sino que además la localización a nivel de píxel de cada una de estas categorías.

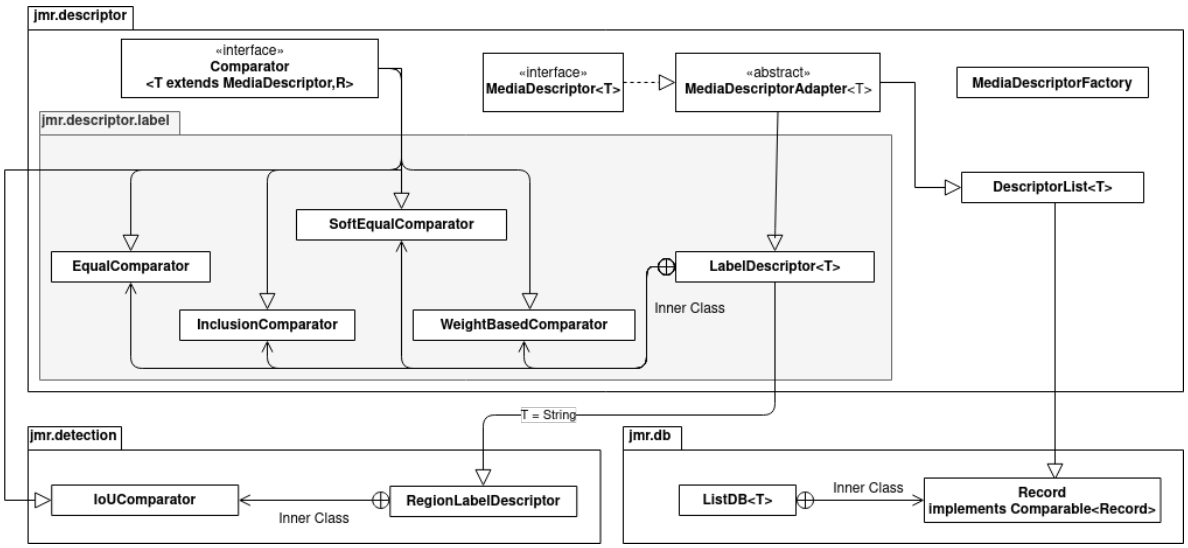

Figura 14.2.: Diagrama de herencia y clases internas para los descriptores y comparadores de etiquetas. [**?**]

Para la extracción de estas características para la creación de este descriptor, se ha definido un clasificador llamado *AttentionClassifier* que recibirá la url de la imagen cuyas características se desean extraer y, a través del uso de la clase *TCPClient*, se conectará con un servidor de python llamado *server.py* que segmentará semánticamente la imagen. Con esta información, almacenada en un objeto de la clase *RegionClassification*, se construirá correctamente un objeto *RegionLabelDescriptor* que podrá ser comparado utilizando los comparadores internos de su clase madre *LabelDescriptor* y el comparador *IoUComparator* que se definirá y creará para este descriptor concreto.

Un detalle importante a tener en cuenta, se trata de que, mientras que en los descriptores de color se definen con *T=BufferedImage*, para el nuevo descriptor que estaremos creando se utilizará *T=String* siendo este *string* la dirección donde esta almacenada en nuestro dispositivo la imagen representante. Se ha elegido este método para facilitar la conexión TCP,

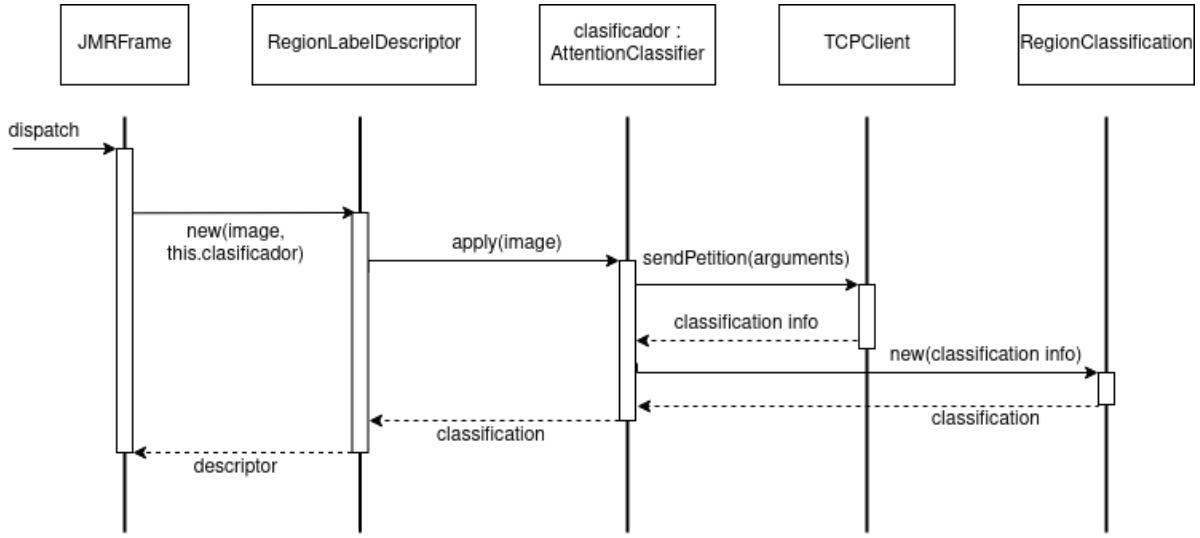

Figura 14.3.: Diagrama de secuencia que muestra la creación de un descriptor del tipo *RegionLabelDescriptor*

así como su velocidad, pero tendremos la desventaja de que no podremos crear una base de datos que contemple tanto descriptores de color como descriptores de regiones al estar trabajando sobre tipos distintos.

Resumiendo, se deberán de implementar las clases *RegionLabelDescriptor*, *AttentionClassifier* y *RegionClassification* para tener la funcionalidad del descriptor deseado. Además, también se implementará la clase *IoUComparator* para añadir una nueva opción de comparación entre descriptores.

En la sección de implementación, se describirá el funcionamiento de *IoUComparator* así como el funcionamiento y utilidad de otras clases desarrolladas para facilitar la comunicación con *JMRFrame*, en particular se tratará de la clase *MultiRegion* y de clases que tendrán la funcionalidad de renderizar la información de los cuadros combinados a implementar en la interfaz de usuario.

15. Implementación

Para la implementación con el lenguaje Java se ha utilizado el entorno de desarrollo *Apache Netbeans IDE 12.1*, sobre todo teniendo en mente el desarrollo de la interfaz gráfica que se ve sumamente facilitado gracias a las herramientas de la que dispone esta interfaz de desarrollo. Mientras que para el código en python y la propia memoria se ha utilizado el editor *Atom* que permite una comunicación interactiva con el repositorio de GitHub utilizado para este proyecto [**?**] en el cual se pondrá encontrar todo el código desarrollado.

Además, se podrán encontrar en dicho repositorio algunos de los múltiples entrenamientos de redes neuronales realizados así como cuadernillos de *Jupyter Notebook* que mostrarán los diversos detalles de estos. Así, se podrá ver de forma interactiva los detalles analizados, en caso de ser de interés, sin necesidad de extender más de lo necesario esta memoria.

15.1. Comparadores

Antes de proceder a explicar la implementación de la clase *IoUComparator* es necesario que se haga una pequeña pausa para comprender el funcionamiento de los comparadores de los ya existentes, en particular los pertenecientes a *LabelDescriptor* para asegurarnos tener una buena compatibilidad con ellos y poder comprender mejor la futura implementación de *IoUComparator*.

Dados dos descriptores, la imagen de la función de comparación entre ambos será $[0, +\infty]$, donde el resultado 0 indicará que bajo esta comparación son iguales y cualquier otro valor significará que son distintos indicando una medida de la diferencia de relativa a esos dos descriptores bajo dicha comparación. En particular, el valor $+\infty$ representará la máxima diferencia posible, también interpretable como incomparables.

Sabiendo que un descriptor de tipo *LabelDescriptor* posee un vector de etiquetas de tipo *string*, se dirá que un *LabelDescriptor* t está incluido en otro *LabelDescriptor* u si cada etiqueta perteneciente al vector de etiquetas de t esta incluida en el vector de etiquetas de u. En particular, si existen etiquetas repetidas en t bastará con que estas aparezcan una única vez en u para que se siga considerando como verdadera la relación $t \subset u$, o t incluido en u.

Dicho esto, podemos explicar el comportamiento de los descriptores sin pesos de *LabelDescriptor*. Sean t y u dos *LabelDescriptor*, el comparador

- *InclusionComparator* devolverá la igualdad, es decir, el valor 0 si $t \subset u$. Se retornará $+\infty$ si no se cumple la relación.

- *SoftEqualComparator* devolverá la igualdad, es decir, el valor 0 si $t \subset u$ o $u \subset t$. Se retornará $+\infty$ si no se cumple ninguna relación.

- *EqualComparator* devolverá la igualdad, es decir, el valor 0 si $t \subset u$, $u \subset t$ y tanto t como u poseen la misma cantidad de etiquetas. Se retornará $+\infty$ si no se cumplen ambas relaciones.

Por otro lado, si se le añade un valor de pesos no nulo a cada una de las etiquetas de un *LabelDescriptor*, por defecto, se utiliza el *WeightBasedComparator* que permite elegir entre la utilización de comparación con igualdad o inclusión de etiquetas, además de calcular el valor absoluto de la diferencia de pesos de las etiquetas. Se permite elegir entre devolver el máximo, mínimo, media aritmética o media euclídea de los resultados de cada valor absoluto de la diferencia de las etiquetas como valor real, teniendo en cuenta que se devolverá, además, $+\infty$ en caso de que este sea el valor devuelto por la operación de igualdad o inclusión seleccionada.

Una vez explicado esto, se procederá a realizar las explicación de la implementación del *IoUComparator* que ha sido desarrollado para este proyecto.

Comenzando, realiza una comparación por etiquetas utilizando *EqualComparator* o *InclusionComparator* según se desee, devolviendo $+\infty$ en caso de que este sea el resultado de esta primera comparación.

Si se obtiene un valor distinto, se procede a unificar temporalmente las categorías duplicadas de forma que las siguientes operaciones se realicen de forma independiente para cada categoría distinta. Seguidamente, para cada categoría se calcula la cantidad de píxeles que pertenecen a la intersección de esta categoría para el *RegionLabelDescriptor* t y el *RegionLabelDescriptor* u, de la misma forma se contabiliza para la unión y se divide el valor de la intersección entre la unión.

Con esta operación, obtendremos tenderemos al valor 0 cuantos menos píxeles coincidan en ambas imágenes y obtendremos el valor 1 si todos los píxeles de la categoría seleccionada coinciden. Sin embargo, estos valores no comparten el mismo criterio que compartían los comparadores de la clase madre, por lo que aplicaremos una transformación no lineal para seguir el mismo criterio. En concreto, se hará:

$$\frac{1}{IoU} - 1,$$

que devolverá 0 si ambas comparten los mismos píxeles en la categoría y tenderá a $+\infty$ cuantos menos píxeles coincidan.

Finalmente, de forma similar a como sucedía en *WeightBasedComparator*, se permite elegir entre el máximo, mínimo, media aritmética o media euclídea para obtener el resultado final a partir de los resultados parciales correspondientes a cada categoría, siendo este el valor devuelto para la comparación de los dos descriptores.

FIXME: Añadir un diagramaita que simplifique la explicación

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.

[AAB+15] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org. [Citado en pág. 63]

[AMMIL12] Yaser S. Abu-Mostafa, Malik Magdon-Ismail, and Hsuan-Tien Lin. *Learning From Data*. AMLBook, 2012. [Citado en pág. 9]

[aut] Varios autores. Convolutional neural networks for visual recognition. [Citado en págs. ix, ix, ix, 4, and 5]

[BCM05] Antoni Buades, Bartomeu Coll, and Jean-Michel Morel. A non-local algorithm for image denoising. In *Proceedings of the 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05) - Volume 2 - Volume 02*, CVPR '05, pages 60–65, Washington, DC, USA, 2005. IEEE Computer Society. [Citado en págs. 18 and 19]

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

[BSCD17] Navaneeth Bodla, Bharat Singh, Rama Chellappa, and Larry S. Davis. Soft-NMS – Improving Object Detection With One Line of Code. *arXiv e-prints*, page arXiv:1704.04503, April 2017. [Citado en pág. 26]

[C+15] Francois Chollet et al. Keras, 2015. [Citado en págs. 41 and 63]

[cla] Classification models zoo - keras (and tensorflow keras). [Citado en pág. 63]

[COC] Coco - common objects in context. [Citado en pág. 41]

[Cyb89] George Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of control, signals and systems*, 2(4):303–314, 1989. [Citado en pág. 11]

[DDS+09] 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. [Citado en pág. 63]

[GBC16] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. http://www.deeplearningbook.org. [Citado en págs. 8, 9, and 10]

[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. 25]

[Gir15] Ross Girshick. Fast R-CNN. *arXiv e-prints*, page arXiv:1504.08083, April 2015. [Citado en pág. 25]

[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. 26]

[HMvdW+20] Charles R. Harris, K. Jarrod Millman, Stéfan J van der Walt, Ralf Gommers, Pauli Virtanen, David Cournapeau, Eric Wieser, Julian Taylor, Sebastian Berg, Nathaniel J. Smith, Robert Kern, Matti Picus, Stephan Hoyer, Marten H. van Kerkwijk, Matthew Brett, Allan Haldane, Jaime Fernández del Río, Mark Wiebe, Pearu Peterson, Pierre Gérard-Marchant, Kevin Sheppard, Tyler Reddy, Warren Weckesser, Hameer Abbasi, Christoph Gohlke, and Travis E. Oliphant. Array programming with NumPy. *Nature*, 585:357–362, 2020. [Citado en pág. 63]

[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. [Citado en págs. IX and 31]

[HSW89] K. Hornik, M. Stinchcombe, and H. White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989. [Citado en pág. 12]

[Hun07] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007. [Citado en pág. 64]

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. [Citado en págs. IX, IX, IX, IX, IX, IX, X, X, 31, 33, 34, 35, 36, 37, 38, 39, 40, and 63]

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

[JE99] Margaret Graham John Eakins. *Content-Based Image Retrieval*. University of Nothmbria at Newcastle, 1999. [Citado en pág. 55]

[JWC+20] Alexander B. Jung, Kentaro Wada, Jon Crall, Satoshi Tanaka, Jake Graving, Christoph Reinders, Sarthak Yadav, Joy Banerjee, Gábor Vecsei, Adam Kraft, Zheng Rui, Jirka Borovec, Christian Vallentin, Semen Zhydenko, Kilian Pfeiffer, Ben Cook, Ismael Fernández, François-Michel De Rainville, Chi-Hung Weng, Abner Ayala-Acevedo, Raphael Meudec, Matias Laporte, et al. imgaug. `https://github.com/aleju/imgaug`, 2020. [Citado en págs. 41 and 63]

[KH91] Kurt and Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991. [Citado en págs. 11 and 12]

[KL19] Patrick Kidger and Terry Lyons. Universal Approximation with Deep Narrow Networks. *arXiv e-prints*, page arXiv:1905.08539, May 2019. [Citado en pág. 13]

[LDG+16] Tsung-Yi Lin, Piotr Dollár, Ross Girshick, Kaiming He, Bharath Hariharan, and Serge Belongie. Feature Pyramid Networks for Object Detection. *arXiv e-prints*, page arXiv:1612.03144, December 2016. [Citado en pág. 27]

[LMB+14] Tsung-Yi Lin, Michael Maire, Serge Belongie, Lubomir Bourdev, Ross Girshick, James Hays, Pietro Perona, Deva Ramanan, C. Lawrence Zitnick, and Piotr Dollár. Microsoft COCO: Common Objects in Context. *arXiv e-prints*, page arXiv:1405.0312, May 2014. [Citado en pág. 41]

[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]

[LPW+17] Zhou Lu, Hongming Pu, Feicheng Wang, Zhiqiang Hu, and Liwei Wang. The Expressive Power of Neural Networks: A View from the Width. *arXiv e-prints*, page arXiv:1709.02540, September 2017. [Citado en pág. 13]

[MRS08] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to Information Retrieval*. Cambridge University Press, USA, 2008. [Citado en pág. 55]

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

[pdt20] The pandas development team. pandas-dev/pandas: Pandas, February 2020. [Citado en pág. 63]

[PLCD16] Pedro O. Pinheiro, Tsung-Yi Lin, Ronan Collobert, and Piotr Dollàr. Learning to Refine Object Segments. *arXiv e-prints*, page arXiv:1603.08695, March 2016. [Citado en págs. IX and 29]

[RDGF15] Joseph Redmon, Santosh Divvala, Ross Girshick, and Ali Farhadi. You Only Look Once: Unified, Real-Time Object Detection. *arXiv e-prints*, page arXiv:1506.02640, June 2015. [Citado en pág. 26]

[RF16] Joseph Redmon and Ali Farhadi. YOLO9000: Better, Faster, Stronger. *arXiv e-prints*, page arXiv:1612.08242, December 2016. [Citado en pág. 27]

[RF18] Joseph Redmon and Ali Farhadi. YOLOv3: An Incremental Improvement. *arXiv e-prints*, page arXiv:1804.02767, April 2018. [Citado en pág. 27]

[RFB15] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-Net: Convolutional Networks for Biomedical Image Segmentation. *arXiv e-prints*, page arXiv:1505.04597, May 2015. [Citado en págs. IX, IX, IX, IX, IX, IX, X, X, 32, 33, 34, 35, 36, 37, 38, 39, and 40]

[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 págs. 25 and 27]

[Ten] TensorFlow. Image segmentation. [Citado en pág. 32]

[Tiu] Ekin Tiu. Metrics to evaluate your semantic segmentation model. [Citado en pág. 41]

[Tya17] Vipin Tyagi. *Content-Based Image Retrieval: Ideas, Influences, and Current Trends*. Springer, Singapore, 2017. [Citado en págs. X, 55, and 56]

[vdWSN+14] Stéfan van der Walt, Johannes L. Schönberger, Juan Nunez-Iglesias, François Boulogne, Joshua D. Warner, Neil Yager, Emmanuelle Gouillart, Tony Yu, and the scikit-image contributors. scikit-image: image processing in Python. *PeerJ*, 2:e453, 6 2014. [Citado en pág. 64]

[VGO+20] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. [Citado en pág. 63]

[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. [Citado en págs. 17 and 52]

[WGGH17] Xiaolong Wang, Ross B. Girshick, Abhinav Gupta, and Kaiming He. Non-local neural networks. *CoRR*, abs/1711.07971, 2017. [Citado en pág. 29]

[WM10] Wes McKinney. Data Structures for Statistical Computing in Python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 56 – 61, 2010. [Citado en pág. 63]

[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, 23, and 26]